CServ: An Internetwork Architecture Supporting Mission-Critical Messaging with Probabilistic Performance Guarantees

by

Matthew F. Carey

S.M. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2011

B.S. Electrical Engineering
Boston University, 2009

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2016

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
January 4, 2016

Signature redacted

Certified by .....
Vincent W. S. Chan
Joan and Irwin M. (1957) Jacobs Professor of Electrical Engineering
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering
Chair, Department Committee on Graduate Students

CServ: An Internetwork Architecture Supporting Mission-Critical Messaging with Probabilistic Performance Guarantees

by

Matthew F. Carey

Submitted to the Department of Electrical Engineering and Computer Science on January 4, 2016 in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering and Computer Science

ABSTRACT

We present the fundamental framework for a multi-service, heterogeneous internetworking architecture that provides probabilistic, end-to-end quality of service guarantees to the data application for short critical messages prior to transmission. The class of critical network messages spans many applications in the civilian and defense network realms, including command and control of crucial public infrastructure, early warning alerts, and military command dissemination. These network applications share a need for a high probability of successful delivery with stringent delay requirements. The IP Internet is not well-suited to bear mission-critical messages across administrative domain boundaries with these demands because IP is a purely best effort network service and Border Gateway Protocol intentionally obscures the detailed administrative domain state information needed to describe the performance of internetwork paths. Current solutions to critical messaging involve the quasi-static provisioning of circuit connections, an approach that is not scalable for general mission requirements. The Critical Service architecture presented in this work directly addresses the need to generate *a priori* probabilistic guarantees without explicit path reservation overhead in an on-demand and dynamic fashion.

The Critical Service architecture leverages a logically-centralized control plane, divorced from the network data plane, which aggregates a compact set of performance state information from participating networks. While these networks retain autonomy and routing control, we introduce the State Measurement Service necessary to characterize the intranetwork services offered by these networks to critical message datagrams. The learned performance measurements are aggregated and pruned by a local controller such that only the minimal required internal state is revealed to the centralized arbiter. Internetwork routing decisions for critical messages are made on a transaction-specific basis using this global state information and the specified service demands of the network application. An algorithmic method is developed that discovers internetwork paths and composes diversity-routed solutions that satisfy the requested minimum reliability and maximum tolerable delay bound performance requirements. A form of source routing using the computed internetwork service enforces the network-granularity routing solution.

Thesis Supervisor: Vincent W. S. Chan
Title: Joan and Irwin M. (1957) Jacobs Professor of Electrical Engineering

Acknowledgment

Despite the name on the cover, this thesis is not the product of an individual. It is only through the inspiration, guidance, and support of many that this work came to be, and I am honored to acknowledge those contributions here.

First and foremost, I owe the deepest gratitude to my research advisor, Professor Vincent W. S. Chan, for everything he has done for me during my tenure at MIT. His guidance served as the crux of my education. Not only did he help to inspire and shape this research, but helped develop my methodology and calculated manner of approaching all of life's challenges. From my first MIT semester as a pupil absorbing the basics in his Heterogeneous Networks class to my last MIT semester as a PhD student under his tutelage, I learned something new from our interactions each and every day. I feel that not only have I gained a great mentor through this process, but also a great friend.

The other members of my doctoral thesis committee panel served as very important contributors to my education and research as well. I offer my thanks to both Professor Anantha Chandrakasan and Alan Kirby for their patience, enthusiasm, and assistance throughout the process of forming and composing this dissertation. Without their expertise and advice, this work would not be nearly as complete or as intelligible. I also want to extend my gratitude to John Chapin, in absentia, for his help towards developing my clarity in preparation and presentation of research material early in my graduate student life.

I am thankful for the faculty both at MIT and at Boston University who have provided mentorship, support, and dedication to my education throughout the years. In particular, I acknowledge Professor Muriel Médard, Professor Robert Gallager, Professor Mark Horenstein, Professor David Starobinski, and the late Professor Sasha Taubin. These individuals both served as sources of inspiration and provided me opportunities to become acquainted with the excitement of pursuing the scientific and engineering unknown.

My time at MIT would not have been quite as enjoyable, productive, or mentally rewarding without my fellow researchers in the "Chan Clan." Throughout the years, my officemates Shane Fink and Manishika Agaskar have been a constant source of support, riveting discourse, and hearty laughter. All other members of the Communication and Network Group of the Research Laboratory of Electronics, including Henna Huang, Anny Zheng, Esther Jang, John Metzger, Andrew Moran, Antonia Feffer, Katherine Lin, David Cole, Lei Zhang, Andrew Puryear, Mia Yinuo, Lillian Dai, Etty Lee, Guy Weichenberg, Joseph Junio, and others, have similarly contributed to my academic development and made coming to the office every day a pleasing adventure. I am also indebted to Donna Beaudry for her assistance with logistics throughout the years and for her own delicious "research."

I am very appreciative of my "research cousins" across the hall in the Network Coding and Reliable Communication Group. Many of us began the doctoral program at MIT as a group, and they have pushed me to success through the desire to compare with their impressive feats, academic contributions, and buoyant spirits.

I am forever thankful for my other wonderful friends from MIT for keeping me sane and becoming lifelong companions in the process (some going so far as to serving as groomsmen in my wedding), especially Bridget Sharei, Collin Mechler, Anisa Mechler, and Ryan Thurston, among everyone else.

Finally, I would like to recognize the most important people in my life: my family. My parents, Kathleen and Francis, instilled in me the importance of education and a passion for knowledge at a very young age, ultimately making this academic journey possible. More importantly, they have given me their unconditional love and encouragement throughout my life, always supporting my varied aspirations and goals. I am lucky to have such a great brother, Robert; I could not ask for a better sibling or friend. I am also blessed to have recently gained a second family, including Isaac, Jean, Ian, Emily, William, and Helena. They have been my sanctuary away from home, accepting and supporting me from our first meeting. And even more significantly, they have trusted me with the hand of their charming and gorgeous daughter, Taryn.

It is to my wife, Taryn, that I dedicate this dissertation. On September 13, 2015, as I was finishing this research and writing this document, I had the honor and pleasure of marrying my best friend and soulmate. I will forever be thankful for her companionship in my life. Without her love, patience, and support, this would not have been possible.

Matthew F. Carey
Cambridge, Massachusetts

The research in this thesis was funded, in part, by the U.S. Defense Advanced Research Projects Agency (DARPA) and the U.S. Office of the Secretary of Defense (OSD). Without this support, the work presented herein would not have been possible.

The views expressed in this article are those of the author and do not reflect the official policy or position of the U.S. Department of Defense or the U.S. Government.

Contents

Chapter 1

Introduction

The modern Internet is one of society's most important complex engineering systems. It facilitates communication and the exchange of ideas between individuals separated by vast distances, both nationally and intercontinentally. Initially designed as a platform for file exchange and text-based communication between a subset of academic institutions as ARPANET [1] (see Fig. 1-1 for the logical network map circa 1977), the Internet has evolved in unpredictable ways to support many well-known communication services, reshaping them along the way. The world has gradually moved from print to digital publishing, where news stories and magazine articles are increasingly accessed through a vast set of interconnected hypertext documents known as the World Wide Web (WWW). Traditional telephony has evolved into Voice over IP (VoIP), eschewing switched circuit-based service. Television broadcast and on-demand video services are now delivered via IP Television (IPTV), an Internet service that embodies the aggressive growth of Internet data traffic. In 2011, Internet video traffic, including web-based video, IPTV, on-demand video, and peer-to-peer (P2P), amounted to 51 percent of all consumer data traffic, and it is forecasted to encompass 86 percent of consumer data traffic by 2016 [2].

What is fascinating is that a system engineered to provide file exchange services has been successfully extended to support such varied applications with different service demands. Consider the service demands of file transfer compared to that of voice or streaming video delivery. File exchange, such as email service, requires high reliability. A file may be segmented and transmitted in multiple datagrams, but the requirement is that all of these datagrams arrive at their destination without error. A single datagram lost or received in error could result in the corruption of the file. However, we do not impose a strict time deadline on the transfer of the file; the file may arrive at the destination within seconds or

it may take minutes or longer. On the other hand, voice and video streaming impose nearly the opposite demand on the network. Voice and video encoders create datagrams in such a way that not all of them need to arrive at the destination to hold a conversation or successfully play a video (generally, the quality of the service is allowed to gracefully degrade as fewer datagrams in a stream arrive successfully at the destination). In this way, these services do not place strict reliability requirements on the network. However, there is a stringent delay requirement on each encoded datagram. For voice, there is a fixed deadline for the reception of some datagrams in order to avoid unnatural speaker interruption which disrupts the conversation. In the case of streaming video, there is no use for a datagram after its scheduled playback deadline. In spite of their differences from the exchange of text files, both of these services have thrived and driven the use of the Internet. In no small part, the use of the Internet to deliver these services is expected to push global Internet traffic beyond a zettabyte per year in 2015 [2].

At its heart, Internet Protocol (IP), the convergence network layer of the packet-switched Internet, provides a "best-effort" datagram delivery service. The protocol specification explicitly acknowledges that the following problems may occur during the delivery of datagrams:

- duplication of datagrams;
- delayed or out-of-order delivery;
  data corruption and datagram loss [3].

These errors and losses necessitate the Transmission Control Protocol (TCP), an endpoint belt-and-suspenders endpoint transport layer that enforces the reliable transfer of information across the unreliable network and its underlying physical substrates. Based on the tenets of the Internet architecture, the intelligence of the system is expected to reside at the endpoints rather than within the network itself. The three functions of TCP are:

1. to provide end-to-end reliability through a datagram retransmission protocol, typically called an Automatic Repeat Query (ARQ) protocol;
2. to match rates between the source and destination hosts through a dynamic, variable-sized transmission window (admission control);
3. and to work in conjunction with IP to manage network congestion and enforce a notion of fair allocation of the network's shared resources.

We do not cover the functionality and implementation of TCP here in detail, but a thorough overview is provided in [3]. The TCP/IP protocol suite was designed with the early driving applications of the Internet in mind, such as file transfer, focusing on the reliable delivery of all datagrams in a transaction without enforcing strictly specified time deadlines.

Emergent applications, such as video and voice, with wildly different service requirements have been designed to use the TCP/IP Internet as their transport, even though this system is not engineered to provide their ideal service class. Since the Internet layer is a "dumb" best-effort service and the intelligence resides in the transport layer, these new applications only require software modifications at the endpoints rather than network infrastructure upgrades. In this sense, TCP/IP has been immensely successful, enabling rapid innovation without fundamental Internet architecture changes.

However, we consider a class of network applications in this dissertation that require *both* guaranteed highly reliable message delivery and guaranteed stringent time deadlines, demands that the current Internet architecture is not designed to meet simultaneously. What kinds of data applications require guarantees on both of these service demands simultaneously? We begin by considering some non-exhaustive application examples from the civilian network space:

- **High-frequency automated algorithmic trading** – High-frequency trading is a strategy that uses automated algorithms to hold very short-term positions in financial instruments with electronic trading capabilities [4]. A lost or unexpectedly delayed trade execution could cost a trading firm millions in profit. We consider this application in more detail in Section 1.1.

- **Emergency alerts** – Sensor systems can be deployed to detect and provide warning for natural disaster, such as monitoring for seismic activity that precedes an earthquake. These networked sensors can generate emergency warning messages that can be used to proactively mitigate the impact of such a catastrophic event. However, if the alert message is lost or delayed, the information may not reach decision makers or countermeasure systems in time, if at all.

- **Command and control of crucial infrastructure** – The national power grid is becoming an increasingly networked and automated system (the "smart" grid), where control is often enacted from operations centers [5]. Critical command messages to the grid's supervisory

control and data acquisition (SCADA) system or distributed intelligent power creation, harvesting, and distribution agents are of utmost importance to the success of this networked infrastructure. Should these command messages be delayed or lost, particularly if they are generated in response to the detection of anomalous events, the results on the system could be ruinous and jeopardize national security.

We can also consider some examples from the defense network space, since many military operations are becoming increasingly reliant upon the network for critical communications [6]:

- **Leadership command dissemination** – The delivery of orders down the military chain of command from human to human is crucial to the success of military operations. In the modern military, this communication may occur using networked electronic devices. We consider that the loss or delay of a time-sensitive command message may result in the desynchronization between military units that are engaged in a joint effort.

- **Fire-control systems** – A fire-control system requires the coordinated interaction between a gun data computer, a director calculating trigonometric firing solutions, and a radar tracking mechanism to facilitate the targeting functionality of a weapon. The networked messages between these subsystems require strict reliability and bounded time deadlines in order to ensure that the projectile accurately meets its intended target.

- **Tactical early warning alerts** – These detection systems and consequent early warning alert messages are much like those described for the civilian network above. An example in this space might be a satellite constellation that actively scans for indication of a missile launch and reports this event to military command, as illustrated in Fig. 1-2. A successful launch detection and alert allows for the deployment of countermeasures, whereas a lost or delayed early warning alert message can result in devastating casualties.

These examples are by no means an exhaustive coverage of the applications that require both highly reliable message delivery and stringent time deadlines simultaneously. We consider the defense network space in more detail in Section 1.2.

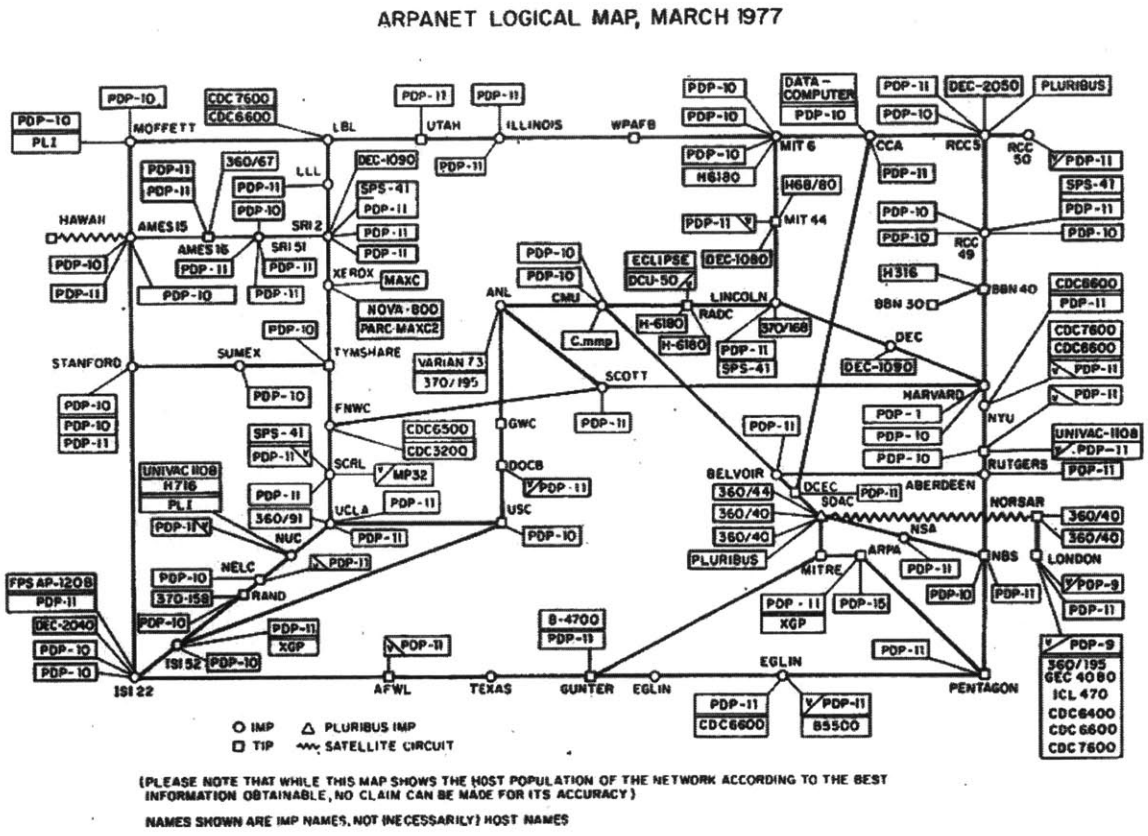


Fig. 1-1: Reproduced from The Computer History Museum, this map shows the logical interconnection network between hosts participating in the ARPANET as of March 1977. Considering that the modern Internet developed from this project, it is worth noting the stark contrast in scale between this network and the Internet which now reaches nearly one-half of the world's population [7].

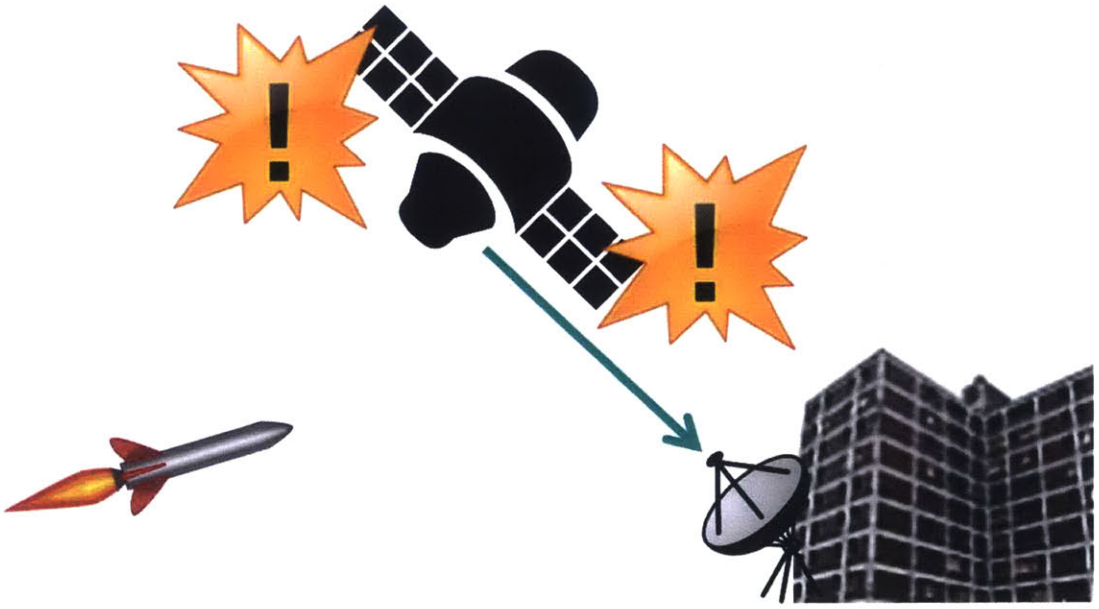


Fig. 1-2: This cartoon illustrates tactical early warning alert generation, an application in the defense network space requiring simultaneous stringent reliability and delay deadlines for the resulting data message.

1.1 Service Demands of High-Frequency Trading

As previously introduced, high-frequency trading is a trading strategy that uses automated algorithms to hold short-term positions in financial instruments with electronic trading capabilities [4]. At the heart of high-frequency trading is the objective to exploit securities volatility, sometimes to the tune of fractions of a cent on the trade [4]. Although this represents a small margin, firms balance this small margin with high trading volume. To take advantage of slight price variations, these strategies move in and out of positions rapidly.

Firms generally disaggregate trade flow into components, such as application, middleware, server, and network, and attempt both to optimize and characterize latency in each functional area through microsecond-accurate measurements [8]. While high-frequency transactions used to have an execution requirement of several seconds, the requirement has decreased by 2010 to milliseconds or even microseconds [9]. Depending on the transaction, a network data packet indicating buy or sell that is lost or delayed in the network could cost a high-frequency trading firm millions in profit.

Because the current public Internet architecture is deemed not well-suited to bear these trade executions with guaranteed reliability and delay, trading firms currently purchase expensive highly over-provisioned network services (capable of satisfying peak demand), reserving resources that are dedicated to their company's transactions and not multiplexed with exogenous data traffic. Headquarters where high-frequency transactions are initiated are also established as close as possible to trade servers to avoid the need for any network routers in the communication path between the endpoints, as depicted in Fig. 1-3. Reliability and delay performance guarantees are then enforced through carefully-constructed service level agreements with the network service provider and lengthy legal battles [10], not through technological means. A network architecture designed to natively provide reliability and delay guarantees simultaneously for these types of transactions and messages would likely represent a much more cost-efficient solution for the algorithmic trading firm compared to these approaches.

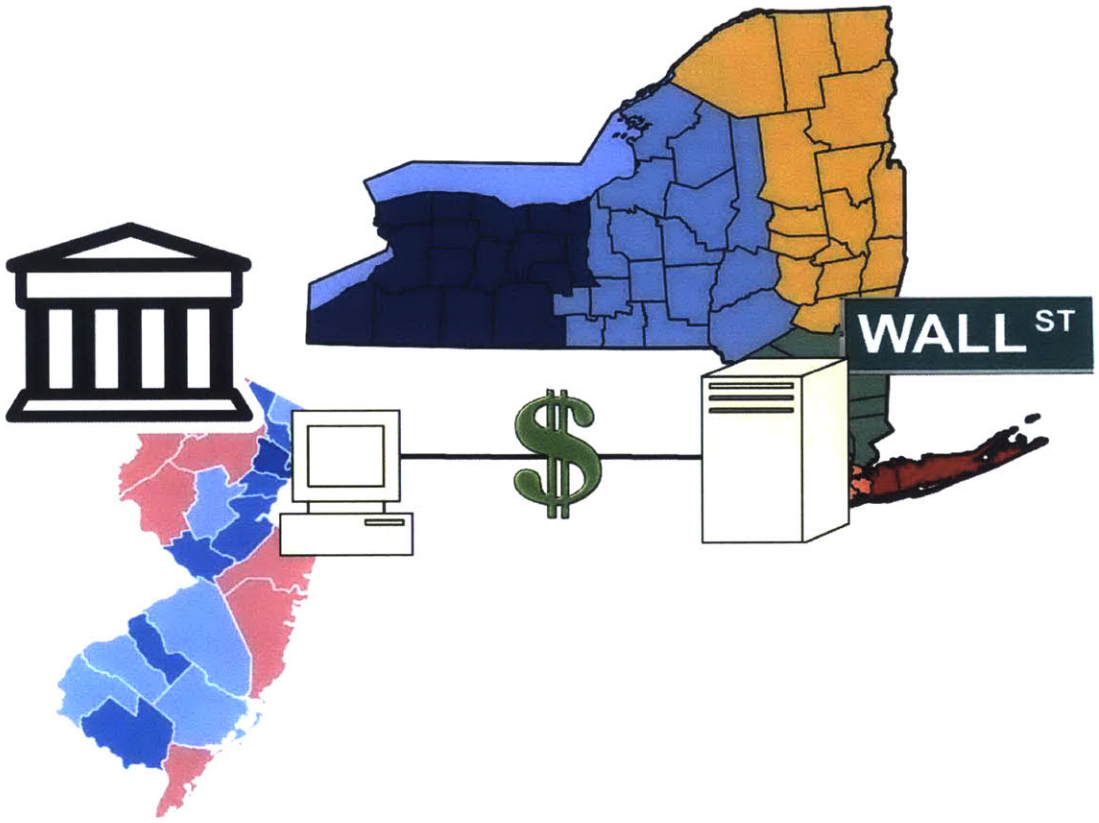


Fig. 1-3: This illustration shows the typical network approach of the modern day high-frequency trading firm. The financial transaction center is set up geographically close to trade servers and the company purchases highly-overprovisioned network circuits between them to circumvent routers and exogenous data traffic flow.

1.2 Service Demands of the Tactical Defense Network

Tactical defense data networks are highly heterogeneous collections of internetworked systems spanning multiple physical communication modalities, including fiber optics, free space optics, SATCOM, and tactical edge wireless networks. Many aspects of military operations are becoming increasingly reliant on these data networks [6] even though they do not currently provide data delivery assurances as they typically implement IP/TCP protocols and services. Service interruptions due to datagram loss or network disconnection could have severe operational consequences. In addition to benign sources of data loss and corruption that most civilian networks must handle, such as link failures and network congestion, military networks must bear the additional burden that they are high value targets for adversarial action and thus often subjects of both kinetic and cyber-attack.

In [11], the authors identify a set of defense network applications that they describe as "connectivity-centric" military applications, or those which require a network service that maximizes the probability of on-time delivery to mission-critical messages. In Fig. 1-4, we reproduce a tabular summary of the identified defense applications from this work and their identified differentiation in service demands compared to the Internet's best-effort network service (referred to as the "high-capacity service" in [11] and in Fig. 1-4). The specifications in Fig. 1-4 are estimated application communication needs, not precise or official requirements.

Consider the applications identified in Fig. 1-4, several of which were previously identified in this introduction chapter. These applications generally want to transmit small messages, on the order of one kilobyte, with strict time deadlines, on the order of seconds. We had previously mentioned leadership command dissemination as one of the defense network applications that requires simultaneous high reliability and stringent delay requirements. The authors of [11] identify that this network application would like the delivery of the top priority message of one kilobyte within one second. They also identify a second priority attachment that may accompany the high priority command message, but we focus on the top priority data. Next consider the emergency alert application, also previously mentioned in this introduction as an application in this space. The authors of [11] also identify this application as one that would like the delivery of a critical message of one kilobyte within one second. More applications are described in Fig. 1-4, most of which are seeking the delivery of short messages on the order of one or several kilobytes within a few seconds. The primary distinction between the network service that these

applications require compared to those that only need best-effort network service are that these applications want "very high" reliability and "hard deadlines" for the transaction delay.

However, we have indicated that the military already relies upon the data network for many of these operational applications, even though the modern IP/TCP Internet does not provide the reliability and delay assurances strictly required. What are the current approaches in the defense network to meeting these application needs? Much like the solutions to the "high-connectivity" service in the high-frequency trading space identified in Section 1.1, the defense network currently utilizes techniques that isolate traffic flows and overprovision resources in order to create guaranteed network service between endpoints. Specifically, many military communications leverage pre-configured, dedicated circuits between endpoints. As the terminology indicates (it is rooted in the telephony network heritage), circuits are provisioned and reserved end-to-end paths that offer predictable communication delay and constant rate. Through reservation and provisioning, circuit-oriented communication isolates the traffic from the effects of multiplexing datagrams with exogenous traffic which introduces delay variability and possibly datagram loss. Further, the use of excess resource headroom through heavy overprovisioning minimizes the occurrence of data corruption or loss during flight.

The problem with these approaches is that they offer very quasi-static communication channels between endpoint hosts. After hefty manual configuration to plan operations and allocate network circuits between critical entities, the network does not dynamically acclimatize to a changing operational picture that diverges from the original plan. In the very fluid modern operational theater, especially at the tactical edge, this lack of adaptation can prove disastrous for the mission if conditions veer into the unexpected.

App	Priority 1 Message	Priority 2 Attached data	Time deadline
Command Dissemination	1 KB	100 KB (image/map)	1 sec
Emergency Alerts	1 KB	100 KB (image/map)	1 sec
Op. Coordination	1 KB	0	4 sec
Air Asset Retask	10 KB	0	60 sec
Blue Force Track	0.2 KB	0	1 sec
Over Air Rekey	5 KB	0	20 sec
Network Crisis Recovery	1.5 KB	1 MB (e.g. logs/counters)	4 sec

Attribute	High-Capacity Service	High-Connectivity Service
Rate	Moderate to very high	Low to moderate
Delay	Best-effort: normally less than application timeouts	Hard deadline e.g. 1 sec
Reliability	Best-effort	Very high
Security	Minimal	Very high
Cost/MB	Very low	High if need be

Fig. 1-4: These tables describing the "high-connectivity service" and a set of defense network applications in that service space are reproduced from [11] with added emphasis on the service attributes central to the discussion in this dissertation.

1.3 Mismatch of TCP/IP Internet to the Desired Service

The TCP/IP Internet does not provide the network behavior or functionality that we need to guarantee reliability and delay performance for the classes of applications previously described. We discuss a few sources of the suitability mismatch here, but this is by no means an all-inclusive consideration.

The IP network service uses shortest path routing to determine the path between source and destination based on minimizing some cost metric that is determined by the network administrator. This often means that the endpoint application has no knowledge of the reliability or delay performance of the chosen path to the destination. If the datagram transaction crosses Internet network domains, this concern is further complicated by the fact that different network administrators may select different cost metrics for routing purposes. Thus, there is no guarantee that the path used for transmission of the datagram is the most reliable end-to-end path or the minimal delay end-to-end path. Even if the path happens to be one or both of these, there is no way for the network to relay or quantify that for the endpoint application in order to determine if the path performance meets the application service demands (such as those discussed in Sections 1.1-1.2).

Although the TCP transport layer provides a retransmission mechanism to achieve eventual end-to-end transaction reliability, it does not account for the urgency of the message in that retransmission; there is no distinction between a message that requires high reliability and stringent delay deadline service or the best-effort message that just needs to get to the destination at some future time (such as an email exchange). The retransmission of a lost or corrupted datagram is treated just the same as the original datagram transmission, regardless of the datagram class. Furthermore, since there is no quantification of the end-to-end reliability of the network path, the first transmission and subsequent retransmissions of the datagram could be over a path with very low reliability or it could be over a very reliable path. Retransmission over the same low-reliability path is unlikely to result in the high-reliability and stringent end-to-end delay service required by the class of data applications that we are interested in. There is no *a priori* understanding of the performance of the path or the transmission, hindering the IP network from providing reliability or delay guarantees to the application.

1.4 Border Gateway Protocol: the Internetworking Impediment

There is an additional impediment to providing guaranteed service in the modern TCP/IP Internet, and that is the standard internetwork routing protocol, Border Gateway Protocol (BGP) [12]. The Internet is composed of interconnected network systems, called Autonomous Systems (ASes). According to [13], the AS "is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy" that it presents to the Internet. As part of the IP protocol suite, BGP enables the forwarding of datagrams across administrative boundaries allowing one AS to exchange datagrams with another. To do so, BGP is used to exchange IP address prefix reachability advertisements between neighboring or peering ASes. These advertisements specify the destination addresses reachable by way of that AS network and, if applicable, the path of ASes through which the datagram would traverse to ultimately reach the destination prefix. Thus, if it so chooses, an AS can re-advertise a path to a particular address prefix after receiving a reachability advertisement for that prefix from its neighboring AS peer. An internal version of BGP is further employed to distribute the information from these reachability advertisements within an AS. Through this protocol behavior, the federated Internet appears to the source endpoint application as one large network where any IP address prefix is universally reachable.

As part of a BGP design tenet, the reachability advertisements contain no more performance state information than the descriptive AS-to-AS path to the advertised destination prefix. They do not provide details about the delay of that internetwork path or its reliability. Although individual network providers and administrators have full visibility into their detailed network telemetry (up to the point that their network equipment allows them), they are reluctant to share the detailed state of their network with AS peers. This reluctance stems from economic incentives; if a network administrator knows the state of its competitor, it can leverage this knowledge to suck data traffic load to its own network and generate additional revenue. Alternatively, a network provider may not want to reveal to its competitor the detailed internal state of its network when it is oversubscribed and struggling to keep up with demand. For these reasons, BGP intentionally obscures the exchange of network state information between neighboring networks. Instead, the protocol is designed to enforce configurable local policy decisions and enable fair competition between network service providers.

In terms of AS internal state exchange, at most BGP allows for the use of a Multi-Exit Discriminator (MED) attribute to be attached to the advertised AS path. The MED is an abstract value with no concrete physical interpretation; it does not represent any specific path performance metric, such as reliability or delay. It is used to indicate to an AS neighbor the preferred peering connection if there are multiple peering connections between the two ASes. This attribute is generally only used by an AS neighbor if it has no *local* preference for one peering connection or another; it may be completely ignored by the AS receiving the route advertisement if it does not favor its own intra-AS routing policy. Furthermore, this attribute is non-transitive. The AS receiving the route advertisement does not include this MED attribute with the AS path if it chooses to re-advertise the BGP route to another of its AS peers. If it includes a MED attribute with this re-advertisement, it chooses its own MED value based on its preferred peering connection with the AS neighbor it re-advertises the BGP path to. This behavior means that the MED attribute cannot be leveraged to reliably pass network state information between ASes even if desired. Ultimately, there is no global state information available across administrative boundaries in the Internet due to the intentional obscuration of state by BGP [14], and this inhibits the desire to make performance guarantees to the identified class of data applications with simultaneous high reliability and hard delay deadline requirements.

In order to make performance guarantees to the interesting class of applications, we need some understanding of the performance of end-to-end paths. Many of these paths, especially in defense networks which involve the interconnection of many disparate, heterogeneous systems, involve internetwork hops between administrative domains. With the IP Internet and the BGP internetwork routing protocol, we are hamstrung by the lack of state knowledge of the constituent networks and their advertised paths. Although the obscuration of state is sensible in some civilian networks that involve competition between network providers, BGP is not necessary the correct internetwork solution in other network deployment scenarios, such as with tactical defense networks. In the case of defense networks, there is often only one "network provider," the national interest. For this reason, state obscuration by default is not necessary the best approach to internetworking when the network aims to provide guaranteed service. Even with the exposure of minimal local network performance information, useful end-to-end performance metrics can be composed for network data applications requiring rigorous service levels.

1.5 Proposals for Dynamic Service-Oriented Networking

In Sections 1.1-1.2, we discussed some current approaches to guaranteed service networking in the civilian and defense network domains. These approaches, such as static planning and provisioning, dedicated circuit-oriented communication, and traffic isolation, represent static or quasi-static service solutions that cannot evolve and adapt at the speed of the network. In many operational scenarios, network allocation and pre-configuration is impossible to predict with complete accuracy. Particularly in the scope of military networks, fluid operational pictures make it challenging to plan communications between all necessary parties. In this section, we consider some existing proposals and techniques for dynamic service-oriented networking. The discussion of these approaches elucidate the current space of networking architectures in this area and highlight the lack of an architecture that provides explicit end-to-end internetwork performance guarantees of the nature required by the class of applications we have introduced, namely those that require datagram transmission with simultaneous high reliability service and stringent delay deadlines.

1.5.1 Dynamically-signaled Virtual Circuits

The existing networking technique that gets closest to the goal of providing guaranteed end-to-end service is that of dynamically-signaled virtual circuits (VCs). VCs are end-to-end packet-switched paths between source and destination hosts that mimic permanent circuit-oriented communication by reserving network switching and transmission resources along the path and installing VC label state on the intermediate routers. The design of early VC technologies reflected the interest of telephone service providers to use them to carry real-time audio and video traffic over datagram networks [15]. Like permanent connection-oriented circuits, VCs provide guaranteed service by establishing virtual bit pipes between hosts that present fixed transmission rate and well-characterized delay and delay jitter to the endpoints. However, unlike permanent circuits that rely heavily on manual configuration, virtual circuits use signaling protocols to dynamically establish VCs, reserve network resources, and install the necessary router state on-demand. The dynamic signaling of VCs generally requires a round trip worth of communication over the path to reserve and confirm the reservation of the path resources. Once a VC has been set up, the communicating hosts can exchange data with some guaranteed level of service that can be characterized in terms of its end-to-end reliability and delay performance.

There are many network architectures and mechanisms, past and present, which implement VCs on a connectionless datagram network, including X.25 [16], Frame Relay [17], ATM [18], and Multiprotocol Label Switching (MPLS) [19]. MPLS is one of the more ubiquitous VC technologies in the modern network, encapsulating various network protocols (including IP) and using short label state information to route labeled datagrams over reserved paths rather than complex routing table lookups. Additionally, there are several standardized protocols for signaling dynamic MPLS VC tunnels, including the Label Distribution Protocol (LDP) [20] and the traffic engineering extension to the Resource Reservation Protocol (RSVP-TE) [21].

We consider that the manual pre-configuration of permanent circuits between every pair of possible communication peers in a network is infeasible for reasonable size networks. This level of configuration and management overhead becomes intractable for networks of reasonable size. Additionally, as the number of hosts in the network increases, the required number of permanent circuits grows quadratically. Instead, can we use transaction on-demand dynamically-signaled VCs between internetwork communication endpoints to generate guaranteed service paths for the interesting class of data applications? If we assume that all network domains in the internetwork path support the establishment of inter-domain VCs (which is not necessarily a given), consider the following internetwork example illustrated in Fig. 1-5. In this instructive toy example, using a defense network framework, we assume that military headquarters wants to disseminate a critical command to an officer in the field with guaranteed end-to-end delay of at most one second (see Fig. 1-4 for this service requirement). The network path that connects the command base to the officer involves the transit of multiple network domains, including transmission over a ground fiber optics backbone connecting the base to a satellite ground terminal, a SATCOM relay in geosynchronous orbit, and a tactical edge wireless hop. Considering the one round trip time required to dynamically reserve the tunnel, this VC establishment phase may at minimum require two-thirds of a second due to the geosynchronous satellite relay hop which can easily contribute up to one-third of a second delay in each direction. Ignoring the transmission delays in the other network domains in the path, this setup time plus the command message transmission delay is already riding the very ragged edge of the one second service requirement. This example demonstrates that dynamically-signaled VCs are not necessarily feasible service solutions in the domain of some of the driving data applications that we are interested in. We would like a dynamic solution that avoids this per-transaction path round trip time for service setup.

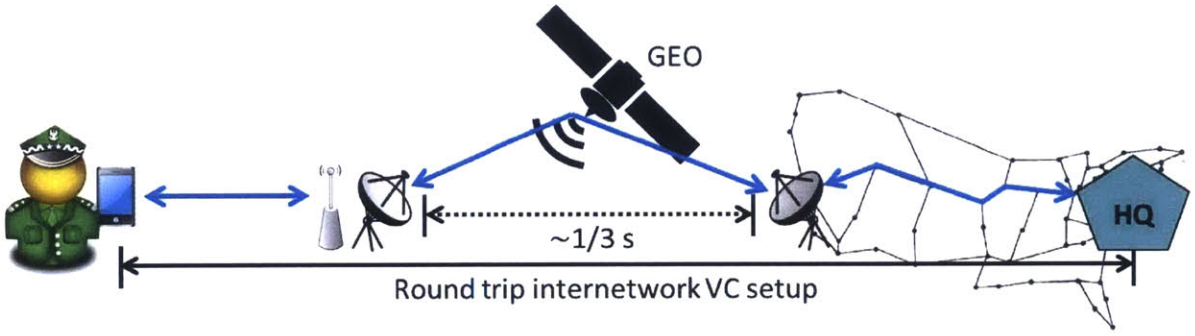


Fig. 1-5: The round-trip time required to establish a dynamically-signaled internetwork VC prior to data transmission is limited by the delay of the path, which in this example is dominated by the SATCOM relay in geosynchronous orbit (and can be on the order of 1/3 second).

1.5.2 QoS-modified Border Gateway Protocol

In [22], the authors propose to add Quality of Service (QoS) state information as attributes to BGP advertisement update messages to supplement the specified AS path information, addressing the opaqueness of the protocol as described in Section 1.4. To quantify and exchange QoS state, the proposal requires that interior gateway routing protocols (IGPs), such as Open Shortest Path First (OSPF) [23] [24] or Intermediate System to Intermediate System (IS-IS) [25] [26], are QoS-aware and implement traffic engineering extensions (e.g. OSPF-TE [27] and ISIS-TE [28]). Their internetwork routing protocol modification includes several enhancements to BGP that enable inter-domain routes to be selected by an AS using the local QoS-aware IGP state and BGP advertisements from peers with summarized QoS state for the advertised paths. These inter-domain routing decisions can be based on path mean delay, minimum bandwidth, and datagram reliability. The selected routes are then enforced either by explicit source routing [29] or inter-domain dynamically-signaled MPLS forwarding.

Although this proposal enables internetworking with some performance quantification, a feature we are unable to achieve with BGP alone, it directly violates the privacy design precept of the protocol by exchanging internal network performance state directly with AS peers. After the receipt of a BGP route advertisement with attached performance attributes for that AS path, an AS neighbor has learned some internal state of a network that it may be in competition with. We would like to avoid directly exposing internal network state to neighboring networks in a service solution for the interesting class of applications.

Additionally, since BGP is a policy-oriented protocol, an AS must trust that it is receiving advertisements for the best AS paths from its peers, or at accurate performance metrics for the paths. If the objective of the network is to offer guaranteed inter-domain service that meets the performance demands of the interesting class of data applications, this approach does not afford much opportunity for path choice or flexibility in service composition. If a BGP neighbor advertises one AS path for a particular destination prefix with one set of QoS state, that may be the network's only known option for reaching the intended destination even if other options exist. Should the performance of this advertised path not meet the needs of the data application, the network receiving this advertisement is not aware of any other opportunities to reach the destination prefix with superior end-to-end performance. An ideal

architecture would be aware of the breadth of opportunities for composing end-to-end internetwork paths, which would allow for efficient routing and decision making at the global scale.

Once a path is chosen, even if its one-way transmission delay does meet the needs of the data application, dynamically-signaled inter-domain VCs may still be necessary to enforce the selected internetwork path. This means that we are potentially back in the same situation as identified in Section 1.5.1 where the per-transaction round trip overhead associated with VC setup precludes the ability of the tunnel to ultimately satisfy the performance requirements.

1.5.3 Differentiated Services

Differentiated Services (DiffServ) [30] is a coarse-grained, class-based mechanism for traffic management that is integrated with IP. This technique uses IP-based QoS markings on individual datagrams to specify router per-hop behaviors, indicating the level of treatment that datagram should receive compared to those of other classes. The details of how individual routers deal with DiffServ markings are configuration specific, but they may involve prioritized queueing and switching or guaranteed rate service for a particular service class. This service-oriented networking mechanism does not entail any per-transaction setup or resource reservation. As long as routers in the network are DiffServ capable and have stored pre-defined per-hop behaviors for a given class marking, the DiffServ datagram needs only to be marked by the source or network edge router as a member of that traffic class to receive treatment as such.

The issue with this approach to service-oriented networking is that it does not provide any end-to-end guarantees. Marking a datagram as a member of a particular DiffServ class indicates to the routers that the sender wants the datagram to be treated with some priority compared to the other markings. The actual implementation and realization of this prioritization in the network is not assured. Since datagram treatment is configuration specific, one router may treat a class of datagrams completely differently than a neighboring router, or it may not necessarily respect the DiffServ class marking at all. This lack of consistency makes it difficult to predict or characterize end-to-end behavior within a network domain, let alone inter-domain end-to-end performance. Ultimately, the DiffServ architecture is useful to indicate relative datagram priority, requesting that a class of traffic receive a distinguished level of treatment with respect to another.

1.5.4 Software-defined Networking

Software-defined networking (SDN) is an emerging research and development area in the service-oriented networking domain. The scope of this area is fairly broad and often difficult to capture succinctly and inclusively, particularly as the field is experiencing rapid growth among both the research and commercial communities. It is generally accepted that there are two characteristics that define SDN architectures:

1. the decoupling of the network control and forwarding planes;
2. and the programmability of the control plane [31].

Neither of these characteristics is particularly new in and of itself; the novel aspect of SDN comes from the fact that it provides programmability *through* the decoupling of the network control from the data plane. In the extreme, intelligence previously required at the router can be moved to the network controller, allowing for simplification of the switching device architecture. All routing decisions can be made by the logically-centralized control plane, and the data plane is only required to enforce the rules composed by this entity. The logically-centralized controller is exposed as a programmable interface, introducing flexibility into the offered network services. With such a general definition, it is impossible to describe the scope of SDN in this introduction chapter. We refer the reader to [31] and [32]; these surveys discuss the SDN concept, the benefits, the challenges, the security concerns, and some of the previously proposed architectures.

We identify the SDN research area as an attractive opportunity for providing end-to-end internetwork performance guarantees. The existence of a logically-centralized control plane introduces the opportunity to aggregate necessary performance state from disparate network domains to characterize end-to-end inter-domain path behavior without the need to expose this information directly to neighboring networks. However, most existing SDN proposals have focused on other applications, such as implementing virtual private networks (VPNs) or green computing architectures [31]. Proposals to use the SDN approach for the composition of specific network services have focused on the incorporation of organization-specific policy (e.g. Health Insurance Portability and Accountability Act compliance) or application servers (e.g. servers running virus scanning programs) [33], rather than the formation of performance-guaranteed network services that meet the specified QoS needs of network applications.

With this in mind, we present our proposed architecture which provides explicit end-to-end internetwork service guarantees to critical network applications, such as those previously identified in this chapter, that require high reliability service with hard delay deadlines.

1.6 The Critical Service Architecture Proposal

In this dissertation, we propose an *internetwork* architecture that is natively designed to serve the needs of the interesting class of data applications previously identified. We call this the *Critical Service architecture*, or CServ architecture. As individual network domains have deep visibility into their own detailed state and operating conditions, we consider that the internetwork transmission of datagrams with strict reliability and delay requirements provides the most challenge to the current IP architecture with BGP as the internetwork routing protocol.

We denote the class of network applications that want simultaneous high reliability and stringent delay deadlines for their datagram transmissions, some of which were described in Sections 1.1 and 1.2, as *critical service (CServ) applications*. Similarly, we identify the datagram traffic they generate as *critical service (CServ) datagrams*. These are the most critical of all applications using the network – those that control crucial infrastructure, relay time-sensitive military commands, execute high-risk algorithmic trades, and provide early alerts about impending situations or dangers to systems and command centers. As motivated by these examples, the CServ datagram traffic class uses short critical messages, usually on the order of one kilobyte, to transit the necessary information. For the purposes of this dissertation, we assume that these transmissions are unicast, meaning from one specific source host to one specific destination host. If multicast transmission is required, it can be treated as the superposition of multiple unicast transmissions. And, importantly, this CServ traffic represents a very small fraction of the total network traffic, on the order of less than one perfect of all datagrams in the network. The network applications generating datagrams in the critical service class must ensure that these are the most important messages in the network. If too many datagrams are generated in the critical service class, they quickly begin to appear as best effort datagrams respectively. There must be a clear and significant distinction within network applications between typical best effort datagrams, such as IP datagrams, and critical service datagrams.

The CServ architecture is a multi-service, packet-switched internetwork architecture designed to natively support two distinct traffic classes: critical service datagrams and standard best-effort datagrams. While the network service provided for best-effort datagrams is functionally equivalent to that offered by IP networks, the experience of the critical service class diverges significantly. Prior to the transmission of a CServ datagram in the CServ architecture, the source host application receives two explicit end-to-end guarantees regarding the internetwork service offered for that datagram. Namely, the source host application receives a guarantee on the reliability of that transmission, or the one-shot probability of success that the CServ datagram will arrive at the specified destination host, and a probabilistically bounded guarantee on the end-to-end transmission delay.

In providing these guarantees, the CServ architecture has a very specific goal. Unlike the specification of "connectivity-centric applications" in [11] which generally desire the maximized probability of on-time delivery to support mission-critical messages, the CServ architecture takes the performance requirements of the specific mission-critical application and specific datagram transmission in terms of reliability and allowable delay and explicitly computes an internetwork (or possibly an intranetwork) service solution that satisfies those demands. The solution is tailored to the needs of the application.

In computing the dynamic and on-demand application-specific network service for CServ datagrams, the CServ architecture does not require any per-transaction end-to-end reservation (such as needed for the creation of internetwork VCs). Simultaneously, the CServ architecture continues to allow for autonomous control within individual network domains in the internetwork topology; network providers and administrators are not required to implement any new routing or forwarding solution within the core of their network domain if desired. The CServ architecture levies only minimal upgrade requirements on the edge routers within a domain, as well as the inclusion of a novel logically-centralized control entity. Furthermore, network providers and administrators do not need to directly expose internal network state to neighboring or peering networks. The CServ architecture requires only minimal exposure of internal network state at a high description level to a logically-centralized control structure, and the minute details of the internal state of the network (such as topology, loading, and network health) are free to remain concealed from other network providers or administrators.

There are several pillars to the CServ architecture that makes its design possible, several of which share traits with the related proposals for dynamic service-oriented networking in Section 1.5. We summarize those vital aspects of the architecture here:

1. a *hierarchical control structure* allows individual network domains to retain control over their own internal routing and forwarding decisions while a logically-centralized global control plane can make internetwork routing decisions;
2. a *state measurement and reporting service* serves as the glue between the hierarchical control layers, allowing individual networks to estimate and report the performance of their preferred intranetwork routing service solutions in terms of reliability and delay which empowers the global controller to determine internetwork routing service;
3. an *internetwork diversity routing strategy* enables a highly-reliable and survivable one-shot internetwork messaging service that can meet the service requirements of critical network applications;
4. and *priority queueing and transmission* for CServ datagrams where available (but not strictly needed in all networks) ensures some distinction in data plane service for the most important messages in the network compared to the best effort messages.

This list of central features for the CServ architecture indicates that it is not wholly detached from other service-oriented network proposals. For example, the separation of the control plane from the data plane shares the primary characteristic of an SDN, but the CServ architecture takes this separation a step further and introduces the notion of hierarchical control plane design. Similarly, priority queueing and transmission for one traffic class and not another resembles the intent of DiffServ, but the CServ architecture places this capability in a holistic internetwork framework that can leverage this technique to actually provide end-to-end explicit performance guarantees.

Although none of these architectural concepts are particularly novel alone (several of these architectural building blocks were indeed identified as necessary for a survivable tactical defense network in [34]), the framework and interaction between these architectural components is what sets the CServ architecture apart as a fresh service-oriented network proposal. In the remainder of this thesis, we discuss these components and their interaction in much more detail after introducing the operation of the CServ architecture more thoroughly in Chapter 2.

1.7 Thesis Organization

The following provides a brief outline of the dissertation document.

Chapter 2 provides further motivation for the CServ architecture and its operation before walking through the details of a CServ transaction. The description of the transaction flow includes pre-transaction network state maintenance, pre-transmission control plane transaction setup, and the data plane internetwork transmission of the CServ datagram.

Chapter 3 focuses on the operation of the data plane, specifying the data plane header control information required to simultaneously enforce the routing decisions of both the global internetwork controller and the local intranetwork routing and forwarding policies.

Chapter 4 examines the State Measurement Service that provides the connection between the different control levels in the control plane hierarchy. This service is used to learn the performance of the network services offered by the different network domains without disclosing detailed state and topology information. In this chapter, we discuss several protocol options that realize the goals of this architectural component.

Chapter 5 considers the representation of the network at a global level and the operation of the logically-centralized top-level controller. As a chapter highlight, the algorithm used to determine internetwork service that meets the transaction-specific demands of the critical service applications is described and analyzed.

Chapter 6 concludes the dissertation and, as this work represents an initial architecture consideration and description, discusses some necessary directions for future research on the CServ architecture design.

Chapter 2

CServ Architecture Overture

This chapter serves as an overture to the remainder of the thesis; the objective is to introduce the reader to the Critical Service (CServ) architecture and its operation. After specifying the design goals of the architecture, we present the primary components, both hardware and software, through a step-by-step walkthrough of the flow of a typical CServ transaction. While not a thorough coverage of all details, this is intended to familiarize the reader with the different aspects of the architecture and promote the in-depth treatment and discussion of these components in the subsequent chapters. Several architectural design features lend themselves to alternative implementations, and this chapter also serves to introduce some of these considerations and trade-offs.

The structure of the chapter is as follows. In Section 2.1, we explicitly describe the goal of the CServ architecture, as seen from the viewpoint of the end-user of the service. Section 2.2 outlines assumptions about the network structure used in the remainder of the dissertation and the significance of the validity of these assumptions. A step-by-step presentation of the end-to-end flow of a standard CServ transaction, from transaction setup to data plane transmission, follows in Section 2.3. Finally, Section 2.4 considers an alternative approach to the CServ transmission flow, framing further discussion for architectural implementation alternatives and options as the treatment of the CServ architecture progresses throughout the document. We conclude the chapter in Section 2.5 and highlight the architectural components to be covered in more detail in the remainder of the document.

2.1 Design Objectives of the CServ Architecture

As motivated by the discussion of the driving application spaces and related service-oriented architectures in Chapter 1, we present the objective statement of the CServ design.

CServ Architecture Objective Statement – *The objective of the CServ architecture is to provide explicit, probabilistic guarantees on the performance of the end-to-end internetwork transmission of critical service datagrams, which exist alongside best-effort traffic, from source host to destination host.*

These probabilistic guarantees are based on the most recently available global network state information, and they are presented to the source host end-user application prior to the transmission of the critical datagram. In doing so, the end-user is endowed *a priori* with an understanding of how its internetwork transmission of the important datagram will fare, a vital component of planning and risk management when transmitting critical, time-sensitive data over an unreliable set of network substrates.

We break down the different aspects of the design objective statement, beginning with the critical service datagram itself. The payload supported by the service is a short message, approximately one kilobyte, such that the CServ datagram with header control information is approximately the size of one standard Internet Protocol (IP) datagram (1.5 kilobytes, based on the Ethernet v2 maximum transmission unit [35]). The structure of the CServ datagram header and payload are further discussed in Chapter 3. The size of the datagram should be sufficient to bear entire transactions for most applications that wish to leverage the critical service class, including the examples discussed in the defense network application space [36] and the commercial high-speed algorithmic trading space. If an application generates a critical message payload that exceeds one kilobyte, the application should fragment the data and transmit the message as multiple independent CServ transactions. The decision to constrain the size of the payload supported by the critical service is manifold:

1. It avoids head-of-the-queue blocking by an "elephant"-sized CServ transaction.
2. It promotes high-availability of the service for many CServ users.
3. It enables the architecture to use CServ datagrams for active probing purposes without needlessly degrading excess network bandwidth.
4. It allows for an alternative CServ implementation proposal where there is no service class distinction between CServ datagrams and IP-like best-effort datagrams (to be discussed later).

The second component of note from the objective statement is the existence of the critical service class alongside best-effort traffic. *Best-effort delivery* service does not provide any guarantee on message performance or specify any particular quality of service level. An everyday example of this type of service is the postal service – delivery of letters are not scheduled in advance, the delivery of letters may be delayed unexpectedly if there is a burst of postal service demand, and the sender of a letter is not informed of letter delivery (at least for the baseline postal service level). Compare this with a communication network best-effort delivery service, where the most ubiquitous example is the network layer Internet Protocol (IP) [29]. Internet Protocol datagrams may be lost, arbitrarily delayed, corrupted, or duplicated as they traverse the network. Endpoint transport layer protocols and applications implement the necessary additional end-to-end services on top of this inherently unreliable data delivery network service. Many network applications have been developed in this way, including file exchange using the supplemental reliability of Transmission Control Protocol (TCP) [37] and Internet streaming video through specific video encoding techniques (e.g. MPEG-4 AVC [38] [39]) that accept periodic datagram loss at the network level. However, as discussed in Chapter 1, the best-effort service of IP is not sufficient for explicit end-to-end quality of service guarantees across network boundaries, such as those that the CServ architecture is designed to provide. Thus we differentiate between the two different classes of service in the CServ architecture: *critical service* and *best-effort*. We assume that the best-effort class continues to provide network service for those applications it is well-suited for, whereas the critical data class is only used by the high priority subset of network applications that require its *a priori* end-to-end guarantees. Internet Protocol can be used to implement the best-effort service to ease the transition from today's predominantly IP-centric networks to the CServ architecture, but this is not required in the design. Furthermore, a central concern in the design of the CServ architecture is to ensure that the operation of the critical service class does not needlessly degrade the operation of the best-effort service class, even though critical service data is tacitly a higher-class network citizen compared to best-effort data. (We note here that this prioritization between the classes may be explicit depending on whether or not parts of the network implement priority queueing and transmission for CServ traffic; more on this to come later.)

The third component of the objective statement we discuss further is the specification of internetwork critical datagram transmission. Most network systems are in fact a network of networks, such as the Internet, a federated collection of many interconnected Autonomous Systems (ASes). ASes are a collection of IP address prefixes under the control of a single administrative domain (or entity, such as

an Internet Service Provider) that present a common routing policy to the rest of the Internet [40]. Network administrators are afforded wide latitude in the design and implementation of the internals of their network, such as with the selection of hardware and software vendors, topology optimization and routing strategies, and security and management practices. The autonomous but interconnected structure of the Internet allows for growth at scale, as well as healthy competition. For the purposes of conveying data between ASes, Border Gateway Protocol (BGP [12]), part of the Internet Protocol Suite standard, enables forwarding of IP datagrams across administrative boundaries. Through this federated internetworking behavior, the Internet appears to the end-user as one large entity where all IP address prefixes are universally reachable. We anticipate that future network architectures will also be implemented as an interconnected set of systems, where different systems may be administered by disparate organizations or entities. While network providers have full visibility into their detailed network telemetry, they are understandably resistant to share this information with their competitors who may operate neighboring (or peering) networks. As discussed in Chapter 1, this desire for network state privacy is readily observed in the design of BGP, which advertises IP address prefix reachability to neighboring networks without any path performance metrics. For these reasons, we consider the eminent challenge of the CServ architecture to be the implementation of guaranteed-performance paths across network boundaries, where detailed network state is frequently not exchanged. Even with common administrative control over different networks and the elimination of the need for state obscuration, full state exposure of many networks poses a scalability concern that the CServ architecture design aims to avoid. Before moving on, we mention here that intranetworking service with performance guarantees can be achieved since the necessary state is readily available within a network domain. In general, we need to understand intranetwork performance before we can characterize internetwork performance, and thus we consider this as a subproblem of internetworking critical service.

The part of the architecture objective statement that indicates the characterization of performance end-to-end, from source host to destination host, is the fourth aspect that we highlight for discussion. There are several implications that can be collected from this phrase. First, we are interested in capturing the performance of the entire internetwork path that connects the two communicating parties, not the performance of a subset of the path. This is particularly significant in light of the above consideration of the challenges of internetworking, where state of the different network components of the route are traditionally not revealed globally. Second, we consider a host-centric communication model where the

two endpoints of the critical data exchange are individual network hosts. Specifically, this means that a critical message is generated by a specific host on the network and is ultimately destined for another explicitly named host on the network. This is in contrast to some future network architecture proposals (e.g. the eXpressive Internet Architecture [41] and Named Data Networking [42] Future Internet Architectures) that shift the network addressing focus from hosts to content; in these frameworks, the addressing scheme specifies desired content requests rather than the host on which the content resides. And lastly, our architectural emphasis is on unicast critical communication between two hosts, the source and the destination, and multicast critical messaging transactions are currently implemented as multiple unicast CServ transactions initiated by the same source host. That being said, the presented CServ architecture can be extended to address explicit multicast CServ transactions.

The last, and arguably most crucial, component of the architecture objective statement is that of *a priori* probabilistic guarantees on critical datagram performance. To begin, we clarify what we mean by performance. In the CServ architecture, we have two primary metrics of concern that together describe the end-to-end performance of a critical datagram: *reliability* and *delay*. With the intention of endowing end-user applications with knowledge of how the CServ datagram will fare before it is transmitted, we posit that these two metrics summarize the information necessary to encapsulate that knowledge. Let us formally define the two metrics:

> **Definition 1** – The *reliability* of a CServ datagram is the probability that the datagram arrives successfully (in other words, the datagram is not dropped, lost, or corrupted in transit) at the intended destination host at some time after transmission by the source host. As a probability, the support of reliability is the set of real values between 0 and 1, inclusive.
>
> **Definition 2** – The *delay* of a CServ datagram is the total elapsed time between the generation of the datagram at the source host and the reception of the datagram by the destination host. As an elapsed time, the support of delay is the set of real, nonnegative values.

Before a CServ message is transmitted, the source host application is aware of the probability that the message will arrive successfully at the specified destination and the elapsed time the CServ datagram will take to arrive (if it does indeed arrive). However, this form of performance guarantee is not yet reasonable. The delay in a datagram network from source host to destination host along a specified path

cannot be characterized by an absolute value; although transmission delay over the path may be predicted precisely, variable cross-traffic patterns and queueing delay components make the end-to-end delay random. Consider the simple example of Section 2.1.1, which illustrates this point. End-to-end delay from a source host to a destination host in a datagram network should be treated as a distribution. For this reason, we propose a more appropriate form of delay performance guarantee, a probabilistic guarantee. Specifically, before a CServ message is transmitted, the source host application is aware of the probability that the message will arrive successfully at the specified destination and the maximum elapsed time the CServ datagram will take to arrive, if it does so, within some probabilistic margin of certainty. With this information, the application that generates the critical message can employ the necessary form of planning and risk management required for its specific task and the purpose of the CServ datagram.

Before moving on, we note that reliability and delay are not independent metrics; networking techniques generally can realize increased reliability at the expense of increased delay or decreased delay at the expense of decreased reliability. In fact, the former is exactly the operational mode of the Internet's TCP [43] which provides reliable end-to-end connections over the inherently unreliable network, while the latter is the operational mode of the User Datagram Protocol (UDP) [44] which emphasizes reduced latency over individual datagram reliability. Consider that TCP employs an Automatic Repeat Query (ARQ) error control method at the endpoints that leverages acknowledgments, timeouts, and retransmission of data packets to ensure that all data in a transaction successfully reaches the destination, all of which incur additional delay.

2.1.1 Probabilistic Delay Guarantees

In this section, we motivate the need for probabilistic delay guarantees in the CServ architecture using a simple, but illuminating, example, which provides necessary insight into a more rigorous and complex network model. The objective of the CServ architecture is to provide *a priori* guaranteed service to critical datagrams, so we must consider the extent to which performance can be guaranteed over a communication network composed of unreliable components and substrates. Ideally, we would like to provide an absolute end-to-end guarantee of the following form to the source host prior to transmission: "The critical datagram generated by source host s will arrive at destination host d within one second." The example discussed here shows that we must reconsider the form of the guarantee.

Consider the example network segment depicted in Fig. 2-1. This segment could represent the edge of a larger network topology, where router R is an edge access router connected to destination host d via some reliable communication link of constant rate, denoted r [bits/second]. For illustration, we stipulate that all datagrams arriving at R are destined for d. As shown in the top diagram of Fig. 2-1, these arriving datagrams from an arbitrary number of sources are multiplexed into a single service queue, and the aggregate arrival rate of these datagrams is denoted λ_d [datagrams/second]. Without the complication of multiple sources for datagrams arriving at R destined for d, this situation is equivalently depicted as in the bottom diagram of Fig. 2-1.

For this analysis, we assume a deterministic datagram arrival process, denoting datagram interarrival time as τ [seconds], and thus the packet arrival rate $\lambda_d = 1/\tau$. We also assume a deterministic datagram length of l [bits], a reliable access network communication channel to destination host d at the given rate r (in other words, there is no datagram packet loss due to bit errors over the channel), and a constant propagation delay between R and d denoted by t_p [seconds]. In practice, this reliable, constant-rate channel would likely represent a direct connection from an output port of router R to the destination host d by some wired transmission medium (e.g. twisted pair or fiber optics) or a local area network (LAN) broadcast topology using wired transmission media (e.g. Ethernet [45]). We let t_x [seconds] denote the transmission delay of a datagram on the channel from R to d. Given our previous assumptions, we have:

$$t_x = \frac{l}{r}. \tag{2.1}$$

An illustration of the datagram arrival and transmission process at router R is shown in the blue box of Fig. 2-1 where the datagram interarrival time exceeds the transmission delay (i.e. $\tau > t_x$).

For analysis purposes, we assume that the depicted system is empty at time $t = 0$, and we consider a stream of m arriving datagrams, denoting the elapsed time between the arrival of the i^{th} datagram at R and its delivery at d, or datagram delay, as d_i, $i = 1,2,\dots,m$. The objective of the analysis is to characterize the datagram delay of all datagrams in the stream. To do so, we need to consider two distinct cases:

I. $\tau \geq t_x$;

II. and $\tau < t_x$.

The primary distinction that we immediately make between the cases is that a buffered transmission queue is required for the multiplexed input stream at R for Case II, but not in Case I.

We begin with the analysis of Case I, which covers when the interarrival time of the datagram arrival process exceeds the transmission delay of the datagram as given in Eq. (2.1). In this situation, the transmission of the previous datagram in the stream will have completed by the arrival of the next datagram, so no transmission queue forms at the input port of R and the transmission buffer remains empty. The datagram delay for each datagram is thus the physical limit per packet, denoted t_{xp}, or the transmission delay plus the propagation delay from R to d. More concretely, we have:

$$d_i = t_{xp} \triangleq t_x + t_p, \tag{2.2}$$

where $i = 1,2, \ldots, m$. The datagram delay is a constant for all m datagrams in the stream.

Now we proceed with the analysis of Case II, which covers when the interarrival time of the datagram arrival process is less than the transmission delay of the datagram as given in Eq. (2.1). This case requires use of the input transmission queue at R to buffer datagrams as they await access to the output port and communication channel. When the datagram interarrival time is less than the transmission delay of the previous datagram, the new datagram arrival will need to wait while the previous arrival is served by the router and transmitted via the interface to destination host d. Using the previously addressed assumption which states that the system is empty at an arbitrary time $t = 0$, the datagram delay of the first datagram in the stream will be the physical minimum t_{xp}, while the datagram delay of all subsequent datagrams in the stream will be strictly greater than t_{xp} due to time waiting in the transmission buffer. We let w_i, $i = 1,2, \ldots, m$, represent the time spent in the transmission queue by the i^{th} datagram in the stream. Consider the illustration of Case II, as shown in Fig. 2-2, which depicts the total time of each of the first four datagrams of the stream spent at router R, both waiting in the queue and in transmission. As we can readily glean from the figure, the total time spent at R increases monotonically for each subsequent datagram arrival because the time spent waiting in the queue

increases for each datagram in the stream. Noting that $d_i - w_i = t_{xp}$, $i = 1,2,\ldots,m$, we find the general expressions for both w_i and d_i, $i = 1,2,\ldots,m$, are as follows:

$$\begin{aligned} w_i &= (t_x - \tau)i - (t_x - \tau), \\ d_i &= (t_x - \tau)i + (t_p + \tau), \end{aligned} \tag{2.3}$$

where $i = 1,2,\ldots,m$. From these expressions, we observe that both transmission queueing delay and datagram delay increase linearly with the datagram index i which represents the datagram's relative position in the stream. Both the transmission queueing delay and the datagram delay are shown for an illustrative example in Fig. 2-3 assuming a large-enough buffer at R such that the buffer never fills to capacity and no datagrams in the stream are dropped.

Comparing the results of Case I in Eq. (2.2) (where the implied queueing delay for each datagram is zero) with the results of Case II in Eq. (2.3), we see that we can only provide an absolute guarantee on datagram delay for all datagrams in a stream under Case I. All datagrams arriving at R and destined for d are delivered in t_{xp} seconds. However, this is only possible under a very specific set of assumptions that govern this example. Most critically, the interarrival time of datagrams at R destined for host d must be consistently greater than the transmission time of a datagram. If the datagram arrival process were not deterministic, it is not sufficient for the average interarrival time to satisfy this property; the minimum interarrival time of the process must be greater than the transmission time of a datagram, otherwise there is a nonzero probability that a transmission queue would form at R which would violate the absolute delivery guarantee of t_{xp} seconds from router R to destination host d.

If we consider Case II, it is clear that for any fixed time horizon chosen as the absolute guarantee on datagram delay from R to d, the datagram delay will surely exceed this value for some datagram in the stream as the index increases. In a non-idealized case, the transmission buffer would be finite length, allowing us to bound the queueing delay based on the buffer capacity. However, we would then need to account for the effect of blocking (datagrams arriving at R when the transmission queue is full) in the guarantee, further obstructing the ability to form an absolute guarantee on delay. A similar result for Case II can be developed for a non-deterministic, random datagram arrival process where the average interarrival time is less than the transmission delay of a datagram.

The question then becomes: is it possible to ensure that a network always operates under the conditions of Case I? Let us revisit the conditions that enable Case I in the example. First, relaxing the deterministic arrival process assumption, the maximum aggregate arrival rate of datagrams at R destined for host d is constrained to be less than the router's service rate for these datagrams, which is r/l [datagrams/second]. Our simple example does not stipulate the full network topology; however, these datagrams destined for d are possibly generated by many source hosts which may be operating in different network segments. During actual operation, we would need to know the set of all possible sources generating datagrams for d ahead of time in order to constrain the individual source datagram generation rates to satisfy this property. The objective of the CServ architecture is to allow on-demand access to the critical service, and thus we do not have prior knowledge of the CServ datagram traffic flow. And second, we have completely ignored all exogenous traffic flow in this example. As described, all datagrams arriving at router R are destined for d. The router does not have to share its resources with any cross traffic destined for other output ports. If we were to include the effect of cross traffic that competes for router processing time in the example, we would need to further introduce constraints on its aggregate arrival rate (and enforce some sort of fair queueing policy) in order to operate under the conditions of Case I. And this does not yet introduce the notion that most datagrams from the arbitrary set of source hosts would likely traverse multiple routers along the path to R, each of which would be subject to these constraints on aggregate cross traffic. It should be clear by now that the conditions necessary to operate under Case I couple the traffic state of the network in its entirety, making it infeasible to ensure that the network can always operate in such a way. In fact, these observations allow us to make a significant assertion:

> **Assertion –** An absolute guarantee on end-to-end delay in a packet network cannot be made without constraining the maximum aggregate datagram arrival rate at each and every network queue to be less than the datagram service rate for that queue at all times.

Put another way, iron-clad performance guarantees require end-to-end circuit reservation without statistical sharing or multiplexing of resources such as in a datagram or packet network. Consequently, the CServ architecture focuses on providing *a priori* probabilistic delay guarantees to the service end-user in the form discussed previously. Before a CServ message is transmitted, the source host application is given the maximum elapsed time the CServ datagram will take to arrive at the destination,

if it does so, within some *probabilistic margin of certainty*. This probabilistic margin is formalized later when we discuss the details of a CServ transaction.

Before concluding the discussion of this section, we briefly discuss another simplification made in the preceding example. Specifically, we have assumed a reliable communication channel between R and d, which is unrealistic for most data networks. We should account for the possibility of datagram loss due to data corruption during transmission; this corruption could be a result of natural noise processes or multiple-access interference, for example. A theoretical problem of two armies, physically separated by a hostile army, attempting to coordinate their attack is presented in [46]. The two friendly armies must use messengers to synchronize their battle plan, but these messengers may be killed or captured indefinitely by the enemy army. This problem is an analogy for reliable communication over an unreliable channel. In fact, it can be shown that no strategy exists that allows the two armies to synchronize their attack with absolute certainty, although a solution can be found if we relax the problem and require only a specific probability of synchronization. This message further drives home the need for probabilistic guarantees in the CServ architecture, and it explains the need for the *reliability* metric in addition to a probabilistic delay guarantee.

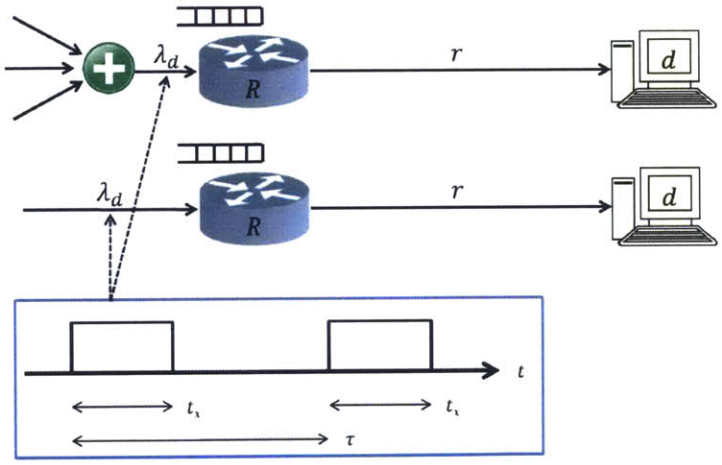


Fig. 2-1: Two equivalent views of an example network segment used to illustrate the need for probabilistic delay guarantees rather than absolute delay guarantees. In this illustration, the datagram interarrival time exceeds the transmission delay.

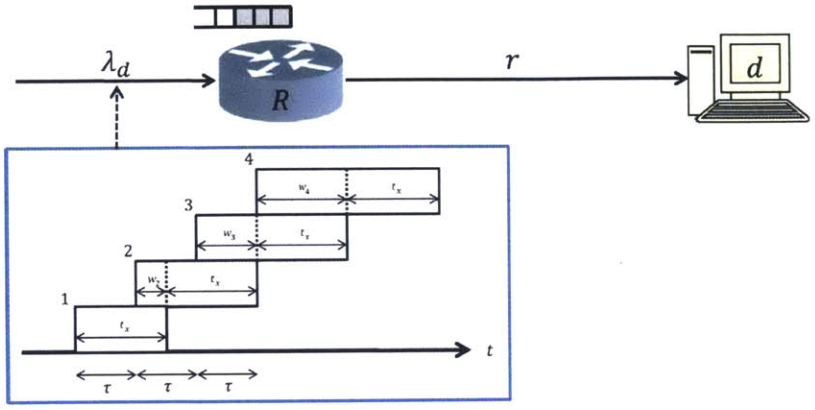


Fig. 2-2: Another illustration of the example network used to illustrate the need for probabilistic delay guarantees. In this version, the datagram interarrival time is less than the transmission delay.

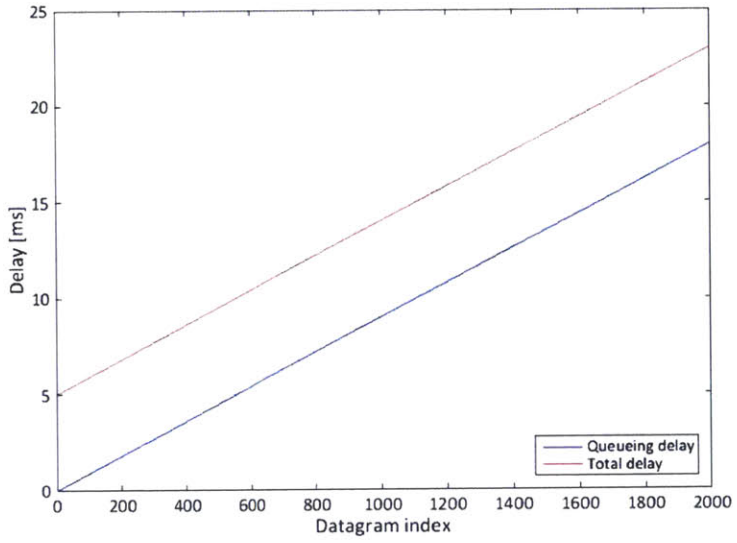


Fig. 2-3: Plot of both queueing delay and datagram delay (total delay) as a function of datagram index in the stream for $r = 1$ Gbps, $l = 10$ kb, $\tau = 1$ μs, and $t_p = 5$ ms. The expressions are found in Eq. (2.3), and they assume a large enough queueing buffer such that it never fills to capacity and no datagrams in the stream are dropped.

2.2 Network Models and Assumptions

In this section, we address some of the network model assumptions used throughout the dissertation to both simplify and unify the discussion from chapter to chapter. Specifically, we consider the network structure, develop the necessary terminology to describe the structure, and we make comments on the rigidity of the assumptions. In many cases, the assumed network structure is not fundamental to the CServ architecture or its operation; rather, we make the assumptions to help formalize the discussion of the architecture and its protocols. We comment here briefly on the necessary degree of adherence to the assumptions and how it impacts the CServ design, and we continue to address these issues throughout the rest of the document when appropriate.

2.2.1 The Subnet

The *subnet* is the central logical participant of the CServ network and global internetwork routing architecture. Specifically, internetwork paths for critical datagrams (where the *inter-* prefix indicates the forwarding of a datagram across subnet boundaries) follow an explicit subnet-to-subnet granularity route, where the internal routing behavior of the subnet appears to be a "black box" to the global network. The design choice that designates the subnet as the central participant in internetworking is discussed in depth throughout the dissertation.

Specifically, we define the subnet as follows:

> **Definition 3** – The *subnet* is an interconnected set of routers that implement a unified routing solution and share a common network controller.

The definition of the subnet may initially give the impression that the subnet is synonymous to the Internet Autonomous System (AS). According to [13], the AS "is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy" that it presents to the Internet. Whereas the definition distinguishes the AS unit by the prefix reachability policy that it chooses to announce externally, the subnet is distinguished by two important factors:

1. a set of routers that agree upon an *internal* routing strategy;
2. and the existence of *a shared network controller.*

The subnet does not necessarily correspond one-to-one with an Internet AS; the definition of the CServ subnet is more specific than that of an AS. The CServ architecture introduces the notion of a logically-centralized controller entity which is called the *subnet controller (SC)*. The specific responsibilities of this controller entity are discussed later, but for now it suffices to say that the SC is accountable for the aggregation of performance state information needed for CServ internetworking functions. Furthermore, all routers in the subnet implement a common set of routing protocols and forwarding techniques, or a common routing strategy. For example, all routers may be configured to implement a form of shortest path routing, such as commonly seen with the Internet, and IP-based data plane forwarding. One such routing protocol is Intermediate System to Intermediate System (IS-IS) [26], a link-state routing protocol that uses Dijkstra's Algorithm to compute shortest paths. Alternatively, routers in a subnet can jointly support more than one routing and forwarding method; for example, a subnet's routers may employ a hybrid of IP-based shortest path routing and administrator-managed virtual circuit (VC) label-based connections, such as those that could be implemented with Multiprotocol Label Switching (MPLS) [19]. All datagram exchange between subnets is brokered by routers that function as subnet gateways, isolating the internal routing and forwarding details of each subnet. In theory, an AS could be comprised of multiple CServ subnets as long as the host reachability advertisement announced externally is unified across the constituent subnets. We conclude this discussion and do not invite further direct comparison between the two definitions because of the critical point of differentiation for the CServ subnet – the addition of the logically-centralized SC.

Our definition of the subnet and its isolation from other subnets via gateway routers lends itself directly to distinguishing among heterogeneous network segments, and this is precisely how we use the subnet terminology and structure in the CServ architecture. There are many forms of network heterogeneity exhibited by modern data networks. The following non-exhaustive list identifies some common forms of network heterogeneity:

1. physical communication modality heterogeneity (e.g. twisted pair, fiber optics, RF wireless, satellite communication, and laser optics);
2. hardware and software component heterogeneity;

3. network resource provider and/or administrator heterogeneity (e.g. service providers);
4. security architecture and protocol heterogeneity;
5. and network management framework heterogeneity.

One might suggest that the first item on the above list is the most fundamental form of communication network heterogeneity. The interconnected set of networks that form the modern Internet represents a mix of these communication modalities, specifically fiber optics in the long-haul backbone and twisted pair copper and RF wireless at the access edge. All of these communication modalities exhibit different properties and present different physical layer behavior to the higher network layers. For example, wired connections, such as copper and fiber links, are generally stationary channels with independent, low-rate symbol errors and short propagation times (even transiting a continent), whereas satellite connections exhibit channel fading due to atmospheric turbulence, long-term attenuation from weather effects, and long propagation delays (approximately quarter-second round trip times for systems in geosynchronous orbit). There is no single statistical channel model that uniformly captures all of these communication substrates. Furthermore, the modern Internet exhibits all other forms of network heterogeneity listed above as well, although many are transparent to the end-user other than the vast selection of providers, hardware, and software available to access the Internet and set up home or office networks.

Since the subnet naturally encapsulates important points of divergence that represent network heterogeneity, such as communication substrates, service providers and administrators, datagram routing strategies, and separation of network control responsibilities, the subnet is the chosen as the logical participant of the internetwork at the global level. The CServ architecture introduces a logically-centralized global controller entity which is called the *master controller (MC)*, and the network is represented as a collection of interconnected subnets at this level of control. An illustrative example is depicted in Fig. 2-4, which shows the interconnection of multiple subnets that are differentiated both among communication modality and service provider. By design, subnets then exhibit a notion of statistical independence due to their intentional heterogeneity. Even under adversarial attack, we assume that their dissimilitude renders them "independent enough" such that significant targeted effort is required to undermine operation in each and that the compromise of one subnet does not directly correlate to the compromise of its peers. This is an important, but possibly fragile, assumption that is later invoked when we describe the global internetworking algorithm that supports CServ operation

across the disparate subnets. For this reason, we keep this property and its potential frailty under reflection as we proceed. Consider, for example, that the three fiber backbone subnets with different providers in Fig. 2-4 may in fact share the same physical plant, using fibers in the same cable or conduit. The infamous "backhoe attenuation" problem, where a backhoe unintentionally digs a fiber cable out of the ground during excavation work, could render all three of these subnets inoperable simultaneously in this case. This is an example where the "independent enough" property does not hold; rather, these three subnets are highly correlated since they share the same physical plant. Maintaining the property of statistical independence between subnets in the CServ architecture is not without challenge and does not come unintentionally. The network planner must be aware of such pitfalls and very deliberately provision subnets to support this assumed property.

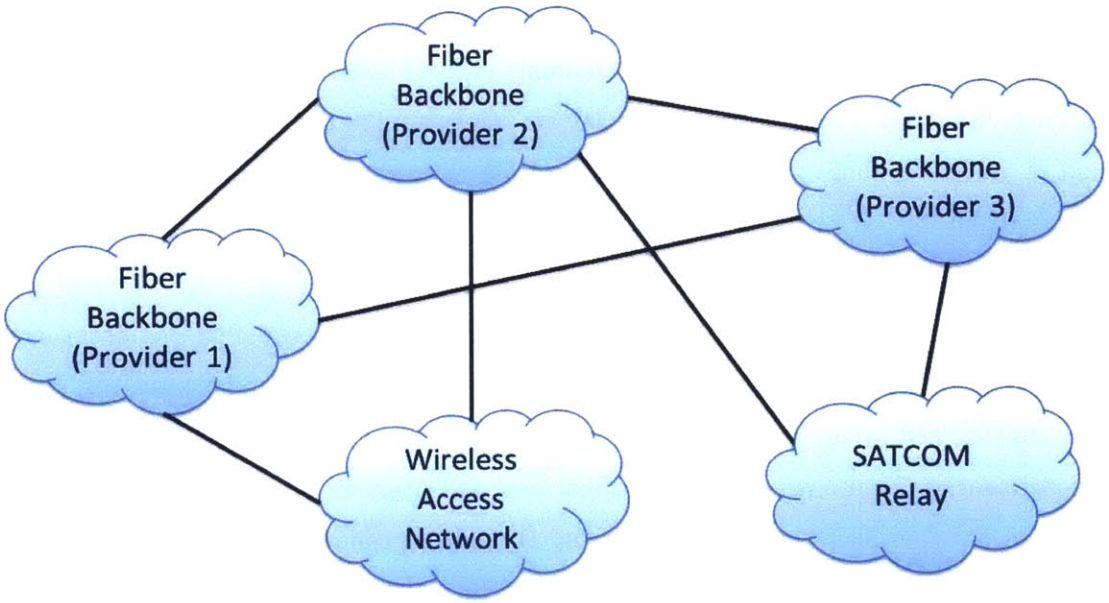


Fig. 2-4: An example network at the global abstraction level, where the subnet is the central logical participant. Here, the three peering fiber backbone subnets are distinguished by their network resource providers, whereas the RF wireless access subnet and satellite relay subnets are differentiated from the fiber subnets by their physical communication substrates.

2.2.2 Internal Subnet Structure

In order to both simplify and homogenize the discussion of the CServ architecture, we standardize the internal subnet structure and terminology with a generic model that can readily be adapted to many subnet realizations. In this section, we begin with definitions of the different components of the generic subnet structure. As illustrated, the example closely resembles a fiber Wide Area Network (WAN) backbone or a Metropolitan Area Network (MAN). However, we illustrate its generality by applying it to a very specific subnet example, namely a SATCOM relay.

The generalized subnet structure is shown in Fig. 2-5. We do not assume any particular internal subnet topology; the depiction in the figure is purely for illustration. The focus of the terminology is to distinguish a hierarchical relationship between endpoint hosts and different types of routers in the subnet. For the purpose of this section, we ignore the distinction between a standard router and its counterpart "active" version; this differentiation is addressed in the subsequent section. Referring to the entities highlighted in Fig. 2-5, we discuss in more detail four of the primary components of the subnet model:

1. access networks;
2. access routers;
3. core routers;
4. and gateway routers.

Access networks are local area connections between endpoint host machines and their upstream access routers. Host machines that are CServ-enabled are the source and destination points of CServ transactions, or the exchange of a critical message datagram. But as depicted in the illustration, not all hosts that are part of the subnet are necessarily CServ-enabled. The technology and topology of access networks may take many different forms. For example, as illustrated in Fig. 2-5, the hosts may be arranged in a logical bus, ring, or tree topology, among others. Furthermore, the actual access communication technology may be, for example, passive optics, edge RF wireless, or twisted pair Ethernet interconnected with multiple layer-2 switches, among others.

Together, *access routers*, *core routers*, and *gateway routers* and their interconnections form the layer-3 core routing topology of the subnet. The naming distinction stems from the specific roles that they play. At their heart, the primary responsibility of all three devices is to parse and forward datagrams appropriately (although what exactly "appropriately" means is at the crux of the discussion in the rest of this dissertation!), much the same as Internet routers. However, the types of connections each support distinguishes between the three flavors in our nomenclature.

The access router provides upstream admittance for endpoint hosts in access networks to the subnet core network. In addition to access networks, the access router can also connect to other access routers, core routers, or gateway routers.

The core router fulfills a role similar to the access router in the core topology of the subnet network, except that the core router does not provide direct upstream access for an access network. The core router simply functions as an interconnection between other core routers, access routers, and gateway routers.

Lastly, the gateway router functions as a core router with the additional responsibility of providing connection to, or *peering* with, neighboring subnets. These routers sit at the logical edge of the subnet and provide the interface to other subnets; the links illustrated in Fig. 2-4 between subnets represent connections between gateway routers at the edge of the subnet peers. Gateway routers may have connections to other gateway routers with the same subnet, as well as core routers and access routers. Note that although the illustration of Fig. 2-5 depicts the gateway router as belonging to a particular subnet, this may not be the case. For uniformity in the dissertation, we represent a peering connection as a communication link between two distinct gateway routers at the edge of each subnet peer; however, this may actually be implemented as one gateway machine with a virtual link between the two routing interfaces. If BGP is used for best effort service as in the TCP/IP Internet, gateway routers act as border routers in the BGP nomenclature and maintain peering external BGP (eBGP) connections.

We illustrate the generality of this network model by using the terminology to describe a very specific subnet instance that differs significantly from the one portrayed in Fig. 2-5, namely a geosynchronous SATCOM relay. This subnet does not support any access networks, and the only internal "router" in the subnet is the satellite transponder. Specifically, the ground terminals can be considered the gateway

routers for the subnet, and the satellite itself can be treated as core router. That being said, most SATCOM relay transponders operate on a bent pipe principle and do not perform traditional layer-3 routing; the transponder only amplifies the signal and shifts it to the downlink frequency channel. But there have been attempts to implement IP routing functionality on the satellite, such as Cisco System's Internet Routing in Space (IRIS) Router which was launched on board the Intelsat IS-14 [47]. For this reason, we use the general approach of treating the SATCOM relay as a core router. This subnet example is depicted in Fig. 2-6 using the same color identification scheme as in Fig. 2-5. This specific subnet example illustrates that a subnet instance does not necessarily contain all of the primary components of the subnet model established in this section. In fact, minimally, a subnet may be composed only of interconnected gateway routers, with no additional core routers, access routers, or access networks.

In the dissertation, we use this generic subnet structure to streamline the discussion, but we note that the subnet structure does not need to strictly follow the preceding description. For example, a router could simultaneously function as both an access router and a gateway router if it peers with neighboring subnets and also provides admittance to the subnet core for an access network. That being said, the router is then responsible to fulfill the duties of both network components (to be discussed and clarified later).

Since the primary focus of the CServ architecture is on internetworking with performance guarantees, most CServ transactions that we discuss originate in one subnet, leave the subnet, and terminate at a destination host in a separate subnet. For this reason, we adopt the convention for the terms "upstream" and "downstream" shown in Fig. 2-7. This terminology naturally induces a sort of hierarchical relationship between the endpoint hosts and the routers. Logically, the depiction of a connected subnet could be rearranged with gateway routers as the root(s) of a rooted tree topology (and at the top of the hierarchy), the access routers as the next level of the tree, and the hosts as the leaves of the tree (and at the bottom of the hierarchy). We stress that this is a logical depiction, and the links between the levels of the tree would not directly correlate with the physical links depicted in a representation such as Fig. 2-5.

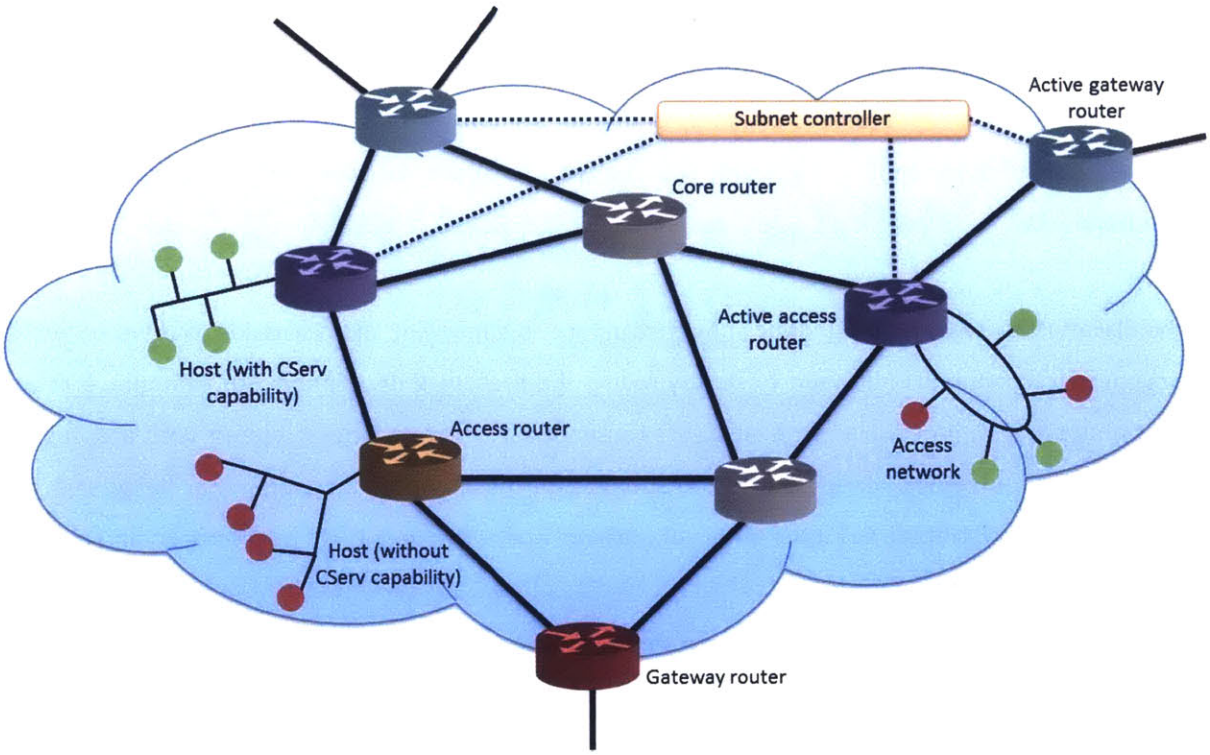


Fig. 2-5: A representation of the internal components of a subnet and the corresponding terminology, shown for a generic example topology.

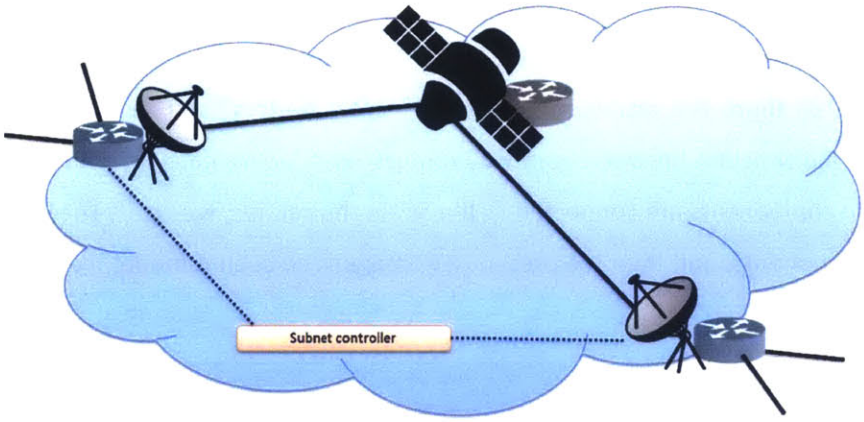


Fig. 2-6: Application of the subnet model to a specific subnet example, namely a geosynchronous SATCOM relay interconnecting two ground terminals. The satellite relay is represented as a router for generality even though it may not necessarily perform the functions of a layer-3 device.

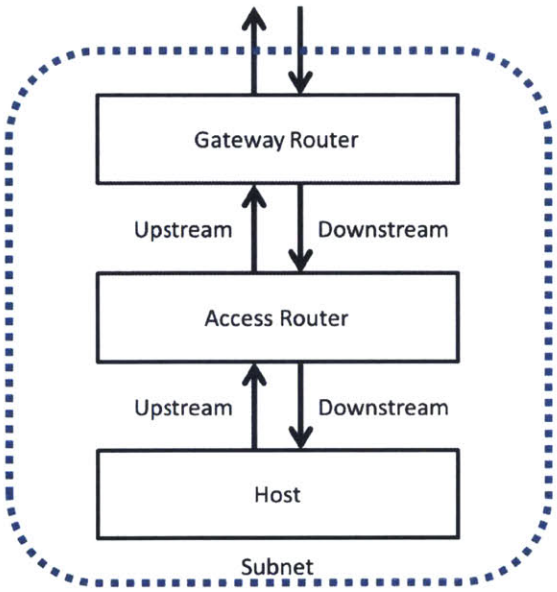


Fig. 2-7: The standardized use of the "upstream" and "downstream" terminology and their relationship to the internal components of the subnet.

2.2.3 Active versus Passive Routers

Considering Fig. 2-5, there is a distinction between "access routers" and "active access routers," and similarly there is a distinction between "gateway routers" and "active gateway routers." Only the active versions of these components are connected to the SC. In this section, we clarify the difference between these subnet components, and then we present a homogenized subnet model that we use for most of dissertation discussion.

We have previously defined access routers and established what differentiates them from core routers and gateway routers in Section 2.2.2. In doing so, we discussed that access routers provide upstream admittance for access networks of endpoint hosts. As depicted in Fig. 2-5, endpoint hosts may either support CServ operation or not. A host machine that supports CServ operation can initiate the transaction of a critical message, whereas other hosts cannot. In order to facilitate the transmission of this critical message, the upstream access router must support processing of CServ datagrams and run the necessary CServ protocol, specifically that which is used to actively learn the details of the subnet performance (these protocols, part of the State Measurement Service, are discussed in detail in Chapter 4). Furthermore, this access router must report the learned performance metrics to the logically-centralized SC. We call an access router that supports CServ operation and protocols an *active access router*. Not all access routers need to be active access routers, but an active access router is required to support CServ-enabled hosts in a constituent access network for which is provides upstream access to the subnet core. The implicit benefit here is incremental deployment; within a subnet, only access routers connecting access networks with CServ-enabled hosts need to be upgraded to active access routers. Later, we discuss additional benefits as they relate to architecture scalability.

Analogously, we draw the distinction between gateway routers and their active counterparts. As established in Section 2.2.2, gateway routers sit at the logical edge of the subnet and provide peering connections with neighboring subnets. With the architecture objective of providing internetwork service with *a priori* guarantees, CServ datagrams are expected to traverse multiple disparate subnets by way of these peering points from source host to destination host. Like access routers, the upstream gateway routers used to bear CServ transactions between subnets must support processing of CServ datagrams and run the CServ protocol used to learn and report the details of the subnet performance to the SC. Borrowing from the access router nomenclature, we call a gateway router that supports CServ operation

and protocols an *active gateway router*. Not all gateway routers need to be active gateway routers, but an active gateway router is required to bear critical CServ datagrams between neighboring subnets. Without the presence of active gateway routers, a peering connection between subnets is not considered when determining the appropriate internetwork paths to convey the CServ message from source host to destination host. Again, this flexible requirement on gateway router technology in a subnet provides the benefit of incremental adoption and deployment, but we show later that the penetration level of active gateway router upgrades is a fundamental concern to the success of the global internetworking routing algorithm.

Although access routers and gateway routers without CServ capabilities cannot directly support CServ protocols and operation, they may still participate in intrasubnet CServ service. All routers in a subnet, active or not, implement a unified subnet-internal routing solution. This internal routing strategy is used to bear CServ datagrams between entities in the subnet's set of active access routers and active gateway routers, and the paths used to do so may traverse access routers, core routers, and gateway routers that are not active. For example, these components may forward CServ datagrams internally within the subnet core using IP-based forwarding tables or MPLS label-switching mechanisms. We expound upon this in the following sections with visual illustrations of the flow of a CServ transaction.

Having made the distinction between subnet routing components and their active complements, we typically rely on a homogenized subnet model throughout the rest of the dissertation for analysis, as shown in Fig. 2-8. In this model, all hosts are assumed to be enabled with CServ capability, all gateway routers are assumed to be active gateway routers, all access routers are assumed to be active access routers, and core routers can be treated as active access routers with no connected CServ-enabled endpoint hosts. Although this homogeneity is certainly not required in practice as previously discussed, this representation helps to simplify architecture analysis, particularly architecture scalability considerations. In fact, this simplified subnet model usefully encourages a pessimistic scalability analysis which guides worst-case discussions. Before moving on, we note that we may sometimes drop the "active" terminology from access and gateway routers in this and the following chapters. When the discrepancy between a legacy router and its upgraded CServ-enabled active counterpart is necessary, we emphasize routers that do not support CServ processing and protocols as *non-active* routers.

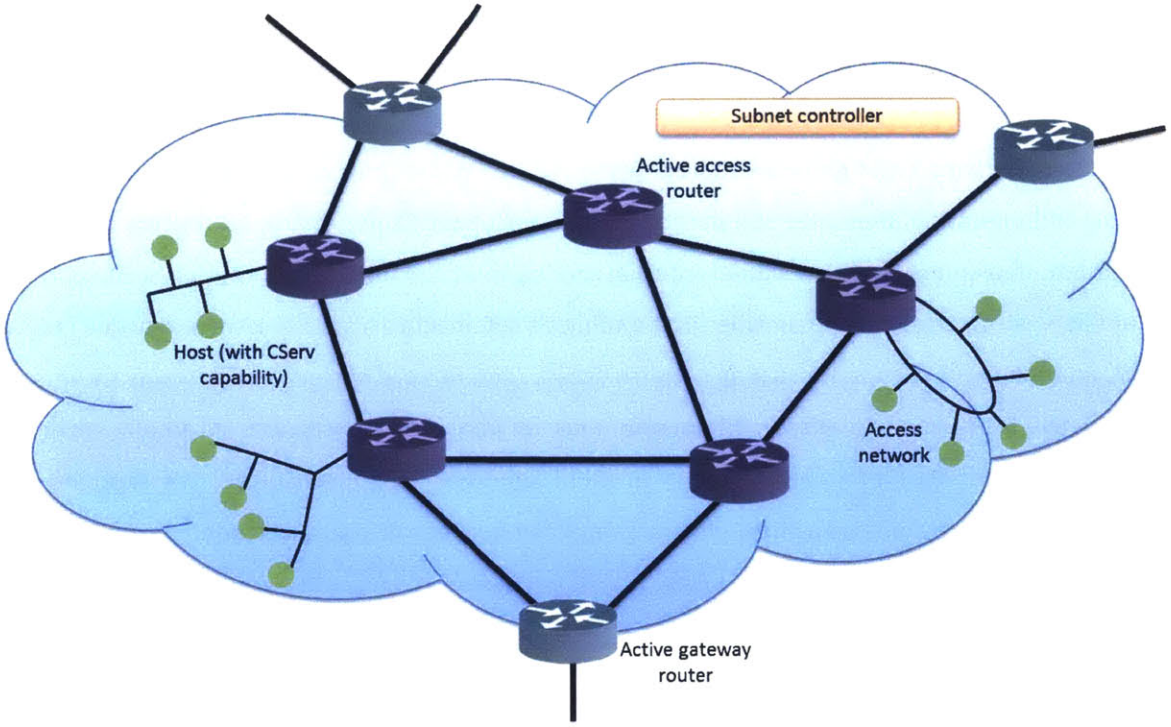


Fig. 2-8: This homogenized subnet internal representation is used throughout the dissertation to unify the architecture description and analysis. Logical connections to the SC are omitted for illustration clarity.

2.3 Flow of a CServ Transaction

We proceed to walk through a CServ transaction step-by-step, including a description of the network state pre-transaction, the transaction setup phase, and the CServ datagram transmission phase. In doing so, we introduce the fundamental components and algorithms that support the CServ architecture (although the details are left for treatment later in the dissertation).

For the purpose of this description, we assume that the network subnets implement some form of IP interior gateway protocol, such as Open Shortest Path First (OSPF) [23] [24] (the details of the exact protocol are immaterial for our purposes here) and IP-based datagram forwarding. This interior gateway protocol serves as the *CServ Intranetwork Service* for each component subnet of the global network. The treatment here shows that subnets in the CServ architecture can operate much the same as ASes in the current IP Internet, implementing the internal routing protocol of their choice; only the gateway routers and access routers for CServ-enabled endpoint hosts need to be upgraded to support processing of CServ datagrams and implementation of the CServ protocols. This overview discussion is continued in Section 2.4, where we present an alternate way to implement the CServ Intranetwork Service within the subnet that takes full advantage of the capabilities of a logically-centralized SC. Although we see that this implementation requires more pervasive adoption of CServ-capable routers within the subnet core.

Before we begin with the description of the transaction flow, we introduce some transaction-specific notation that is used to differentiate subnet components based on their relationship to the CServ critical message intersubnet path. The terminology that we present here is illustrated in Fig. 2-9. The *source subnet* is the subnet that contains the access router that provides upstream connection to the source host of the CServ transaction in its routing core; analogously, the *destination subnet* is the subnet that contains the access router that provides upstream connection to the destination host of the CServ transaction in its routing core. We note here that endpoint hosts may be *multihomed* – provided upstream core access via multiple access routers in the same subnet or multiple access routers in disparate subnets. In the latter case, there may be more than one source subnet or more than one destination subnet. However, we assume that the source and destination hosts are single-homed for the current discussion and that there is a unique source subnet and destination subnet. A *transit subnet* is a subnet that the CServ datagram traverses on its path from source subnet to destination subnet, but that does not contain either the source or destination host. And lastly, we formalize the use of the modifiers

ingress and *egress*. An ingress gateway router is one that provides entry to a subnet for a CServ datagram from a peering subnet, whereas an egress gateway router is one that serves as the "last hop" for a CServ datagram in a subnet before it is handed off to a neighboring subnet peer. A gateway router can serve as either an ingress or an egress gateway, and the distinction depends on the orientation of the critical message intersubnet path. Fig. 2-9 also shows the *CServ Application Programming Interface* (API), which serves as the communication interface between the CServ-enabled host and the logically-centralized MC. We present the high-level functions of the API in the following overview of the CServ transaction setup.

For the purpose of describing the flow of the CServ transaction, we use a specific network example, as shown in Fig. 2-10. An arbitrary human-readable decimal addressing scheme is used in this illustration to distinguish between the different network components, although IPv4 [48] or IPv6 addressing [49] could be used in practice (among other less widely adopted alternatives).

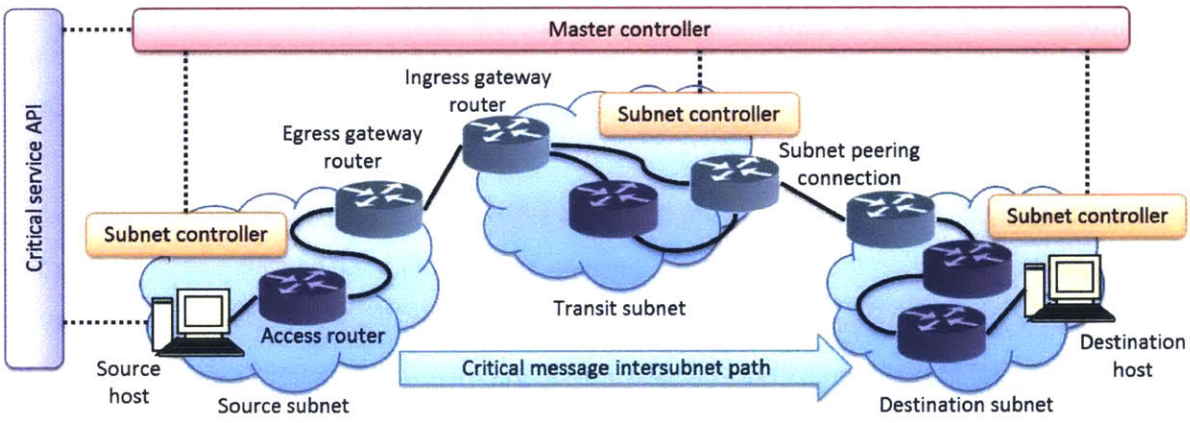


Fig. 2-9: This shows CServ transaction-specific terminology, where the source, destination, and intersubnet path of the CServ datagram determines the relative names of the network components.

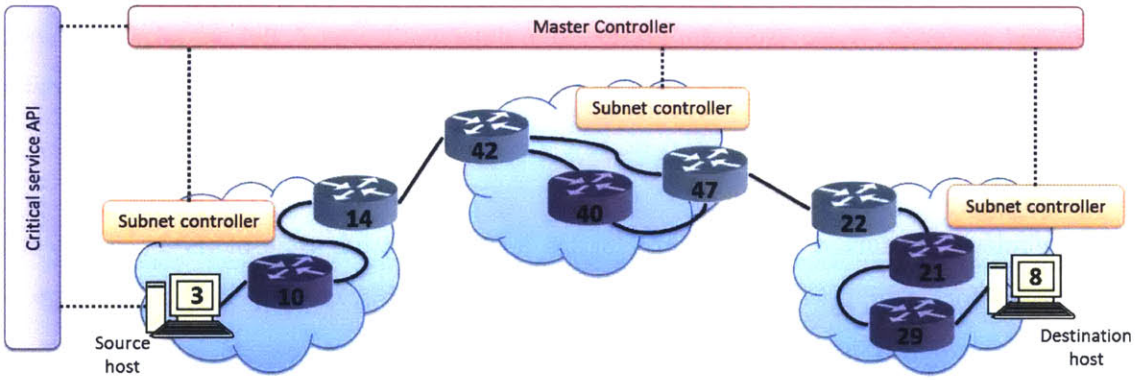


Fig. 2-10: This example network is used to describe the basic flow of a CServ transaction in Sections 2.3.1-2.3.3, including pre-transaction state maintenance, transaction setup, and critical message datagram transmission.

2.3.1 Pre-Transaction State Maintenance

The key to the success of the *CServ Internetwork Service* architecture is the learning, collection, and maintenance of the necessary subset of network state information. Specifically, in order to provide *a priori* end-to-end guarantees on critical message performance, the logically-centralized MC must have some knowledge of the performance offered by the underlying subnets that bear the datagram from source host to destination host. It is with this state information that the MC is able to discover and compose subnet-to-subnet granularity paths that meet the CServ host's performance requirements. In this section, we do not delve into the details of the state learning, collection, and maintenance techniques and protocols. These issues are presented in detail in Chapter 4. But we take a moment to discuss the necessary pillars of subnet state collection and maintenance that precede and enable a CServ transaction.

First and foremost, the subnet is responsible for the learning and reporting of the necessary CServ performance metrics that are used by the MC for CServ internetworking service discovery and composition. As previously discussed, a subnet maintains a unified routing solution, where the choice of the routing policy is left to the subnet administrator. Consistent with this routing policy, the subnet is required to measure and learn the CServ Intranetwork Service performance in terms of the CServ performance metrics between its CServ network components using the *State Measurement Service*. Specifically, the CServ performance metrics must be captured between each pair in the set of (active) access and gateway routers. Furthermore, each access router is responsible for measuring and learning some succinct set of CServ performance metrics that capture the upstream and downstream performance to their constituent access networks. The learned CServ performance metrics are reported to the local SC, which is responsible for the maintenance and synchronization of the most up-to-date subnet-internal state. It is then the responsibility of the SC to report updates regarding the necessary condensed subset of this collected state to the global MC system. For now, we suppress the discussion of which state is necessary at the global level and leave that for later in the dissertation. The MC is responsible for the maintenance of the state required to perform service discovery and computation for CServ end-to-end intersubnet paths.

Together, the SCs and the MC form the fundamental control hierarchy of the CServ architecture, separating the control responsibilities based on their respective domain. The SC is responsible for

control and state aggregation at the local subnet level, while the MC is responsible for the aggregation of state necessary only for the composition of internetwork service and control of routing at this subnet-to-subnet granularity. Both levels of control require logical centralization, even if they are implemented as physically distributed controllers to mitigate single point-of-failure pitfalls inherent in physically-centralized systems. There is a breadth of literature in the software-defined network research area that is concerned with the design and synchronization of distributed but logically-centralized controllers through distributed system algorithms, and this is not the focus of this document. Furthermore, a robust *off-band control channel* is provisioned to connect the CServ-enabled hosts and routers to the SCs, and the SCs to the MC. As we describe later, the off-band control channel is used both by the CServ-enabled hosts to request CServ internetwork service and by the active subnet routers to report measured CServ performance metrics. Reliability, availability, and predictable signaling delay with minimal jitter are vital concerns for the design of the control channel. Likely, a heavily overprovisioned control network with a simple topology, such as a tree, to avoid contention between end-users would best suit the need for this system. This concept is illustrated in Fig. 2-11. The dissertation does not focus on the design and allocation of the off-band control channel, but its use and requirements are considered in detail.

Second, at a more fundamental level, the subnet is responsible for reporting the host membership of the subnet – that is, the addresses of the CServ hosts that have upstream access to the subnet core through an active access router. As another use of the robust off-band control channel, a simple association protocol can be used to register CServ-enabled hosts and routers with the SC when they join the subnet, while a complementary dissociation protocol can be used to remove registered CServ devices when they leave the subnet. The local SC can, in turn, report aggregated sets of addresses registered with CServ capabilities to the MC. This is a low overhead maintenance requirement for the CServ control hierarchy that generates infrequent updates. We discuss later when describing the MC service discovery and composition algorithm that the MC also needs to know the address of the access router that provides this upstream connection to the subnet core. The hierarchical addressing scheme of IPv4 and IPv6 naturally lends itself to this need, but we do not require the use of these specific addressing structures in the CServ architecture. The choice of addressing framework is considered in more detail later in the dissertation.

Lastly, each subnet is responsible for the measurement, collection, and maintenance of the state information required to operate its internal routing solution. For example, this interior routing method

may be some form of IP interior gateway protocol, such as OSPF or IS-IS which only require distributed link state protocol updates, or it could involve the use of pre-established MPLS circuits, or it may even implement some local form of the MC intersubnet service discovery and composition protocol (more on this briefly in Section 2.4). The CServ architecture does not specify the use of any particular interior routing protocol or forwarding mechanism. In this way, subnet administrators are free to implement a strategy that best suits their network capability and business needs. However, this chosen interior routing strategy serves as the CServ Intranetwork Service for transiting critical messages. The routing solutions dictated by the policy are those that are used to learn and measure the CServ performance metrics that are in turn employed by the MC for internetwork service discovery and composition. For this reason, the unified routing strategy of a subnet is critical both to its capability to bear critical datagrams and to accurately measure and report that capability.

In our step-by-step example in this section, we assume that the network subnets each implement some form of IP interior gateway protocol, such as OSPF, and IP-based forwarding. In this scenario, the subnet administrators would only need to upgrade gateway routers to support CServ service (and any subnet access routers that provide upstream connection to an access network with CServ-enabled hosts). Other core and access routers in the subnet could operate in legacy modes without upgrade or replacement. OSPF is an IP link state protocol that uses Dijkstra's Algorithm [50] to find shortest network paths. After the exchange of link state protocol updates, each router in the subnet builds a graph that represents the interior subnet topology map and then computes shortest paths based on the interior routing metrics. The results of this computation are generally summarized in the router's forwarding information base (FIB), often implemented with fast hardware lookup mechanisms such as ternary content addressable memory (TCAM), which allows for quick resolution of a datagram's IP destination address. In Fig. 2-12, we show abbreviated portions of the FIBs at a subset of the routers in each subnet that could result from the use of OSPF as an interior routing policy. Although the route options are limited and obvious in both the source and destination subnets, we note that the shortest path in the transit subnet is the two-hop path between the gateway routers rather than the direct path. We also note that, in practice, the "Next hop interface" field would be populated by the MAC address of the network interface card for the appropriate outgoing router interface rather than the address of the next hop router; this convention is used in the illustration for simplicity.

Prior to the initiation of the CServ transaction, the state related to the interior routing policies of each subnet would already be established – the FIBs would be populated at each router with appropriate forwarding rules. This is not calculated and populated on-demand when the CServ transaction begins. Furthermore, the CServ performance metrics related to these routing policies and the CServ Intranetwork Service paths would have been measured by the subnets, and the SC and MC systems would have the most up-to-date performance metrics stored in memory. The ongoing maintenance of state is a fundamental requirement in the CServ architecture because it reduces the amount of pre-transaction overhead and improves network response time when a CServ message is generated. Otherwise, the on-demand, per-transaction learning and reporting of the necessary CServ performance metrics would severely limit the ability of the service to provide reasonable end-to-end delay and impede the efficacy of the design. In the next section, we describe the remaining pre-transaction communication and computation overhead, noting here that the execution speed of this process is of integral concern since pre-transaction delay contributes to the overall critical datagram delay and the ability of the CServ architecture to provide probabilistic performance guarantees.

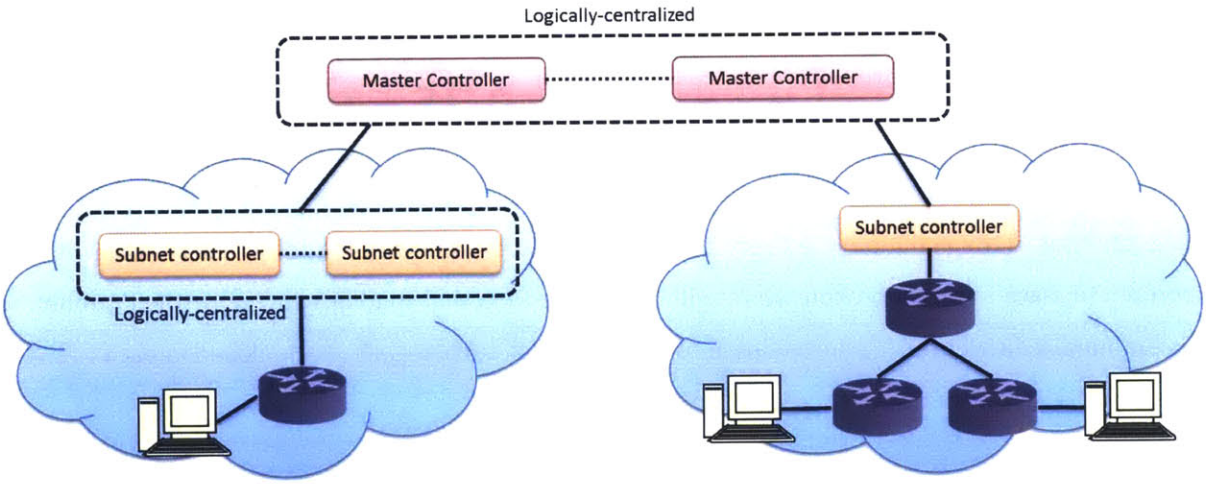


Fig. 2-11: This is a notional illustration of the control hierarchy composed of logically-centralized SCs at each subnet and a global, logically-centralized MC. This illustration also depicts the robust off-band control channel structure connecting CServ-enabled devices and the controllers themselves. This control channel is provisioned to bear service requests/responses and state measurement reports.

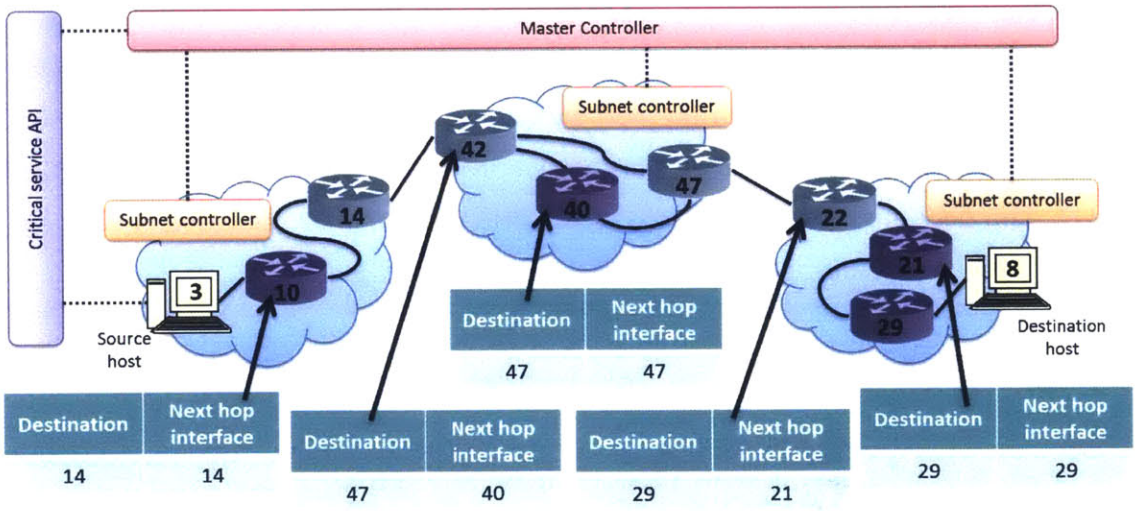


Fig. 2-12: Prior to the initiation of a CServ transaction, routers populate their forwarding tables with the appropriate state information consistent with the unified internal routing strategy of their subnet. In this example, a distributed IP shortest path routing protocol, such as OSPF, is employed in each network subnet and is used to build the forwarding tables at each router.

2.3.2 CServ Transaction Setup

With the necessary state learning and collection procedures in place and routine state maintenance ongoing, the network is ready to handle a critical message datagram transaction. We describe the transaction-specific control plane process that precedes the data plane transmission of the critical datagram in this section.

In the first step of this pre-transmission transaction setup, the creator of the critical message requests a desired level of service in terms of the CServ performance metrics. Once the critical datagram payload is generated at the source host, the source host initiates a CServ Request (CSR) via the CServ API, as depicted in Fig. 2-13. The purpose of the CSR is to describe the desired level of service for the critical datagram in a structured way, which in turn is used by the MC to algorithmically determine the ability of the network to bear the critical message with this level of service. The CSR contains the following fields:

- **Source host** – The source host includes its own network address in the CSR (where the addressing scheme depends on the network implementation). This address is used by the MC as the starting point of the CServ transaction in the service discovery and composition algorithm, and it is also used as the return address for the eventual service response generated by the MC.

- **Destination host** – The destination address for the CServ critical message datagram is specified. For now, we assume a unicast messaging service, so this is a unique host address. (In the future, we can consider expansions to the CServ architecture that allows for addressing multicast groups. For now, multicast can be implemented as the superposition of multiple unicast transactions.) This address is used by the MC as the terminal point of the CServ transaction in the service discovery and composition algorithm.

- **Primary CServ performance metric** – For the purpose of service discovery and composition at the MC, the source host specifies which of the two CServ performance metrics is deemed "more critical" to the application, either the message reliability or delay. This performance metric is used to algorithmically discover intersubnet paths in the MC service discovery and composition algorithm, rather than jointly optimizing over some heuristic combination of the disparate metrics. The MC algorithm and considerations are covered in detail in Chapter 5.

- **Minimum reliability** – The CServ source host application specifies a numerical value for the minimum tolerable end-to-end reliability, or the complement of the datagram loss probability (for more detail, see Definition 1). The support of this value is the real numbers between 0 and 1, inclusive. We note that a value of 0 imposes no service constraint on the achievable reliability of the service, whereas a value of 1 is infeasible to achieve in any real network deployment. We anticipate that reasonable and useful minimum reliability requirements will be greater than 0.9. For the purpose of discussion, the minimum reliability value is represented as a 32-bit single-precision floating point number [51], although any numerical representation format that allows for appropriate precision in the valid parameter range would suffice for the implementation of this field.

- **Maximum delay** – The CServ source host application also specifies a numerical value for the maximum tolerable end-to-end delay for the critical message datagram from the time that the CSR is generated to the time that the critical datagram is received by the destination host (including the delay incurred for the pre-transaction control process overhead). For a more precise description, see Definition 2. The support of this parameter is the nonnegative real numbers. We note that a value of 0, however, is infeasible to achieve in any real network deployment. Based on the driving design applications as discussed in Chapter 1, we anticipate that useful maximum delay requirements will be on the order of seconds. For the purpose of discussion, the maximum delay value is represented as a 32-bit unsigned integer that specifies the number in terms of microseconds. This format allows the CServ-enabled application to make requests up to a maximum delay value of approximately 4,295 seconds (or approximately 71.5 minutes).

- **Minimum path diversity** – Although we have yet to discuss in detail the role that path diversity plays in the CServ architecture (see Chapter 5), we note here that the source host application has the opportunity to request some minimum number of end-to-end disjoint paths. The support of this parameter is positive integer values. Ample discussion of the use of diversity routing follows later in the dissertation, but for now we summarize the dialogue by saying that end-to-end path diversity allows both for improved CServ transaction reliability and for survivability against unpredictable "Black Swan" events [52] that are not part of the CServ

performance metric learning model. Considering survivability against unpredictable events alone, we suggest that an implementation of the CServ architecture uses 2 as a default value for this field if the user does not explicitly specify a value for this field. However, we note, without further detail, that the use of end-to-end path diversity requires additional pre-transmission computation overhead and delay at the MC. For that reason, the MC limits the maximum number of end-to-end disjoint paths that it can discover (not to the mention that the network topology itself may further limit this number). In the MC service discovery and composition algorithm to be introduced shortly, the actual minimum path diversity used is the minimum between the CServ user-specified value and the MC's maximum number of end-to-end disjoint paths considered by the service discovery and composition algorithm. Only a few bits are required to represent this value in the CSR – somewhat arbitrarily, we choose 8 bits for the time being, allowing for 255 as the maximum value for minimum path diversity (which is far beyond a useful request in practice).

- **Generation timestamp** – Lastly, the source host application places a timestamp in the CSR that represents the time of generation of the request for critical service and serves as the official "start of the clock" for the maximum tolerable delay of the transaction. The CServ architecture assumes access to a synchronized global clock, such as that available through Global Positioning System (GPS) timing.

These constitute the most important fields of the CSR. Additional control information may be useful in implementation, such as a request sequence number that distinguishes between two identical CSRs from a particular source host destined for a particular destination host with the same CServ performance metric requirements. Additionally, some effort is required to ensure the integrity and validity of the CSR. Beyond a typical checksum to ensure that the data is not modified intentionally or naturally in transit, we would want to consider the use of a cryptographic signature to validate the authenticity of the CSR-generating source. This would help to subvert attempts to forge CSRs or to mount a denial-of-service (DoS) attack on the MC. Security concerns of the CServ architecture are discussed in more detail as future work in Chapter 6.

The CServ architecture design uses a robust off-band control channel to bear CSRs via the CServ API and connect CServ-enabled hosts to the logically-centralized MC. This off-band control channel additionally

carries state measurement updates from the subnet routers to the SC and, in turn, the MC. As we see later, predictable timing and reliable transmission on the control channel is essential to accurate MC service computation, so careful attention must be given to the control plane design.

The next step of the pre-transmission transaction setup process is the MC service discovery and computation procedure. After receiving the CSR, the MC executes the Critical Service Discovery and Composition Algorithm (CSDCA) to determine whether or not the network can bear the critical service datagram to its intended destination host with the requested level of service, and, if so, how exactly that level of service can be achieved (see Fig. 2-14). In addition to the CSR, the global subnet-level network topology description, learned as part of the state measurement and maintenance procedure, is an input to the CSDCA. Specifically, this global network representation visualizes the intersubnet topology as a graph of interconnected active gateway routers joining the constituent subnets. The graphical representation of the global network and the rationale behind the representation is covered in depth in Chapter 5. Furthermore, the details of the CSDCA are discussed in full in that chapter as well. For now, it suffices to say that after the execution of the algorithm, the MC decides if it can support the requested service or not. If it can support the requested service, it has found an explicit intersubnet path or set of diversity paths in terms of the intermediate gateway router hops from source host to destination host that support the service level based on the known state information.

The third and final step of the control plane transaction setup involves returning the result of the CSDCA execution to the source host application via the CServ API on the off-band control channel. This response takes the form of either a *service access* or *service denial*. A service access response grants the source end-user application permission to transmit the critical datagram payload in some specified manner; namely, the response itself bears the explicit intersubnet route or set of routes that the CServ datagram must follow from the source host to the destination host – we call this the *CServ Internetwork Service path or paths*. As the illustration of Fig. 2-15 shows, these routes begin with the source subnet upstream access router (which is particularly useful when the source host is multihomed), followed by a string of subsequent gateway routers (that describe the path from subnet to subnet), and ending with the destination subnet upstream access router (which is particularly useful when the destination host is multihomed). We note again here that the service access response prescribes the intersubnet path that the CServ datagram must follow, but it does not specify the intrasubnet path that the datagram follows within any particular subnet. Consider the example of Fig. 2-15, the service access response dictates that

the CServ datagram transits ingress gateway router 42 and continues on to egress gateway router 47, but it does not specify if the datagram should follow the one-hop route interconnecting these gateway routers or if it should take the two-hop route that traverses access router 40 in the core of the transit subnet. This determination is left up to the subnet administrator and the CServ intrasubnet service routes established by the subnet's unified internal routing policy. In this way, the CServ service access response only provides the source host to destination host route or set of routes at subnet-level granularity, while the paths internal to subnets along the route are left up to the discretion of the subnet network provider.

Alternatively, a service denial response indicates to the source end-user application that the MC has determined that the network cannot provide the requested level of service for the CServ transaction. So long as the request is reasonable, the CServ network should be designed and dimensioned to avoid this response. In the future development of the CServ architecture, we suggest the inclusion of an intermediate response, the *service counteroffer*. This would allow the MC to propose what it determines to be its best available service level, even if it does not meet the original demands of the CSR. It would then be up to the requesting application to decide whether or not to transmit the datagram at this newly offered level of service. That being said, this scheme would require a different paradigm of CServ application design; the application requests a particular level of service, knowing that it is willing to accept service at a degraded level (meaning with either less reliability, longer end-to-end delay, or both).

In addition to the type of response and explicitly-routed address string or strings of the service access response, additional control information may be required in the responses from the MC. As in the CSR, a response sequence number that distinguishes between two identical service responses for a particular source host and destination host pair may be useful. The response should include a timestamp that indicates the latest time at which the source host can transmit the CServ datagram on the data plane without violating the computed service. Even though the CSDCA execution attempts to account for the transmission time of the control messages in the pre-transmission transaction setup, this "belt-and-suspenders" field ensures that the source application is protected against any improbable and unforeseen delays or disruptions in the robust off-band control channel. Lastly, methods to ensure the authenticity of the response and the integrity of the returned explicit route information, such as a cryptographic signature, would be helpful to provide defense against a malicious actor trying to spoof the MC service.

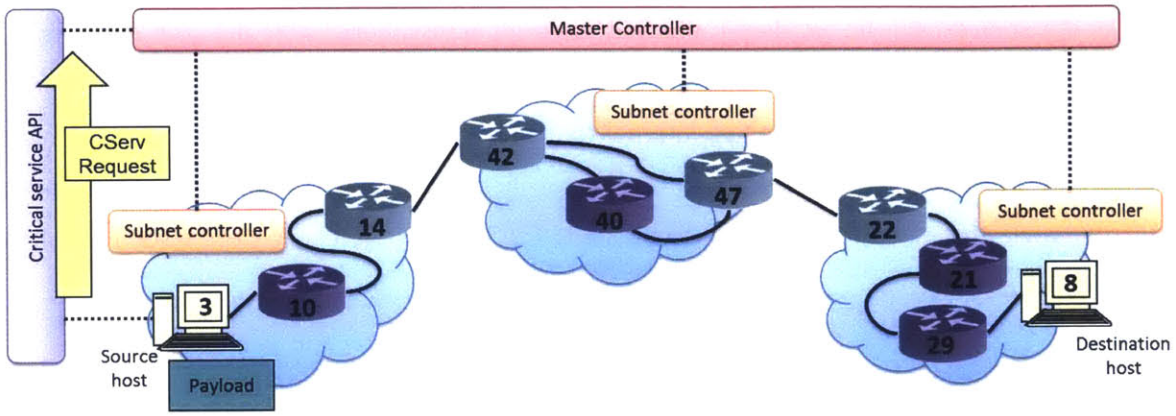


Fig. 2-13: When the critical datagram payload is created at the source host, the CServ source application generates a CServ Request via the CServ API which is transmitted to the MC on the off-band control channel.

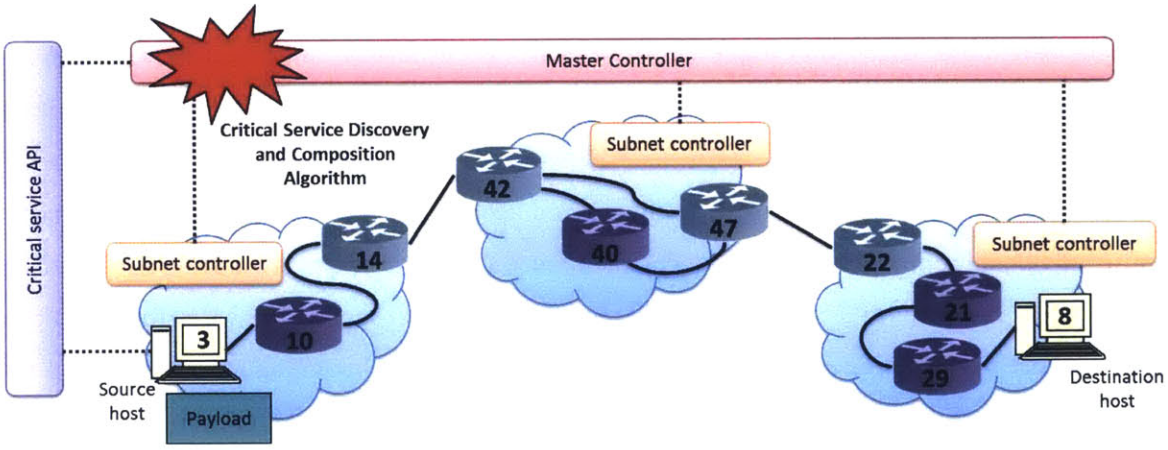


Fig. 2-14: After receiving the CSR, the MC executes the discovery and composition algorithm which determines the availability of an intersubnet path or set of paths that meet the service requirements required by the pending critical message transaction.

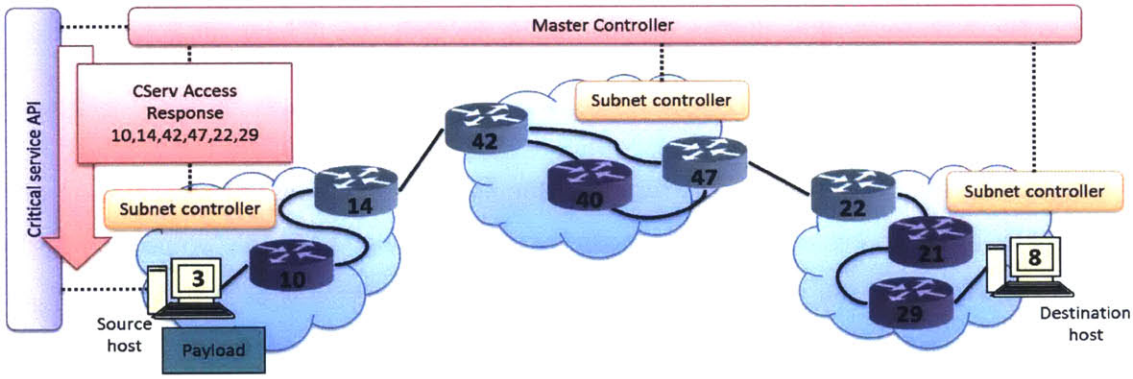


Fig. 2-15: Following execution of the CSDCA, the MC forms a service response, either a service access or service denial, and returns the result to the source host application using the CServ API on the off-band control channel.

2.3.3 CServ Datagram Transmission

Following the on-demand, transaction-specific setup process described in Section 2.3.2, the critical message is ready for internetwork data plane transmission. This process bears the message from the source host to the destination host according to the intersubnet route or routes specified by the service access response of the MC. Along the way, the path followed by the critical message as it traverses each subnet along the intersubnet route depends on the routing strategy and policy of that individual subnet. In the following step-by-step example using the network of Fig. 2-10, the subnet internal routing state of Fig. 2-12, and the service access response of Fig. 2-15, we describe this process as the critical message traverses a three-subnet data plane path from source host to destination host, including the source subnet, one transit subnet, and the destination subnet. Consistent with the subnet internal routing state illustrated in Fig. 2-12, we assume that each subnet along the path uses IP forwarding mechanisms for their CServ Intranetwork Service.

Upon reception of the MC service access response, the source host begins the transmission of the critical message by preparing the CServ datagram, as shown in Fig. 2-16. This preparation process involves the generation of the *CServ Internetwork Service header*, which is used to encapsulate the critical message payload to form the *internetwork CServ datagram*. This header bears the explicit CServ Internetwork Service path computed by the MC to satisfy the service demand of source host's CServ application, as depicted in the figure. There are other components of the CServ Internetwork Service header, but we save the discussion of the details of these fields for Chapter 3. Once the internetwork CServ datagram is prepared, the source host transmits the datagram to its upstream access router. If the source host is multihomed and has network access through multiple upstream access routers in the same subnet or multiple upstream access routers in disparate subnets, the first address of the CServ Internetwork Service path in the datagram header specifies the correct transmission interface.

When the upstream access router receives the CServ datagram, it has the responsibility to prepare the internetwork CServ datagram for *CServ Intranetwork Service*, as illustrated in Fig. 2-17. Throughout this step-by-step example, we show that it is always the responsibility of the first active router that encounters the internetwork CServ datagram within a subnet to prepare it for intrasubnet transmission, although this first active router is not always an access router (in fact, it is frequently an ingress gateway router). Using the next hop of the explicit CServ Internetwork Service path (which will be a gateway

router in the source subnet), the access router forms the appropriate *CServ Intranetwork Service header* according to the subnet internal routing policy. In this case, using IP forwarding mechanisms, the access router generates an IP header (either IPv4 or IPv6) and uses the next hop address of the CServ Internetwork Service path as the destination address in the header. If the subnet allows for priority queueing and transmission at its routers, the datagram header can be marked to indicate the desire for priority treatment compared to best effort traffic. In the case of IP intranetwork service, this can be done using the DiffServ field in the header [30]. After populating the other necessary IP header fields, the access router encapsulates the internetwork CServ datagram, where it becomes the payload of an IP datagram. The IP datagram header is then matched against the forwarding table at the access router to determine its outgoing, or next local hop, interface, and the IP datagram is then transmitted on that interface. The internetwork CServ datagram is tunneled through the subnet inside an IP datagram, the network transport of the subnet's CServ Intranetwork Service. In the case of our example, the next hop of the Internetwork Service path is gateway router 14, so access router 10 forms an IP header with 14 as the destination IP address. Since gateway router 14 is directly connected to access router 10, the forwarding table at router 10 specifies that the next hop interface for the CServ Intranetwork Service is that which connects to router 14.

When the source subnet egress gateway router receives and processes the IP datagram, it finds that it is the destination of the CServ Intranetwork Service path. As shown in Fig. 2-18, it removes the leading IP header (the CServ Intranetwork Service header) to reveal the internetwork CServ datagram. It then uses the next hop of the explicit CServ Internetwork Service path (which is often a peering gateway router in a neighboring subnet) to decide how to forward the CServ datagram. Although the forwarding table state of the egress gateway router is not shown in the illustration, gateway routers in peering subnets should know how to forward datagrams between them. Since both gateway routers must be active (CServ-capable), encapsulation in an IP datagram is not necessary for this transmission, but it can be used if the exchange policy between the peering subnets requires it. In our example, egress gateway router 14 of the source subnet transmits the internetwork CServ datagram to the next hop in the CServ Internetwork Service path, which is peering ingress gateway router 42 of the neighboring subnet.

The ingress gateway router of the transit subnet receives the internetwork CServ datagram and assumes responsibilities similar to the upstream access router in the source subnet; we illustrate in Fig. 2-19 that the ingress gateway router prepares the internetwork CServ datagram for intrasubnet transmission.

Using the next hop of the explicit CServ Internetwork Service path (which again will be a gateway router in the transit subnet), the ingress gateway router forms the appropriate CServ Intranetwork Service header according to the transit subnet's internal routing policy. Again, since we assume IP forwarding mechanisms, the ingress gateway router creates an IP header and uses the next hop address of the CServ Internetwork Service path as the IP destination address in the header. The gateway router encapsulates the internetwork CServ datagram as the payload of an IP datagram, preparing the internetwork CServ datagram for an IP transport tunnel, and then the IP datagram header is matched against the gateway router's forwarding table state to determine its outgoing interface. In the example, the next hop of the CServ Internetwork Service path is gateway router 47, so the ingress gateway router 42 forms an IP header with 47 as the destination IP address. The forwarding table state of router 42 then specifies that the IP datagram should be transmitted on the outgoing interface for router 40 for the CServ Intranetwork Service.

In Fig. 2-20, we show the actions of the core access router in the transit subnet when it receives and processes the critical message encapsulated in an IP datagram. The router checks the destination address of the CServ Intranetwork Service header, an IP header in this case. Upon finding that it is not the destination, the core access router consults the state of its forwarding table to look up the outgoing interface for the datagram and transmits it on that interface. In the example, router 40 is not the destination of the IP datagram, so it performs a forwarding table lookup for the destination IP address. The forwarding information base of router 40 specifies that the IP datagram should be transmitted on the outgoing interface for router 47 for the CServ Intranetwork Service. Note that this process does not require that router 40 is a CServ-enabled, or active, router. What we have described in this paragraph is standard IP datagram processing.

Upon datagram reception, the egress gateway router of the transit subnet performs the same processing tasks as the egress gateway router of the source subnet. As shown in Fig. 2-21, it finds that it is the destination of the CServ Intranetwork Service path by checking the destination IP address of the CServ Intranetwork Service header (the IP header, in this case). It then strips away the leading IP header to reveal the internetwork CServ datagram. The next hop address of the explicit CServ Internetwork Service path in the CServ Internetwork Service header is used to determine how to forward the CServ datagram. The transit subnet egress gateway router of our example, router 47, transmits the

internetwork CServ datagram to the next hop in the CServ Internetwork Service path, which is peering ingress gateway router 22 of the neighboring subnet.

When the ingress gateway router of the destination subnet receives the internetwork CServ datagram, it performs the same processing tasks as the ingress gateway router of the transit subnet previously described. The difference for the destination subnet is that the next hop address of the explicit CServ Internetwork Service path is not a gateway router, but the access router that provides upstream connection for the destination host. So when the ingress gateway router of the destination subnet generates the appropriate CServ Intranetwork Service header according to the subnet's internal routing policy (in this example, an IP header), it uses the access router IP address as the destination field of the header. This is depicted in Fig. 2-22. After encapsulating the internetwork CServ datagram as the payload of the intranetwork CServ datagram (which is an IP datagram), the ingress gateway router uses its local forwarding table state to determine the appropriate outgoing interface. In the example, the next hop of the CServ Internetwork Service path is access router 29, so the ingress gateway router 22 creates an IP header with 29 as the destination IP address. The forwarding table state of router 22 then specifies that the IP datagram should be transmitted on the outgoing interface for router 21 for the CServ Intranetwork Service.

Fig. 2-23 illustrates the role of the core access router of the destination subnet (router 21 in the example), which performs in the exact same manner as the core access router of the transit subnet. After the datagram is transmitted to and received by the upstream access router of the destination subnet, it finds that it is the destination of the CServ Intranetwork Service header (or in this example implementation, the destination of the IP header). It then strips away the leading IP header to reveal the internetwork CServ datagram, as shown in Fig. 2-24. Upon examining the explicit CServ Internetwork Service path in the CServ Internetwork service header, the upstream access router of the destination subnet determines that it is at the end of the explicitly-routed path, the last hop. This signals to the access router that it provides upstream access to the destination host in one of its connected access networks. Using the destination host address in the CServ Internetwork Service header (which is not shown in the cartoon of Fig. 2-24), it transmits the internetwork CServ datagram on the appropriate outgoing interface. In the example, the upstream access router for the destination host, router 29, removes the CServ Intranetwork Service header, recognizes that it is the last hop of the CServ

Internetwork Service path, and then transmits the internetwork CServ datagram via the appropriate connected access network to destination host 8.

Finally, completing our step-by-step example, the destination host receives the internetwork CServ datagram and removes the leading CServ Internetwork Service header to reveal the critical message payload, as shown in Fig. 2-25.

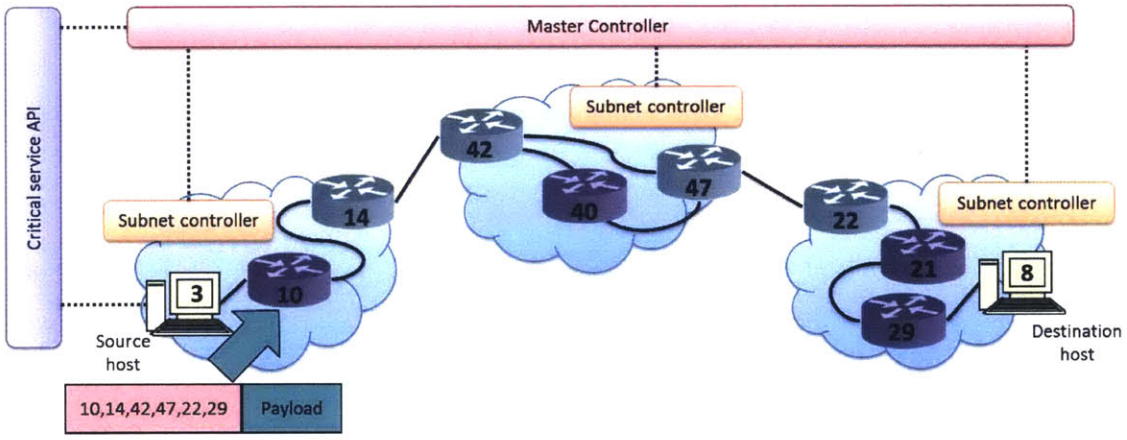


Fig. 2-16: Prior to passing the CServ datagram to the upstream access router prescribed by the MC-computed route, the source host prepends the CServ Internetwork Service header which contains the explicit subnet-to-subnet path information.

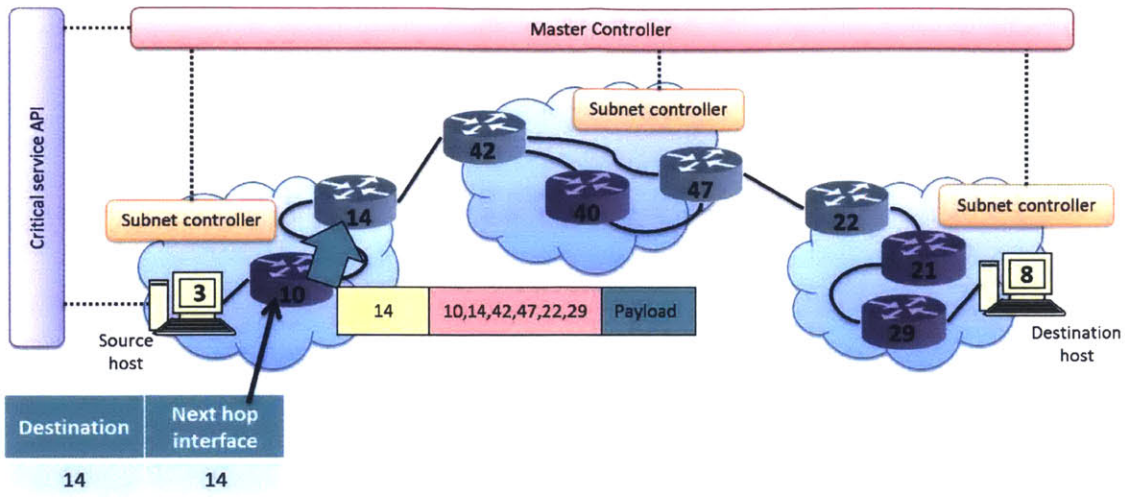


Fig. 2-17: Upon receiving the CServ datagram, the access router uses the CServ Internetwork Service path next hop address (here, router 14) and the subnet's routing policy to generate and prepend the CServ Intranetwork Service header, which in this example is an IP header (with destination address 14). In this way, the CServ datagram is encapsulated in an IP datagram.

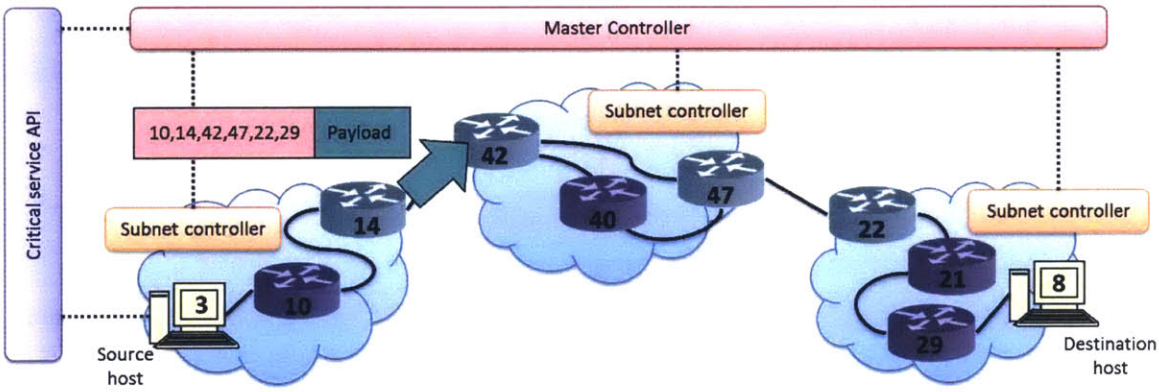


Fig. 2-18: The source subnet egress gateway is the "destination" for the CServ Intranetwork Service, so it removes the leading header from the CServ datagram. It then uses the exposed CServ Internetwork Service header to forward the datagram to the appropriate peering subnet gateway (here, router 42).

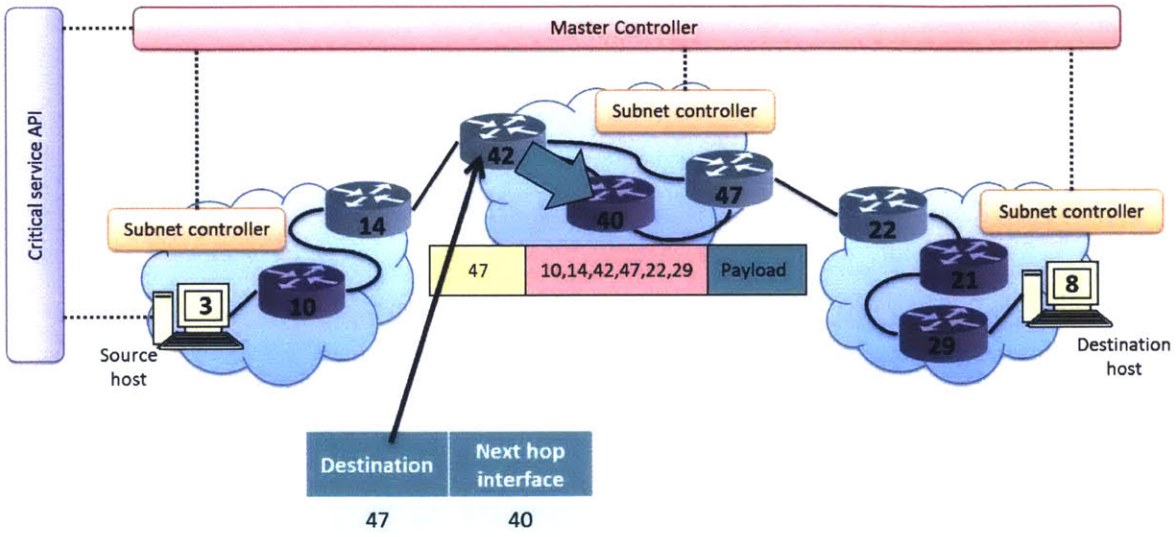


Fig. 2-19: Just like the upstream access router in the source subnet, the ingress gateway of the transit subnet uses the CServ Internetwork Service path next hop address (here, router 47) and the subnet's routing policy to generate and prepend the new CServ Intranetwork Service header, an IP header here (with destination address 47). In this way, the CServ datagram is encapsulated in an IP datagram.

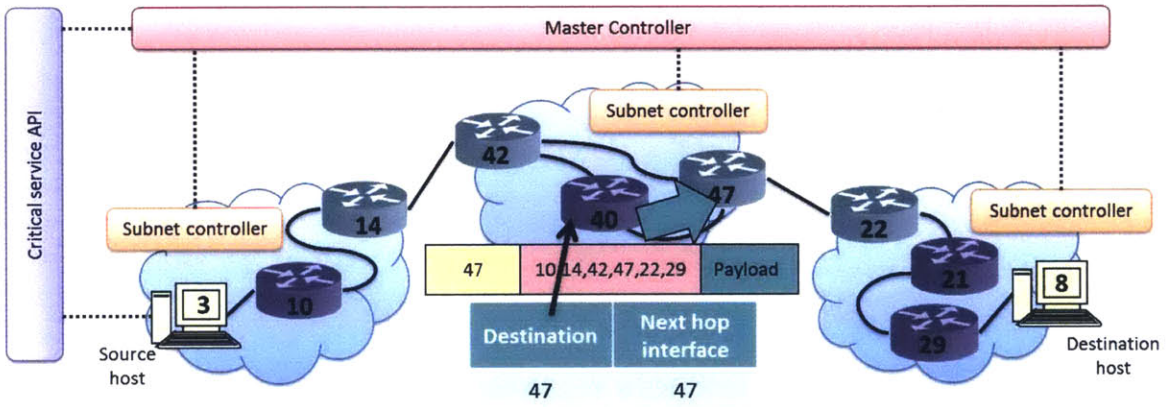


Fig. 2-20: The CServ Intranetwork Service header is used to direct the CServ datagram through the transit subnet according to the appropriate CServ service path and unified subnet routing policy as stored by the forwarding table state of the routers. In this example, router 40 need not be active or CServ-enabled, as the CServ datagram is encapsulated as an IP datagram.

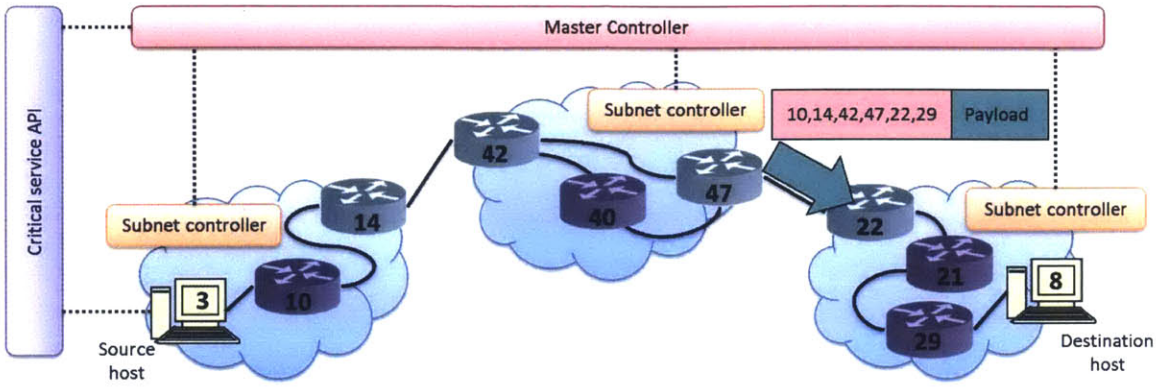


Fig. 2-21: As in the source subnet, the transit subnet's egress gateway is the "destination" for the CServ Intranetwork Service path, so the gateway router removes the leading header from the CServ datagram. The exposed CServ Internetwork Service header is then used to forward the datagram to the appropriate peering subnet gateway router (here, router 22).

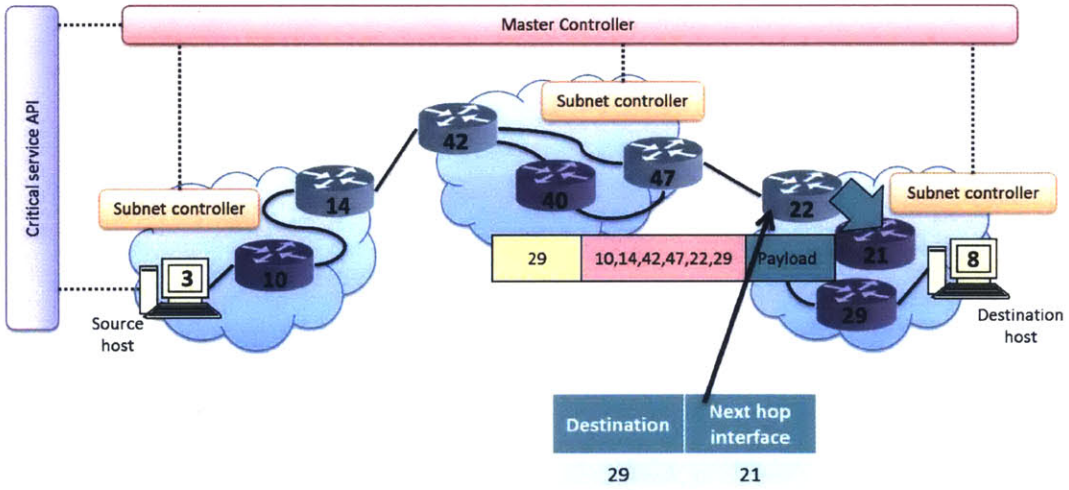


Fig. 2-22: Like the ingress gateway of the transit subnet, the ingress gateway of the destination subnet uses the CServ Internetwork Service path next hop address (here, router 29) and the subnet's routing policy to generate and prepend a new CServ Intranetwork Service header, an IP header in this example (with destination address 29).

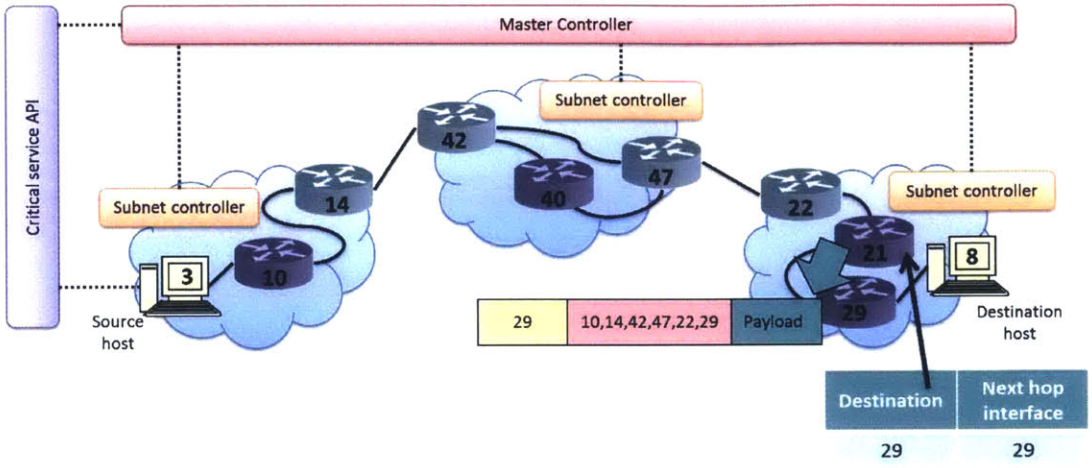


Fig. 2-23: The CServ Intranetwork Service header is used to direct the CServ datagram through the destination subnet. In this example, router 21 need not be active or CServ-enabled, as the CServ datagram is encapsulated as an IP datagram.

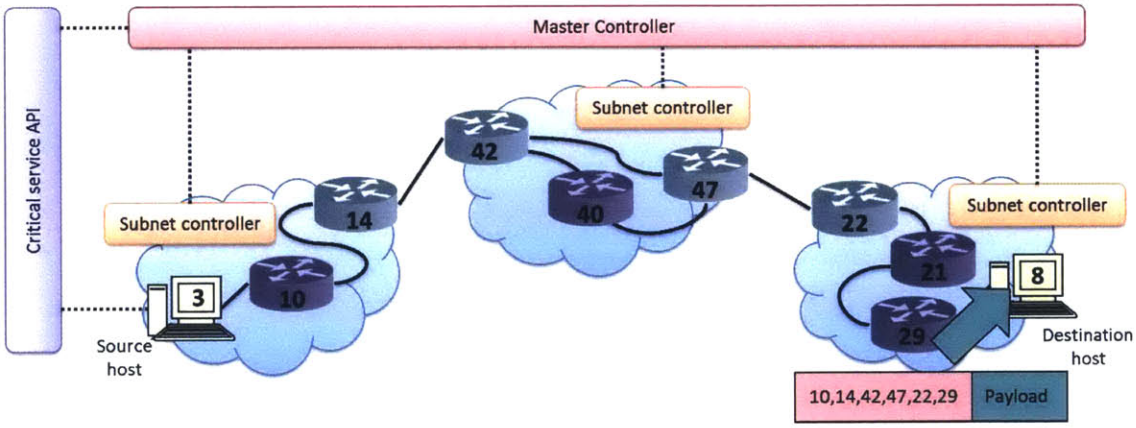


Fig. 2-24: The destination host's upstream access router is the "destination" for the CServ Intranetwork Service path, so it strips the leading CServ Intranetwork Service header from the CServ datagram. The exposed CServ Internetwork Service header indicates to router 29 that it is the last hop in the CServ Internetwork Service path, so it forwards the CServ datagram to the destination host in its connected access network.

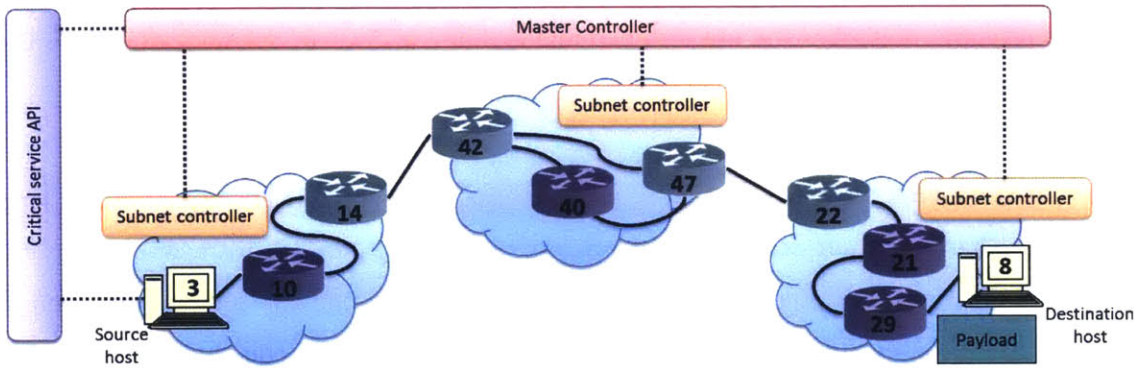


Fig. 2-25: Upon receiving the CServ datagram, the destination host removes the leading CServ Internetwork Service header to reveal the payload of the critical datagram that was initially generated by the source host.

2.3.4 End-to-End Internetwork Diversity Routing

We briefly reconsider the previous example of Sections 2.3.1-2.3.3 with a new network topology, as shown in Fig. 2-26. In this network, there is more than one option for an intersubnet path from source subnet to destination subnet, unlike in our previous topology example of Fig. 2-10 with a degenerate CServ Internetwork Service solution. As previously mentioned, the use of diversity routing solutions is a central feature of the CServ architecture since it allows both for an improvement in end-to-end reliability and a degree of survivability against unpredictable, high-impact events. Imagine now that the MC returns the service access response of Fig. 2-27 after reception of the CSR and execution of the CSDCA. This response prescribes the use of a two-fold end-to-end diversity solution, where the CServ datagram should be transmitted over two CServ Internetwork Service paths.

Note that the intersubnet paths included in the example response of Fig. 2-27 are disjoint except for the two endpoints where the routes converge – the access routers that provide upstream connection for the source and destination hosts. In our context, we consider these disjoint intersubnet paths even with the common access router endpoints. In fact, we typically generalize and describe two paths as subnet-disjoint paths as long as the only subnets they share are the source and/or destination subnets (this relaxed definition allows them to even share the same source subnet egress gateway router or destination subnet ingress gateway router). Without this relaxed definition of disjoint internetwork paths, the network topology places a tight constraint on the existence of disjoint internetwork paths. Before moving on, we mention that CServ-enabled hosts multihomed to distinct subnets provide the CSDCA with the additional flexibility to find truly subnet-disjoint internetwork paths.

Upon receiving the service access response with a two-fold diversity routed solution, the source host replicates the critical message payload and prepares two separate internetwork CServ datagrams before transmission, as depicted in Fig. 2-28. A distinct CServ Internetwork Service header is generated for each internetwork CServ datagram, with one of the explicit CServ Internetwork Service paths placed in each header. Although we continue to delay discussion of the full header details until Chapter 3, we note that the CServ Internetwork Service header contains a session sequence number – a unique increasing counter based on the source host and destination host double. This value identifies the critical message payload, so each of the internetwork CServ datagrams generated during the preparation process are

endowed with the same value. After prepending the CServ Internetwork Service headers to each of the datagrams, the source host transmits them, in series, to the appropriate upstream access router.

After replication, preparation, and transmission of the two internetwork CServ datagrams by the source host, these datagrams are treated during data plane transmission just as described in Section 2.3.3. It is shown in Fig. 2-29 that each datagram is processed by the upstream access router as before, where the next hop address in the explicit CServ Internetwork Service path is used in the determination of the CServ Intranetwork Service path. In this example, the two internetwork CServ datagrams have different next hop addresses, and this is reflected in the prepended CServ Intranetwork Service headers and the forwarding action at the source subnet upstream access router.

If both internetwork CServ datagrams reach the destination host successfully, the replication of the critical message payload is quenched through the use of the session sequence number in the CServ Internetwork Service header. The destination host is responsible for maintaining a counter state for the source host and destination host double, and repeated receptions of the same critical message payload are discarded.

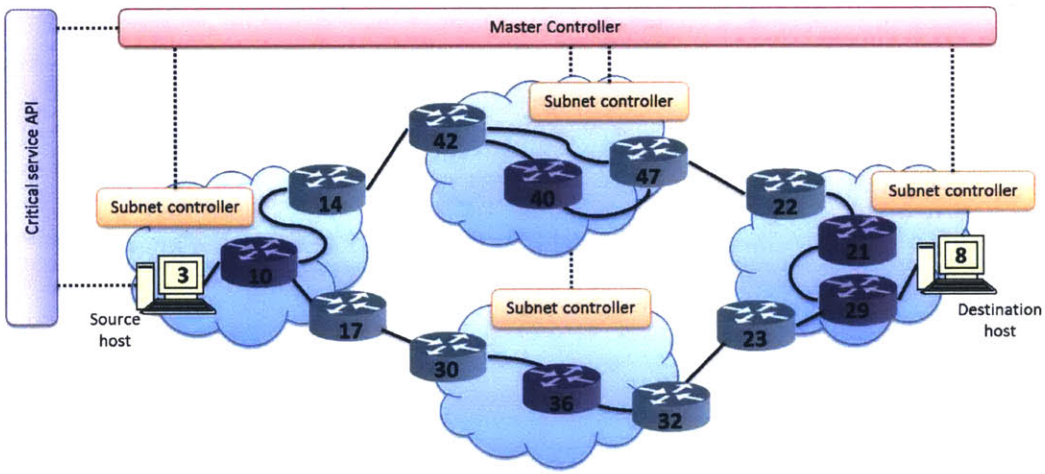


Fig. 2-26: This modified example network topology is used to describe the transmission of a CServ datagram when the MC service access response indicates the use of end-to-end diversity routing to achieve the desired level of service, as in Section 2.3.4.

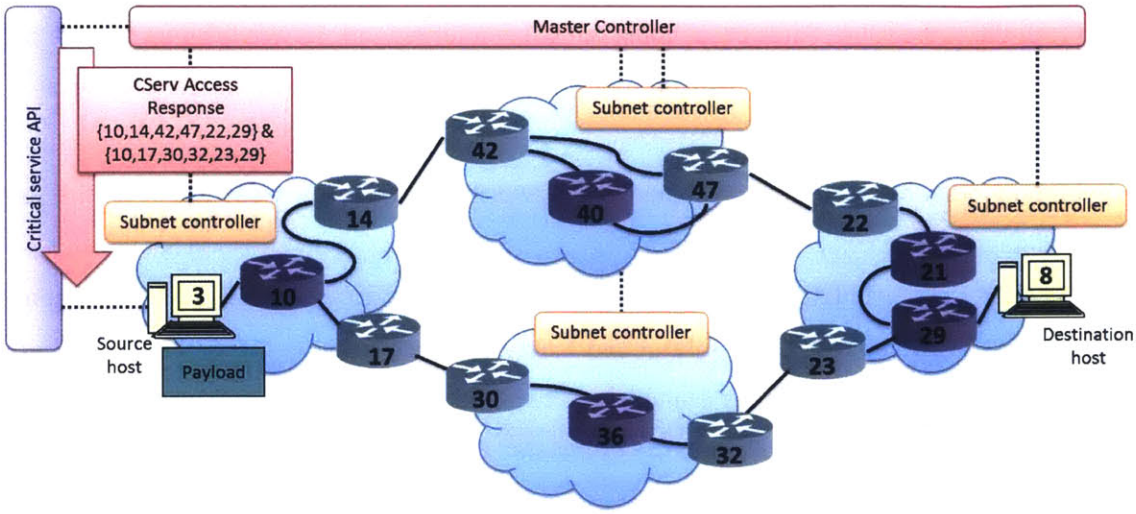


Fig. 2-27: Following execution of the CSDCA, the MC prescribes the use of two-fold end-to-end diversity routing in its service access response to satisfy the demands of the CSR.

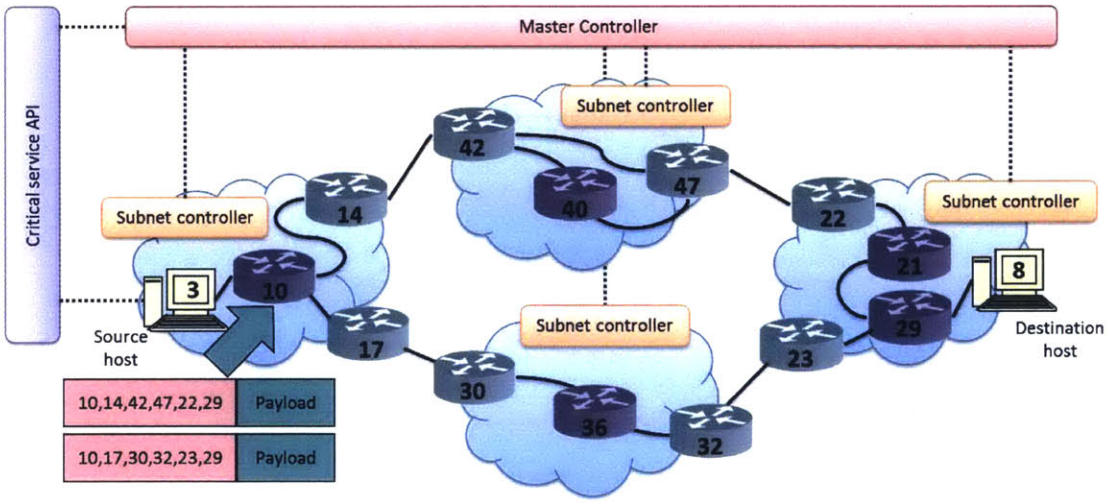


Fig. 2-28: Before transmission, the source host replicates the payload and prepends each of the MC-computed intersubnet routes to one of the datagrams. The source host then transmits each of the two CServ datagrams in series to the upstream access router prescribed by each particular path (in this example, the same upstream access router for both datagrams).

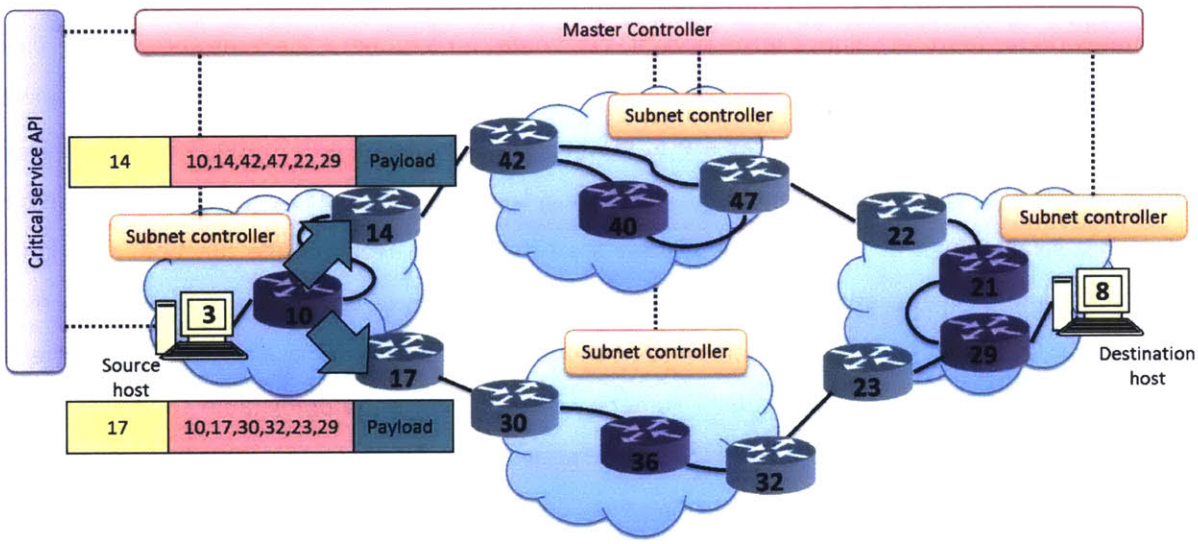


Fig. 2-29: Just as in Fig. 2-17, the access router uses the CServ Internetwork Service path next hop address in each CServ datagram and the subnet's routing policy to generate and prepend the appropriate CServ Intranetwork Service header for each datagram. The transmission of each datagram follows as previously described in Section 2.3.3.

2.4 Alternate Implementation of CServ Intranetwork Service

As previously stressed, the CServ architecture does not rely on IP as the subnet network transport for CServ Intranetwork Service. In fact, IP routing and forwarding represents what is likely the most basic intranetworking service since most current networks support this capability in the core. A subnet administrator would only need to upgrade the necessary gateway routers and access routers to support CServ service, while the CServ Intranetwork Service could run using native IP capabilities and legacy equipment in the rest of the subnet backbone.

Other network transport techniques serve as viable candidates for a subnet's CServ Intranetwork Service according to the network topology and the policy of the subnet operator. For example, we have previously discussed VC designs, such as MPLS, in Chapter 1. Rather than IP routing and forwarding, static or dynamically-signaled VCs could be used to form label-switched paths between pairs of active routers in a given subnet as long as these routers are capable of performing the functions of a label switch router and/or a label edge router. These paths would need to be established prior to the arrival of a particular CServ datagram rather than signaled and reserved on-demand, as the signaling overhead and label state management at the subnet routers would present an unpredictable source of delay which would not be accounted for in the CServ Internetwork Service path. The use of VCs allows for quasi-static traffic planning and engineering, directing CServ sessions as desired through the subnet core along pre-specified routes. When the internetwork CServ datagram arrives at an ingress gateway router in the subnet, the gateway would be responsible for determining the appropriate label-switched path based on its internetwork next hop and for prepending the necessary MPLS label information as the CServ Intranetwork Service header (these tasks satisfy the job description of a label edge router in addition to those of an active gateway router in the CServ architecture).

A subnet need not rely on one network transport to serve as its CServ Intranetwork Service. For example, a subnet administrator may choose to employ a hybrid IP/MPLS approach, where MPLS label-switched tunnels are established for subnet transit between gateway routers, but IP forwarding is used to direct CServ datagrams destined for a local access network and its upstream access router. This type of approach can reduce implementation cost and simplify the administrative burden of a traffic engineering technique like MPLS.

Additionally, we propose an even more powerful option for CServ Intranetwork Service. If a subnet has widely adopted the use of CServ-enabled routers in the network core, then the subnet administrator has the option of employing a form of centrally-computed explicit path routing such as performed by the MC at the global level. The CServ architecture already requires the subnet to implement a logically-centralized SC for the use of CServ performance metric state collection, summarization, and reporting. The existence of a controller entity with a thorough view of the subnet along with this detailed collection of state provides the opportunity to implement a version of the CSDCA for computation of CServ Intranetwork Service paths. Although we have yet to discuss the specifics of the algorithm (see Chapter 5), the ability to discover paths based on their CServ metric performance and compose diversity-routed service solutions opens a potent traffic engineering opportunity to the subnet provider.

While the objective of the CSDCA is to find internetwork paths and compose service solutions based on a specific CServ request, this algorithm can be adapted for the subnet to maintain intranetwork paths and diversity-routed solutions that meet administrator-defined requirements in terms of the CServ performance metrics, reliability and delay. Once discovered and composed at the SC, the CServ Intranetwork Service solutions can be distributed to the active subnet routers and stored as entries in their data plane forwarding tables. Consider the example of Fig. 2-30, which illustrates how the explicitly-routed entries could be stored at several routers in the network. The forwarding table lookup for the next hop along the CServ Internetwork Service path now yields an explicit intrasubnet hop-by-hop path or set of paths. Upon arrival of an internetwork CServ datagram at the ingress gateway router of a subnet (or an upstream access router in the source subnet), the router retrieves the explicit CServ Intranetwork Service path and prepends it to the internetwork CServ datagram as part of the CServ Intranetwork Service header, which looks much like the CServ Internetwork Service header. Importantly, this lookup may actually return a diversity-routed set of paths such as depicted in Fig. 2-31, which can be used to provide additional reliability and survivability during subnet transit. In this case, the ingress gateway router or upstream access router replicates the incoming CServ datagram and generates a separate CServ Intranetwork Service header for each copy before transmitting them along the specified routes. As with end-to-end internetwork diversity routing, repetitive copies of the same CServ datagram are quenched at the terminus of the CServ Intranetwork Service path (either at the specified egress gateway router or the destination host's upstream access router). Routers along a specified intranetwork path now need only to refer to the CServ Intranetwork Service path in the CServ datagram

control information to determine the next hop interface for the CServ datagram, circumventing any further forwarding state lookup.

The bottom line is that, using this alternative CServ Intranetwork Service implementation, the internetwork CServ datagram is encapsulated in an intranetwork CServ datagram that provides access to an explicitly-routed CServ Intranetwork Service tunnel (or set of tunnels), affording the network administrator much greater control over the subnet's performance. The caveat is that the subnet routers must be CServ-enabled and capable of processing explicitly-routed CServ Intranetwork Service headers and forwarding these CServ datagrams, much like how active gateway routers and access routers must be ready to process and interpret CServ Internetwork Service headers. In Chapter 3, we formalize the explicitly-routed CServ Intranetwork Service header and datagram format alongside the CServ Internetwork Service header and datagram format.

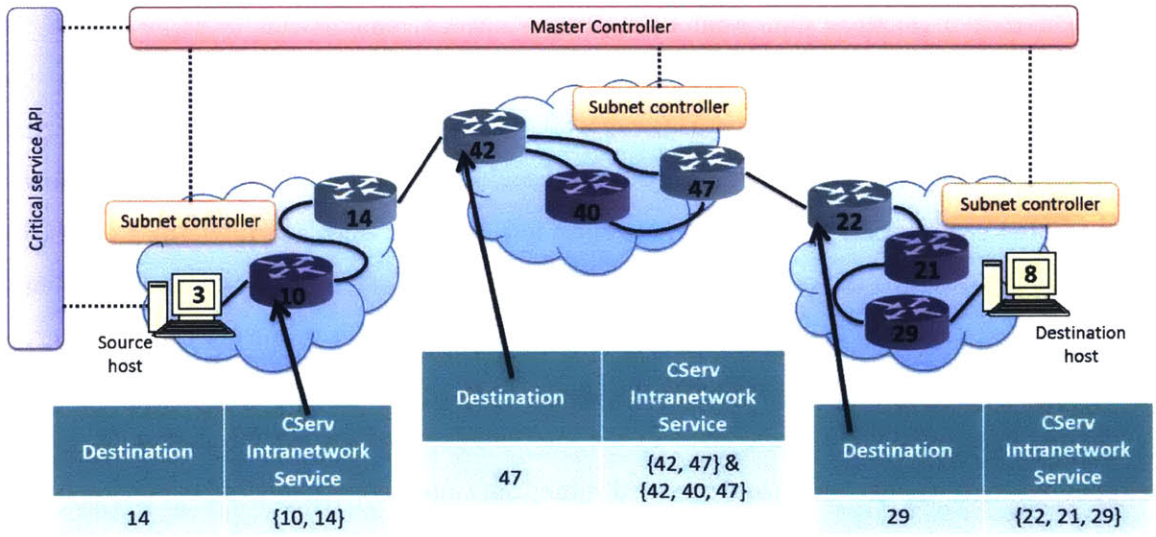


Fig. 2-30: Prior to the initiation of a CServ transaction, routers populate their forwarding tables with the appropriate state information consistent with the unified internal routing strategy of their subnet. In this example, the subnets use an explicitly-routed path solution where the full hop-by-hop path is stored in the forwarding tables. The explicit path is then prepended to the CServ datagram and used for forwarding much like the CServ Internetwork Service header.

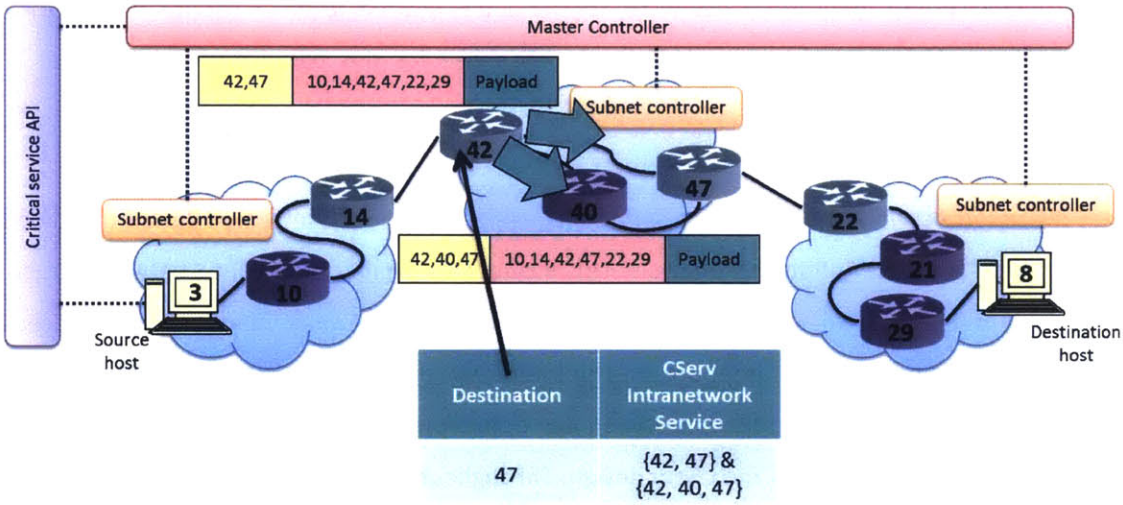


Fig. 2-31: When the CServ datagram arrives at the ingress gateway router, it prepends the full explicitly-routed path to the datagram as the CServ Intranetwork Service header. In this example, we illustrate that this provides the flexibility to use subnet-internal diversity routing to improve the reliability performance of the subnet. The ingress gateway router replicates the incoming CServ datagram, appending a different explicitly-routed path to each, and the egress gateway router (here, router 47) quenches the datagram replication if both copies arrive successfully.

2.5 Conclusion

In this chapter, we have introduced the CServ architecture and walked through an overview of its operation. We began with a description and discussion of the internetwork architecture objective, complete with definitions of the important terms used to specify this objective. The CServ design goals had been previously motivated by the example driving applications and use cases discussed in the beginning of the dissertation document. The use of probabilistic performance metrics, both for CServ datagram reliability and end-to-end delay, to stipulate the desired level of service was motivated through a simple analytic example that illustrated the need for this paradigm, showing that strict deterministic guarantees cannot be made in a data network.

We next discussed network models and assumptions used throughout the remainder of the thesis to unify the architecture description and analysis. This included treatment of the internetwork structure, the subnet unit and its internal composition, and the differentiation between CServ-enabled devices and legacy hardware. Although the assumptions on the internetwork and subnet structure do not need to hold strictly for CServ architecture implementation and operation, we noted that this set of conventions helps to streamline the coverage of the CServ design. Throughout the document, we highlight aspects of the architecture that do not rely on the network model assumption set where appropriate.

Finally, the operation of the CServ architecture was described using an example critical datagram transaction. This description included the pre-transaction state collection and maintenance process, the initiation of the transaction through the CServ API and the pre-transmission overhead, and the transmission of the internetwork CServ datagram from source host to destination host as it traverses multiple disparate subnets. Initially, this example is presented using legacy IP routing and forwarding techniques for the CServ Intranetwork Service in the constituent subnets along the internetwork path; later, with ubiquitous intrasubnet adoption of CServ-enabled hardware, we showed the opportunity to employ more sophisticated routing and forwarding techniques to implement the CServ Intranetwork Service which are more closely aligned with the CServ Internetwork Service routing and forwarding mechanisms. This discussion highlights the opportunity for flexible implementation and incremental deployment of the CServ hardware and software. As we present the CServ architecture components in more detail in the subsequent chapters, this is a theme that we frequently revisit. Subnet administrators are not required to immediately upgrade all control system and routing core hardware and software to

support all of the CServ architecture features. However, eventual comprehensive adoption of the new technology allows for the implementation of a more robust and reliable intranetwork service, which in turn is favorable for the discovery and composition of internetwork service.

We have introduced the primary components of the CServ architecture with this design overture. In the following chapters, we consider the critical features that enable the CServ objective in more detail. Specifically, we discuss the following:

- the CServ Intranetwork and Internetwork Service datagram format and processing (Chapter 3);
- the CServ performance metric learning, collection, and maintenance techniques (Chapter 4);
- the internetwork topology representation and manipulation at the global level (Chapter 5);
- and the MC Critical Service Discovery and Composition Algorithm operation (Chapter 5).

The final chapter of the dissertation considers some open problems, particularly those addressing the architecture security framework, and future work opportunities to further develop the CServ architecture.

Chapter 3

CServ Internetwork and Intranetwork Service Headers

This chapter discusses the header control information used to direct the Critical Service (CServ) datagrams from source host to destination host. Following the pre-transmission transaction setup procedure discussed in Chapter 2, the CServ datagram is transmitted on the data plane, and it is the responsibility of the header control information to guide the message to its destination along a path that is part of a service solution that meets the critical service metric requirements of the transaction. This end-to-end path from source host to destination host includes at least one intranetwork path, or one confined to a subnet, and often involves an internetwork path, or a subnet to subnet granularity path from the source subnet to the destination subnet. For the purpose of this chapter, we focus the discussion on an end-to-end path that involves an internetwork route.

As introduced in the overture of Chapter 2, the CServ internetwork architecture includes the use of both *CServ Internetwork Service* and *CServ Intranetwork Service*. The CServ Internetwork Service is an internetwork path or set of internetwork paths that satisfies the service demands of the CServ Request (CSR). This service solution is computed by the Critical Service Discovery and Composition Algorithm (CSDCA) at a logically-centralized *master controller* (MC) entity, the details of which are discussed further in Chapter 5. As part of the CSR solution, a subnet-to-subnet granularity path in the CServ Internetwork Service is called a *CServ Internetwork Service path*. The details of this path are carried in the *CServ Internetwork Service header* which is prepended to a CServ message payload to create the internetwork CServ datagram. Similarly, the CServ Intranetwork Service is defined as an intranetwork path or set of intranetwork paths that bears the internetwork CServ datagram from CServ-enabled router to CServ-enabled router with some measured and reported level of performance. Namely for the purposes of the internetwork CServ datagram, the CServ Intranetwork Service represents the intranetwork path or set of intranetwork paths between an access router and an egress gateway router,

an ingress gateway router and an egress gateway router, or an ingress gateway router and an access router that serve to carry the internetwork CServ datagram between the designated CServ-enabled in its explicit CServ Internetwork Service path. As part of the CServ Intranetwork Service, a router-to-router granularity path in the service is called a *CServ Intranetwork Service path* (note that these routers are not necessary CServ-enabled active routers). The header control information prepended to an internetwork CServ datagram and used to bind the CServ datagram to a path in this service is called the *CServ Intranetwork Service header*, however we mention now that this header does not necessarily include the explicit details of the path (unlike the CServ Internetwork Service header).

Although we do not finalize all of the details of the CServ datagram header control information in this chapter, this coverage concentrates on describing the type of information necessary for successful data plane operation rather than the specifics. The decision on some of the protocol details are outside the scope of this dissertation, as highlighted during this chapter. However, we consider some alternative implementation options and underscore the differences between them where applicable. Furthermore, this level of discussion allows for the sizing of the header control information and the CServ datagram which is utilized for analysis in the remainder of the thesis coverage.

The outline of the chapter is as follows. In Section 3.1, we present some initial considerations that are used in the creation of the CServ datagram headers. Although the details of these aspects are beyond the scope of the document, this discussion presents some basic principles and deliberations by which the header control information fields are sized for the purpose of later analysis. Section 3.2 presents the structure and describes the fields of the CServ Internetwork Service header. Then in Section 3.4, the CServ Intranetwork Service header is addressed. This conversation is less structured than that of Section 3.2 since the details of the CServ Intranetwork Service header depends on the routing and forwarding strategy chosen by a specific subnet administrator to implement its CServ Intranetwork Service. Rather than focus on any given possibility, we present a few non-exhaustive options and consider the corresponding CServ Intranetwork Service header. Finally, we conclude the chapter and discussion of data plane control fields in Section 3.6.

3.1 Preliminary Discussions on Header Control Information

The formulation of the CServ header control information is presented in this chapter as a preliminary design. Fundamental fields for CServ service are presented, but the details of their implementation (including the number of bits allotted to each) may later be reconsidered. Other less fundamental fields and control information are also described, along with the rationale for their inclusion, but the implementation details of these control fields are only presented at an overview level. Further research is required to fully understand the requirements of this set of control information. That being said, we generally employ an approach of overprovisioning for the different header control fields in this chapter in order to facilitate worst-case analyses later in the thesis document.

In this section, we consider three topics that ultimately need more future research to define their requirements in the context of the CServ architecture:

1. addressing;
2. datagram integrity and authentication;
3. and signaling between protocol layers.

In spite of the incomplete design specification, we contemplate possible implementations for each and justify the initial sizing for their respective control fields in the CServ service headers.

3.1.1 Addressing

The addressing scheme, or identification structure for network subnets, routers, and host interfaces, used in the CServ architecture is not yet specified definitively. We begin this discussion by presenting the desired properties of a candidate addressing framework, and then we proceed to give a suggestion for its implementation and the associated rationale. That being said, our suggestion in this section is merely one possibility, as we conclude this section by highlighting a drawback of the suggested scheme that merits future attention.

We identify three primary desired properties of the addressing framework used in the CServ architecture:

1. a hierarchical structure;
2. sufficient addressing space;
3. and interoperability with addressing used for best effort datagrams.

In Section 2.2 of Chapter 2, we introduce a natural hierarchy between the primary components of the network model: the endpoint hosts, the routers, and the subnet. Endpoint hosts can be considered the lowest member of the hierarchy. Routers (specifically access routers) provide upstream access for endpoint host nodes into the core of the subnet network, and thus routers can be considered the next level of the hierarchy. Finally, a subnet consists of a collection of cooperating routers that form its core network – the top level of the hierarchy. An ideal addressing framework would provide the flexibility to represent this natural hierarchy within the addresses themselves. For example, knowledge of a router's network address would indicate its subnet membership, and knowledge of a terminal host's network address would similarly indicate both its provider subnet membership and the address of its upstream access router.

Address exhaustion occurs when the unallocated pool of network address space has depleted, and it is a well-known problem in the context of the Internet and the Internet Protocol Version 4 (IPv4) addressing space. IPv4 uses 32-bit addresses, a field length which allows for $2^{32} \approx 4.3$ billion unique addresses [29]. The top-level exhaustion of IPv4 space occurred on January 31, 2011 [53], and the exhaustion of IPv4 address space allocated to the North American regional Internet registry occurred on September 24, 2015 [54]. The proliferation of Internet-connected devices, each of which may have multiple IPv4 addresses corresponding to different network interfaces, and the inefficient allocation of address space has led to the exhaustion of the available pool, even with architectural modifications to delay the inevitable (such as the transition from classful network addressing to Classless Inter-Domain Routing and the introduction of network address translation). The long-term mitigation technique for this problem has been the gradual transition to Internet Protocol Version 6 (IPv6) [55], which employs 128-bit addresses rather than 32-bit addresses, allowing for $2^{128} \approx 3.4 \times 10^{38}$ unique addresses. An appropriate choice for the CServ architecture addressing framework should avoid the address exhaustion issue to a reasonable extent.

A third desirable property for a candidate addressing framework is interoperability with the addressing scheme used for best effort traffic in the CServ architecture. As a CServ-enabled endpoint host may both

generate CServ datagrams or best effort traffic, it befits the design to choose an addressing framework that generalizes to both services rather than to differentiate between addresses or addressing schemes based on the intended network layer service. The latter introduces an additional level of unnecessary complexity. Furthermore, and although outside the scope of this document, this interoperable framework enables customers of the CServ class and the best effort class to leverage common support infrastructure, such as address resolution services like the Domain Name System [56] [57].

As the Internet addressing framework is gradually shifting from IPv4 to IPv6 in response to the exhaustion of IPv4 address space, we identify IPv6 as a promising candidate for the addressing scheme in the CServ architecture. We present a brief overview of the IPv6 address structure, and then we consider its viability in terms of the three identified desired properties.

IPv6 uses 128-bit addresses that are intended to permit hierarchical address allocation to facilitate efficient route aggregation. Generally, these addresses are represented as eight groups of four hexadecimal digits (or 16 bits) separated by colons, such as illustrated in Fig. 3-1. The specification introduces several address classifications based on the routing methodology, such as anycast and multicast, but we focus on the unicast address format here since the critical datagram service is design for unicast message transmission. As depicted in Fig. 3-1, the most significant 64 bits are reserved for the network prefix used for routing, while the least significant 64 bits are used as an interface identifier to pinpoint the host's network interface. The network prefix is further decomposed into routing prefix and subnet identification fields, which are variable in length but typically 48 bits and 16 bits respectively. The routing prefix, a /48 (or sometimes as little as a /56 [58]), is generally distributed by local Internet registries to end sites, such as companies and organizations, and serves as the network address. The subnet identification field provides the end site the flexibility to configure a hierarchy of smaller networks within that space (note the different use of the term "subnet" in the description of the IPv6 architecture) [49]. The standardized 64-bit field for the host interface identification allows for automatic derivation from its unique 48-bit media access control (MAC) address to form an EUI-64 [59], although alternate mechanisms to obtain the identifier through an address server, random choice, or manual configuration are available.

As considered above, the flexibility of the 128-bit IPv6 address is intended to promote the creation of a hierarchical routing space. This suits our first desired property. Second, the approximately 3.4×10^{38}

possible unique addresses provide more than sufficient addressing space to avoid near-term address exhaustion, meeting our second desired property. Third, the adoption of IPv6 in the Internet for best effort Internet Protocol (IP) service makes this addressing framework very attractive for interoperability purposes. The CServ architecture can make use of IPv6 for the best effort traffic class, while simultaneously leveraging the IPv6 addressing scheme to identify subnets, routers, and hosts for the CServ traffic class. Furthermore, the prescribed procedure to generate interface identifiers from unique MAC hardware addresses ensures uniqueness among addresses used for both CServ-enabled hosts and those without CServ capability.

We suggest the use of the 128-bit IPv6 address as shown in Fig. 3-1, reallocating some of the most significant bits in the network prefix to suit our network hierarchy. Adopting the terminology of the CServ architecture, we allot the most significant 32 bits for the Subnet Identifier, where we now use the "subnet" as introduced in Chapter 2. Although subnets in our architecture do not necessarily correspond one-to-one with Internet Autonomous Systems (ASes), we posit that the magnitude of their respective sets could be more or less comparable. As of November 2015, the number of unique ASes in the Internet routing system is approximately 52,180 [60]. AS Numbers (ASNs) were originally 16-bit integers, but the standard was later updated to 32-bit integers to account for growth in the number of ASes [61]. This 32-bit field allows for more than 4 billion unique ASes, and should easily suit the requirements for subnet addressing given that there are less than 53,000 ASes in the Internet routing system. The size agreement between Subnet Identifier and ASN also enables the opportunity to leverage Border Gateway Protocol to serve the subnet internetworking connectivity needs of the best effort traffic class. The next 32 bits in the address are reserved for router identification within the subnet, again providing address space for more than 4 billion routers per subnet routing core (a unique address can be assigned to each router interface if required for implementation, as there is abundant address space). The least significant 64 bits are left unmodified, as we allow for the use of the same techniques introduced by the IPv6 addressing framework for automatic host interface ID generation.

In the following discussion, we employ some standard representations of IPv6 addresses as text as outlined in [62]. Namely, we suppress leading zeros in each 16-bit run, and multiple consecutive 16-bit runs of all zeros are replaced by an empty group using two colons (i.e. "::").

The address structure can then be used in the following way. A subnet address has the form of 2001:db8::, for example, where the first 32 bits specify the subnet and the remaining 96 bits are all zeros. The address of a router should reflect its membership within the core routing plane of a subnet by using the same most significant 32 bits as the subnet. For example, an access or gateway router in subnet 2001:db8:: could have the address 2001:db8:0:1::, where the 32 bits following the subnet identification specify the router and the least significant 64 bits are all zeros. Lastly, the address of a host interface then reflects both the upstream access router which provides network access into the subnet routing core and the subnet membership of that access router. As an example, a host interface connected to access router 2001:db8:0:1:: could have the address 2001:db8:0:1::1, where the least significant 64 bits are generated from the interface's hardware MAC address. This use of the addressing framework immediately indicates information about other addresses in the hierarchy. For example, given a host's interface address such as 2001:db8:0:1::1, application of a bitwise AND operation with the router bitmask (ffff:ffff:ffff:ffff::) or the subnet bitmask (ffff:ffff::) directly reveals the addresses of the host interface's upstream access router and the host interface's connected subnet, respectively. Similarly, application of a bitwise AND operation with the subnet bitmask on a router address, such as 2001:db8:0:1::, reveals the subnet membership of that router device, whereas application of a bitwise AND operation with the router bitmask on a router address can be viewed as a trivial identity operation.

Although not necessarily unique to this addressing framework, we note that an endpoint host with multiple network interfaces has multiple network addresses since each interface has a unique hardware address. The source of a CServ transaction likely wants the critical message to reach the intended destination host with multiple network interfaces regardless of which interface the message is received on. Thus, even if one address is used to specify the destination of the transaction, the CServ architecture should have the capability to resolve that address into all possible interface addresses for the same endpoint host. The primary benefits of multiple network interfaces are availability and survivability, particularly if the multiple network interfaces are used for multihoming. Consider the two multihoming examples in Fig. 3-2 and Fig. 3-3. In the first example, the host device is redundantly connected to two upstream access routers in the same subnet via its multiple network interfaces. The multiple connections render the host still available even if one of the interfaces goes down or if one of the upstream access routers fails, a highly desirable property if this host is either the source or intended destination of a CServ transaction. Alternatively, in the second example, the host device is connected to upstream routers in two different subnets via its multiple network interfaces. In this scenario, the host

additionally remains reachable by one subnet even if the other subnet completely fails or becomes compromised, an even more resilient use of the dual interfaces. In both examples, the interface addresses would be different given our proposed scheme even if we unified the least significant 64 bits, or the interface identifier, in the address, since the upstream access router (and possibly connected subnet) addresses are unique. The use of this addressing framework requires the ability to bind multiple unique addresses as belonging to the same host device, and it is possible that there are addressing frameworks that handle the multiple interface and multihoming issues more efficiently.

Before we continue to the next section, we reiterate that the use of IPv6 is only one of many alternatives for an addressing implementation in the CServ architecture. For the reasons previously identified, we believe that it is a promising candidate, particularly because of its acceptance in the context of the Internet's best effort service. Other addressing frameworks not yet explored, however, may exhibit the same desirable properties and even more effectively handle some of the identified challenges. The majority of the discussion in the rest of this document does not rely on the specifics of IPv6, but rather only on the capability to use the addressing structure in a hierarchical manner. That being said, we adopt 128-bit address fields when analyzing the size of header control information because of the broad addressing space it provides and the flexibility available with that address length.

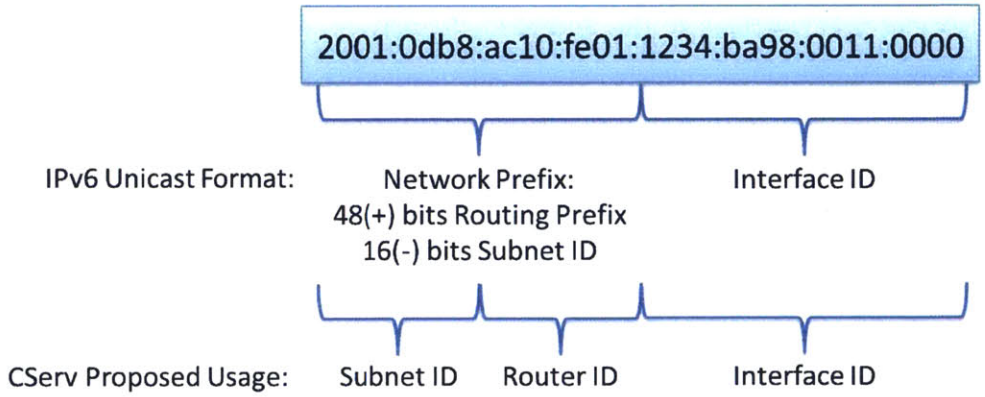


Fig. 3-1: This shows the detailed structure of a 128-bit IPv6 unicast address in hexadecimal notation and a proposed restructuring for possible use in the CServ architecture.

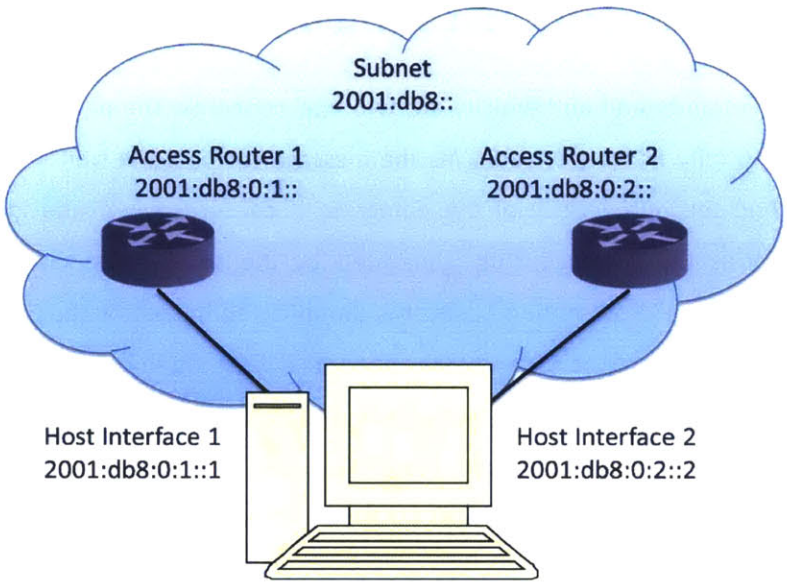


Fig. 3-2: In this multihoming example, the host is connected to two upstream access routers in the same subnet via its multiple network interfaces.

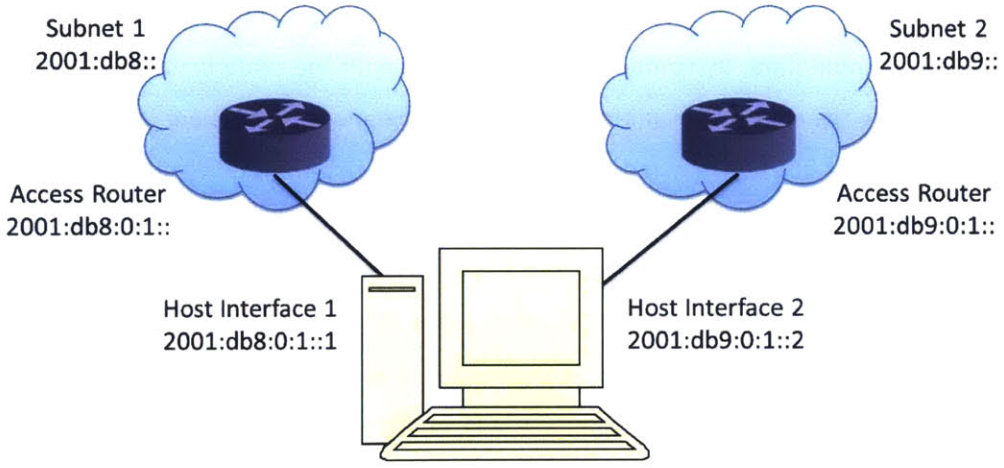


Fig. 3-3: In this multihoming example, the host is connected to two upstream access routers in two disparate subnets via its multiple network interfaces.

3.1.2 Datagram Integrity and Authentication

The CServ datagram integrity and authenticity are two vital concerns. The *integrity* of the datagram is concerned with the validity of the payload. Was the message corrupted in transit from source host to destination host? And the *authenticity* of the datagram is concerned with the identification of the originating source. Was the message truly generated by the source host claimed in the control information of the datagram? Both of these concerns should be addressed by the CServ architecture, as they are central to the success of the CServ messaging service designed to bear the most important data in the network. A CServ datagram that arrives at the destination host with a corrupted message or, worse, that has been forged by an adversarial third party is either useless or harmful to the recipient. In this section, we very briefly consider these topics for the purpose of sizing the control information fields in the CServ datagram headers. However, a more thorough treatment of these considerations is necessary. In particular, the security of a system designed to handle mission-critical messaging is of utmost concern; we deliberate upon the many outstanding security questions in the future work section of Chapter 6.

First, we consider how we can protect the integrity of the CServ datagram. A typical approach to this problem in data transmission is the inclusion of a checksum in the datagram header, or a small-size hash of the data. Checksum functions are typically designed to output a significantly different value even for a slight change in the input data. By computing the checksum with the same function at the destination and comparing with the value transmitted with the datagram, the recipient can verify with high probability that the data has not changed or been corrupted during transit. We note here that the next generation of Internet network service, IPv6, does not compute or include a checksum in the header control information [55] because link level protocols are assumed to almost always provider a stronger data integrity check (such as a cyclic redundancy check) and the higher layer reliable transport protocol, Transmission Control Protocol (TCP), provides end-to-end checksum protection [63]. As it has been observed that the TCP checksum is not redundant [64], we choose to reintroduce an integrity check in the CServ Internetwork Service header because the CServ messaging architecture does not assume nor require the use of a higher-layer transport protocol, and thus the CServ Internetworking Service serves as an end-to-end protocol. We assume the use of a simple 16-bit checksum, as in TCP, and allow for this in the allocation of the header control fields.

We additionally consider allocating control fields for the use of verifying the authenticity of the CServ datagram, a concern at the crux of a network service that bears the most important messages including early warning notifications and command and control messages. A forged CServ datagram could have severe consequences. Although we do not give the full treatment of the consideration here, we introduce some of the issues in the decision upon one authenticity protocol or another.

As a typical approach, cryptographically-generated codes are often attached to network messages to give reason to believe the message was created by the specified sender. These authenticity measures are at the center of many security-centric Internet protocols, including IPSec and Transport Layer Security. Broadly speaking, cryptographic techniques can be decomposed into symmetric schemes and asymmetric schemes. In a symmetric scheme, the two communication endpoints have a shared secret key that is used to generate and validate the transmitted authentication code, called a message authentication code. These techniques require that the two endpoints have established and agreed upon a shared secret in advance of initiating the authenticated communication. In an asymmetric scheme, a private key is used to generate an authentication code which is transmitted with the message, called a digital signature in this context. This signature can be verified, but not forged, using an openly announced public key in the key pair. This technique requires that the recipient has the message source's public key prior to initiating the authenticated communication.

The security level of these authentication schemes is frequently characterized by *bits of security*, where a scheme with n bits of security requires $O(2^n)$ operations on average for an adversary to compromise the secret key. Symmetric schemes, such as those that generate keyed-hash message authentication codes, are designed to achieve a level of security equal to their key length. Asymmetric schemes do not meet this property; they require longer keys (and often longer signatures) to attain the same bits of security. For example, a popular asymmetric public-key signature scheme uses the RSA cryptosystem. The level of security with a 1024-bit key, and consequently 1024-bit digital signature, in RSA is approximately equal to an 80-bit symmetric key algorithm [65]. Alternatively, the Digital Signature Algorithm (DSA), a variant of the ElGamal algorithm [66], produces a 320-bit signature with a 1024-bit key and 80 bits of security. An elliptic curve version of DSA (ECDSA) allows for shorter key sizes for a given level of security, although the signature length remains unchanged. Generally, elliptic curve-based security has a level of security roughly half its key length. For 80 bits of security, ECDSA only needs a 160-bit public key, but the resulting signature remains 320 bits in length. Using either DSA or ECDSA, 112

bits of security can be realized with 448-bit signatures, and 128 bits of security can be realized with 512-bit signatures [67]. Generally, 128 bits of security is widely considered to be out of reach for conventional digital computing techniques in the foreseeable future.

The question of how to size the header control field allocated for authentication depends jointly on the cryptographic authentication scheme (symmetric or asymmetric), the authentication code generation algorithm, and the level of security desired. We consider it infeasible for all possible pairs of CServ communication endpoints to exchange secret keys. For m total CServ-enabled devices, this would require $O(m^2)$ secret key pairs to be prearranged in a distributed fashion prior to the transmission of CServ datagrams. And this distribution of secret keys would require some method that does not utilize the network itself, as this would negate the "secret" aspect. This consideration rules out symmetric signature schemes. Focusing then on asymmetric schemes which require the distribution of m public keys in the open, the choice boils down to the overhead we are willing to accept in the CServ datagram headers and the level of security needed. This decision on a cryptographic signing scheme remains an open problem in the CServ architecture and requires more detailed consideration. Suffice it to say, we choose to reserve 512 bits in the CServ datagram header for an authenticating signature, and we find that this allocation already imposes a heavy burden in terms of overhead. Standards recommendations require at least 112 bits of security [68]. This excludes the use of RSA, which would need 2048-bit signatures [65] and much more space in the CServ datagram header. However, 112 bits of security can be achieved using DSA or ECDSA with only 448-bit signatures, or we can leverage all 512 available bits in the authentication field and realize 128 bits of security with one of these schemes. The complexity of signature generation and verification should additionally be considered in the future when determining which scheme best suits the CServ architecture, as well as any other candidate schemes that are not discussed here.

3.1.3 Signaling Between Protocol Layers

An important consideration in packet encapsulation and processing is that of indicating the protocol type of the encapsulated payload. This is necessary to indicate to the router or other network device how to handle the internal datagram or packet once the outer protocol information is stripped away. Otherwise, the router does not know how to process or interpret the information in the header of the next protocol layer. For example, if the next protocol layer is an IP datagram, then the router needs to

direct the datagram into the IP forwarding processing pipeline after stripping away the data link layer information. Alternatively, if there is a Multiprotocol Label Switching (MPLS) label present, then the router needs to direct the datagram into the label lookup processing module. Most protocols reserve fields in their control headers specifically to embed a code representing the next layer protocol type.

In the context of the CServ architecture, there are two main considerations:

1. How does the data link layer protocol signal that the encapsulated datagram is a CServ datagram?
2. And does the CServ datagram need the capability to signal the type of a higher layer protocol?

Although we do not specify all possibilities or standardize any set of type codes, we discuss some of the options that could be used to address these considerations in order to choose appropriate control field sizes in the CServ headers.

We begin by addressing the first question above. First, let us consider how this is done in the Internet in the context of the IP stack. The lower layer protocol indicates the encapsulation of an IP packet through a field in its control header. As discussed earlier in this chapter, there are multiple flavors of IP in the wild, namely IPv4 and IPv6. Although the IP header itself has a field that encodes the particular version used, this differentiation is typically made in the encapsulation type field by the lower layer protocol as well. As an example, let's consider the Ethernet frame [45], a data link layer protocol that can encapsulate an IP packet. The header of the frame includes a two octet field called EtherType, which can be used to indicate the protocol type of the payload. Specifically, the hexadecimal code 0x0800 is used for IPv4 and 0x86dd for IPv6. As another example, the EtherType code 0x8847 represents that the encapsulated datagram has an MPLS unicast label. Other data link layer protocols have similar methods of identifying their encapsulated payload type.

The CServ architecture could take this approach. In fact, this could be considered the "correct" approach in that it conforms to the standard method of introducing new protocol formats. The challenge here is that all possible data link layers in the network need the flexibility to encode several types of encapsulated CServ datagrams. Consider the generic decomposition of a CServ datagram in Fig. 3-4. As introduced in Chapter 2, the CServ Intranetwork Service header may in fact be an IP header, an MPLS

label, a CServ proprietary header, or another format. Additionally, the CServ Internetwork Service header, which is always CServ proprietary format, may appear alone at the network edge without a CServ Intranetwork Service header. This wide range of encapsulation possibilities makes it difficult to guarantee that the lower layer protocol has enough freedom to express this variety, even if it can be assumed that it is easy enough to reserve these type codes for CServ purposes.

Alternatively, we propose a second approach that may be better suited to the variety of possible datagram formats and should ease adoption of the CServ architecture, even if it represents a sort of system hack. Since IP packet formats are widely recognized by network devices and the 4-bit Version header field is only sparingly assigned, we suggest borrowing the Version field from IP and taking advantage of the sparsely-populated code to indicate CServ datagram types. Consider that codes for IPv4 and IPv6 are the only two widely used in IP (more precisely, codes for decimal values 1-3 and 10-14 are unallocated). With this approach, CServ headers, including the proprietary Intranetwork Service header, the proprietary Internetwork Service header, and the optional State Measurement Service headers (to be discussed in more detail in Chapter 4), lead with a 4-bit field in the most significant location mimicking an IP header and specifying the type of CServ header. Data link layer protocols can then encapsulate any datagram that leads with a CServ header using an IP protocol code, which is already assumed to be supported by link layer technologies such as Ethernet. The challenge with this approach is to ensure that CServ-enabled routers are configured to process the 4-bit Version field correctly and interpret CServ datagram types, distinguishing them from actual IP datagrams.

Additionally, IPv4 and IPv6 both have a one octet field in the header to indicate the next header type (or protocol type). This is frequently used to indicate the encapsulated higher layer protocol, such as TCP or User Datagram Protocol (UDP). As of the time of publication, there are 110 unassigned codes for this field in the decimal range of 143-252. We take this approach and suggest the inclusion of a 4-bit Next Header code in all CServ headers. The architecture currently does not need the flexibility of a full octet-length field, and this should suffice to indicate the type of the next header, be it a CServ datagram payload (no additional header), a CServ Internetwork Service header, or a state measurement header. This Next Header code enables us to build arbitrary header stacks in the form of Fig. 3-4 where each header segment can indicate its own format and the header format of the encapsulated datagram type. Note that there may be some implementation-specific exceptions to this design, since we do not have design control over some CServ Intranetwork Service header formats (such as IP headers or MPLS

labels). While IP headers have unused codes in the Next Header/Protocol field that we can leverage, a format such as an MPLS label does not. Consider an example where MPLS is used as the CServ Intranetwork Service and thus the CServ Intranetwork Service header is an MPLS label. These labels do not have fields available to indicate the next header protocol in their control overhead. However, since MPLS supports the transport of IP packets, this approach fits naturally with next headers that are *mimicking* IP headers with the leading Version field.

We now briefly address the second question posed above. As previously mentioned, IP provides support to specify the higher layer endpoint transport protocol that it encapsulates, such as TCP or UDP. The CServ data plane protocol, as described in this dissertation, represents both the network protocol and the transport protocol. Since our data transaction of interest is a short, critical payload with stringent time deadline requirements that may be on the order of the one-way transmission time, we do not necessarily have the luxury to implement a higher layer transport protocol to ensure end-to-end delivery reliability through retransmission protocols. That being said, there is no technical barrier to developing a transport protocol that treats CServ purely as a network service. The specification of reliability and delay requirements through the CServ API presents the opportunity to design unique transport protocols that leverage these CServ performance metrics for more efficient retransmission decisions. Further consideration of this possible future development is out of the scope of this work, but we note that any unused codes in the 4-bit Next Header field could be used to specify the encapsulated higher layer transport protocol if it exists (otherwise, the CServ Internetwork Service header always encapsulates the CServ message payload). If greater flexibility is demanded in the future, this control header field could be expanded to an 8-bit length since the current four bits need to be jointly shared with the codes for encapsulated CServ network service header layers.

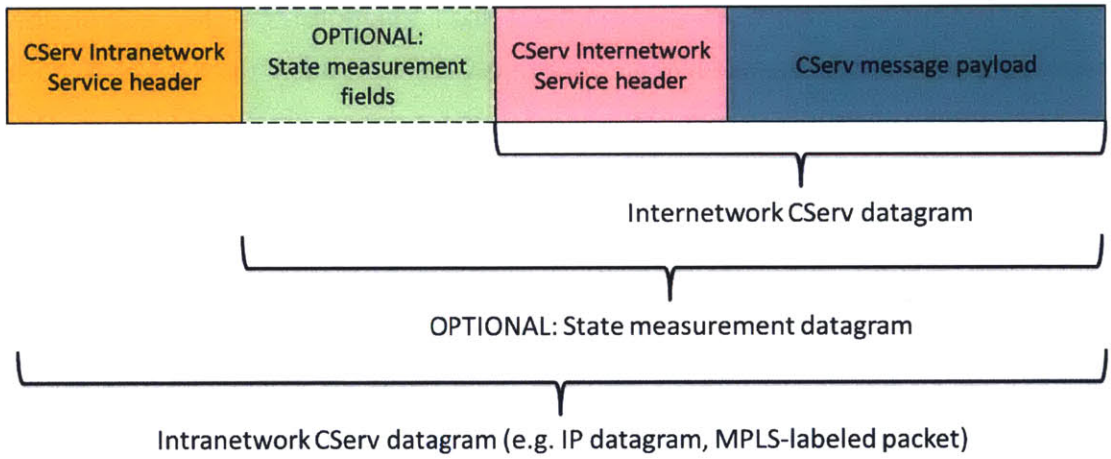


Fig. 3-4: This is the general format for the encapsulation of a CServ payload, including the placement of optional State Measurement Service fields that depend on the CServ performance metrics state learning protocol used. Without the optional fields, the CServ Intranetwork Service Header directly encapsulates the Internetwork CServ datagram.

3.2 CServ Internetwork Service Header

With the preliminary considerations of Section 3.1 under our belt, we proceed to discuss the structure of the CServ Internetwork Service header, describing the use of the control fields. This is followed by an analysis of the overhead imposed by this header layer.

The organization of the CServ Internetwork Service header is depicted in Fig. 3-5. This format is used as the first CServ header layer for all CServ datagrams; there is only one type of CServ Internetwork Service header. That being said, this is a variable length header, as we describe as we walk through the descriptions of the illustrated fields. As more functionality is considered for the CServ architecture, the design of this header control information may need to be modified, as we mentioned in the chapter introduction. However, we consider that this is the fundamental information required for successful CServ data plane operation as outlined in this document.

The following describes the purpose and use for each field in the CServ Internetwork Service header:

- **Version (VER) –** As discussed in Section 3.1.3, this leading 4-bit Version field allows the internetwork CServ datagram to masquerade as an IP packet, which also leads with a 4-bit version code. We employ an unused code (codes for decimal values 1-3 and 10-14 are unallocated) to specify that the following is a CServ Internetwork Service header. In this way, lower layer data link protocols can encapsulate the internetwork CServ datagram as an IP packet with the appropriate signaling in its header control information. The only layer-3 routers that should encounter an internetwork CServ datagram are CServ-enabled active devices by design, and they should be capable of interpreting the Version field code to distinguish the internetwork CServ datagram from an IP packet.

- **Next Header (NH) –** The 4-bit Next Header field is used to indicate the format of the encapsulated datagram header, if indeed there is one, as introduced in Section 3.1.3. For the time being, we assume that there is no encapsulated header for internetwork CServ datagrams and the default Next Header code (say, for example, 0000) is a sort of null value that indicates the encapsulated payload is the CServ message data to be delivered to the destination host's CServ application. However, as discussed before, this field can later be used to indicate the

existence of a transport protocol control header if one is developed that supports CServ network service. Furthermore, all CServ header formats begin with the 4-bit Version and 4-bit Next Header fields, and the latter is needed for other CServ header formats (to be discussed in Section 3.4).

- **Path Length (PL) –** The Path Length field is used to indicate the number of CServ routers in the explicit CServ Internetwork Service path, the details of which are included later in the CServ Internetwork Service header. The decimal value encoded in this field corresponds to the number of addresses in the explicit path. For example, the decimal value of the PL field would be 8 for a path with two transit subnets since the full path includes the source and destination subnets and two CServ routers per subnet. Although most paths are only expected to traverse a handful of subnets, four bits would allow for only a maximum of 15 routers in the CServ Internetwork Service path. We choose eight bits for this field to allow for additional path length flexibility (up to 255 routers in the CServ Internetwork Service path), but we do not expect the full flexibility of this field to be often used. In fact, the MC's service discovery and composition algorithm may limit the length of candidate CServ internetwork paths. Indirectly, this field also specifies the total size of the CServ Internetwork Service header since it controls the length of the variable part of the header as shown in Fig. 3-5, and thus delineates the boundary between the CServ Internetwork Service header and the CServ data payload (or, possibly, the next header if there is support for one in the future).

- **Payload/Data Length (DL) –** This field is used to encode the size of the encapsulated payload in bytes. As mentioned above, the payload is currently the bare CServ message data (as indicated by the default Next Header code). If support for a higher layer transport protocol is developed in the future, this value would specify the size of the encapsulated protocol data unit payload (both the transport control information header and the CServ message data). With 16 bits, this field can encode payloads up to 65,535 bytes. Our driving applications are expected to transmit CServ message data on the order of one kilobyte, making this field sufficiently expressive for these purposes.

- **Checksum (CS) –** As motivated in Section 3.1.2, we include a 16-bit Checksum field to store the value of simple hash of the internetwork CServ datagram to ensure its integrity as it traverses

the network. This integrity protection takes the place of similar checks in Internet transport protocols (such as TCP), since the CServ network service plays the dual role of the network and transport layers in the CServ architecture.

- **Source Address (SA)/Destination Address (DA) –** These self-explanatory fields are used to store the network addresses of the source and destination hosts for the unicast CServ transaction, the same addresses specified in the CSR to request CServ network service. Per the discussion of 3.1.1, we allocate 128 bits for each network address.

- **Sequence Number (SN) –** We use a sequence number to identify unique CServ message data payloads between a particular source and destination host. Since the CServ Internetwork Service may specify the use of subnet-disjoint path diversity to meet the demands of the CSR, the sequence number is required to alert the destination host to multiple receptions of the same CServ message data such that the two messages are not treated as unique transactions. In other terms, the {SA, DA, SN}-tuple is used to quench CServ message replication for diversity-routed solutions at the internetwork endpoint. This does require stateful CServ processes at the endpoints of the CServ transaction to maintain a current sequence number for the {SA, DA} pair. Although it likely represents an overly conservative allocation, we mark off 32 bits for the Sequence Number field, which allows for more than four billion CServ transactions between a pair of endpoint hosts before the value wraps around. It is very feasible that fewer bits are actually required for this field.

- **Expiration Time (ET) –** This 64-bit field can be considered a "belt-and-suspenders" check to ensure that the destination host receives the CServ message data within the maximum tolerable delay specified by the source host CServ application. This value, the current time plus the maximum tolerable delay, is calculated from the moment that the source host application generates the CServ message and CSR. By placing this timestamp in the CServ Internetwork Service header, the destination host can verify that the internetwork CServ datagram has been received by the intended deadline by checking that the current time is later than the value in the Expiration Time field. This may be useful for the CServ application, which can implement an appropriate policy for CServ datagrams that are received by the destination host application after the deadline. The 64-bit timestamp conforms to many standard system time

representations, such as the nanosecond-accurate real-time clock supported by Linux kernels which uses 64 bits to specify the number of nanoseconds since the Unix Epoch. With 64 bits, it would take more than 584 years for this timer to wrap around. Note that the CServ architecture assumes access to a synchronized global clock, such as that available through Global Positioning System (GPS) timing.

- **Authentication Mark (AM)** – Per the discussion in Section 3.1.2, we allocate a 512-bit field for CServ datagram authentication. Although we do not formalize the cryptographic algorithm used to generate digital signatures in the CServ architecture at this time, the consideration earlier in the chapter describes some of the options and tradeoffs.

- **Hop $x \in \{1, 2, \ldots, h\}$ Address** – The final part of the CServ Internetwork Service header is the explicit representation of the CServ Internetwork Service path determined by the MC and the CSDCA. This path consists of h 128-bit network addresses, where h is the decimal integer specified by the Path Length field. Each of these addresses corresponds to a CServ-enabled router, either an access router or a gateway router in the CServ nomenclature, in the path from the source host to the destination host, and it represents the explicit subnet-granularity path that the internetwork CServ datagram follows through the network.

It is possible that additional fields may be necessary in the CServ Internetwork Service header as additional functionality is developed for the CServ architecture. However, this formulation gives us a foundation in order to analyze the protocol overhead.

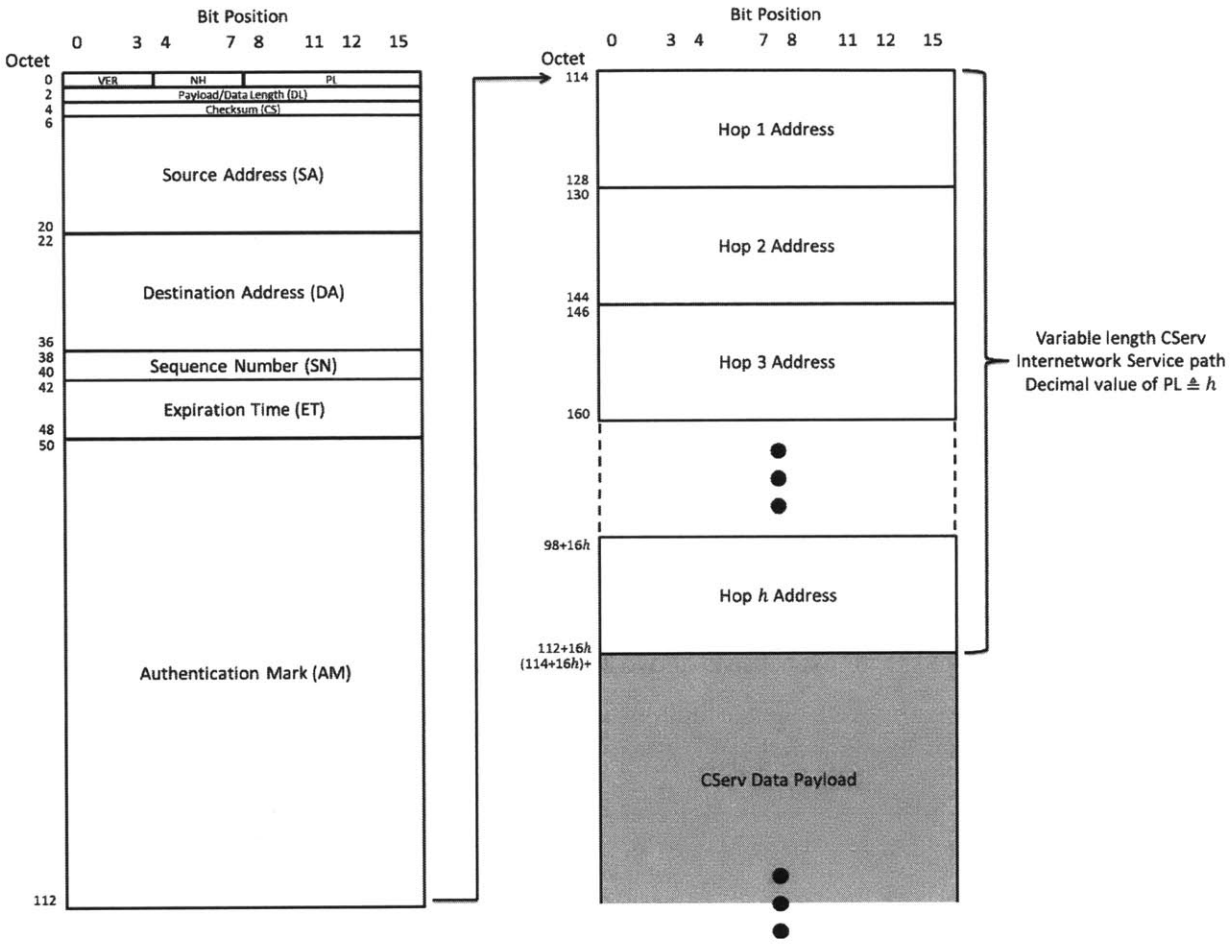


Fig. 3-5: The variable length format of the CServ Internetwork Service header is depicted here, to scale, based on the sizing discussions in Sections 3.1 and 3.2. The realized header length depends on the value stored in the Path Length (PL) field.

3.2.1 Size Analysis of the CServ Internetwork Service Header

We consider the control overhead imposed by the CServ Internetwork Service header as described in this section. In particular, we want to analyze the range of possible header sizes based on the header format and system parameters. For the purpose of analysis and computation of the overhead efficiency, we denote the size of the CServ message data payload as $l_{message}$ and assume a payload of one kilobyte.

We begin with the analysis of the lower and upper bounds on the CServ Internetwork Service header size. The lower bound for internetwork service requires a CServ Internetwork Service path specified by four hops, namely the source host's upstream CServ access router and egress gateway router in the source subnet and the destination host's ingress gateway router and upstream CServ access router in the destination subnet. The upper bound, alternatively, is determined by the Path Length field, which encodes the length of the variable part of the header, the sequence of CServ Internetwork Service path hops. As Path Length occupies an 8-bit field, it is able to encode unsigned integers up to 255. We denote these two values as h_{min} and h_{max}, respectively. Referring to Fig. 3-5, the byte (or octet) numbering system to the left of the header description guides the size analysis by indicating the leading byte index for each header row (and each row consists of two bytes). The general expression for the length in bytes of the CServ Internetwork Service header that encodes a CServ Internetwork Service path with h hops, denoted $l_{inter}(h)$ is:

$$l_{inter}(h) = 114 + 16h. \tag{3.1}$$

We can use the expression in Eq. (3.1) to compute the lower and upper bounds on the CServ Internetwork Service header as follows:

$$\begin{aligned} l_{inter}^{min} &\triangleq l_{inter}(h_{min}) = 114 + 16h_{min} = 114 + 16(4) = 178, \\ l_{inter}^{max} &\triangleq l_{inter}(h_{max}) = 114 + 16h_{max} = 114 + 16(255) = 4194. \end{aligned} \tag{3.2}$$

We note the large range imposed by these bounds: $178 \leq l_{inter}(h) \leq 4194$. In particular, the upper bound on the CServ Internetwork Service header is four times the size of the expected CServ message

data payload of one kilobyte. The CSDCA, which computes CServ Internetwork Service paths, limits the possible path lengths by the algorithm design since it requires bounded and well-defined execution time. Thus, even though the header is technically capable of encoding paths with 255 CServ router hops, this full flexibility is not used in practice.

The typical CServ Internetwork Service header is expected to fall near the lower end of these bounds. We define a typical CServ Internetwork Service path as one with two transit subnets. Therefore, the typical CServ Internetwork Service header needs to encode eight hops, namely the source host's upstream CServ access router and egress gateway router in the source subnet, the ingress-egress gateway router pairs in each transit subnet, and the destination host's ingress gateway router and upstream CServ access router in the destination subnet. We denote the typical number of hops in a CServ Internetwork Service path as $h_{typical}$. With Eq. (3.1), we have the following for the typical CServ Internetwork Service header length in bytes:

$$l_{inter}^{typical} \triangleq l_{inter}(h_{typical}) = 114 + 16h_{typical} = 114 + 16(8) = 242. \tag{3.3}$$

Using the values for l_{inter}^{min}, $l_{inter}^{typical}$, and l_{inter}^{max} we can compute the fractional overhead required by the CServ Internetwork Service header with respect to the CServ message data payload, $l_{message}$. We denote the fractional overhead for the CServ Internetwork Service header that encodes h hops as $\Gamma_{inter}(h)$, and we have the following:

$$\begin{aligned}
\Gamma_{inter}^{min} &\triangleq \Gamma_{inter}(h_{min}) = \frac{l_{inter}^{min}}{l_{message}} = \frac{178}{1000} = 17.8\%, \\
\Gamma_{inter}^{typical} &\triangleq \Gamma_{inter}(h_{typical}) = \frac{l_{inter}^{typical}}{l_{message}} = \frac{242}{1000} = 24.2\%, \\
\Gamma_{inter}^{max} &\triangleq \Gamma_{inter}(h_{max}) = \frac{l_{inter}^{max}}{l_{message}} = \frac{4194}{1000} = 419.4\%.
\end{aligned} \tag{3.4}$$

Considering the range on the CServ Internetwork Service header length, the range in overhead values for a constant CServ message data payload is not surprising. Considering these results, we can accept a reasonable overhead which is less than 25% for typical CServ Internetwork Service paths. However,

without bounding the length of these paths, the protocol control overhead can grow out of control and become quite unreasonable with respect to the message size.

3.3 State Measurement Service Header

A critical component of the CServ architecture is the measurement of subnet internal CServ performance metrics which are used to discover CServ Internetwork Service paths and compose end-to-end service that meets the demands of the CServ application. This is necessary since the MC otherwise has no direct visibility into the topology or routing state of the individual subnets, which are left to operate autonomously in terms of their routing and forwarding strategy in order to best meet the needs of their own system. To this end, the CServ design uses internetwork CServ datagrams to measure the actual performance of CServ Intranetwork Services as they traverse the network.

The full details of subnet state learning and reporting protocols are covered in Chapter 4. State measurement service headers prepended to the internetwork CServ datagram could be considered, informally, as part of the CServ Intranetwork Service header since they are specific to the subnet. We choose to treat them separately because:

1. individual subnets may employ different state measurements protocols and thus require different state measurement service headers;
2. these headers are strictly optional since not every internetwork CServ datagram is used as a state learning probe;
3. and CServ Intranetwork Service headers may be standard network protocols (such as IP or MPLS) so we differentiate CServ-exclusive fields by identifying state measurement service headers as their own header layer in the stack.

For the purposes of this chapter, we only wish to characterize the protocol overhead required in the datagram header, so the remainder of this section treats the optional state measurement service headers generally and simply provides an upper bound on the size requirement without details. As introduced in Section 3.1.3, each CServ proprietary header begins with a 4-bit Version field identifying the type of header (in this case, the specific state measurement protocol header), followed by a 4-bit Next Header field that allows for the specification of the next header in the layered header stack

(typically the CServ Internetwork Service header in this situation). Following this initial octet, we allot 36 maximum additional bytes for the fields required by the state measurement service header. Denoting the maximum size of the state measurement service header in bytes as l_{sms}^{max} (and noting that the size is independent of any CServ Intranetwork Service path length), we claim here without further detail that $l_{sms}^{max} = 37$. We denote the maximum fractional overhead for the state measurement service header as Γ_{sms}^{max}, and we have the following:

$$\Gamma_{sms}^{max} = \frac{l_{sms}^{max}}{l_{message}} = \frac{37}{1000} = 3.7\%. \tag{3.5}$$

This is the maximum overhead imposed by the state measurement service header, a fairly trivial amount compared to the requirement of the CServ Internetwork Service header. In fact, the overhead for one of the state learning protocols that does not represent this maximum is closer to 1%. We leave the remainder of this discussion for Chapter 4, which is devoted to learning the state of CServ performance metrics.

3.4 CServ Intranetwork Service Header

The CServ Intranetwork Service header used within a subnet depends on the routing and forwarding policies of the individual subnet. As previously discussed, the subnet administrator may choose to employ IP-based longest-prefix matching as the forwarding mechanism along shortest-path routes in order to minimize the number of CServ router upgrades required within the subnet's routing core. Alternatively, the subnet administrator may want to use a label-based forwarding mechanism to implement CServ-enabled router to CServ-enabled router virtual circuits (VCs). These CServ Intranetwork Service tunnels serve as a hook for efficient traffic engineering decisions. The subnet administrator may even choose to pervasively adopt CServ-enabled routers in the routing core and then employ centralized route computation and explicit path forwarding just as the CServ architecture uses on a per-transaction basis for the CServ Internetwork Service (see Section 2.4).

These are among a few of the many possible implementations of the CServ Intranetwork Service. Alongside the breadth of choices for the service comes a breadth of possibilities for the CServ Intranetwork Service header. For example, an IP-based longest prefix matching scheme requires an IP

header, a labeled VC requires an MPLS label, and explicit path forwarding in the CServ architecture can use a CServ Intranetwork Service header that closely resembles the CServ Internetwork Service header. We briefly consider each of these options in more detail and analyze their respective overhead requirement, noting that there are many other possibilities for CServ Intranetwork Service that are not covered in this section.

3.4.1 IP-based CServ Intranetwork Service

We first consider the use of IP-based forwarding within a subnet for CServ Intranetwork Service, where each router uses longest-prefix matching to determine the next-hop interface for the intranetwork CServ datagram. The attractiveness of this option is that much of a subnet's routing core can operate using legacy equipment without upgrade, since most network providers already support IP mechanisms. The only routers that need to be CServ-enabled are those directly providing upstream access to access networks with CServ-enabled end hosts and those gateway routers that are part of the CServ internetwork routing topology.

The discussion here focuses on IP encapsulation with the IPv6 header format as specified in [55] since we are assuming 128-bit network addresses. The IPv6 header has a 40 byte structure, as long as no extension headers are employed. The Version field of the header would be used to indicate that it is indeed an IPv6 header (using the decimal value 6, or binary value 0110). Just as for CServ proprietary headers, the Next Header field would be used to encode the header type of the encapsulated payload (which, considering the discussion of Section 3.1.3, could masquerade as an IP header itself even if it is not). The Source Address field would be used for the address of the entry point into the subnet's IP domain, which is generally the ingress active gateway router or the upstream active access router generating and prepending this IPv6 header. The Destination Address field stores the next address in the explicit path sequence of the CServ Internetwork Service header, which serves as the departure point from the subnet's IP domain. The other header fields are populated just as they would be for normal IP service operation, where the 8-bit Traffic Class field for differentiated services (DiffServ [30]) can indicate the request for priority queueing and switching where available in the subnet through the Expedited Forwarding Per-Hop Behavior defined in [69].

For the purposes of this discussion, we consider the use of IPv6 in its basic form without extension headers. We use l_{intra}^{IPv6} to denote the size of the CServ Intranetwork Service header in bytes when using IP-based forwarding mechanisms, and thus $l_{intra}^{IPv6} = 40$. As before, we can consider the fractional overhead levied by using this CServ Intranetwork Service, which we represent by Γ_{intra}^{IPv6}, and we have the following:

$$\Gamma_{intra}^{IPv6} = \frac{l_{intra}^{IPv6}}{l_{message}} = \frac{40}{1000} = 4\%. \tag{3.6}$$

3.4.2 MPLS-based CServ Intranetwork Service

Next we consider the use of MPLS label-based forwarding within a subnet for CServ Intranetwork Service, where the next-hop interface for the intranetwork CServ datagram is dependent only on the label value and pre-established label-switched path. The attractiveness of this option is the opportunity for the network administrator to deploy simple traffic engineering solutions that direct CServ traffic between CServ-enabled routers. MPLS is also designed to support the tunneling of IP packets, and the CServ architecture is designed such that the encapsulated internetwork CServ datagram (or state measurement datagram) can masquerade as an IP packet until the leading Version code is processed by the endpoint CServ router, acting as a label edge router. (If necessary, the MPLS label allocates three experimental bits in the control information for a service code which can be used to signal the next header type in the header stack, or alternatively to request priority label-switching where available in the subnet.)

The format of the MPLS label is standardized in [19], which describes a short 32-bit label containing four fields. When an internetwork CServ datagram enters the MPLS domain, the ingress gateway router, acting as a label edge router, maps the next address in the explicit path sequence of the CServ Internetwork Service header to the appropriate label-switched path that provides the CServ Intranetwork Service. The corresponding 20-bit MPLS label (which totals 32 bits with the additional MPLS control information) is prepended to the CServ datagram. At the egress gateway router (or upstream access router in the destination subnet), the MPLS label is popped as the CServ datagram exits the MPLS domain. We do not cover the implementation details of MPLS route determination or the label distribution protocol in this document, although we note for the interested reader that two

standardized protocols for the management of MPLS paths are the Label Distribution Protocol [70] and an extension of the Resource Reservation Protocol (RSVP) for traffic engineering purposes [71].

When using MPLS for the CServ Intranetwork Service, we denote the size of the CServ Intranetwork Service header in bytes as l_{intra}^{MPLS}. Based on the discussion above, $l_{intra}^{MPLS} = 4$. The fractional overhead imposed by using this CServ Intranetwork Service, represented by Γ_{intra}^{MPLS}, is correspondingly:

$$\Gamma_{intra}^{MPLS} = \frac{l_{intra}^{MPLS}}{l_{message}} = \frac{4}{1000} = 0.4\%. \tag{3.7}$$

The data plane protocol overhead associated with MPLS-based CServ Intranetwork Service is negligible, but this does not account for the control plane overhead required for the management of label-switched paths and the distribution of the associated labels.

3.4.3 Explicit Path Forwarding as CServ Intranetwork Service

As the last example of CServ Intranetwork Service and its associated protocol control overhead, we consider the scheme introduced in Section 2.4. This approach leverages the existence of the subnet controller (SC) and the pervasive collection of state for CServ internetworking purposes to perform logically-centralized route computation for CServ Intranetwork Service paths. These routes are distributed to and stored by the CServ-enabled gateway and access routers, which prepend them to transiting CServ datagrams to explicitly guide them through the subnet.

Using the SC to determine the intranetwork service gives the subnet administrator the opportunity to maintain CServ Intranetwork Service paths that conform to defined standards in terms of the relevant CServ performance metrics, reliability and delay. Forwarding based on explicit route description in the intranetwork CServ datagram avoids costly lookups, such as required when doing IP longest-prefix match forwarding. Furthermore, this route computation approach opens the door for the use of router-disjoint path diversity between CServ-enabled routers to improve the subnet's transit reliability. On the flip side, this routing and forwarding strategy requires pervasive adoption of CServ-enabled routers in the subnet since each router along the explicit path must be capable of processing CServ formats. Additionally, explicit path computation requires significant processing effort at the SC to maintain

$n(n-1)$ CServ Intranetwork Service solutions for n active CServ-enabled routers in the subnet's routing core, and thus the solutions may be slower to update in response to the dynamic state evolution of the subnet.

When an internetwork CServ datagram enters the domain of a subnet using the CServ explicit path forwarding approach for CServ Intranetwork Service, the ingress gateway router or upstream active access router uses the next hop address in the CServ Internetwork Service path to look up the appropriate stored CServ Intranetwork Service. The resulting path (or set of paths if diversity routing is specified in the solution) is used to generate the CServ Intranetwork Service header. This header could take the same form as that depicted in Fig. 3-5 for the CServ Internetwork Service header, but we can bowdlerize some unnecessary fields to reduce the significant overhead imposed by this large set of control information (as shown in the fractional overhead analysis of Eq. (3.4)).

Let's consider the essential header fields required for the CServ Intranetwork Service; the resulting format for a CServ Intranetwork Service header using explicit path forwarding is illustrated in Fig. 3-6. As with other CServ-proprietary formats, the most significant 4-bit position is reserved for the Version (VER) field, which is used to mimic the leading IP field and signal the flavor of CServ header to follow. The next four bits, called Next Header (NH), are allocated to encode the header type of the datagram encapsulated by this intranetwork CServ datagram, typically a CServ Internetwork Service header or one of the state measurement service headers. The subsequent 8-bit Path Length (PL) field is used to encode the decimal number of hops in the CServ Intranetwork Service path, which is later specified in the header. The value of this field also controls the overall size of this variable length header format since it is dependent on the number of hops in the CServ Intranetwork Service path. The next 16 bits, the Payload/Data Length (DL) field, encode the decimal number of bytes of payload encapsulated by this intranetwork header. With 16 bits of precision, this can represent up to 65,535 bytes, which is significantly more than necessary for our purposes. This is followed by two 128-bit fields, one for the Entrance Address (EA) and one for the Departure Address (DA). These addresses are retrieved from the CServ Internetwork Service header. The Entrance Address is the address of the router that serves as the ingress into the routing core of the subnet (and the router that is generating this CServ Intranetwork Service header), while the Departure Address is the next hop address in the CServ Internetwork Service path specified in the CServ Internetwork Service header. These addresses are followed by a 32-bit Sequence Number (SN) that is used to quench diversity at the Departure Address router if diversity

routing is used for the CServ Intranetwork Service, and then a 64-bit Origin Timestamp (OT) that holds a value representing the current time when the CServ Intranetwork Service header is created. A sequence of h' 128-bit addresses completes the CServ Intranetwork Service header, where h' is the decimal value specified in the Path Length field. This sequence of addresses represents the explicit CServ Intranetwork Service path that the intranetwork CServ datagram follows to transit the subnet, and the sequence does not need to specify the first and last hop since these addresses are included in the Entrance Address and Departure Address fields, respectively.

Comparing this format to the CServ Internetwork Service header, shown in Fig. 3-5, we note that the 16-bit Checksum and the 512-bit Authentication Mark fields have been discarded. The inclusion of the Checksum in the CServ Internetwork Service header was intended to replace the weak checksum usually included in the transport layer protocol header. Stronger data integrity checks at the data link level are assumed, and thus we do not need another weak checksum in the network protocol header stack. Although the omission of the Checksum does not significantly impact the amount of overhead, the removal of the Authentication Mark field represents a hefty reduction in control information overhead compared to the CServ Internetwork Service header (where the Authentication Mark field represents 56.14% of the fixed-length section of the control information overhead). The authenticity check is used by the destination endpoint host to have reason to believe that the critical message data originated from the source host claimed in the internetwork CServ datagram. At the most fundamental level, this verification of authenticity is not necessary for CServ Intranetwork Service since the receiving router at the end of the CServ Intranetwork Service path is not the recipient of the critical message data.

Let us analyze the protocol overhead associated with this CServ Intranetwork Service. As with the CServ Internetwork Service header, the size of the control information overhead depends on the value encoded in the Path Length field. We can consider the lower and upper bounds on the CServ Internetwork Service header size, as well as an expected typical size. The lower bound for a subnet's intranetwork service requires a path specified by only two hops if the routing core ingress router (either a gateway router or access router) is adjacent to the routing core egress router at the network layer. In other words, if the router specified by the Entrance Address is connected to the router specified by the Departure Address through one of its interfaces with no additional routers in between, then only two hops are required for intranetwork service. Since these two hops are specified by the Entrance Address and the Departure Address fields in Fig. 3-6, the decimal value for the Path Length field in this case is 0.

Using h' to generically represent the decimal value encoded in the Path Length field, we denote the minimum valid value for h' as h'_{min}. As just explained, $h'_{min} = 0$. Note that this is different from the lower bound value encoded in the Path Length field when we analyzed the size of the CServ Internetwork Service header. The upper bound on the size of the CServ Intranetwork Service header, just like the CServ Internetwork Service header, is determined by the maximum integer value that can be encoded in the Path Length field specifying the intermediate sequence of CServ Intranetwork Service path hops between the endpoints. We denote the maximum valid value for h' as h'_{max}, as we have that $h'_{max} = 255$ as in the CServ Internetwork Service header analysis.

Referring back to Fig. 3-6, the byte (or octet) offset numbering system to the left of the header description guides the size analysis by indicating the leading byte index for each header row (and each header row consists of a pair of bytes). The general expression for the length, in bytes, of the CServ Intranetwork Service header that encodes an explicit CServ Intranetwork Service path with h' hops, denoted $l_{intra}^{path}(h')$, is:

$$l_{intra}^{path}(h') = 48 + 16h'. \tag{3.8}$$

We then use the expression in Eq. (3.8) to compute the lower and upper bounds on the CServ Intranetwork Service header for CServ explicit path forwarding as follows:

$$\begin{aligned} l_{intra}^{path-min} &\triangleq l_{intra}^{path}(h'_{min}) = 48 + 16h'_{min} = 48 + 16(0) = 48, \\ l_{intra}^{path-max} &\triangleq l_{intra}^{path}(h'_{max}) = 48 + 16h'_{max} = 48 + 16(255) = 4128. \end{aligned} \tag{3.9}$$

As in the CServ Internetwork Service header case, we note the wide latitude for header sizes imposed by these bounds: $48 \leq l_{intra}^{path}(h') \leq 4128$. In practice, the CServ Intranetwork service paths are limited by several factors that further bound the realized header sizes away from the theoretical upper bound $l_{intra}^{path-max}$, including:

1. the number of unique CServ-enabled routers in the subnet's routing core;
2. the diameter of the subnet's routing core graph;

3. and the CServ performance metrics associated with any candidate paths between a pair of CServ-enabled routers in the routing core.

So even though the header format is capable of encoding paths with 255 intermediate CServ router hops between the routers that serve as the ingress and egress from the subnet's routing core, this full flexibility is generally not exploited for practical reasons.

As is the case with the CServ Internetwork Service header, the typical CServ Intranetwork Service header for CServ explicit path forwarding is expected to fall near the lower end of the above bound. For analysis, we define a typical CServ Intranetwork Service path as one with four intermediate router hops connecting the endpoint routers (those specified by the Entrance Address and the Departure Address fields). Therefore, the typical header needs to encode four hops in the variable-length section of the header. We represent the typical number of intermediate hops in a CServ Intranetwork Service path as $h'_{typical}$, and we have $h'_{typical} = 4$. With Eq. (3.8), we have the following for the typical CServ Intranetwork Service header size, in bytes, when used for CServ explicit path forwarding:

$$l_{intra}^{path-typical} \triangleq l_{intra}^{path}(h'_{typical}) = 48 + 16h'_{typical} = 48 + 16(4) = 112. \tag{3.10}$$

Comparing this result with Eq. (3.3), we note that this is less than half the typical size for the CServ Internetwork Service header.

Using the values for $l_{intra}^{path-min}$, $l_{intra}^{path-typical}$, and $l_{intra}^{path-max}$, we can compute the fractional overhead required by the CServ Intranetwork Service header for CServ explicit path forwarding with respect to the CServ message data payload, $l_{message}$. We denote the fractional overhead for the CServ Intranetwork Service header for CServ explicit path forwarding that encodes h' intermediate router hops as $\Gamma_{intra}^{path}(h')$, and we have the following:

$$\Gamma_{intra}^{path-min} \triangleq \Gamma_{intra}^{path}(h'_{min}) = \frac{l_{intra}^{path-min}}{l_{message}} = \frac{48}{1000} = 4.8\%,$$

$$\Gamma_{intra}^{path-typical} \triangleq \Gamma_{intra}^{path}(h'_{typical}) = \frac{l_{intra}^{path-typical}}{l_{message}} = \frac{112}{1000} = 11.2\%, \quad (3.11)$$

$$\Gamma_{intra}^{path-max} \triangleq \Gamma_{intra}^{path}(h'_{max}) = \frac{l_{intra}^{path-max}}{l_{message}} = \frac{4128}{1000} = 412.8\%.$$

Although comparatively large to an IPv6 header or an MPLS label as the CServ Intranetwork Service header (compare with Eq. (3.6) and Eq. (3.7)), the typical overhead for a CServ Intranetwork Service header for CServ explicit path forwarding seems acceptable (we try to put this into perspective in the subsequent section of this chapter). However, without bounding the length of the CServ Intranetwork Service paths, the protocol overhead can grow out of control and become quite unreasonable with respect to the critical message data size.

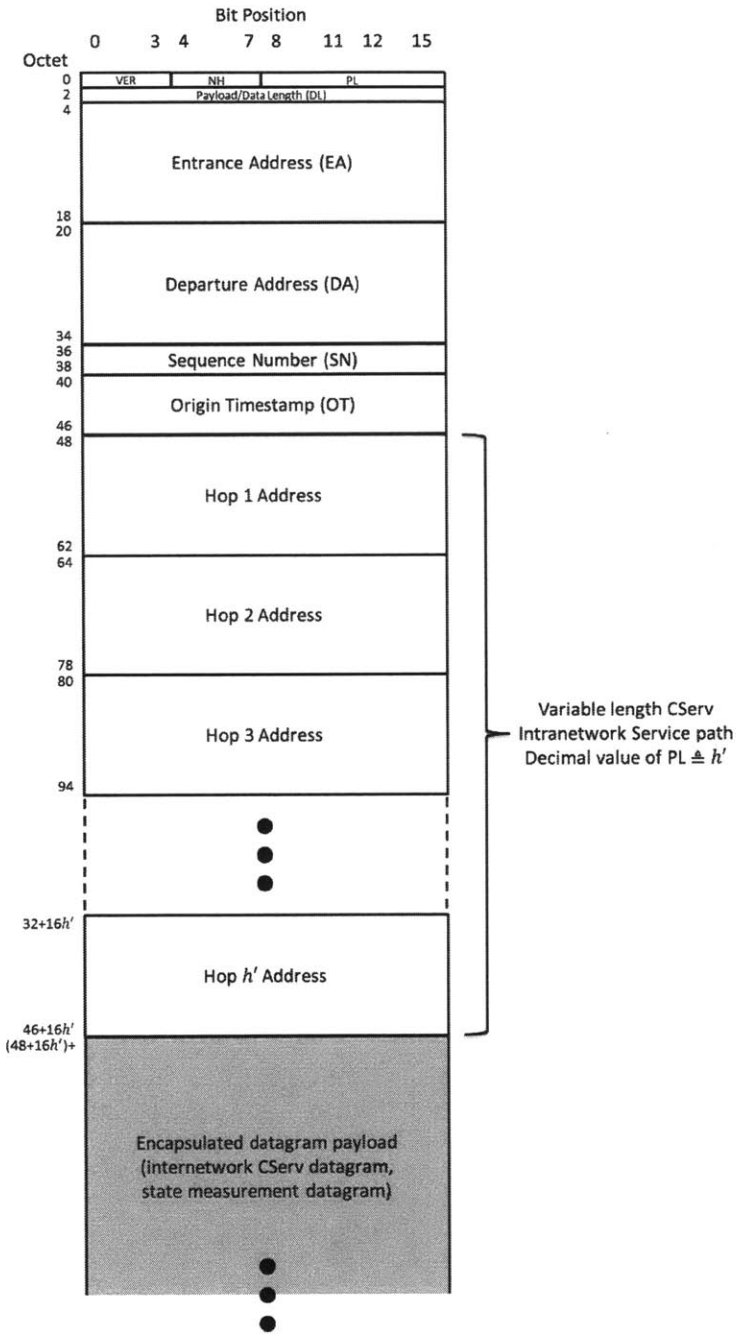


Fig. 3-6: The variable length format of the CServ Intranetwork Service header used for CServ explicit path forwarding service is depicted here, to scale, based on the discussion of Section 3.4.3. The realized header length depends on the value stored in the Path Length (PL) field.

3.5 CServ Datagram Protocol Overhead Analysis

In this section, we put the CServ protocol overhead together as a holistic picture, synthesizing the analyses from Sections 3.2-3.4. Although there are several flavors of CServ header stacks as described throughout the chapter, we consider a few representative examples and use the typical CServ Internetwork Service path and typical CServ Intranetwork Service path lengths.

Example 1: ***Intranetwork CServ Datagram with State Measurement Header using IPv6 CServ Intranetwork Service***

$$l_{intra}^{IPv6} + l_{sms}^{max} + l_{inter}^{typical} = 40 + 37 + 242 = 319 \text{ octets}$$
$$\Gamma_{intra}^{IPv6} + \Gamma_{sms}^{max} + \Gamma_{inter}^{typical} = 4\% + 3.7\% + 24.2\% = 31.9\%$$

Example 2: ***Intranetwork CServ Datagram with State Measurement Header using MPLS CServ Intranetwork Service***

$$l_{intra}^{MPLS} + l_{sms}^{max} + l_{inter}^{typical} = 4 + 37 + 242 = 283 \text{ octets}$$
$$\Gamma_{intra}^{MPLS} + \Gamma_{sms}^{max} + \Gamma_{inter}^{typical} = 0.4\% + 3.7\% + 24.2\% = 28.3\%$$

Example 3: ***Intranetwork CServ Datagram without State Measurement Header using CServ Explicit Path Forwarding***

$$l_{intra}^{path-typical} + l_{inter}^{typical} = 112 + 242 = 354 \text{ octets}$$
$$\Gamma_{intra}^{path-typical} + \Gamma_{inter}^{typical} = 11.2\% + 24.2\% = 35.4\%$$

Example 4: ***Intranetwork CServ Datagram with State Measurement Header using CServ Explicit Path Forwarding***

$$l_{intra}^{path-typical} + l_{sms}^{max} + l_{inter}^{typical} = 112 + 37 + 242 = 391 \text{ octets}$$
$$\Gamma_{intra}^{path-typical} + \Gamma_{sms}^{max} + \Gamma_{inter}^{typical} = 11.2\% + 3.7\% + 24.2\% = 39.1\%$$

We note that the actual protocol overhead may vary from the numbers calculated above. Not only are the CServ Internetwork Service header and the CServ Intranetwork Service header for CServ explicit path forwarding path-length dependent, but the state measurement header value we use is a maximum. The actual details of the state measurement protocols are discussed in Chapter 4, and at that time we describe a protocol with a state measurement header of only 13 bytes.

The next observation we make is that the CServ architecture imposes a relatively large network protocol overhead for its datagrams (although, one may understandably argue that the state measurement protocol is outside the traditional network protocol layer). Even with reasonable internetwork and intranetwork path lengths, the fractional overhead borders on 40%. Each additional transit subnet in the CServ Internetwork Service path would contribute another 32 bytes to the overhead, whereas each additional intermediate hop in the CServ Intranetwork Service path (assuming the use of CServ explicit path forwarding) contributes another 16 bytes to the protocol overhead. This is a direct result of encoding explicit paths of 128-bit addresses directly into the critical message datagram header. Although this burden is a requirement of the CServ architecture at the Internetwork Service level, we can avoid the overhead explosion within the subnet by using a protocol that does not grow with the length of the CServ Intranetwork Service path. For example, MPLS as the CServ Intranetwork Service uses pre-computed service tunnels between CServ-enabled routers, giving the same control as directly encoding that path, hop-by-hop, into the datagram control information. That being said, modifying the CServ Intranetwork Service path using CServ explicit path forwarding only requires the manipulation of the forwarding table state on the ingress router, whereas changing an MPLS label-switched path requires installing new forwarding state on each router along that path. The centralized computation of explicit paths also facilitates the use of diversity-routed solutions for improved CServ performance metrics. The trade-off between CServ Intranetwork Service options such as MPLS and CServ explicit path forwarding is not as clean-cut as comparing the respective protocol overhead burden and demands careful consideration of other factors.

Before moving on, we pose one more important consideration with respect to the CServ datagram protocol overhead examples above. The CServ message transaction should avoid datagram fragmentation between the transaction endpoints as this incurs fragmentation/reassembly overhead and additional delay. The objective of the CServ critical messaging service is to send *one* datagram with *a priori* performance metric guarantees. This formulation is complicated if the critical message needs to

be fragmented into multiple CServ datagrams at the CServ source host. Although the CServ specification does not assume control over data link layer designs and transmission protocols, we briefly consider some ubiquitous examples and the interplay with the CServ datagram size.

We would like to present a datagram for transmission that does not require host endpoint fragmentation. Although we control the CServ-specific network protocols, namely the CServ Internetwork Service and the CServ explicit path forwarding service for CServ Intranetwork Service, we need to be wary of the restrictions of other network protocols, such as the use of IPv6 as a CServ Intranetwork Service or any data link layer transmission protocols that restrict the unit transmission size. By precept, IPv6 does not perform fragmentation to simplify router processing requirements (although we note that optional header extensions allowing IPv6 fragmentation do exist for the extreme case). The IPv6 specification requires endpoints to perform path maximum transmission unit (MTU) discovery and then present appropriately sized payloads for IPv6 network transit such that the IP datagrams do not exceed the path MTU. Additionally, IPv6 makes the hard promise to deliver any IPv6 packet smaller than or equal to 1280 octets without the need for network layer fragmentation (if a link transmission protocol cannot convey a 1280 octet payload, IPv6 requires the existence of link-specific fragmentation and reassembly at a lower layer) [55].

Considering Example 1 above, it is clear that the CServ datagram exceeds 1280 octets for reasonable CServ Internetwork Service path lengths and a one kilobyte CServ message payload, even if we remove the state measurement service header. With fewer transit subnets in the CServ Internetwork Service path or a smaller CServ message payload, the datagram could be condensed to fewer than 1280 octets, but this cannot be guaranteed. However, the 1280 byte clause is just a minimum guarantee of IPv6. The reality is that many practical path MTU values are closer to 1500 bytes; this is because the maximum payload size of the Ethernet v2 data link frame is 1500 octets, and thus the maximum datagram size at the network layer is 1500 octets [35]. If we assume this as a reasonable path MTU, then we have room for the encoding of additional transit subnets in the CServ Internetwork Service path without worrying about the need for endpoint fragmentation. The encoding of each additional transit subnet in the CServ Internetwork Service path requires an additional 32 bytes, meaning that we could extend the path to include another five transit subnets (for seven total transit subnets in the source subnet to destination subnet internetwork path) without exceeding the 1.5 KB MTU constraint.

The bottom line is that the ideal CServ datagram would never exceed the source host to destination host path MTU, avoiding the need for CServ message fragmentation at the endpoint and, additionally, avoiding the increased delay and jitter associated with any data link layer fragmentation/reassembly. While 1500 octets is a reasonable assumption on network-wide minimum MTU for the purposes of this document (and there are certainly data link protocols with larger MTU values, such as IEEE 802.11 [72] with a 7981 octet MTU), its future acceptance network-wide must be enforced and not just assumed. Endpoint hosts do not have the luxury to perform path MTU discovery prior to each CServ datagram transaction since this would incur a prohibitive amount of pre-transmission delay (see [73] and [74] for the description of a path MTU discovery protocol). Thus, the CServ architecture needs to be aware of the network minimum MTU to work within the parameters this restriction presents and avoid CServ datagram fragmentation. This may be challenging in light of Examples 3 and 4 above, where there are two sources of overhead control information variability – the CServ Internetwork Service path length and the CServ Intranetwork Service path length. In our analysis using typical path lengths, we see that there is not much more than 100 bytes worth of headroom remaining to encode longer internetwork or intranetwork paths without bumping up against the 1500 byte ceiling. But in a subnet with pervasive CServ-enabled router adoption and the use of CServ explicit path forwarding, it is not a stretch to imagine that the interconnection link transmission technologies could be upgraded to allow for a larger subnet-wide minimum MTU than the suggested 1.5 KB baseline.

3.6 Conclusion

In this chapter, we covered the datagram formats and required control information used to direct CServ datagrams from source host to destination host along the data plane of a CServ Internetwork Service path, traversing multiple subnets with provide their own CServ Intranetwork Service. The CServ datagram design takes full advantage of the power of encapsulation, which allows for the tunneling of CServ-proprietary datagram formats through a subnet which provides some other CServ Intranetwork Service, such as IP or MPLS. The CServ message data payload is encapsulated in an internetwork CServ datagram, which in turn can be encapsulated in one of various types of intranetwork CServ datagrams based on the CServ Intranetwork Service provided by each subnet along a CServ Internetwork Service path. Additionally, although without any specifics, we identified the position in the CServ datagram encapsulation process for the header fields related to CServ performance metrics measurement. As we

introduced in Chapter 2, the use of CServ datagrams to learn the performance of CServ Intranetwork Service paths is a fundamental requirement of the CServ architecture.

Through the description of the header formats and encapsulation methods, we alluded to the CServ datagram processing required at CServ-enabled routers in the network, particularly those routers that define a CServ Internetwork Service path or those that sit at the edge of a CServ Intranetwork Service path (the subnet domain entrance and departure routers). These routers use the information in the CServ Internetwork Service header to generate appropriate CServ Intranetwork Service headers corresponding to the next hop in the CServ Internetwork Service path before enqueueing these datagrams for priority switching and transmission (when available). The CServ Intranetwork Service header is then responsible for bearing the CServ datagram across the data plane of the subnet to the CServ-enabled departure router. In the next chapter, we describe additional CServ-enabled router processing requirements related to the measurement and reporting of CServ performance metric state.

Along the way, we considered some implementation details not yet formalized in the CServ architecture, such as the addressing scheme, the integrity and authentication protocols, and cross-layer signaling methods. These discussions were essential for the sizing of the respective control information fields, but the final decision on the protocols or methods has been left for future determination. The authentication field, in particular, is a dominant contributor to the overhead levied by the CServ Internetwork Service header. An authentication scheme requiring smaller digital signatures or authentication codes, yet with a realizable implementation within the delay-constrained CServ messaging framework, would be greatly welcomed as it would free up space in the CServ header overhead for the encoding of longer CServ Internetwork Service paths or longer explicit CServ intranetwork paths.

Finally, we analyzed the typical overhead incurred by the different header layers within this stacked CServ datagram format for several cases. This effort is used extensively in the subsequent chapter in the analysis of CServ performance metric state measurement protocols. Since we propose to use CServ datagrams to infer the state of the network, we need to consider the burden that this datagram traffic places on the data plane. It is necessary to bound the CServ traffic far away from the network routing and transmission capacity since the CServ performance metrics necessarily suffer as the load increases (not to mention that it could render the best effort network service inoperable). This control

information overhead analysis also highlights a point of potential fragility for the CServ architecture; the explicit encoding of path information in the CServ datagram can lead to datagrams that are either too large for the network or that adversely impact other CServ datagrams. While all CServ datagrams are intended to bear short message payloads, control information explosion can make one CServ datagram appear as an elephant in comparison to the mouse, or a CServ datagram with minimal control information overhead.

Chapter 4

CServ Performance Metric Learning Protocols

Learning and reporting the state of the subnet in terms of the CServ performance metrics is a critical pillar of the CServ architecture. Leaving the internal subnet routing and forwarding control to the subnet administrator is an important aspect of the design, minimizing the network details that the master controller (MC) needs to manage while simultaneously allowing the subnet provider the freedom to make internal policy decisions that align with their network capabilities and business needs. This freedom of internal implementation, however, requires that the subnet measure and announce the resulting CServ performance metric state in order for the MC to make any meaningful end-to-end determination on critical message internetwork service. By learning and reporting the reliability and delay statistics for the chosen CServ Intranetwork Service paths that bear CServ datagrams between CServ-enabled gateway (and access) routers, the subnets enable the logically-centralized control entity to compose CServ Internetwork Service paths that meet the performance requirements of a CServ Request (CSR) specifying a desired level of reliability and bound on tolerable source host to destination host service delay, all without disclosing the details of the subnet's internal routing policy or network topology.

In this chapter, we describe two possible protocols that empower the subnet to measure CServ performance metrics for the chosen CServ Intranetwork Service paths. While the objective of these two protocols is the same, we motivate the inclusion of both in this document. The first protocol uses *Learning Sessions* to learn the CServ performance metrics, while the second protocol shifts the workload to the network routers and employs infrequent *Collector datagrams* to amalgamate the details for each CServ Intranetwork Service path. Although we consider the usage of these alternatives in more depth as we describe their implementations, we begin here by highlighting a few of the primary considerations

for the selection of one or the other. Either protocol may be deployed within a subnet for the purpose of CServ performance metric learning depending on the network's capabilities.

The Learning Session protocol uses live CServ datagrams to estimate the CServ performance metrics between two CServ-enabled routers in the subnet routing core. Supplementing real CServ traffic with "dummy" CServ traffic that serve the role of active probes, fixed rate CServ traffic sessions between router entities are used to learn the reliability and delay statistics of the CServ Intranetwork Service path (or paths) that connects them logically during fixed interval learning periods. In order to learn the CServ performance metrics with acceptable precision, this technique requires relatively high session rates compared to the alternative Collector approach. The benefits of using these sessions are that:

- it imposes fewer computational requirements at each CServ-enabled subnet router;
- is opaque to any non-active router (one that is not upgraded for CServ support);
- and finds more accurate and robust performance measurements.

Real CServ traffic can expect to encounter the same performance since it is learned using fixed-rate sessions composed of that same type of traffic. The drawbacks of this approach, in addition to the session rate overhead that the subnet must support, are that it introduces a challenging access control problem (if the aggregated real CServ traffic exceeds the session rate, then the measured performance does not accurately indicate the experience of any particular CServ datagram) and that it consumes a significant subnet transmission and switching capacity overhead even when the subnet behavior is stable and the CServ performance metrics are not changing. For this reason, the subnet must be capable of supporting the Learning Sessions with plenty of capacity headroom to simultaneously bear best effort traffic. If CServ Intranetwork Service paths are recomputed, then it must be done with care not to oversubscribe any particular link or router's switching ability since this can lead to subnet instability or degradation of best effort service capabilities. This management and traffic engineering burden may be prohibitive or impossible (depending on routing and forwarding implementations) in some subnets. We consider these trade-offs in more detail after presenting the details of the Learning Session protocol.

Alternatively, the Collector protocol shifts the work to the subnet's routers to reduce the burden on the subnet transmission and switching capacity. This approach requires that all CServ Intranetwork Service

paths traverse only routers that are upgraded for the CServ architecture. The active routers then use all traffic, real CServ datagrams and best effort, to maintain running estimates of the CServ performance metrics each fixed interval learning period. This is done without the supplement of any "dummy" active probe traffic. Infrequent Collector datagrams in the CServ traffic class are then used to amass the CServ performance metric statistics along the chosen CServ Intranetwork Service paths as they traverse the constituent routers. The benefits of using the Collector protocol is that it introduces minimal transmission and switching capacity overhead and provides more flexibility for CServ Intranetwork Service path rerouting without concerns about link or router oversubscription. There are several drawbacks for this approach, however, including that:

- it necessitates a higher penetration rate for CServ-enabled routers in the routing core to support CServ Intranetwork Service paths;
- it requires additional router processing capabilities to monitor datagram performance maintain CServ performance metric estimates;
- and it finds less accurate and robust CServ performance metric estimates since it uses traffic in the best effort class for learning purposes.

In fact, we expect the performance that a real CServ datagram encounters to often outperform the learned CServ performance metrics since CServ traffic gets switching and transmission priority over best effort traffic. From this point of view, the Collector approach finds conservative CServ performance metrics estimates. However, it is not robust against influxes of real CServ traffic that alter the collective subnet traffic statistics during an event that generates many critical messages, whereas the Learning Session approach provides a safeguard against these transient periods of traffic fluctuation through its relatively high-rate traffic sessions that can be "filled" with real critical messages. Again, we consider these trade-offs in more detail after presenting the details of the Collector protocol.

In the end, it is up to the subnet administrator to choose between the two protocols presented in this chapter as candidates for learning and reporting the CServ performance metrics that correspond to their internal CServ Intranetwork Service paths. This decision may be dependent upon the physical network modality and available transmission and switching capacity, the subnet topology, or even the administrator's desire and ability to adopt CServ-upgraded routers in the subnet core. Additionally, there may be other reasonable protocols yet explored that meet the same goals. At the end of the

chapter, we briefly consider a more theoretical approach to CServ performance metrics estimation that could further reduce the learning protocol burden on the subnet. However, further studies and experimentation would be required to determine how much accuracy is lost in such an approach and whether or not the reduction in overhead is worth this compromise. Ultimately, the CServ user and CServ-dependent application needs to be aware of the limitations of the architecture regardless of the state measurement protocols deployed. Although the architecture objective is to provide probabilistic guarantees, these guarantees are made based on the details of the CServ performance metrics learning protocols. And as we see in this chapter, these learning protocols are grounded in measurements from the observed state of the constituent subnets. As such, this state is always necessarily stale to some extent, and these approaches cannot account for the "Black Swan" network events [52] that have not been previously observed or encountered.

In this chapter, we present the details of both the Learning Session and Collector approaches to the subnet CServ state measurement protocol. We analyze these approaches in terms of the network traffic burden that they impose for two representative topologies that notionally represent a sort of upper and lower bound in our context. Finally, we discuss the tradeoffs between these two approaches, the peering gateway router CServ performance metrics learning approach, the access network CServ performance metrics learning approach, the reporting responsibilities of the routers and subnet controller (SC), and a possible direction for a more theoretical approach to CServ performance prediction. This coverage of the state measurement protocol prepares us for discussion of the internetwork graphical representation and MC path discovery and composition algorithm for CServ Internetwork Service in the next chapter.

4.1 Network Model for State Measurement Protocol Discussion

Before we begin with the coverage of the CServ performance metrics state learning protocols, we standardize the network model that we use in the description and analysis of the approaches.

The purpose of the CServ performance metrics learning protocols are to measure the reliability and delay statistics between pairs of CServ-enabled routers in a subnet that may serve as endpoints for CServ Intranetwork Service bearing critical datagrams along their prescribed CServ Internetwork Service path. For the purpose of protocol description and analysis, we are interested in the number of routers in

the subnet routing core topology, the number of routers that are CServ-enabled, and the routing connection topology. We denote the set of routers in a subnet's routing core as N_R, which includes both active CServ-enabled routers and non-active legacy routers that are not capable of processing CServ datagrams or interpreting CServ header control information. For notational simplicity, we define the cardinality of the set N_R as follows: $|N_R| \triangleq n_R$.

As introduced in Section 2.2, the set of routers includes access routers, gateway routers, and core routers, all of which may either be active (upgraded to support CServ datagram processing) or non-active (not capable of CServ datagram processing). We assume in analysis that all routers in N_R are CServ-enabled active routers (although we make comments about situations where there are non-active routers in the subnet core). We denote the set of access routers as N_{R-A}, the set of gateway routers as N_{R-G}, and the set of core routers as N_{R-C}. We have previously discussed in Chapter 2 that a router may technically serve both as an access router and gateway router, for example. However, for simplicity in the discussion and analysis here, we assume that all routers serve in one role exclusively (a gateway router that provides upstream connection for an access network can be logically separated into two routers to align with this assumption). With this assumption, the sets N_{R-A}, N_{R-G}, and N_{R-C} do not intersect, and $N_R = N_{R-A} \cup N_{R-G} \cup N_{R-C}$. Further for the purposes of analysis, we assume a null set of core routers that do not serve the purpose of providing upstream connection for access networks or of providing peering connection to neighboring subnets (i.e. $N_{R-C} = \emptyset$). Core routers capable of processing CServ datagrams are significant only for the Collector protocol or for the forwarding of CServ datagrams that use explicit-path forwarding for their CServ Intranetwork Service. However, they do not need to generate or sink any Learning Sessions or Collector datagrams. With this additional assumption, we have that $N_R = N_{R-A} \cup N_{R-G}$. We use the following notation for the cardinality of the sets N_{R-A} and N_{R-G}: $|N_{R-A}| = n_{R-A}$ and $|N_{R-G}| = n_{R-G}$. Thus, we have $n_R = n_{R-A} + n_{R-G}$.

As far as routing topology is concerned, we employ two distinct topology examples in this chapter which are meant to represent upper and lower bound analysis scenarios for the traffic burden imposed by the state learning protocols. For these topologies, we standardize the number of routers to facilitate comparison between the two. In particular, we choose to consider two topologies where $n_R = 10$. We do not need to distinguish between the cardinality of the access router and gateway router sets for the purpose of the routing core topology learning protocol analysis. The upper bound, or worst-case topology, is a line network, as shown in Fig. 4-1. This topology is a particularly pessimistic case since

there are no routing options. For each pair of routers in N_R, there is only one candidate path connecting them to bear the Learning Session traffic or the Collector datagram. We shall see in the analysis that the links in the middle of the line network need to carry the heaviest transmission burden for the state measurement protocol traffic. On the other hand, the selection of the best-case, or "lower bound," topology requires more careful consideration. The subnet topology that would actually minimize the transmission burden on any given link is a fully-connected topology where each router in N_R is directly connected to all other routers in the set. This requires $n_R(n_R - 1)$ links (or $n_R(n_R - 1)/2$ undirected links) and scales as $O({n_R}^2)$. While this type of topology might be reasonable for small subnets with $n_R = 10$, most subnet realizations will likely have restrictions on router degree based on the physical network layout and cost of link deployment (for example, consider the capital and operational investment required to dig and pull additional fiber cables between sites). For this reason, we prefer to consider a representative best-case topology with constrained router degree rather than the overly optimistic fully-connected case, even though the representative topology is not a true analytic lower bound. Under the all-to-all learning condition that we consider where all $n_R = 10$ routers are CServ-enabled and engaged as endpoints in the state measurement learning protocol, the Petersen graph serves as a reasonable best-case topology to analyze since it belongs to family of Generalized Moore graphs (it is, in fact, a Moore graph) and the all-to-all protocol load can be distributed uniformly over all links in the strongly regular topology. A Generalized Moore graph for a given number of vertices and fixed degree serves provably as a lower bound for the average minimum hop distance between vertices for any irregular topology with the same number of vertices and a maximum degree equal to the Moore graph's degree [75]. The Petersen graph topology is depicted in Fig. 4-2.

For the analysis of the CServ performance metrics learning protocols, we make two additional assumptions for tractability. First, we assume that CServ Intranetwork Service does not use path diversity to connect any pair of routers in the set N_R. The use of diversity routing in CServ Intranetwork Service can increase the reliability of the connection, but it does so at the expense of consuming additional subnet transmission and switching bandwidth. A subnet administrator must take this additional overhead burden into account if diversity routing is used to improve the measured and reported CServ performance metrics. And second, we assume shortest-path routing between the routers in N_R for CServ Intranetwork Service paths where all links are given the same weight. Equivalently, these paths are those with the fewest number of intermediate routers (or fewest hops). This routing strategy may not necessarily be employed by a subnet administrator. In an actual

deployment, traffic engineering techniques, such as Multiprotocol Label Switching (MPLS) virtual circuits or strategic link weighting, could be leveraged to direct CServ Intranetwork Service paths over preferred links or paths. The subnet administrator needs to consider the impact of CServ Intranetwork Service path routing on the distribution of the CServ performance metrics learning protocol burden.

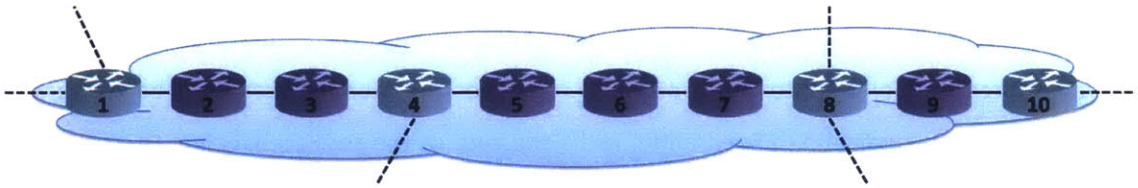


Fig. 4-1: The line network subnet topology with $n_R = 10$ is used as a worst-case (upper bound) example for the state measurement protocol overhead analysis. In this illustration, $n_{R-A} = 6$ and $n_{R-G} = 4$.

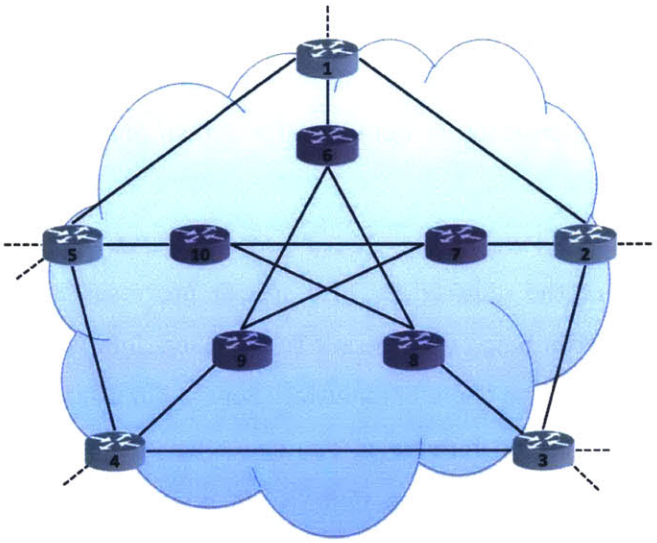


Fig. 4-2: The Petersen graph subnet topology is used as a representative best-case ("lower bound") example, assuming restricted router degree, for the state measurement protocol overhead analysis. In this illustration of the Petersen graph subnet, $n_{R-A} = 5$ and $n_{R-G} = 5$.

4.2 The Learning Session Protocol

As introduced earlier in the chapter, the Learning Session protocol uses live CServ datagrams to estimate the CServ performance metrics between two CServ-enabled routers in the subnet routing core. Supplementing real CServ traffic with "dummy" CServ traffic that serve the role of active probes, fixed rate CServ traffic sessions between router entities are used to learn the reliability and delay statistics of the CServ Intranetwork Service path (or paths) that connects them logically during fixed interval learning periods. The fixed-rate session does the work in this protocol, while the endpoint routers have simple tasks. The source endpoint router is responsible for generating dummy active probe datagrams to supplement the actual transiting CServ traffic to fill the rate of the session, labeling these datagrams with the appropriate state measurement service headers for the Learning Session protocol. The destination endpoint router in the subnet has the responsibility of counting the received datagrams within a fixed-interval learning period to compute the CServ Intranetwork Service reliability and calculating the delay of each datagram while maintaining a running estimate of mean delay and delay variance for each learning period. The destination endpoint router for the session is also responsible for reporting the estimated CServ performance metrics to the SC when they deviate far enough to consider the change in state worth updating in the control hierarchy (this is considered more in Section 4.6 since the reporting responsibilities are the same regardless of the state learning protocol used). Throughout this section, we describe this protocol in more detail and then analyze the burden imposed by its operation.

Before starting, we introduce the requirement of a specific processor at each CServ-enabled active router for the purpose of performing tasks specific to the CServ Internetwork Service and the State Measurement Service (as well as the CServ Intranetwork Service if the subnet administrator chooses to employ a CServ-specific routing and forwarding strategy). The primary processor workhorse of the router architecture, the Network Processing Unit (NPU), is generally pushed to the limits of its processing capability for the purpose of standard processing for arriving datagrams (say the processing of best effort datagrams) and the near-optimal scheduling of the switch fabric for high throughput. The CServ architecture introduces the need for active CServ routers to perform additional processing of CServ Internetwork Service headers, the mapping of CServ Internetwork Service to appropriate intranetwork service, the maintenance of State Measurement Service state, and the reporting of CServ performance metric estimates as they evolve with the network state. In order to avoid taxing the NPU

further, we suggest the use of a separate CServ Processing Unit (CSPU) in the datagram processing path at the CServ-enabled active router to perform these new tasks. This isolates the CServ-specific processing responsibilities from those that are native to the core router design, allowing the NPU to focus on the processing responsibilities that it is designed to handle most efficiently. This should also minimize the impact on router throughput incurred by the introduction of CServ-specific processing.

4.2.1 Endpoints of the Learning Sessions

Within the subnet, all active access routers and gateway routers are participants in the Learning Session protocol. Each router in this set establishes a Learning Session with each other router in this set. This is necessary because we do not assume that CServ Intranetwork Service paths are symmetric between any given pair of routers. Using the notation introduced in Section 4.1, each router in N_R establishes a pairwise Learning Session with each other router in N_R, which means that there are $n_R(n_R - 1)$ Learning Sessions within a subnet. Without the assumption that all routers in N_R are CServ-enabled access or gateway routers, there would be fewer Learning Sessions in the subnet. Non-active routers and core routers that are CServ-enabled but do not provide upstream connection for an access network do not need to generate or sink Learning Sessions. For example, consider the example of Fig. 4-3. This illustrates a subnet with two CServ-enabled gateway routers and one active access router providing upstream connection for an access network with CServ-enabled hosts. The other routers in the subnet are core routers and a non-active access router. The Learning Sessions are depicted following some arbitrarily determined CServ Intranetwork Service paths as arrows that are color-coordinated with the color used for the router's identifying address. As we observe, only six total Learning Sessions are necessary even though there are six routers in the subnet routing core.

The responsibility of facilitating these Learning Session connections is that of the SC. All CServ-enabled access and gateway routers in a subnet need to associate themselves with the SC such that it has a database of the internal routers participating in the CServ architecture. The SC then distributes the compiled list of addresses of the CServ-enabled routers to the members of the set. Once this list has been distributed from cold start, only incremental list updates are required when a new router associates with CServ or when a router previously in the set disassociates with CServ (possibly because all of the CServ-enabled endpoint hosts in its downstream access network have disconnected). The

concept of CServ-association is considered briefly in Chapter 5, although the protocol details of this elementary process are not fleshed out in whole in this document.

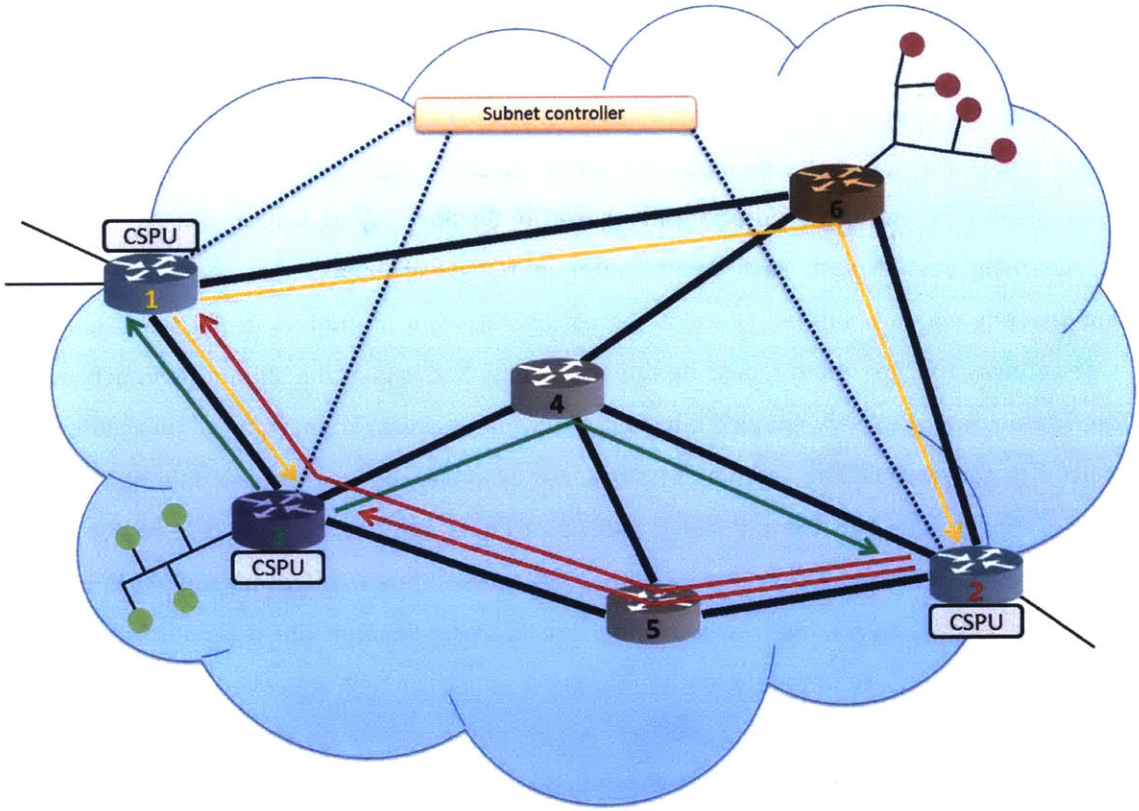


Fig. 4-3: This example subnet illustrates the Learning Sessions required for three active routers in the routing core. The color-coordinated arrows indicate the CServ Intranetwork Service paths connecting the active access routers and gateway routers. The CServ Processing Units (CSPUs) are the processors responsible for CServ-specific tasks at each active router.

4.2.2 Datagrams in the Learning Session

In this subsection, we consider a Learning Session from router A and router B within the subnet of interest, where $A, B \in N_R$. Router A may be an access router and router B may be a gateway router. As another example, both routers A and B may be the gateway routers. Depending on the CServ Intranetwork Service employed by the subnet to connect routers A and B and the topology of the routing core, this Learning Session may traverse multiple intermediate routers in N_R between A and B.

As mentioned previously, the Learning Session is composed of two types of traffic:

1. *passive* real CServ datagrams;
2. and *active* "dummy" probe datagrams.

As a network that supports the transmission of critical messages, there exists some rate of real CServ datagrams that flow from router A to router B according to the route specified in their respective CServ Internetwork Service paths computed by the MC. This rate is designed to be very low and generally varies over time. These real CServ datagrams are used opportunistically as part of the Learning Session. We call these datagrams *passive* probes because router A does not need to actively generate them to learn the performance of the CServ Intranetwork Service to router B. As real CServ traffic arrives at router A, it considers the next hop in the CServ Internetwork Service path specified in their CServ Internetwork Service headers. This is necessary for the determination of the appropriate CServ Intranetwork Service. Simultaneously, it uses this CServ Intranetwork Service destination address to assign the CServ datagram to the appropriate Learning Session. Before enqueueing the datagram in the router A's interface input buffer, the state measurement service header for Learning Sessions is prepended to the internetwork CServ datagram, followed by the CServ Intranetwork Service header. The real CServ datagram that is transiting router A to router B is now part of the Learning Session established between the two routers to measure the CServ performance metrics of the CServ Intranetwork Service.

By the design and intention of the critical messaging service, the rate of the real CServ datagrams traversing router A to router B is low (and also time dependent, since certain events may generate service transients with an influx of critical messages). This rate is generally expected to be insufficient to

estimate the CServ performance metrics accurately, particularly the reliability of the CServ Intranetwork Service between routers A and B. For this reason, we supplement the real CServ datagrams with active "dummy" probe datagrams in the CServ class to generate the full rate of the Learning Session. We call these *active* probes because router A must proactively form and transmit these datagrams in order to learn the CServ performance metrics. During a learning interval, a predetermined number of CServ datagrams are needed in the Learning Session between router A and router B. The difference between this number and the number of opportunistic real CServ datagrams are the number of active probe datagrams that router A must generate and transmit for the Learning Session during each learning interval.

These active probes are formed to resemble real CServ datagrams and are indistinguishable except that the Source Address in the CServ Internetwork Service header is the address of router A and the Destination Address in the CServ Internetwork Service header is the address of router B with no other hops specified in the CServ Internetwork Service path (i.e. the Path Length field is set to a decimal value of 0). This is because these "dummy" probes are not actually internetwork CServ datagrams. The use of a router as the CServ Internetwork Service header Destination Address signals that these are active probes in a Learning Session that can be discarded by the endpoint router (router B in our nomenclature here). Furthermore, these active probes can carry "dummy" placeholder critical message payloads; the actual CServ message payload contents are never examined or used. The tradeoff between using a "dummy" CServ message payload or not is considered further during the analysis of this protocol. As these active probes are generated and released, router A is responsible for prepending a state measurement service header for Learning Sessions and the appropriate CServ Intranetwork Service header before enqueueing them in the input buffers of the router interfaces.

4.2.3 Measurement Components of the Learning Session

We need to specify the measurement components of the Learning Session protocol in order to fully understand what is captured by the CServ performance metrics reported to the control hierarchy of the CServ architecture. The state measurement protocols report a compact set of statistics, namely the reliability, the mean delay, and the delay variance or standard deviation, for CServ Intranetwork Service; this compact set captures contributions from many different sources of delay and datagram integrity loss as a datagram traverses the path or paths specified by the CServ Intranetwork Service. Identifying

these contributions not only helps motivate the Learning Session protocol details, but it lends insight into the composition of CServ Internetwork Service performance from the performance of the constituent CServ Intranetwork Services. We want to ensure that the design of the Learning Session protocol does not neglect the measurement of any sources of delay or degraded reliability, and we want to be certain that the composition of CServ performance metrics for CServ Internetwork Service paths using the reported CServ Intranetwork Service performance does not omit any delay or integrity degradation contributions.

For discussion, again consider a Learning Session from router A and router B within the subnet of interest, where $A, B \in N_R$. Router A may be an access router and router B may be a gateway router, both routers A and B may be the gateway routers, or router A may be a gateway router and router B may be an access router. Depending on the CServ Intranetwork Service employed by the subnet to connect routers A and B and the topology of the routing core, this Learning Session may traverse multiple intermediate routers in N_R between A and B. We illustrate this scenario in Fig. 4-4 with no intermediate routers between A and B, highlighting the sources of reliability loss and delay along the datagram path. For the purpose of this section, we adopt the terminology of the CServ Intranetwork Service and denote router A as the *Entrance Router* and router B as the *Departure Router*. This terminology stems from the view of CServ Intranetwork Service as an opaque tunnel that chauffeurs the internetwork CServ datagram from one address specified in the CServ Internetwork Service path to the next without MC control over the forwarding mechanism or path used to do so. In the following conversation, we consider the different contributions to the measurement of the reliability and delay values captured by the Learning Session estimation of these CServ Intranetwork Service performance metrics.

When the internetwork CServ datagram arrives without error at the input interface of the Entrance Router, the CSPU determines the appropriate CServ Intranetwork Service based on the subnet's routing and forwarding strategy and the next hop in the datagram's CServ Internetwork Service path. The internetwork CServ datagram is then prepended with a State Measurement Service header for a Learning Session (see Section 4.2.4 for details of the State Measurement Service header) before it is encapsulated as a protocol data unit of the CServ Intranetwork Service using the corresponding CServ Intranetwork Service header. The router's NPU may take further actions to process the arriving datagram after the CSPU. After being processed, the intranetwork CServ datagram is then enqueued in

the input buffer (priority input buffer, if available) of the Entrance Router's input interface. The intranetwork CServ datagram then waits for access to the router's switching fabric. There is one primary source of datagram loss during this phase; during periods of heavy utilization, the input buffer may overflow (or approach overflow, depending on the drop policy of the router) and the datagram may be discarded. There are two primary sources of delay during this phase. First, the initial internetwork CServ datagram processing by the CSPU and NPU introduces a relatively deterministic delay component along the datagram path. Second, the waiting time in the input buffer contributes a random delay component depending on the occupancy of the input buffer and the scheduling of the router's switching plane.

Once the intranetwork CServ datagram is cleared for access to router A's switching fabric, the datagram is dequeued from the interface's input buffer and switched to the appropriate output interface based on the CServ Intranetwork Service header. There is one opportunity for loss during this phase, in general; in the case of imperfect scheduling of the switching plane by the NPU, two datagrams may collide during this process, garbling the contents of the datagram. This is an unlikely opportunity for datagram loss since most routers avoid contention in their switch scheduling algorithm. This phase also contributes a relatively deterministic delay component as the datagram is transmitted between router interfaces via the switching fabric.

After being switched to the appropriate router output interface, the intranetwork CServ datagram is, generally, enqueued once more in the output buffer (priority output buffer, if available) of the Entrance Router's output interface. This buffer meters access to the output's datagram transmission interface. As with the input interface's input buffer, this buffer may overflow during periods of heavy utilization, leading to datagram loss. Additionally, waiting time in the output buffer contributes another random delay component along the datagram path depending on the occupancy of the output buffer and the rate of the transmission interface.

Once the output transmission interface is available for the intranetwork CServ datagram, the intranetwork CServ datagram is transmitted on the interface destined for the Departure Router, router B. (Note here that the intranetwork CServ datagram might be destined for an intermediate router if there are additional hops along the CServ Intranetwork Service path. We use this simple scenario for ease and clarity of discussion.) This datagram transmission may be over fiber optics, radio frequency channels, or other physical data transmission media. The transmission of the datagram between the

routers, at a minimum, incurs additional delay in the form of transmission delay as the data is pushed onto the communication substrate and propagation delay between the routers. These components of delay are generally predictable, deterministic values. The transmission of the intranetwork CServ datagram may also contribute to datagram loss if the data is corrupted in transit (due to interference, multipath fading, dispersion, or atmospheric scintillation, among other actors in data corruption) and cannot be recovered by the physical and data link control layers' integrity check and forward error correction techniques. The processing required by these lower layer protocols are considered part of the data transmission delay; these are often very fast operations implemented in hardware and are not expected to contribute significantly to the estimated delay. We also note that in some cases there may be layer-2 devices used to switch the intranetwork CServ datagram between routers, such as routers A and B in this case. These switches do not process the CServ Intranetwork Service header (or any other CServ header), and we consider their behavior invisible to the CServ architecture. However, any additional contributions to delay or reliability loss made by these lower layer devices would be captured by the Learning Session protocol.

After successful transmission, the intranetwork CServ datagram arrives at the Departure Router at the end of the CServ Intranetwork Service path, or router B in our illustration. This may be after traversing multiple intermediate routers along the CServ Intranetwork Service path, although this is not shown in the example of Fig. 4-4. Once the lower layer protocols confirm that the intranetwork datagram has arrived without error, the datagram is surrendered to the Departure Router's CSPU and NPU for processing (as discussed for the Entrance Router). The delay incurred during this processing is not included in the measurement of the delay by the Learning Session protocol, otherwise this would lead to double-counting of the contribution as these learned performance metrics are composed to find the performance of the end-to-end CServ Internetwork Service path. However, all previously discussed sources of datagram loss and delay are captured by the Learning Session protocol state measurement service. And if there are any intermediate routers between the Entrance Router and the Departure Router, the same four identified phases of reliability degradation and delay would be captured as well by the Learning Session protocol for each intermediate router in the CServ Intranetwork Service path.

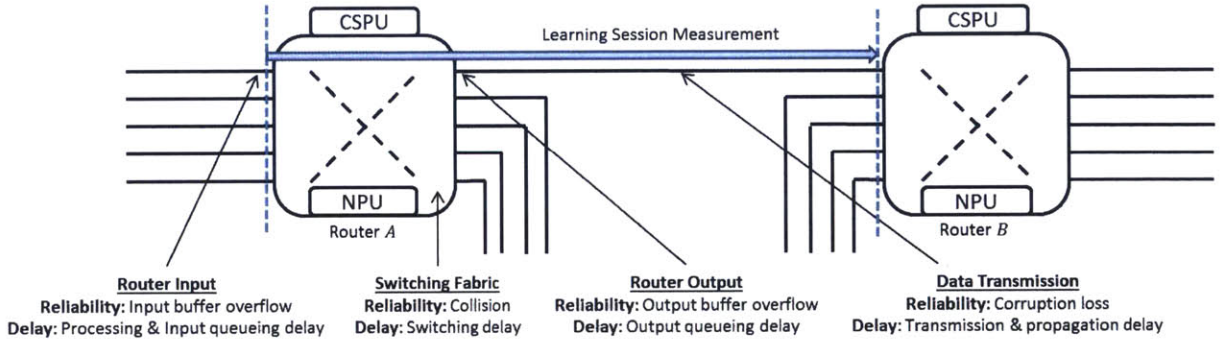


Fig. 4-4: The contributing components of the CServ performance metrics estimated by the Learning Session protocol are shown here for a pair of active routers. If there were an intermediate router along the path, the same four identified phases of datagram loss and delay for that router would be additionally captured by the protocol.

4.2.4 State Measurement Service Header for Learning Sessions

In Chapter 3, a placeholder was included for the use of the State Measurement Service in the stacked CServ header description. Although a bound for the maximum length required for this header was presented, we did not discuss the details of the control information that would be included. We now describe the State Measurement Service header for Learning Sessions in terms of the fields that it includes and the use for these fields.

The State Measurement Service header is prepended to the internetwork CServ datagram when it arrives at an Entrance Router (an active access router or an active gateway router) that determines the appropriate CServ Intranetwork Service for the particular next hop in the CServ Internetwork Service path. The Learning Session protocol prefers the use of real CServ datagrams before the generation and inclusion of active "dummy" probe datagrams in the CServ class to meet the required Learning Session rate. Thus, if a subnet is using the Learning Session protocol to estimate the CServ performance metrics of the CServ Intranetwork Service, then the State Measurement Service header for Learning Sessions is always included when a real CServ datagram traverses the Entrance Router. The exception to this rule is if the rate of real CServ traffic exceeds the Learning Session rate during a learning period; in this case, those datagrams beyond the prearranged number to be transmitted in the Learning Session do not get State Measurement Service headers for Learning Sessions. However, this situation should never occur in the wild. The rate of real CServ datagrams arriving at the Entrance Router and destined for a particular Departure Router in the subnet should be significantly less than the Learning Session rate. When the Learning Session protocol generates active probe datagrams to fill the rate of the Learning Session, these CServ datagrams always contain the State Measurement Service header for Learning Sessions.

The format of the State Measurement Service header is shown in Fig. 4-5 and placed in the context of the intranetwork CServ datagram as discussed in Chapter 3. The following describes the purpose and use for each control field in the State Measurement Service header for Learning Sessions:

- **Version (VER)** – As for all other CServ headers, this leading 4-bit Version field is used to specify the type of State Measurement Service header (in this case, the State Measurement Service header for Learning Sessions). Since this field is based on the IP header field, we employ an unused code (codes for decimal values 1-3 and 10-14 are unallocated) to specify that the

following is a State Measurement Service header for Learning Sessions. This code allows the Departure Router to correctly process the intranetwork CServ datagram when it arrives.

- **Next Header (NH)** – This 4-bit field is used to encode a value specifying the format of the encapsulated datagram header, which is typically the CServ Internetwork Service header. Although there is only one flavor of CServ Internetwork Service header currently, future CServ architecture developments may call for different types of CServ Internetwork Service headers. This field is included for encapsulation flexibility, and it is the joint existence of the Version and Next Header fields in all CServ proprietary header formats that allows for dexterity in stacking different chains of headers together at the CServ network service level.

- **Sequence Number (SN)** – As we discuss shortly, the estimators for CServ performance metrics using Learning Sessions involve counting datagrams and identifying unique datagrams. Since CServ Intranetwork Service may employ diversity path routing (for example, using the centrally-computed explicit path forwarding technique introduced in Section 3.4.3), we do not want to double-count replicated datagrams that are routed over multiple disjoint paths for increased CServ Intranetwork Service reliability. For this reason, active routers using Learning Sessions maintain a set of monotonically increasing sequence numbers, one for each other active router that they establish a Learning Session with in the subnet. Simultaneously, they also maintain a set of sequence numbers that represent the largest value that they have received from each other active router in the subnet. When a passive, real CServ datagram joins a Learning Session (or when an active dummy CServ datagram is generated to fill the rate of a Learning Session), the Entrance Router includes the next sequence number value for the Departure Router of the CServ Intranetwork Service in the Sequence Number field of the header. Furthermore, if the CServ Intranetwork Service specifies that the datagram should be replicated and transmitted over multiple paths, the same value is used for the Sequence Number field in both copies of the datagram. This field allows the Departure Router to both track the number of lost datagrams in the Learning Session and identify duplicates of a datagram that were generated for diversity routing purposes to avoid double-counting. We use a 32-bit Sequence Number field in the header, allowing for over 4 billion unique datagrams before the counter wraps around. Note that this is the same Sequence Number field used in the CServ Intranetwork Service header for

Explicit Path Forwarding as introduced in Section 3.4.3. The Sequence Number is also included here in case a different forwarding and routing strategy is used for CServ Intranetwork Service.

- **Origin Timestamp (OT)** – The Origin Timestamp field is used to calculate the CServ Intranetwork Service delay experienced by an intranetwork CServ datagram in a Learning Session. When the State Measurement Service header for Learning Sessions is generated and prepended to the internetwork CServ datagram by the Entrance Router, the current time value is included in this 64-bit field (the choice of a 64-bit field was previously explained in Section 3.2). The Departure Router of the CServ Intranetwork Service can then identify the realized delay of the intranetwork CServ datagram by finding the difference between the time it arrives at the Departure Router and the value contained in the Origin Timestamp field. Note that this is the same timestamp as in the Origin Timestamp field used in the CServ Intranetwork Service header for Explicit Path Forwarding as introduced in Section 3.4.3. The Origin Timestamp is also included here in case a different forwarding and routing strategy is used for CServ Intranetwork Service and, consequently, a different header without this information.

Using the octet offsets indicated in Fig. 4-5, we see that the State Measurement Service header for Learning Sessions requires a total of 13 bytes of overhead (not the maximum 37 bytes allowed in the analysis of Chapter 3). Additionally, both the Sequence Number and Origin Timestamp fields are redundant if the CServ Intranetwork Service uses the CServ Intranetwork Service header for explicit path forwarding (see Section 3.4.3).

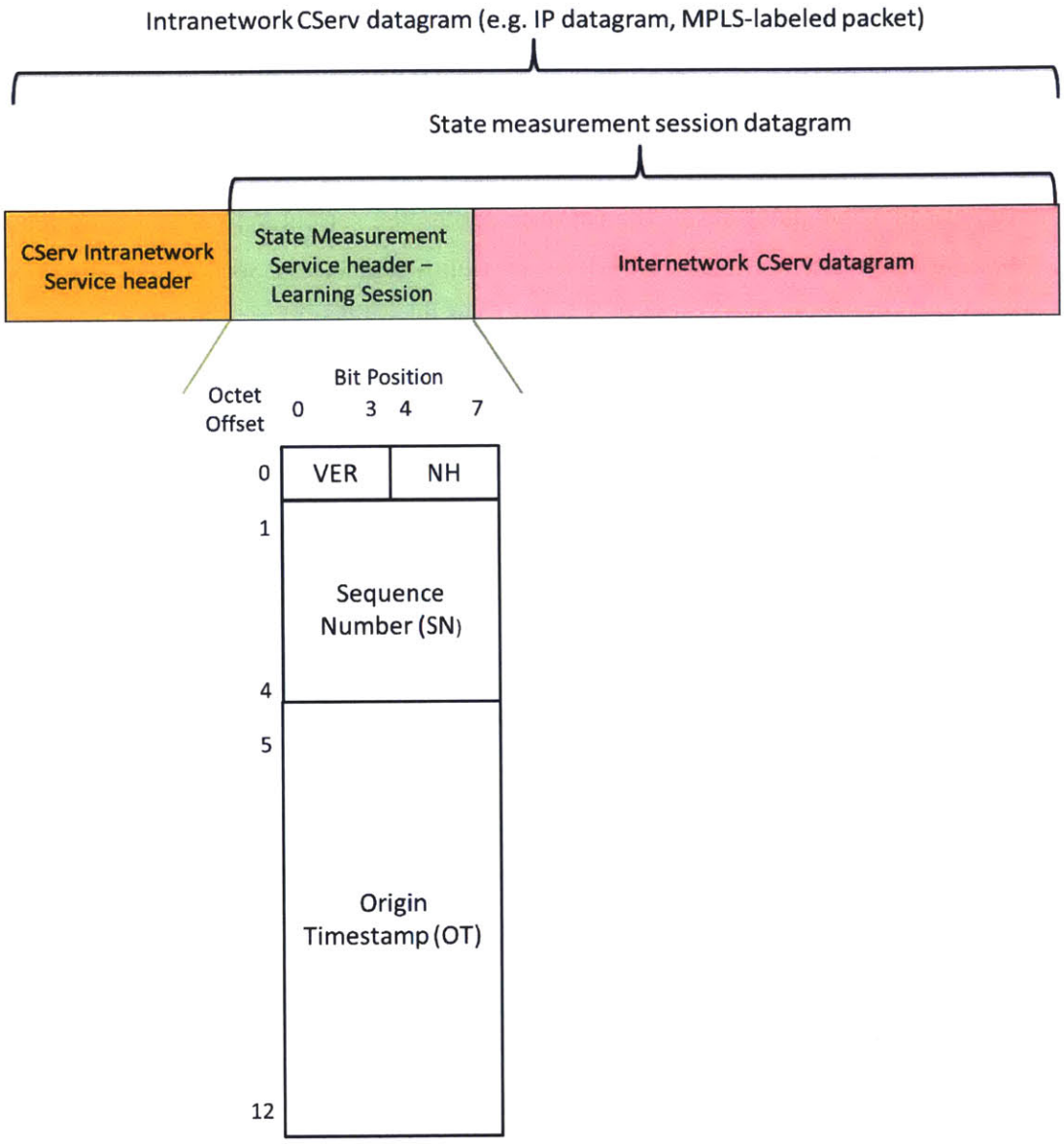


Fig. 4-5: The fixed length format of the State Measurement Service header for Learning Sessions is depicted here, to scale. The use of the Learning Session is encoded in the Version (VER) field.

4.2.5 Learning Session Parameters of Estimation

Before we introduce the estimators used by the Departure Router for the CServ performance metrics, we consider a few of the fundamental parameters of the Learning Session estimation procedure. We have alluded to these parameters in the discussion thus far, but we formalize them here. We then discuss that these parameters are not independent considerations, but rather they are intimately coupled.

The first parameter that we consider is the *Learning Session rate*. We denote this rate as λ_S [datagrams/second]. Active routers in a subnet using the Learning Session protocol to estimate the CServ performance metrics of CServ Intranetwork Service paths maintain fixed rate Learning Sessions at this specified and prearranged rate. Note that we do not quantify the Learning Session rate as a raw data rate in bits per second. This is because passive real CServ datagrams that join a Learning Session are not necessary uniform in length due to varying critical message payload sizes and variable length CServ Internetwork Service headers. As mentioned in Section 4.2.4, the Learning Session rate should be greater than the ingress rate of real CServ traffic arriving at any given active Entrance Router and destined for any other active Departure Router in the subnet that may be reasonably observed. We use the modifier "reasonably" since it is impossible to account for network "Black Swan" events [52] without a process that strictly polices the admission rates of real CServ traffic.

The second parameter that we discuss is the *learning interval*. We denote the learning interval length as τ [seconds]. During each learning interval, an independent estimate of the CServ Intranetwork Service performance metrics are generated by the active routers in the subnet. These intervals are synchronized using the assumption of a synchronized clock for the CServ-enabled network devices (for example, a GPS-based clock).

The third parameter that we consider is the *reliability precision*. Reliability precision indicates how we are able to specify the CServ reliability performance metric. For example, specifying reliability as 0.999 ("three nines") requires finer reliability precision than specifying reliability as 0.99 ("two nines"). We use a proxy variable to denote this parameter. Specifically, we consider the total number of Learning Session datagrams transmitted in a learning interval of length τ as the proxy. We denote this as N_τ [datagrams]. With this proxy variable, we see that we can express reliability with a precision of $1/N_\tau$. For example, we

need $N_\tau = 1000$ to specify three nines reliability while we only need $N_\tau = 100$ to specify two nines reliability.

Based on the parameter definitions, we recognize that $N_\tau = \lambda_S \times \tau$, a relationship that conflates the three Learning Session parameters. In general, we would like to be able to express the reliability of CServ Intranetwork Service with a minimum precision of two nines reliability. Achieving finer precision for estimation of reliability requires either a higher Learning Session rate or a longer learning interval. As we shall observe when we analyze the burden of the Learning Session protocol in Section 4.2.8, higher Learning Session rates place a larger transmission and switching burden on the subnet. This corresponds to degrading the subnet's capability to provide adequate best effort network service, or it requires the subnet to be dimensioned and managed to account for the headroom required of the aggregate Learning Sessions burden. Higher Learning Session rates are also more robust to severe actual CServ traffic peak rates that might result from an unexpected network event. Alternatively, a longer learning interval means that the state measurement protocol is slower to react and estimate changes in the state of the CServ Intranetwork Service due to transient congestion conditions, faltering subnet components, or reconfiguration of the CServ Intranetwork Service paths. That being said, a longer learning interval is not as susceptible to frequent state estimation updates due to transient conditions that are short with respect to the learning interval. This may improve the stability of CServ performance metric estimations from one learning interval to the next. Clearly, there is a careful tradeoff that should be considered between the Learning Session rate and the learning interval length required to estimate reliability with a desired level of precision.

4.2.6 Learning Session CServ Performance Metric Estimators

In this section, we describe the estimators used by the Departure Router for a particular Learning Session before specifying the Learning Session protocol in full. Let us consider again the example of a Learning Session from router A and router B within the subnet of interest, where $A, B \in N_R$. Router A may be an access router and router B may be a gateway router, both routers A and B may be the gateway routers, or router A may be a gateway router and router B may be an access router. Depending on the CServ Intranetwork Service employed by the subnet to connect routers A and B and the topology of the routing core, this Learning Session may traverse multiple intermediate routers in N_R between A and B. Router B computes the CServ performance metrics estimations for this Learning Session, as this

is the Departure Router's responsibility. However, we note here for clarity that there would also be a Learning Session from router B to router A to learn the CServ performance metrics of the CServ Intranetwork Service for the reverse direction of subnet traversal – we do not assume that the CServ Intranetwork Service paths are equivalent or that the CServ performance metrics are reciprocal.

First, let's consider the estimator used for the *reliability* CServ performance metric. We use the naïve estimator that considers the ratio of the number of datagrams that arrive successfully in a learning interval to the number of datagrams that should arrive based on the Learning Session rate and the learning interval length. Abstractly, this means that router B computes the reliability as follows:

$$\text{Reliability Estimate} \triangleq \frac{\text{Count of CServ datagrams received successfully in interval}}{\text{Number of CServ datagrams that } \textit{should} \text{ be received in interval}}.$$

We formalize this estimator with the following notation. Let τ_i, $i \in \mathbb{Z}$ represent an indexed learning interval instance of length τ. Let $\hat{\rho}_{\tau_i}^{A \to B}$ denote the reliability estimate for a particular interval τ_i for the CServ Intranetwork Service from active router A to active router B. Let $X_{\tau_i}^{A \to B}$ represent the realized count of successfully received intranetwork CServ datagrams from active Entrance Router A to active Departure Router B for a particular learning interval τ_i. This is the count of datagrams in the Learning Session that arrive successfully at their local destination, router B, while discounting any repetitions from diversity routing using the Sequence Number field in the State Measurement Service header for Learning Sessions and a stateful counter at router B. As established in Section 4.2.5, N_τ denotes the total number of Learning Session datagrams transmitted in a learning interval of length τ. Equivalently, N_τ is the total number of intranetwork CServ datagrams that should be received by the Departure Router in a particular Learning Session during a learning interval of length τ. In Section 4.2.5, we explained that $N_\tau = \lambda_S \times \tau$. Since both the Learning Session rate and learning interval length are uniform for all Learning Sessions within a subnet, N_τ is also a constant value for all Learning Sessions within a particular subnet. Using this notation, we can express the reliability estimator for learning interval τ_i as:

$$\hat{\rho}_{\tau_i}^{A \to B} \triangleq \frac{X_{\tau_i}^{A \to B}}{N_\tau}. \tag{4.1}$$

Second, we discuss the estimators for the delay CServ performance metrics. As previously addressed, we do not consider only the mean delay for the CServ Intranetwork Service, but also the delay variance (or standard deviation). Similar work in the area of providing inter-domain service performance guarantees has typically leveraged only the mean delay values [22]. With the end-to-end mean delay alone, we have no power to characterize the spread of the distribution. However, with the inclusion of the delay variance, we can generate end-to-end delay bounds even though we do not know the details of the delay distributions. The use of the delay statistics is considered in much more detail in Chapter 5.

Recall that the State Measurement Service header for Learning Sessions has a field for the Origin Timestamp (see Section 4.2.4). When the internetwork CServ datagram becomes part of a Learning Session at the Entrance Router (router A in our example) or an active dummy CServ datagram is generated to fill the Learning Session rate at the Entrance Router, the State Measurement Service header is marked with the current timestamp. When the CServ datagrams in the Learning Session arrive at the Departure Router (router B in our example), the CSPU finds the difference between the current time and the time in the Origin Timestamp field of the State Measurement Service header. This realized delay for each errorless CServ datagram in the Learning Session is used to compute a running estimate of both the *mean delay* and *delay variance* statistics during each learning interval of length τ. This use of a fresh, independent estimate for each learning interval period limits the effect of any transient conditions that might induce anomalous estimation outliers during otherwise nominal subnet conditions.

We now proceed to formalize these delay statistic estimators, starting with the mean delay. We create a running estimate for the mean delay that can be updated iteratively upon the arrival of each CServ datagram in the Learning Session during a particular learning interval. Consider the arrival of the n^{th} intranetwork CServ datagram in the Learning Session from router A to router B during a particular indexed learning interval τ_i. We define the delay of this CServ datagram, the difference between the current time and the value of the Origin Timestamp in the datagram's State Measurement Service header, as $d_{n,\tau_i}^{A\to B}$. We denote the current mean estimate as $\hat{\mu}_{n-1}^{A\to B}$ and update this estimate upon the arrival of the n^{th} intranetwork CServ datagram in the Learning Session with a cumulative moving average as follows:

$$\hat{\mu}_n^{A\to B} = \frac{1}{n}\left[(n-1)\hat{\mu}_{n-1}^{A\to B} + d_{n,\tau_i}^{A\to B}\right], n \in \mathbb{Z}^+. \tag{4.2}$$

We arbitrarily define $\hat{\mu}_0^{A\to B} \triangleq 0$, although we note that this does not impact Eq. (4.2) thanks to the $(n-1)$ term during the calculation of $\hat{\mu}_1^{A\to B}$ after the arrival of the first CServ datagram during the learning interval. At the end of the learning interval τ_i, the final estimate of the mean delay is $\hat{\mu}_{X_{\tau_i}^{A\to B}}^{A\to B}$. As previously mentioned, this mean delay estimate starts from scratch with $\hat{\mu}_0^{A\to B} = 0$ during the next learning interval.

The creation of the delay variance estimate proceeds in a similar fashion to that of the mean delay. We also utilize a running estimate for the delay variance that can be updated iteratively upon the arrival of each CServ datagram in the Learning Session during a particular learning interval. We again consider the arrival of the n^{th} intranetwork CServ datagram in the Learning Session from router A to router B during a particular indexed learning interval τ_i and use the definition of the delay of this particular CServ datagram as before. The iterative update of the delay variance must occur after the update of the mean delay. We denote the current mean estimate as $\hat{\mu}_n^{A\to B}$, the previous mean estimate as $\hat{\mu}_{n-1}^{A\to B}$, and the current variance estimate as $\widehat{\sigma^2}_{n-1}^{A\to B}$. Upon the arrival of the n^{th} intranetwork CServ datagram in the Learning Session, we update the delay variance estimate using an online estimator of population variance as follows:

$$\widehat{\sigma^2}_n^{A\to B} = \frac{1}{n}\left[(n-1)\widehat{\sigma^2}_{n-1}^{A\to B} + (d_{n,\tau_i}^{A\to B} - \hat{\mu}_{n-1}^{A\to B})(d_{n,\tau_i}^{A\to B} - \hat{\mu}_n^{A\to B})\right], n \in \mathbb{Z}^+. \tag{4.3}$$

We define $\widehat{\sigma^2}_0^{A\to B} \triangleq 0$, noting that this value does not impact Eq. (4.3) due to the $(n-1)$ term during the calculation of $\widehat{\sigma^2}_1^{A\to B}$ after the arrival of the first CServ datagram during the learning interval. At the end of the learning interval τ_i, the final estimate of the delay variance is $\widehat{\sigma^2}_{X_{\tau_i}^{A\to B}}^{A\to B}$. Just as with the estimate of the mean delay, this estimator starts from scratch with $\widehat{\sigma^2}_0^{A\to B} = 0$ at the start of each new learning interval.

Before we pull everything together into the Learning Session protocol description, we briefly consider how we can mitigate a situation which could artificially inflate the estimated reliability value and bias

the delay estimates. Since CServ Intranetwork Service specifies the routing and forwarding of an intranetwork CServ datagram in a Learning Session, we do not generally need to worry about out-of-order delivery. In some transient scenarios, we may find anomalous CServ performance metric estimations as a CServ Intranetwork Service path changes, but this should be mitigated by the estimation of the next learning interval. However, consider the case where the CServ Intranetwork Service path or paths do not change, but an anomalous subnet event induces unexpected CServ congestion at some intermediate router that holds up the forwarding of many datagrams in a particular Learning Session. After the anomalous "Black Swan" congestion event, these datagrams may be artificially grouped and forwarded together, skewing the reliability, mean delay, and delay variance estimations for the current learning interval. One option is to accept this possible behavior as inevitable and allow for the transient effect to die out, correcting the estimations in the next learning interval. Alternatively, we could allow the subnet administrator to set an upper limit to acceptable CServ datagram delay using CServ Intranetwork Service. If the calculated CServ Intranetwork Service traversal time of the intranetwork CServ datagram in a Learning Session exceeds this value, the datagram is dropped and its delay is not included in the current running estimates of mean delay or delay variance. The issue with this approach lies in correcting tuning the delay upper limit in the subnet and the potential to drop CServ datagrams unnecessarily. The concern of CServ performance metric estimation outliers and related methods to improve estimator stability require additional attention in future research.

4.2.7 Learning Session Protocol Description

With the fundamentals of Learning Sessions under our belt, we are prepared to describe the Learning Session protocol that serves as the State Measurement Service for the subnet, learning the CServ performance metrics that capture the behavior of the CServ Intranetwork Service. This protocol is illustrated in Fig. 4-6 (with no intermediate routers between the Entrance Router and Departure Router for the Learning Session). The following describes the protocol responsibilities for both the Entrance Router and Departure Router of the Learning Session, pulling together the fundamentals from Sections 4.2.1-4.2.6. For the simplicity of discussion, we assume that all subnet routers in N_R are active CServ routers. If there are non-active routers in the subnet's routing core, they should be excluded from the following discussion since non-active routers do not generate or sink Learning Sessions with their subnet peers.

The following protocol applies for every Router $i \in N_R$ and every Router $j \in N_R \backslash \{i\}$. We use the convention that Router i is the Entrance Router and Router j is the Departure Router for the Learning Session.

Entrance Router (Router i) CSPU

- Create and maintain a CSPU process for a Learning Session with each Router $j \in N_R \backslash \{i\}$ using the address set N_R distributed by the SC
    - Terminate any existing Learning Session CSPU process with Router k if it is removed on latest update of address set N_R distributed by the SC (i.e. $k \notin N_R$)
- Maintain a 32-bit incrementing, wrap-around counter for Router j that is used as a sequence number
- Ensure that the rate of CServ datagrams destined for Router j and part of the Learning Session is equal to λ_S
    - Stagger creation and transmission of active dummy CServ datagrams destined for Router j to realize this rate, allowing the transmission of an arriving passive CServ datagram to preempt the transmission of the next active dummy CServ datagram in this process
    - Do not allow the rate of datagrams destined for Router j and part of the Learning Session to exceed λ_S (If the rate of arriving passive CServ datagrams destined for Router j exceeds λ_S, passive datagrams in excess of this rate should not be included as part of the Learning Session and do not get the State Measurement Service header)
- Upon arrival of a passive CServ datagram with Router j as the next hop in the CServ Internetwork Service path:
    - Prepend a State Measurement Service header for the Learning Session and set the fields as follows:
        - Version – Use the encoding that represents a State Measurement Service header for a Learning Session
        - Next Header – Use the appropriate encoding (this code represents the CServ Internetwork Service header for internetwork CServ datagrams)
        - Sequence Number – Use the next value of the incrementing counter for Router j (and increment the counter state for the CSPU process)

        - Origin Timestamp – Use the current time
    - If the appropriate CServ Intranetwork Service for Departure Router j specifies the use of diversity routing over multiple CServ Intranetwork Service paths, replicate the state measurement session datagram the appropriate number of times prior to prepending the appropriate CServ Intranetwork Service headers for each
        - Note that this follows the design that each replication should have the same values encoded in the Sequence Number fields
    - Prepend the appropriate CServ Intranetwork Service header based on Router j (the next hop in the CServ Internetwork Service path) to each replication (if any)
    - Enqueue the passive CServ datagram (or datagrams) in the input buffer of the arriving interface (using the CServ class priority queue if available)
- Upon creation and transmission of an active dummy CServ datagram destined for Router j to realize the Learning Session rate λ_S:
    - Generate a dummy internetwork CServ datagram that mimics a real passive internetwork CServ datagram with the following specifications:
        - Version – Use the encoding that specifies the CServ Internetwork Service header consistent with a real passive internetwork CServ datagram
        - Next Header – Use the default Next Header code that indicates that the capsulated payload is the CServ message data (which is a dummy placeholder in this case)
        - Path Length – Encode the decimal value 0 in this field (there is no real CServ Internetwork Service path since this is a dummy internetwork CServ datagram used only for the Learning Session)
        - Payload/Data Length – Use the correct value to describe the size of the dummy CServ message data payload in bytes (default is one kilobyte, but this can vary between the implementation depending on the needs and capabilities of the subnet)
        - Checksum – This field is unnecessary and can be set to all zeros since the CServ Internetwork Service header integrity check is only used at real CServ transaction endpoint hosts and the integrity check for CServ Intranetwork Service is either at the data link layer and below or included as part of the CServ Intranetwork Service protocol

- Source Address – Use the address of Router *i*, the Entrance Router for the CServ Intranetwork Service and the "source" of the Learning Session
- Destination Address – Use the address of Router *j*, the Departure Router for the CServ Intranetwork Service and the "destination" of the Learning Session (note that it is the use of the Learning Session endpoints as the Source and Destination Addresses, the lack of a CServ Internetwork Service path, and the default dummy CServ data payload that indicate to Router *j* that this is an active dummy probe in the Learning Session and not a real passive internetwork CServ datagram)
- Sequence Number – Use some subnet-specific default value in this field to further indicate that this is a dummy active probe in the Learning Session and not a real passive CServ datagram
- Expiration Time – Use some default value since this field is not used by the Departure Router *j*
- Authentication Mark – For now, use some default value since this field is not used by the Departure Router *j* (however, if necessary, this could be employed in the future to authenticate the active dummy probe in the Learning Session as a legitimate probe generated by Entrance Router *i*)
- Hop Addresses – There are no additional hop addresses encoded since there is no CServ Internetwork Service path for active dummy probes in the Learning Session and the value encoded in the Path Length field is zero
- CServ Data Payload – Fill the payload with the appropriate number of bytes encoded in the Payload/Data Length field (typically all zeros, but it depends on the subnet's implementation of the Learning Session protocol)

- Prepend a State Measurement Service header for the Learning Session, just as for real passive CServ datagrams, and set the fields as follows:
  - Version – Use the encoding that represents a State Measurement Service header for a Learning Session
  - Next Header – Use the encoding that represents the CServ Internetwork Service header for internetwork CServ datagrams
  - Sequence Number – Use the next value of the incrementing counter for Router *j* (and increment the counter state for the CSPU process)

        - Origin Timestamp – Use the current time
    - If the appropriate CServ Intranetwork Service for Departure Router j specifies the use of diversity routing over multiple CServ Intranetwork Service paths, replicate the state measurement session datagram the appropriate number of times prior to prepending the appropriate CServ Intranetwork Service headers for each
        - Note that this follows the design that each replication should have the same values encoded in the Sequence Number fields
    - Prepend the appropriate CServ Intranetwork Service header (to each replication, if applicable) based on Departure Router j, the other endpoint for the Learning Session
    - Enqueue the active dummy CServ datagram (or datagrams) in the input buffer of a *random* interface (using the CServ class priority queue if available) since there is no "arriving interface"
        - This choice should be selected independently for each transmitted active dummy CServ datagram and uniformly over all possible Entrance Router i interface input buffers to avoid any interface-specific measurement dependencies over the Learning Session

Departure Router (Router j) CSPU

- Create and maintain a CSPU process for a Learning Session with each Router $i \in N_R \backslash \{j\}$ using the address set N_R distributed by the SC
    - Terminate any existing Learning Session CSPU process with Router k if it is removed on latest update of address set N_R distributed by the SC (i.e. $k \notin N_R$)
- Maintain a 32-bit incrementing, wrap-around counter for Router i that reflects the largest Sequence Number value seen in the State Measurement Service headers of arriving CServ datagrams in the Learning Session for Entrance Router i
- Maintain values for the last reported CServ performance metric estimates for each Entrance Router i in the subnet
- Upon the successful arrival of an intranetwork CServ datagram with Router j as the Departure Router (or destination router) for the CServ Intranetwork Service, perform the following steps:

- If diversity routing is enabled for the CServ Intranetwork Service, verify that the Sequence Number in the CServ Intranetwork Service header or State Measurement header is greater than the value of the counter for Router i
    - If it is, this is a unique CServ datagram and the counter should be updated to reflect its Sequence Number
    - If it is not, it is a replicated datagram that has already been received and should be discarded to quench diversity replication without updating the counter
- Check the State Measurement Service header for Learning Sessions to verify if the datagram is part of the Learning Session with the CServ Intranetwork Service Entrance Router (or source router) i
    - Unless the rate of real passive internetwork CServ datagrams arriving at Entrance Router i destined for Departure Router j exceeds λ_S, which it should not by design in all but the most extreme conditions, all CServ datagrams should be part of the Learning Session
- If the datagram is part of the Learning Session:
    - Increment the counter for unique CServ datagrams received during the current learning interval
    - Extract the Origin Timestamp value in the State Measurement Service header to calculate the delay of the datagram as the difference between the current time and that extracted value
    - Remove the leading CServ Intranetwork Service header and State Measurement Service header before continuing processing of the encapsulated internetwork CServ datagram
    - If the encapsulated internetwork CServ datagram indicates that it is an active dummy probe datagram, discard the datagram; otherwise, process the CServ Internetwork Service header and use the next hop in the CServ Internetwork Service path to forward the passive real CServ datagram appropriately
- If the datagram is not part of the Learning Session:
    - Remove the leading CServ Intranetwork Service header
    - The encapsulated CServ datagram cannot be an active dummy probe datagram because it is not part of the Learning Session, so process the CServ Internetwork

Service header and forward the real internetwork CServ datagram appropriately based on the next hop in the CServ Internetwork Service path

- During a particular learning interval indexed τ_i:
  - Maintain the count of unique CServ datagrams in the Learning Session with each Entrance Router i that arrive successfully (i.e. after verifying the integrity of each arriving CServ datagram using the link layer integrity check or the CServ Intranetwork Service integrity check if it exists), which generates the value $X_{\tau_i}^{i \to j}$ at the end of the learning interval
  - For each CServ datagram in the Learning Session with each Entrance Router i that arrives successfully, use the calculated datagram delay to update the online cumulative mean delay estimate and the delay variance estimate
- At the end of each learning interval of length τ:
  - Generate the CServ performance metric estimate for reliability and finalize the CServ performance metric estimates for mean delay and delay variance for the Learning Session with each Entrance Router i using the appropriate estimators (specifically, those described in Eq. (4.1)-(4.3) of Section 4.2.6)
  - Report the new CServ performance metric estimates for the Learning Session with each Entrance Router i to the SC if the new values are different enough from the last reported values (see further discussion of this in Section 4.6)
    - Update the last reported CServ performance metrics for the Learning Session with Router i if the new estimates are reported
  - Reset the count of unique received CServ datagrams and zero out the running estimates of the mean delay and delay variance for the Learning Session with each Router i

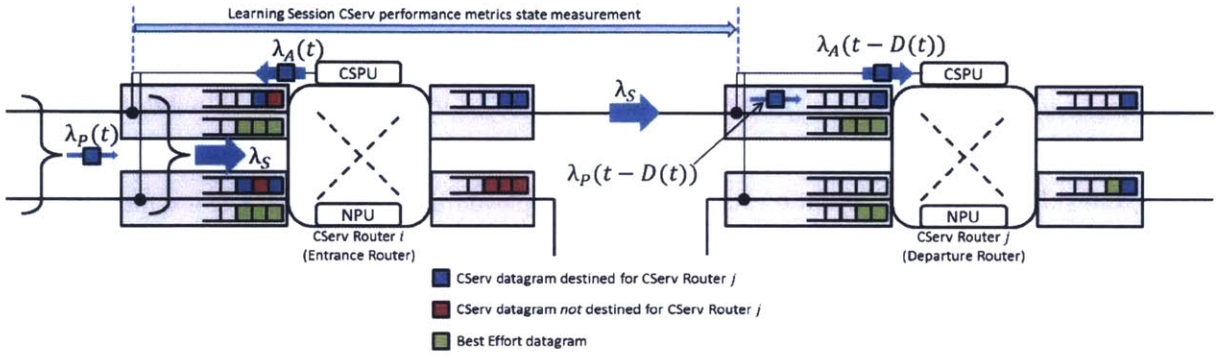


Fig. 4-6: This illustration depicts the Learning Session protocol used to estimate the CServ performance metrics for CServ Intranetwork Service from active Entrance Router i to active Departure Router j. The variable rate of passive real CServ datagrams traversing Entrance Router i en route to Departure Router j is denoted notionally as $\lambda_P(t)$, while the variable rate of active dummy CServ datagrams that the CSPU at Entrance Router i must generate for this Learning Session is denoted notionally as $\lambda_A(t)$. These rates are, in general, time varying and must together satisfy the relationship $\lambda_P(t) + \lambda_A(t) = \lambda_S$, where λ_S is the Learning Session rate. The random process $D(t)$ represents the unknown stochastic delay of the CServ Intranetwork Service path between Entrance Router i and Departure Router j and encapsulates the delay components illustrated in Fig. 4-4. The thickness of the illustrated arrows notionally represents the contributions of active CServ traffic compared to passive CServ traffic in the Learning Session and assumes no datagram loss. Note that priority queueing for CServ datagrams is depicted only for the input interfaces of these routers; the output interfaces show a merged-class buffer. However, in general, priority queueing can be implemented both for input and output router interfaces.

4.2.8 Analysis of the Learning Session Protocol Subnet Burden

Since the Learning Session protocol establishes a fixed-rate session between each pair of active CServ-enabled routers, both access routers and gateway routers, in the subnet, this approach to estimating the CServ performance metrics imposes a heavy burden on the subnet data plane. It is necessary to characterize the data plane overhead since the subnet must be capable of supporting this CServ traffic while maintaining ample headroom for best effort traffic. Otherwise, the Learning Session protocol burden itself inhibits the ability of the subnet to support both network services simultaneously. The Learning Session protocol burden is intimately tied to the number of active CServ-enabled routers in the subnet, the subnet topology, and the routing strategy employed by the subnet for CServ Intranetwork Service. In this section, we consider and analyze the Learning Session burden for the two subnet topologies introduced in Section 4.1, the Petersen graph network and the line network. These topologies serve as representative "lower" and "upper" bounds, respectively, for subnet routing core topologies with bounded degree.

Before discussing the two subnet topologies, we reiterate the three important assumptions made for the tractability of the analysis, first presented in Section 4.1. For the purposes of analysis, we assume the following in this section:

1. All routers in the subnet routing core topology are either active CServ-enabled access routers or active CServ-enabled gateway routers. Thus, all routers in the subnet routing core topology participate in the Learning Session State Measurement Service protocol.

2. The routing scheme used for the purpose of analysis is that of shortest-path routing, where the shortest path is defined by the fewest number of intermediate hops in the path. This assumption is necessary for the analysis of the Petersen graph subnet topology, but not for the line network subnet topology (which presents a degenerate case with no routing decisions).

3. The CServ Intranetwork Service does not employ path diversity between Entrance and Departure routers. There is only one unique CServ Intranetwork Service path connecting any pair of active routers in the subnet routing core as described by the second assumption.

4.2.8.1 Learning Session Burden: The Petersen Graph

We consider the Petersen graph for the subnet routing core topology as a form of "lower" bound for the analysis of the Learning Session protocol data plane burden, as discussed previously in Section 4.1. The Petersen graph topology is shown in Fig. 4-2, where we assume that all vertices in the graph are active CServ-enabled routers participating in the Learning Session protocol. The router numbering (addressing) is arbitrary, as is the distinction between active access routers and active gateway routers, for the purpose of this analysis of the data plane burden in the routing core topology. The Petersen graph routing core topology has 10 routers, so $n_R = 10$ using the notation introduced in Section 4.1. The total number of logical directed links in the topology is 30, where two logical directed links are used to realize each logical undirected link shown in the figure. The degree of each router is 3 in the nomenclature of the undirected graph and as depicted in Fig. 4-2 (the equivalent digraph indegree is 3 and the digraph outdegree is also 3). Finally, the Petersen graph has a graph diameter of 2, which we see when we next consider the form of the shortest-path trees in the topology.

Consider a shortest-path tree rooted at Router 1, as illustrated in Fig. 4-7. The links connecting Router 1 to all other routers in the routing core topology are given an orientation for this shortest-path tree (even though they are bidirectional links in the Petersen graph network), and the dotted blue links are unused for this particular, but unique, shortest-path tree for Router 1. All rooted shortest-path trees have the exact same structure if we consider the other routers as the roots due to the strong regularity of the Petersen graph.

Now consider the nine Learning Sessions generated by Router 1 for all other nine routers in the routing core topology. Using the shortest-path tree rooted at Router 1 shown in Fig. 4-7, we see that one-third of these Learning Sessions generate one hop worth of traffic (those sessions with Departure Routers 2, 5, and 6). The other two-thirds of the Learning Sessions created by Router 1 generate two hops worth of traffic (those sessions with Departure Routers 3, 4, 7, 8, 9, and 10). The $n_R(n_R - 1) = 90$ Learning Sessions correspond to the following total hops worth of Learning Session traffic, which we call "traffic units" in this argument, since one-third of them generate one hop worth of traffic while the remaining two-thirds generate two hops worth of traffic:

$$\left(\frac{1}{3} \times 90 \text{ Learning Sessions } \times 1 \frac{\text{Traffic unit}}{\text{Learning Session}}\right)$$
$$+ \left(\frac{2}{3} \times 90 \text{ Learning Sessions } \times 2 \frac{\text{Traffic units}}{\text{Learning Session}}\right)$$
$$= 150 \text{ Traffic units.}$$

Furthermore, by the strong regularity of the topology, the 90 total Learning Sessions for all 10 active routers in the subnet routing core are distributed uniformly over the 30 logical directed links. Thus, we have that each logical directed link bears the burden of 5 traffic units worth of the total Learning Session traffic.

For generality, we define the average number of Learning Session traffic units per logical directed link in the Petersen graph topology as $C^{Petersen}$. We note that this is not actually an average in the case of the Petersen graph topology, but the use of the average nomenclature is useful in the analysis of the line graph network. As we have just shown, $C^{Petersen} = 5$. Similarly, we define the average data rate of aggregate Learning Session traffic carried per logical directed link in the Petersen graph topology as $C_{r-s}^{Petersen}$. Again, this is an exact value and not actually an average in the case of the Petersen graph, but the terminology is useful in the next section when we discuss the analysis of the line graph subnet routing core topology. For the purpose of analysis, we assume a uniform size for all CServ datagrams in the Learning Session (regardless of whether or not each datagram is a real, passive internetwork CServ datagram or an active, dummy probe CServ datagram), and we define the size of the CServ datagram in the Learning Session as b_S [bits/datagram]. With the Learning Session rate λ_S [datagrams/second], we can further define the data rate of the Learning Session as R_S [bits/second], where we have that:

$$R_S = \lambda_S \times b_S.$$

As previously discussed in Section 4.2.5, λ_S and the learning interval duration τ jointly determine how precisely the Learning Session protocol can estimate a value for the reliability CServ performance metric. Specifically, this precision is dependent upon the number of datagrams transmitted in a Learning Session in a learning interval of length τ, which was given previously as $N_\tau = \lambda_S \times \tau$. With this, we can further specify the data rate of the Learning Session as follows:

$$R_S = \lambda_S \times b_S = \frac{N_\tau}{\tau} \times b_S. \tag{4.4}$$

With these definitions, we have that the aggregate Learning Session data rate that traverses each logical directed link in a Petersen graph routing core topology is:

$$\begin{aligned} C_{r-s}^{Petersen} &= C^{Petersen} \times R_S \\ &= C^{Petersen} \times \left(\frac{N_\tau}{\tau} \times b_S\right) \\ &= \frac{5N_\tau b_S}{\tau}. \end{aligned} \tag{4.5}$$

To get a sense of the aggregate traffic traversing each link, or the data plane transmission burden, we need reasonable values for b_S, N_τ, and τ. Considering the analysis of Chapter 3, we use an approximate value of $b_S = 12{,}000$ [bits], allowing one kilobyte for the CServ message payload and a half-kilobyte for the total header control information, including the CServ Internetwork Service header, the State Measurement Service header for Learning Sessions, and the specific CServ Intranetwork Service header used by the subnet. This choice is also aligned with a typical network maximum transmission unit, as considered in Chapter 3. The choice of N_τ depends on the precision at which we wish to estimate the reliability CServ performance metric. At a minimum, we would like to have two-nine precision, meaning that we need $N_\tau = 100$. (Ideally, we would like finer precision, but we see that this minimal requirement already strains the subnet data plane.) And the choice of τ can be viewed as the measurement granularity with which the protocol can react to a change in subnet state since a new set of CServ performance metrics can only be estimated after τ seconds. In general, we would like this parameter of the Learning Session protocol to be at least on the order of the subnet's "coherence time," or the average time between significant subnet state changes. For the purpose of this analysis, we consider $\tau = 1$ second. A smaller learning interval allows for quicker reactions to a change in state, but it may also trigger state measurement oscillation or instability due to short transients. With these example parameter choices (and approximate Learning Session datagram size) and the expression derived in Eq. (4.5), we have:

$$C_{r-s}^{Petersen} = \frac{5 \times 100 \times 12{,}000}{1} = 6 \text{ Mbps}.$$

While this may be a reasonable state measurement protocol burden for multi-gigabit fiber links, this data plane transmission burden for the Learning Session protocol may be too great for edge wireless links in challenged environments where the CServ architecture is most likely to be used, even with this optimistic subnet routing core topology. Furthermore, this transmission burden increases by an order of magnitude if we either want to capability to estimate three-nine reliability (i.e. $N_\tau = 1000$) or increase the frequency of CServ performance metric estimation such that $\tau = 100$ ms.

We can also consider the data plane switching burden of the Learning Session protocol. Specifically, this is the aggregate rate of Learning Session traffic that each router in the subnet routing core topology must switch to maintain the protocol sessions. If we consider one router in isolation, say Router 1 in Fig. 4-2, we know that there is a total of 15 traffic units worth of Learning Session traffic arriving on its incoming logical edges (there are three directed links incident at Router 1, and each bears $C^{Petersen} = 5$ traffic units of Learning Session traffic). Nine of these traffic units are Learning Sessions destined for Router 1 from the other routers in the subnet routing core topology, and these traffic units do not need to be switched by Router 1 according to the protocol presented in Section 4.2.7. However, Router 1 also generates nine Learning Sessions, or nine traffic units of Learning Session traffic, for the other nine routers in the subnet routing core topology, and these must be switched through Router 1's switching fabric. Thus, we can conclude that a total of 15 traffic units worth of Learning Session traffic need to be switched by Router 1. Due to the strong regularity of the Petersen graph topology, the same scenario holds for all routers in the topology. We can use the Learning Session data rate (using the same parameters as before: $b_S = 12{,}000$, $N_\tau = 100$, and $\tau = 1$) to conclude that the switching throughput for each router must be 18 Mbps just to support the Learning Session State Measurement Service protocol.

The overall router switching bandwidth must be significantly higher to support additional best effort traffic. In general, we would like the Learning Session traffic to take up no more than 10% of the switching or transmission capacity, which implies that we would need active routers with switching throughput of at least 180 Mbps. This requirement may stress the capabilities of all but some of the higher-end router designs [76]. Additionally, the logical links in the subnet with a Petersen graph routing core topology would need to be designed to support at least 60 Mbps according to this guideline.

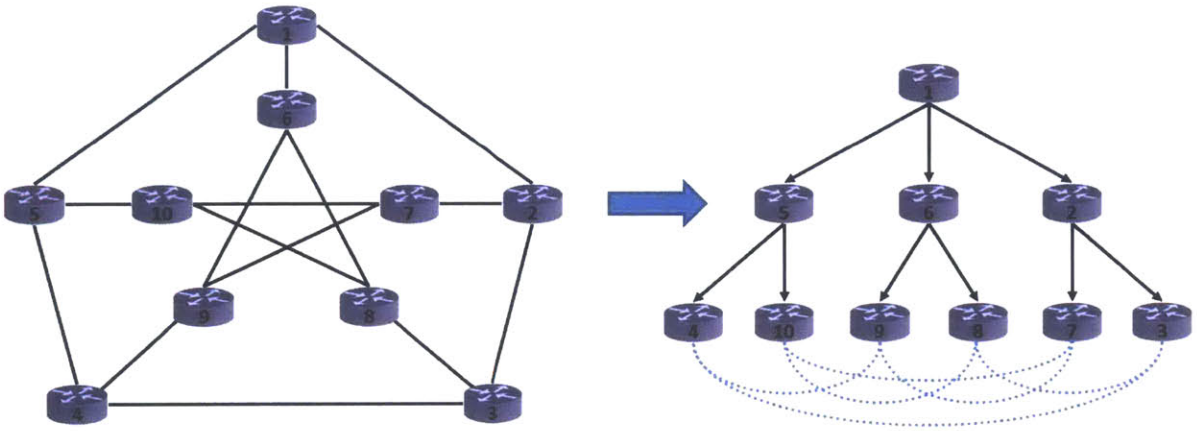


Fig. 4-7: A rooted shortest-path tree is shown for Router 1, where the shortest path is defined as the path with the least number of intermediate hops. The dotted blue links are unused by the shortest-path tree for Router 1.

4.2.8.2 Learning Session Burden: The Line Network Graph

As an "upper" bound to contrast to the optimistic analysis of the Petersen graph, we select the line network graph with $n_R = 10$ for the routing core topology. This example forms an "upper" bound in that it maximizes the average number of hops between vertices, and thus it maximizes the average number of links that the Learning Sessions traverse. The subnet line network topology is depicted in Fig. 4-1, where we assume that all vertices in the graph are active CServ-enabled routers participating in the Learning Session protocol. For the purpose of this analysis of data plane burden in the routing core topology, the router addressing is arbitrary, as is the distinction between active access routers and active gateway routers in the figure. The total number of logical directed links in the topology is 18, where two logical directed links are used to realize each logical undirected link shown in Fig. 4-1. The degree of all routers except Router 1 and Router 10 is 2 (the equivalent digraph indegree is 2 and the digraph outdegree is also 2), while the degree of the other two routers at the ends of the subnet line routing core topology is 1 (the equivalent digraph indegree is and the digraph outdegree is also 1). Finally, the line network has a graph diameter of 9 (consider the path from Router 1 to Router 10).

We note that there are no routing decisions for the line graph. There is one unique path connecting any pair of routers in this subnet routing core topology. However, we see that the Learning Session from Entrance Router 1 to Departure Router 10 traverses 9 logical links between routers, whereas no Learning Sessions generated by Router 5 traverse that many. Unlike the Petersen graph topology, the line network graph does not exhibit the same regularity properties.

If we consider the distribution of the 90 Learning Sessions over the logical directed links in the topology, we see the distribution illustrated in Fig. 4-8. The logical links between Router 5 and Router 6 bear the heaviest burden in terms of the aggregate Learning Session traffic, each carrying 25 Learning Sessions, because they serve as the connection between the left and right "halves" of the topology. Counting up the distribution of the Learning Session load in the figure, we see that the 90 Learning Sessions correspond to 330 hops worth of Learning Session traffic, or traffic units. Thus we note that the average Learning Session traverses 11/3 hops and thus generates 11/3 traffic units worth of burden on the line graph network routing core topology.

Using the nomenclature introduced in Section 4.2.8.1, we define the average number of Learning Session traffic units per logical directed link in the line network graph topology as C^{Line}. Unlike in the case of the Petersen graph routing core topology, this is actually an average value due to the lack of uniformity in the distribution of the Learning Sessions over the links of the topology. Averaging over the number of Learning Sessions traversing each link, we have that $C^{Line} = 55/3$. We additionally consider the worst-case links, the links between Router 5 and Router 6. We define the maximum number of Learning Session traffic units per logical directed link in the line network graph topology as $C^{Line-max}$, and we have that $C^{Line-max} = 25$. As before, we can denote the equivalent average and maximum data rate for the aggregate Learning Session traffic carried per logical directed link in the line network graph topology as C_{r-s}^{Line} and $C_{r-s}^{Line-max}$, respectively.

Following the same analysis as in Section 4.2.8.1, we have that the average and maximum aggregate Learning Session data rates that traverse each logical directed link in a line network graph routing core topology are:

$$\begin{aligned} C_{r-s}^{Line} &= C^{Line} \times R_S \\ &= C^{Line} \times \left(\frac{N_\tau}{\tau} \times b_s\right) \\ &= \frac{55 N_\tau b_s}{3\tau}; \end{aligned} \tag{4.6}$$

$$\begin{aligned} C_{r-s}^{Line-max} &= C^{Line-max} \times R_S \\ &= C^{Line-max} \times \left(\frac{N_\tau}{\tau} \times b_s\right) \\ &= \frac{25 N_\tau b_s}{\tau}. \end{aligned} \tag{4.7}$$

With the same reasonable parameter choices (and approximate Learning Session datagram size) as used in Section 4.2.8.1 and Eqs. (4.6)-(4.7), we have:

$$C_{r-s}^{Line} = \frac{55 \times 100 \times 12{,}000}{3 \times 1} = 22 \text{ Mbps};$$

$$C_{r-s}^{Line-max} = \frac{25 \times 100 \times 12{,}000}{1} = 30 \text{ Mbps}.$$

Considering that 6 Mbps may be an unreasonable rate for the data plane transmission burden of the State Measurement Service protocol, the average rate here is 11/3 times greater than the rate required for the optimistic Petersen graph routing core topology, while the maximum rate is a full five times greater. This Learning Session transmission burden may be reasonable for multi-gigabit fiber wavelengths, but we need to consider an alternative for subnets using other transmission substrates, such as wireless communication. Although State Measurement Service overhead can be reduced if not all routers in the routing core topology participate in the Learning Session protocol (for example, non-active routers that support CServ Intranetwork Service), subnets with limited transmission bandwidth likely still need a State Measurement Service protocol with less data plane burden. In the next section, we discuss an alternative approach to the State Measurement Service protocol that is better suited for subnets with limited transmission capacity in the routing core.

Before proceeding, we can again also consider the data plane switching burden of the Learning Session protocol for this "upper" bound subnet topology example. We focus here on the worst-case routers, specifically Router 5 and Router 6 in Fig. 4-8. For each of these routers, there is a total of 49 traffic units worth of Learning Session traffic arriving on its incoming logical edges. Although nine of these traffic units are Learning Sessions destined for the router in consideration, this router must also generate and switch nine traffic units worth of Learning Sessions destined for the other subnet routers in the routing core topology. Thus, 49 units of Learning Session traffic must be switched by both worst-case routers. With the same parameters as above, we can use the Learning Session data rate to conclude that the switching throughput for these routers must be 58.8 Mbps just to support the State Measurement Service protocol using Learning Sessions. This worst case requires approximately 3.3 times more switching bandwidth to support the Learning Sessions than in the "lower" bound Petersen graph case.

The overall router switching throughput must then be significantly higher to support additional best effort traffic. If we consider the "10% rule" introduced in Section 4.2.8.1, this implies that we would need active routers with switching throughput of at least 588 Mbps, where we have identified in the previous section that 180 Mbps may already strain the capabilities of many typical router switching

designs. Additionally, the worst-case logical links in the subnet with a line graph routing core topology would need to be designed to bear, at a minimum, 300 Mbps.

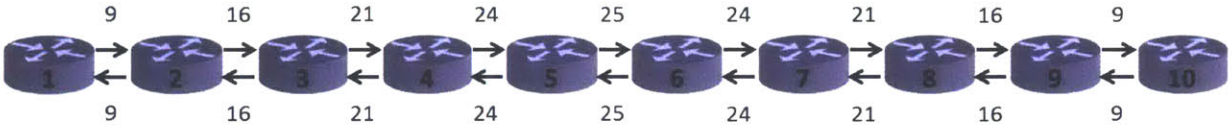


Fig. 4-8: This depicts the number of Learning Sessions supported by each logical directed link in the routing core of a subnet with a line network topology. Note that all routers in the routing core are assumed to be active CServ-enabled routers that are participating in the Learning Session protocol.

4.3 The Collector Protocol

In contrast to the Learning Session protocol for the State Measurement Service, we now present an alternative approach – the Collector protocol. The objective of this protocol is the same as that of the Learning Session protocol; it generates estimates of the CServ performance metrics, both reliability and delay statistics, for CServ Intranetwork Service between two active CServ-enabled routers in the subnet routing core. However, this protocol shifts the estimation workload to the individual routers in the CServ Intranetwork Service path. In the Learning Session protocol, the CServ Intranetwork Service endpoints (Entrance and Departure Routers) assume the majority of the workload; the Entrance Router of the CServ Intranetwork Service is responsible for transmitting a fixed-rate traffic session while the Departure Router computes and tracks statistics based on the realized performance of the traffic session during each learning interval. In the Collector protocol, the routers along the CServ Intranetwork Service path are now responsible for computing and tracking their own performance statistics, and a Collector datagram that traverses the CServ Intranetwork Service path "collects" these statistics once per learning interval. The reporting responsibilities for CServ performance metric updates remain with the destination endpoint router (the Departure Router) for the CServ Intranetwork Service.

Before we begin with the detailed discussion of the Collector protocol, we outline both the benefits of this approach and the sacrifices it makes. This protocol requires significantly less transmission and switching data plane overhead since it does not require the workload of a significant session rate between the CServ Intranetwork Service endpoints. It is this benefit that motivates its development as an alternative to the Learning Session protocol. It also reduces the responsibilities of the endpoint routers; the Entrance Router does not need to concern itself with metering the session rate and generating dummy active probes to supplement the variable rate real passive CServ datagrams, while the Departure Router does not need to maintain complicated running estimates of the CServ performance metrics for each peer active router in the subnet.

On the flip side, the estimates of this protocol are much more conservative compared to ground truth compared to those of the Learning Session protocol. Since real passive CServ datagrams are participants in the fixed-rate Learning Sessions, the estimates accurately capture actual performance of a CServ datagram traversing the CServ Intranetwork Service path and the cross-interaction with other CServ Intranetwork Service paths at intermediate routers. The Collector protocol, on the other hand, does not

have the benefit of high-rate sessions of CServ traffic to form CServ performance metric estimates. Since real passive CServ datagrams are intended to contribute a negligible amount of rate in most all network conditions, the Collector protocol relies on the best effort traffic that traverses each router during a learning interval to measure performance. The performance experienced by a CServ datagram should then outperform these estimates, since CServ datagrams receive priority queueing and transmission where possible. This approach requires the assumption that nominal use of the CServ messaging service does not significantly affect the statistics of the best effort traffic state (and we note here that this assumption may not hold during a significant network "Black Swan" event which generates a sharp increase in CServ messages). Additionally, we describe that this protocol requires both greater penetration of upgraded, CServ-enabled routers in the subnet routing core and more complex datagram processing and state tracking by the CSPU of intermediate active routers in the CServ Intranetwork Service paths.

4.3.1 Endpoints and Participants of the Collector Protocol

Unlike the Learning Session protocol where only the endpoints of CServ Intranetwork Service participate in the State Measurement Service, all routers in the CServ Intranetwork Service paths of the subnet are participants in the Collector protocol.

The endpoints of the Collector protocol are the same as the endpoints of the Learning Sessions, as described in Section 4.2.1. All active access routers and all active gateway routers in the subnet routing core are endpoints of the Collector protocol. The CServ Intranetwork Service connects each router in this set to every other router in this set via some CServ Intranetwork Service path or set of paths. Using the notation of Section 4.1, each router in N_R transmits a periodic *collector datagram* destined for each other router in N_R, which implies that there are at most $n_R(n_R - 1)$ periodic collector datagram exchanges. Without the subnet modeling assumption that all routers in N_R are CServ-enabled access or gateway routers, there would be fewer collector exchanges in the subnet. Core routers that are CServ-enabled can participate in the Collector protocol, as we shall describe in the subsequent paragraph, but they do not need to generate or sink collector datagrams to estimate the CServ performance metrics of any specific CServ Intranetwork Service path. As with the Learning Session protocol, the SC is responsible for facilitating the exchange of collector datagrams between the appropriate endpoints in the subnet

routing core. It distributes a compiled list of addresses of CServ-enabled routers that need to exchange collector datagrams (and this is the list of active CServ-enabled access routers and gateway routers).

Endpoints of collector datagram exchanges are not the only participants in the Collector protocol. Each router that is part of a CServ Intranetwork Service path must be CServ-enabled and participate in the protocol. We refer to these routers as *intermediate routers* in a CServ Intranetwork Service path. It is for this reason that the Collector protocol requires a greater adoption rate of active routers (and, likely, a greater capital investment to upgrade the routing core) than the Learning Session protocol. Each pair of endpoints for CServ Intranetwork Service must be connected by at least one path that contains only active, CServ-enabled routers as intermediate routers. Although a subnet that utilizes the Collector protocol can have non-active core routers, these routers cannot be members of CServ Intranetwork Service paths (unlike with the Learning Session protocol). The determination of CServ Intranetwork Service must be aware of this requirement, generating only CServ Intranetwork Service paths that traverse appropriate active CServ-enabled routers. Otherwise, the Collector protocol is unable to estimate the CServ performance metrics of the specified path. Both active routers that serve as endpoints for the exchange of collector datagrams and intermediate routers have similar responsibilities for maintaining performance statistics; the only difference is that intermediate routers do not need to generate or receive any collector datagrams.

4.3.2 Breakdown of Collector Protocol Measurements

Since the Collector protocol is not an end-to-end estimation of CServ performance metrics for CServ Intranetwork Service like the Learning Session protocol, we need to characterize the components of a CServ Intranetwork Service path and the estimation responsibilities of the constituents of the path to ensure that the collector datagram can capture the same performance contributions as the Learning Session.

Let us begin with a general description of a CServ Intranetwork Service path that uses the Collector protocol to estimate CServ performance metrics. Specifically, consider a CServ Intranetwork Service path that logically connects router A and router B within the subnet of interest, where $A, B \in N_R$. In the terminology of the Collector protocol, we say that router A is the Entrance Router for the collector datagram exchange and router B is the Departure Router for the collector datagram exchange. This

terminology stems from the view of the CServ Intranetwork Service as an opaque subnet tunnel that bears the internetwork CServ datagram between two routers in its CServ Internetwork Service path without the explicit control of the MC. This CServ Intranetwork Service path may or may not contain intermediate routers based on the subnet routing core topology and the routing strategy used to generate the paths. For the purpose of discussion, we assume that there is at least one intermediate router in the CServ Intranetwork Service path, which we designate router C. Router C does not need to be an active access or gateway router, but it does need to be an active CServ-enabled router as discussed in Section 4.3.1.

If the Learning Session protocol were used to estimate the CServ performance metrics of the CServ Intranetwork Service path connecting router A to router B, the estimates would include contributing factors from the enqueueing of CServ datagrams at the input interface buffer of router A until the successful reception of the CServ datagrams at the input interface of router B (see Section 4.2). Using the Collector protocol, these path CServ performance metric estimates are broken into components, where each router in the path is responsible for contributing its component when it is aggregated by a collector datagram. We break the path CServ performance metric measurement into areas of responsibility for each router participating in the Collector protocol, while capturing the same set of end-to-end contributors as the Learning Session protocol. The remainder of this section describes the estimation responsibilities of each active CServ-enabled router that participates in the Collector protocol. When applied to our example for router A and router B, with intermediate router C, we note that it enables the aggregation of the same contributing performance components as a Learning Session between routers A and B.

Each active router participating in the Collector protocol, either as an endpoint of a collector datagram exchange or as an intermediate router in a CServ Intranetwork Service path, has the purview to maintain two sets of estimates that are periodically refreshed (namely, each learning interval). One set of estimates pertains to the communication reliability of the input transmission interfaces, and the other set of estimates captures the router traversal reliability and delay. The separation of responsibility for these estimates is depicted in Fig. 4-9.

First, consider the estimate that concerns the communication reliability of the input transmission interfaces. The transmission of a datagram between two adjacent routers can contribute to datagram

corruption in a variety of ways, including atmospheric interference, multipath fading, dispersion, cross-channel interference, and multiple access collision, among other actors in data corruption. As datagrams arrive on the input interfaces of the router in question, the lower layer protocol integrity checks verify that the datagram has in fact been received without error (or in some cases, if the error can be rectified by the forward error correction mechanism). If the datagram cannot be recovered successfully, it is considered lost and dropped. Even though this processing is frequently performed at the physical or data link control layer, the operations are implemented on the router input interfaces. It is the responsibility of the CSPU to maintain an estimate of the *Last Hop Transmission Reliability*, which can be estimated each learning interval as the ratio of the number of successfully received datagrams to the total number of arriving datagrams at the aggregate router input interfaces. A similar estimator is used for reliability in the Learning Session protocol (see Section 4.2.6), except that this version of the estimator is local to the input interfaces of a particular active router in the subnet. This estimate counts datagrams in both the best effort and CServ classes, and we note here that Collector datagrams are in the CServ class and can also be used to generate new estimates.

Next, consider the set of estimates that cover the traversal of the router itself. As introduced in Section 4.2.3 in more detail, the traversal of the router includes the arrival processing of the datagram to determine its router output interface, the waiting time in the input buffer of the arriving router interface, the transmission of the datagram across the switching fabric at the scheduled time, and the waiting time in the output buffer of the output router interface. Section 4.2.3 further describes the opportunities for datagram loss (or reliability degradation) and delay throughout this traversal process; we do not reiterate the details here. With the Collector protocol, it is the responsibility of the CSPU to maintain three separate estimates for the router traversal process which are generated each learning interval: the *Traversal Reliability*, the *Mean Traversal Delay*, and the *Traversal Delay Variance*. Leveraging information that is generally available from the NPU for the purpose of scheduling the switching fabric, the CSPU calculates the three estimates as follows. The Traversal Reliability is computed, similarly to the Last Hop Transmission Reliability, as the ratio of the number of datagrams successfully transmitted on the aggregate output router interfaces (again in both the best effort and CServ classes, including Collector datagrams) to the total number of datagrams enqueued in the aggregate input router interfaces. In order to calculate the delay statistics, each datagram that is enqueued in an input buffer on the router input interfaces is tagged with a current time timestamp. As datagrams are removed from the output buffers on the router output interfaces, the difference

between the current time and the tagged timestamp value is calculated, representing the traversal delay of that particular datagram. The CSPU maintains running estimates of the Mean Traversal Delay and Traversal Delay Variance that it updates with these computed traversal delay values during each learning interval. The forms of the estimators are the same as in Eq. (4.2) and Eq. (4.3). Note that router-specific datagram timestamp tags and counts should be available from the NPU. However, the NPU interface may require modification to expose this information to the CSPU for the computation of these estimates.

We now return to the example of the CServ Intranetwork Service path that logically connects router A and router B to show that, with this separation of estimate purviews by each router, the Collector protocol can amalgamate this information to measure the performance contributions from the same sources as the Learning Session protocol. When router A, the Entrance Router, generates a Collector datagram destined for router B, it enqueues it in the input buffer of a random input interface of the router, just like a datagram in a Learning Session. Since this Collector datagram has not traversed an intranetwork router-to-router hop prior to router A, it does not collect the current Last Hop Transmission Reliability estimate at router A. However, as it does traverse router A, it collects the current Traversal Reliability, Mean Traversal Delay, and Traversal Delay Variance estimates at router A. Once the collector datagram reaches router C, the intermediate router in the CServ Intranetwork Service path, it collects *both* sets of estimates since it has been transmitted over a previous communication hop to reach router C and it will traverse router C itself. Finally, when the collector datagram arrives at router B, the Departure Router, it only collects the current Last Hop Transmission Reliability estimate at router B since this Collector datagram will not traverse the router. Comparing the contributions captured by this process to that of the Learning Session described in Section 4.2.3, the only contributions that are missing are the transmission and propagation delays between routers in the CServ Intranetwork Service path. These are considered deterministic delay components that can be measured by the transmission of the Collector datagram itself. As a Collector is dequeued from the output buffer of an active router for transmission, it is updated with a current time timestamp. Upon arrival processing at the next router, the difference between the current time and the Collector timestamp, the *Transmission and Propagation Delay*, is collected by the datagram to account for this last component that contributes to the CServ Intranetwork Service delay.

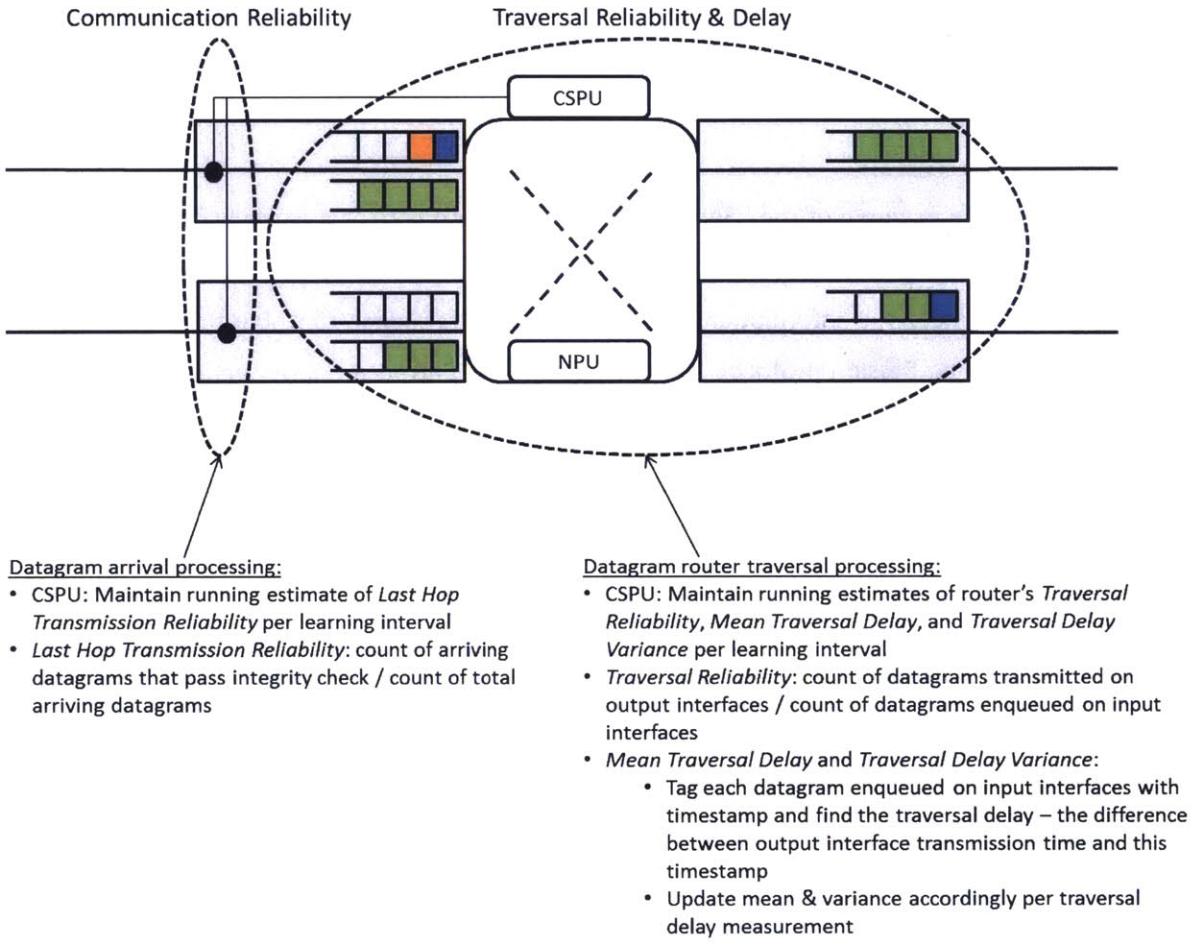


Fig. 4-9: This illustration depicts the separation of estimation responsibilities for the two sets of estimates maintained by an active CServ-enabled router participating in the State Measurement Service Collector protocol.

4.3.3 State Measurement Service Header for Collectors

In this section, we discuss the use of the State Measurement Service header placeholder from Chapter 3 for the purposes of the Collector protocol. Unlike the State Measurement Service header described for Learning Sessions in Section 4.2.4, several of the fields in the State Measurement Service header for Collector datagrams are mutable. That is, these fields are modified in transit as the datagram "collects" the CServ performance metrics for the CServ Intranetwork Service path. The header also utilizes the maximum length bound presented for the CServ overhead analysis of Chapter 3.

To begin, a real internetwork CServ datagram is never used as a Collector datagram, the preferred datagram type in the Learning Session. All Collector datagrams are active probes that are generated by the Entrance Router endpoint and terminated by the corresponding Departure Router endpoint of the particular CServ Intranetwork Service. Since Collector datagrams include mutable fields that are modified frequently during the transit of a CServ Intranetwork Service path, we choose to use active probes so as not to risk corruption of real critical messages during datagram modification.

As an active probe, the Collector datagram retains the structure of a typical internetwork CServ datagram traversing the routing core of the particular subnet in question. That is, it contains a dummy CServ message payload, a CServ Internetwork Service header, a State Measurement Service header for Collector datagrams, and a CServ Intranetwork Service header that binds the Collector datagram to a particular CServ Intranetwork Service path. Although the CServ Internetwork Service header and CServ data payload are not strictly necessary for the workload of the Collector datagram, the standard CServ datagram structure is used since the Collector datagram itself generates measurements of transmission delays along the CServ Intranetwork Service path.

The format of the State Measurement Service header for Collector datagrams is depicted in Fig. 4-10 and placed in the context of the intranetwork CServ datagram, or more specifically the Collector datagram in this case. The following describes the purpose and use for each control field in the State Measurement Service header for Collector datagrams:

- **Version (VER)** – As with all other CServ-proprietary headers, this leading 4-bit Version field is used to specify the type of State Measurement Service header (in this case, the State

Measurement Service header for Collector datagrams). This field is based on the IP header field, so we employ an unused code (codes for decimal values 1-3 and 10-14 are unallocated) to specify that the following is a State Measurement Service header for Collector datagrams. This code allows the active CServ-enabled routers that compose the CServ Intranetwork Service path to correctly process and update the Collector datagram when it arrives. This is a non-mutable field in the State Measurement Service header for Collector datagrams.

- **Next Header (NH) –** The 4-bit Next Header field specifies the format of the encapsulated datagram header. In the case of Collector datagrams, this is the CServ Internetwork Service header. This field is included for flexibility should future CServ architecture developments change the structure of the Collector datagram. Additionally, this field is included to maintain the consistency among CServ-proprietary headers, each of which leads with both the Version and Next Header fields that allow for dexterity in the stacking of headers and generation of datagrams. This is a non-mutable field in the State Measurement Service header for Collector datagrams.

- **Diversity Degree (DIV) –** The 4-bit Diversity Degree field is used to specify the degree of path diversity used by the particular CServ Intranetwork Service, with a default value of 1 (note that this then allows for the use of a maximum of 15 diverse paths for CServ Intranetwork Service). All Collector datagrams that are replicated for the purpose of a diversity-routed CServ Intranetwork Service bear the same value in this field, as well as in the following Sequence Number field. We consider the importance of this field in more detail in Section 4.3.5. This is a non-mutable field in the State Measurement Service header for Collector datagrams.

- **Sequence Number (SN) –** The 28-bit Sequence Number is used in a similar fashion as with the Sequence Number field in the State Measurement Service header for Learning Sessions (see Section 4.2.4). The only difference is that four of the bits have been "stolen" for the Diversity Degree field. Although the Sequence Number is not necessary to count unique arriving datagrams at the Departure Router for the estimation of reliability, it is used to identify datagrams replicated for the purpose of diversity routing. Replicated Collector datagrams sent over the disjoint paths in the same CServ Intranetwork Service carry identical Sequence Number

field values. We consider the importance of this field in more detail in Section 4.3.5. This is the last non-mutable field in the State Measurement Service header for Collector datagrams.

- **Timestamp (TS)** – The Timestamp field is the first mutable of the mutable fields of the State Measurement Service header for Collector datagrams. All following fields are also mutable. The 64-bit Timestamp is used throughout the collection process to calculate the transmission delay experienced by the Collector datagram. Before transmission, the field is marked with the current time. At the receiving end of the transmission, the active router in the CServ Intranetwork Service path uses the difference between the current time and the value in the Timestamp field to compute the transmission delay. Note that this is *not* the same timestamp as in the Origin Timestamp field used in the CServ Intranetwork Service header for Explicit Path Forwarding, which is a non-mutable field and represents the time the intranetwork CServ datagram is enqueued in the input buffer of the Entrance Router's input interface. We discuss the utilization of this Timestamp field in more detail in Section 4.3.5.

- **Reliability (RL)** – The Reliability field is a mutable 64-bit field that encodes a double-precision floating-point number representing the collected reliability of the CServ Intranetwork Service path thus far. When the Collector datagram is generated by the Entrance Router endpoint of the CServ Intranetwork Service, this field is initialized to a decimal value of 1. At each active CServ-enabled router that forms the CServ Intranetwork Service path, this field is updated appropriately according to the most recent Last Hop Transmission Reliability and/or Traversal Reliability estimates. The update process for this field is discussed in more detail in Section 4.3.5. Together with the Delay Mean and Delay Variance fields, this triplet holds the collected CServ performance metrics for the CServ Intranetwork Service path upon arrival at the Departure Router of the CServ Intranetwork Service.

- **Delay Mean (DM)** – The Delay Mean field is another mutable 64-bit field that encodes a double-precision floating-point number representing the collected mean delay statistic of the CServ Intranetwork Service path thus far. When the Collector datagram is generated by the Entrance Router endpoint of the CServ Intranetwork Service, this field is initialized to a value of 0. At each active CServ-enabled router that forms the CServ Intranetwork Service path, this field is updated appropriately according to the computed Transmission Delay and/or the most recent Mean

Traversal Delay and Traversal Delay Variance statistic estimates. The updated process for this field is discussed in more detail in Section 4.3.5.

- **Delay Variance (DV)** – The Delay Variance field is the final field of the State Measurement Service header for Collector datagrams and is also a mutable 64-bit field that encodes a double-precision floating-point number. The current value of this field represents the collected delay variance statistic of the CServ Intranetwork Service path thus far. At the time of generation, the Entrance Router endpoint of the CServ Intranetwork Service initializes the value of this field to 0, and it is updated at each active CServ-enabled router that forms the CServ Intranetwork Service path according to the most recent estimate of Traversal Delay Variance, if appropriate. The updated process for this field is discussed in more detail in Section 4.3.5.

With the convenience of the octet offsets illustrated in Fig. 4-10, we see that the State Measurement Service header for Collector datagrams requires the maximum 37 bytes allotted in the analysis of Chapter 3. This additional overhead compared to the State Measurement Service header for Learning Sessions is due to the need for the header to encode the current collected estimates of the CServ performance metrics for the CServ Intranetwork Service path. Even though double-precision may not be strictly required for any of the CServ performance metrics, Collector datagrams are transmitted infrequently enough that this conservative allocation does not significantly contribute to the subnet transmission and switching burden incurred by the Collector protocol.

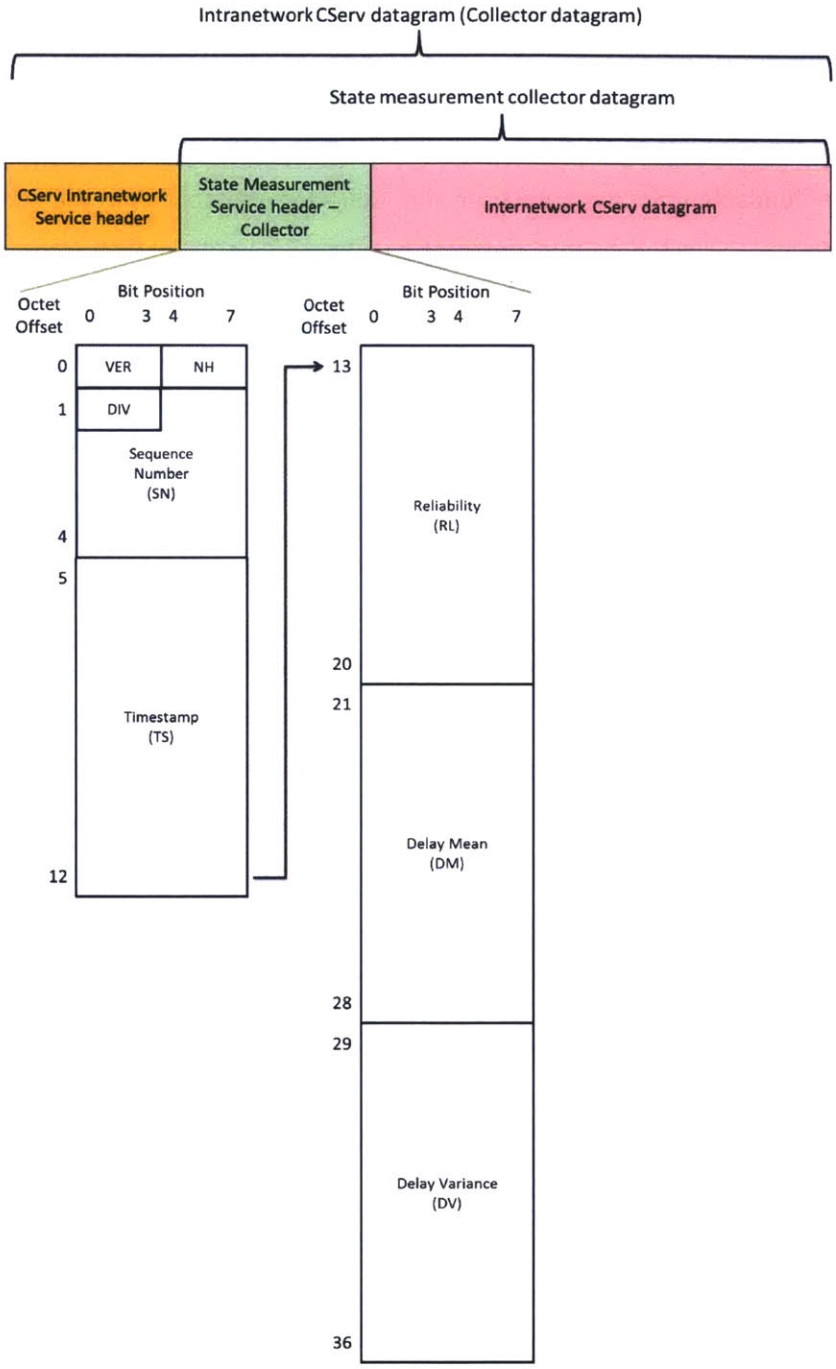


Fig. 4-10: The fixed length format of the State Measurement Service header for Collector datagrams is depicted here, to scale. The use of the Collector protocol is encoded in the Version (VER) field. The Timestamp, Reliability, Delay Mean, and Delay Variance fields are mutable.

4.3.4 Collector Protocol Parameters of Estimation

Before we consider the process of updating the Collector datagram as it traverses the active CServ-enabled routers in the CServ Intranetwork Service path between Entrance Router and Departure Router, we consider the fundamental parameters of the Collector protocol and the generation of the performance estimates collected by the protocol datagrams. The Collector protocol has a reduced parameter dimensionality compared to the Learning Session protocol. Whereas the Learning Session protocol has three fundamental parameters (two of which can be controlled independently), the Collector protocol has only two fundamental parameters (only one of which is independently under the discretion of the subnet administrator).

The first parameter discussed in Section 4.2.5 was the Learning Session rate. There is an equivalent rate for the Collector protocol, the rate at which Collector datagrams are generated by a particular active CServ-enabled Entrance Router in the subnet routing core destined for a particular active CServ-enabled Departure Router. We call this the *Collector protocol rate* and denote it as λ_C [collector datagrams/second]. Active routers in a subnet using the Collector protocol to estimate the CServ performance metrics of CServ Intranetwork Service paths maintain fixed rate Collector datagram exchanges, similar to the Learning Session protocol. However, this is a low rate exchange with respect to the interval of estimate generation, which motivates the use of the "exchange" terminology rather than the "session" nomenclature. This becomes clearer after we introduce the other parameter of the protocol.

The second parameter discussed in the context of the Learning Session protocol was the *learning interval*, which was denoted as τ [seconds]. The same parameter is used for the Collector datagram protocol. During each learning interval, independent estimates are generated by each active CServ-enabled router for the local performance components, the Last Hop Transmission Reliability, the Traversal Reliability, the Mean Traversal Delay, and the Traversal Delay Variance. As with the Learning Session protocol, these intervals are synchronized using the assumption of the availability of a global synchronized clock for the CServ-enabled network devices.

The third parameter considered in Section 4.2.5 for the Learning Session protocol was that of reliability precision, indicating the granularity at which the CServ reliability performance metric could be

estimated. To characterize this parameter, we presented a proxy parameter, the total number of Learning Session datagrams transmitted in a learning interval of length τ. In the Collector protocol, the frequency of Collector datagram generation and transmission does not determine the precision at which the CServ reliability performance metric can be specified. Each active CServ-enabled router in the subnet routing core generates two estimates related to the reliability of a CServ Intranetwork Service path each learning interval: the Last Hop Transmission Reliability and the Traversal Reliability. The precision at which either of these can be estimated depends on the traffic load seen at that particular router during the previous complete learning interval (the aggregate of both best effort traffic and real CServ datagrams), not on a parameter of the Collector protocol. And, in general, an active router can refuse to update the current set of estimates if there is not enough traffic to accurately capture the performance during any particular learning interval.

As far as the number of Collector datagrams exchanged between two active routers in the subnet routing core each learning interval, we do not consider this a formal parameter of the protocol for now. By default, the Collector protocol generates and transmits one Collector datagram between each pair of active routers during each learning interval. However, we note that this could be considered a maximum. In the future, the rate of Collector datagram exchanges could be reduced such that a Collector datagram is exchanged only every few learning intervals in stable subnets. It does not make sense to increase the rate of exchanges beyond this maximum since new performance estimates along the CServ Intranetwork Service path are only generated once every learning interval; additional Collector datagram exchanges per learning interval would be redundant, aggregating the same set of performance estimates along the path.

Based on the parameter definitions, and following the example of Section 4.2.5, we note that $\lambda_C = 1/\tau$ since only (or at maximum) one Collector datagram is exchanged between an active router pair via the CServ Intranetwork Service per learning interval. The subnet administrator's choice of learning interval length becomes the primary consideration in configuring the Collector protocol. If the learning interval is small compared to the delay of the CServ Intranetwork Service paths in a subnet, the performance estimates gathered early in the CServ Intranetwork Service path by a Collector datagram are stale estimates by the time the Collector reaches the Departure Router. In general, the learning interval should not be shorter than the CServ Intranetwork Service path delays, and simultaneously it should be on the order of the coherency of the subnet state.

4.3.5 The Collection Process

Before presenting the "collection process" that updates a Collector datagram as it traverses an active CServ-enabled router participating in the Collector protocol as part of a CServ Intranetwork Service path, we consider the incremental updates to the three CServ performance metrics of interest: reliability, mean delay, and delay variance.

Recall that reliability represents the probability that a datagram transmitted by some source host arrives successfully at the intended destination host, meaning without corruption or loss. For the sake of discussion, assume a toy example scenario where our source and destination hosts are connected by a path with two intermediate routers, routers A and B, and that the reliability of each router is known (as one variable) as ρ_A and ρ_B. Further, we assume that all communication links between the routers along this path are completely reliable. We denote the reliability of the path as ρ. If we can assume that the loss processes captured by the reliability variables are statistically independent, then we can conclude that the reliability of the path between the source and destination hosts $\rho = \rho_A \times \rho_B$. That is, the reliability of the path is the product of the reliability among the statistically independent elements along the path. This requirement of statistical independence between subnet elements is vital to the collection process, but it does require some careful consideration. If the primary source of loss at both routers A and B is subnet congestion, then it is highly unlikely that these two loss processes are statistically independent; the oversubscription of the subnet transmission or switching bandwidth could induce correlation between them. Additionally, consider the case now where the reliability of router A is captured with two separate variables, one denoting the reliability of the buffering process, ρ_A^{buffer}, and the other representing the reliability of the switching process, ρ_A^{switch}. If we can assume statistical independence between these two processes, then the reliability of the path between the source and destination hosts becomes $\rho = \rho_A^{buffer} \times \rho_A^{switch} \times \rho_B$. However, if the buffering process at a particular router is congested and has higher loss, then the likelihood that loss is observed from imperfect router switch scheduling increases. These loss processes are clearly not statistically independent as described.

We utilize the statistical independence assumption throughout the Collector protocol and collection process for implementation simplicity and computation tractability. This assumption enables the use of a compact set of performance estimates with very simple collection update rules. Thus, the reliability of a CServ Intranetwork Service path is a product of the estimated reliability values of the elements that

compose the path. To update the Reliability field, the collected reliability estimate is multiplied with the current value in the field. The resulting value is then used to update the Reliability field of the collector. That being said, it is necessary to remain skeptical of the results and consider sources of correlation between elements in the subnet, particularly during significant subnet transient conditions or instances of network stress.

We apply the same assumption of statistical independence as we collect the delay statistics along a CServ Intranetwork Service path in the Collector protocol. The mean delay of a path is the sum of the mean delays along the path. We note that this statement alone does not invoke the assumption on statistical independence. It holds even if there were correlation between the path delay processes. However, we do need the assumption for the next statement; the delay variance of a path is the sum of the delay variances along the path. With the statistical independence assumption and these statements, we provide the rules to update the delay statistic fields in the Collector datagram during the collection process. To update the Delay Mean field, the collected mean delay estimate (or computed transmission and propagation delay value, as this is treated as a deterministic DC component of delay) is added to the current value in the field. The result then replaces the original value and updates the Delay Mean field. To update the Delay Variance field, the collected delay variance value is added to the current value stored in the field. The resulting value is then used to update the field. As with the reliability collection, we need to maintain a healthy feeling of trepidation with the resulting path delay variance since we have assumed away any correlation between the subnet delay processes.

Endowed with these incremental update rules for the Reliability, Delay Mean, and Delay Variance fields of a Collector datagram, we describe the collection process at an active router participating in the Collector protocol (that is, it is part of the CServ Intranetwork Service). The process is illustrated in Fig. 4-11. For the detailed discussion of the process, we consider a CServ Intranetwork Service path that logically connects router A and router B within the subnet of interest, where $A, B \in N_R$, as in Section 4.3.2. In the terminology of the Collector protocol, we say that router A is the Entrance Router for the collector datagram exchange and router B is the Departure Router for the collector datagram exchange. For the purpose of discussion, we additionally assume that there is at least one intermediate router in the CServ Intranetwork Service path, which we designate router C, where $C \in N_R$.

First, let us consider the Entrance Router, router A. This router serves as the source endpoint of the Collector datagram exchange. When the appropriate CSPU process for this exchange creates the Collector datagram, it initializes the Reliability, Delay Mean, and Delay Variance fields as discussed in Section 4.3.3. The initialization of the Timestamp field is inconsequential, but it can be set to the current time at Collector datagram generation. Before the Collector datagram is enqueued in the input buffer of a random input interface of the Entrance Router, the Reliability field is updated by the current estimate of Traversal Reliability at router A, the Delay Mean field is updated by the current estimate of the Mean Traversal Delay at router A, and the Delay Variance field is updated by the current estimate of the Traversal Delay Variance at router A. These updates follow the rules previously established in this section. The use of a random input interface for the initial enqueueing of the Collector datagram is not fundamental to the aggregation of the final CServ performance metric estimates; this tactic only spreads the load of generated Collector datagram traffic over the router interfaces. When the Collector datagram is dequeued from the output buffer of router A's output interface, the CSPU updates the Timestamp field with the current time immediately before the datagram is sent on the output transmission interface.

Next, consider the intermediate router in the CServ Intranetwork Service path, router C. When the Collector datagram is successfully received on one of its input interfaces, the CSPU initiates the following collection steps. Immediately upon successful arrival, the last hop's realized Transmission and Propagation Delay is computed as the difference between the current time and the value in the Collector datagram's State Service Measurement header Timestamp field. The Delay Mean field of the Collector datagram is then updated by this computed value. Furthermore, the Reliability field is updated by the current estimate of the Transmission Reliability at router C. This completes the datagram arrival collection steps. The CSPU then proceeds to complete the router traversal processing steps as described in the previous paragraph for router A: the Reliability field is updated by the current estimate of Traversal Reliability at router C, the Delay Mean field is updated by the current estimate of the Mean Traversal Delay at router C, and the Delay Variance field is updated by the current estimate of the Traversal Delay Variance at router C. Following the collection of traversal estimates, the Collector datagram is enqueued in the input buffer of the arriving input interface at router C to await scheduled access to the switching fabric. And just as with the Entrance Router, the CSPU updates the Timestamp field with the current time immediately after the Collector datagram is dequeued from the output buffer and immediately before the datagram is transmitted on the output transmission interface.

Lastly, consider the Departure Router in the CServ Intranetwork Service path, and consequently in the Collector datagram exchange. In our example, this is router B. The collection process at the Departure Router only involves collecting the datagram arrival estimates, specifically the last hop's realized Transmission and Propagation Delay and router B's current estimate of the Transmission Reliability. Traversal performance estimate collection does not occur at the Departure Router. So immediately upon successful arrival, the last hop's realized Transmission and Propagation Delay is computed as the difference between the current time and the value in the State Service Measurement header Timestamp field. The Delay Mean field of the Collector datagram is updated by this computed value, and the Reliability field is updated by the current estimate of the Transmission Reliability at router B. Following these updates, the appropriate CSPU process (the peer process with Entrance Router A) discards the Collector datagram after extracting the aggregated CServ Intranetwork Service path estimated CServ performance metrics. Comparing these end-to-end updates from the Entrance Router to the Departure Router with the overview of Section 4.3.2, we see that the finalized CServ performance metric estimates for the CServ Intranetwork Service path match the intended purview of the State Measurement Service for the subnet routing core.

What we have yet to discuss is the combination of Collector datagrams in the collection process when path diversity is used for CServ Intranetwork Service connecting a particular pair of Entrance and Departure Routers. In Section 4.3.3, we allocated a 4-bit State Measurement Service header field for the Collector datagram to specify the number of diverse paths employed to realize the CServ Intranetwork Service. This is necessary because we wish to use the Collector protocol to collect the CServ performance metrics for a CServ Intranetwork Service, not for a particular CServ Intranetwork Service path. If we consider the operation of the Learning Session protocol, we notice that the output CServ performance metric estimators summarize the reliability and delay statistics of the union of the paths if diversity routing is employed. It does not individually characterize the CServ performance metrics of each diverse path in a particular CServ Intranetwork Service.

Consider the case where diversity routing of at least two disjoint paths is used as the CServ Intranetwork Service to connect a particular Entrance and Departure Router pair in a subnet. When the Entrance Router generates the Collector datagram and binds it to the appropriate CServ Intranetwork Service, it must replicate the Collector datagram instance such that it can prepend the appropriate CServ Intranetwork Service header for each diverse path to a separate State Measurement Service datagram.

The appropriate encoding representing the number of replications or number of diverse paths in the service is written into the Diversity Degree field of each State Measurement Service header, and the next sequence number maintained by the Entrance Router CSPU process for the destination Departure Router is applied to the Sequence Number field (which is only 28-bits for Collector State Measurement Service headers). The behavior of the Departure Router in the Collector datagram exchange is modified when path diversity is used to implement the CServ Intranetwork Service. Assuming that value encoded in the Diversity Degree field is greater than one, the Departure Router cannot generate an update to the CServ Intranetwork Service performance metric estimates until it receives all Collector datagrams with the same sequence number value. That is, if the degree of path diversity used is two, then the Departure Router does not finalize new performance metric estimates until it receives two Collector datagrams with the same sequence number. After the first Collector datagram is received on the first path, it extracts the aggregated CServ Intranetwork Service path performance metric estimates from the State Measurement Service header fields and stores them with the appropriate CSPU process while it waits for the arrival of the second Collector datagram. Upon arrival of the second Collector datagram and extraction of its aggregated CServ Intranetwork Service path performance metric estimates for the second path, the CSPU process is able to finalize new CServ performance metric estimates for the CServ Intranetwork Service. Should the second Collector datagram not arrive due to loss or corruption, the extracted information from the first is discarded when Collector datagrams for the Collector datagram exchange in the next learning interval (with a new sequence number) begin to arrive. This behavior is consistent with the update policy when path diversity is not used to implement CServ Intranetwork Service; if the one Collector datagram does not arrive successfully at the Departure Router due to corruption or loss, the current CServ Intranetwork Service performance metric estimates are not updated.

The following process is used to combine the collected metric sets from the Collector datagrams when path diversity is used for CServ Intranetwork Service. For discussion, assume that k-diversity is employed where $k > 1$ for the particular CServ Intranetwork Service between active router $A \in N_R$ and active router $B \in N_R$ (meaning that there are k diverse paths used in the CServ Intranetwork Service to logically connect the Entrance and Departure Router pair). We enumerate the paths as path $i \in \{1,2,\ldots,k\}$, and we define the set of path performance estimates collected by the respective Collector datagrams as $\{\hat{\rho}_i, \hat{\mu}_i, \hat{\sigma}_i^2\}_{i\in\{1,2,\ldots,k\}}$ where $\hat{\rho}_i$ denotes the value from the Reliability field, $\hat{\mu}_i$ denotes the value from the Delay Mean field, and $\hat{\sigma}_i^2$ denotes the value from the Delay Variance field. The CServ

Intranetwork Service performance metric updates occur once all k Collector datagrams with the same value in the Sequence Number field have arrived at the Departure Router. Using the statistical independence assumption, we can abstractly describe the reliability of CServ Intranetwork Service using k disjoint diversity paths as:

$$\begin{aligned}\text{Service Reliability} &\triangleq \Pr\{\text{Transmission over at least one of } k \text{ paths is successful}\}\\ &= 1 - \Pr\{\text{Transmission over none of the } k \text{ paths is successful}\}\\ &= 1 - \prod_{i=1}^{k} \Pr\{\text{Transmission over path } i \text{ is not successful}\}\\ &= 1 - \prod_{i=1}^{k} (1 - \text{Reliability of path } i).\end{aligned}$$

Note that the statistical independence assumption is leveraged in the third line above (as well as in the collection process for each Collector datagram). With the notation we have introduced for this discussion, the updated CServ reliability estimator generated at Departure Router B, $\hat{\rho}_{\tau_i}^{A\to B}$, after the successful reception of the k Collector datagrams with the same sequence number during learning interval τ_i is:

$$\hat{\rho}_{\tau_i}^{A\to B} \triangleq 1 - \prod_{i=1}^{k} (1 - \hat{\rho}_i). \tag{4.8}$$

Note that this formula is consistent with the Collector protocol when diversity routing is not employed for CServ Intranetwork Service (i.e. $k = 1$). Defining the CServ Intranetwork Service delay statistics updates is trickier, and we devote more attention to the matter in Chapter 5 in the context of defining the delay of a CServ Internetwork Service path. For now, we present the update rule without further exposition. As the Departure Router receives the k Collector datagrams from the exchange, it computes the following *Path Delay* estimate, $\hat{d}_i$, for each path $i \in \{1,2,\ldots,k\}$:

$$\hat{d}_i \triangleq \hat{\mu}_i + 10\sqrt{\hat{\sigma}_i^2}. \tag{4.9}$$

The aggregated Collector datagram delay statistics for the path with the maximum estimated Path Delay are maintained as the CServ Intranetwork Service delay statistic estimates (with an arbitrary tie breaker). Formally, the updated CServ mean delay and delay variance estimates generated at Departure Router B, $\hat{\mu}_{\tau_i}^{A\to B}$ and $\hat{\sigma}^2{}_{\tau_i}^{A\to B}$, after the successful reception of the k Collector datagrams with the same sequence number during learning interval τ_i is given by:

$$\left\{\hat{\mu}_{\tau_i}^{A\to B}, \hat{\sigma}^2{}_{\tau_i}^{A\to B}\right\} = \{\hat{\mu}_{\phi}, \hat{\sigma}_{\phi}^2\}, \quad \text{where } \phi = \underset{i\in\{1,2,\ldots,k\}}{\operatorname{argmax}} \hat{d}_i. \tag{4.10}$$

It is worth mentioning that the Collector protocol and collection process opens the opportunity for a centralized version that completely eschews the exchange of datagrams in the subnet routing core data plane. This centralized version leverages the full potential of the logically-centralized SC structure within the subnet. We can consider the collection process described so far as the "distributed Collector protocol," and we briefly outline the "centralized Collector protocol." In the distributed version, the active CServ-enabled routers perform the majority of the performance estimation workload, leaving the Collector datagrams to measure the transmission and propagation delays for a CServ datagram using the CServ Intranetwork Service. If adjacent active CServ-enabled routers in the subnet implement a protocol to measure the transmission and propagation delay between them, the routers in the subnet jointly now have all necessary state estimates and measurements needed to determine the end-to-end CServ performance metric estimates for a CServ Intranetwork Service path. The centralized version of the Collector protocol would have the active routers that can participate in the CServ Intranetwork Service report this full set of estimates to the SC. By means of the appropriate routing policy for the CServ Intranetwork Service, the SC uses the reported estimates from each active router and the network topology to compute the CServ performance metrics for the CServ Intranetwork Services in the subnet. The logically-centralized computation capability of the SC replaces the need for the exchange of Collector datagrams. Furthermore, if the routing and forwarding policy allows, the SC can use the reported performance metric estimates to optimize the CServ Intranetwork Service paths. This makes the determination of CServ Intranetwork Service performance-aware, a much more proactive approach than collecting the estimates of CServ performance metrics with Collector datagrams only after the paths are established. The detailed development of this approach is left as future work in the CServ framework.

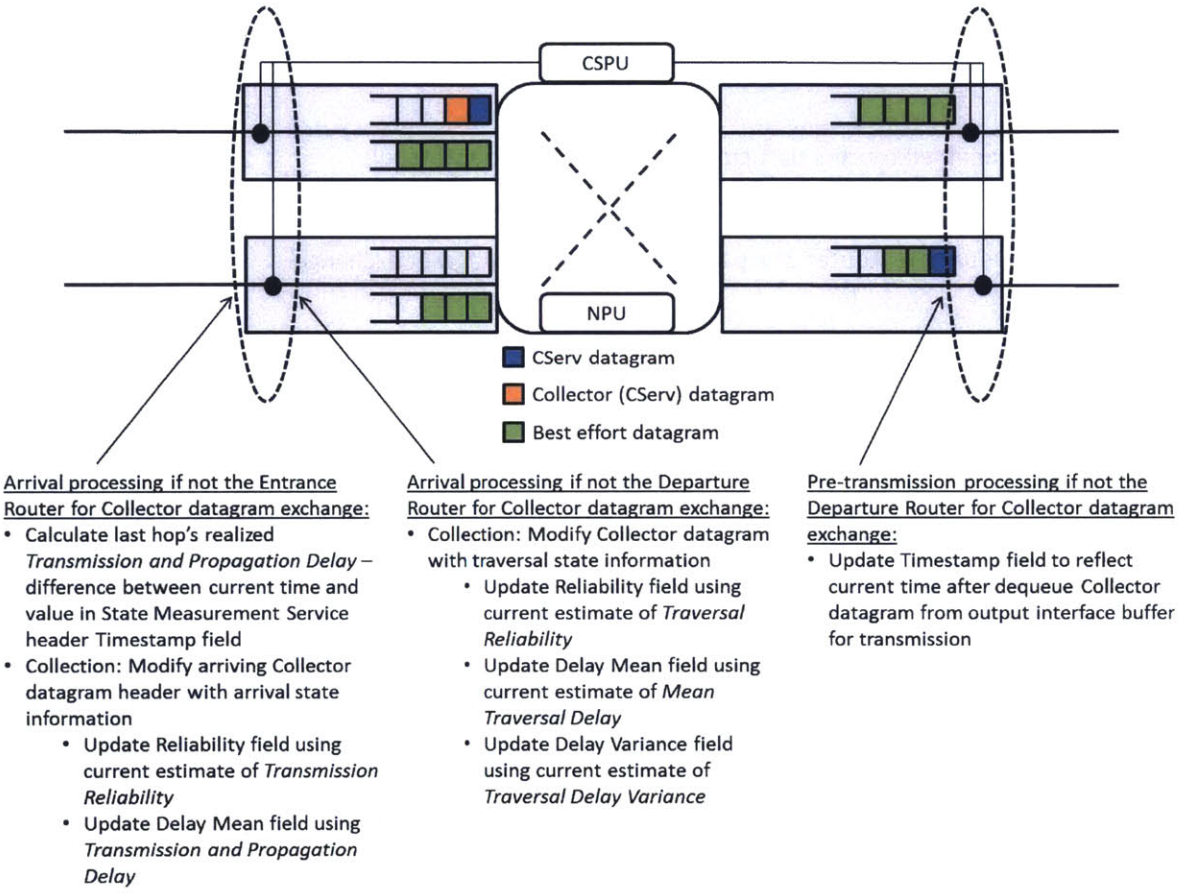


Fig. 4-11: This is a depiction of an active CServ-enabled router that participates in the State Measurement Service Collector protocol, describing the "collection process" used to update Collector datagrams with the appropriate performance estimates and calculations.

4.3.6 Collector Protocol Description

With the fundamental descriptions of the Collector protocol components, we are ready to formalize the protocol responsibilities of the primary participants in the State Measurement Service, the active CServ-enabled routers in the subnet. Pulling together the coverage from Sections 4.3.1-4.3.5, the following outlines the protocol responsibilities for the following router designations:

1. all CServ-enabled routers participating in the Collector protocol;
2. the Entrance Router of a particular Collector datagram exchange;
3. and the Departure Router of a particular Collector datagram exchange.

Note that an active CServ-enabled router participating in the Collector protocol assumes all of the above roles, depending on the endpoints of the Collector datagram exchange considered. For the simplicity of discussion, we assume that all subnet routers in N_R are active CServ-enabled routers participating in the Collector protocol. If there are non-active routers in the subnet's routing core, they should be excluded from the following discussion since non-active routers do not participate in the Collector protocol, nor can they serve as constituents in CServ Intranetwork Service paths in the subnet.

The following protocol applies for every Router $i \in N_R$ and every Router $j \in N_R \backslash \{i\}$. We use the convention that Router i is the Entrance Router for the considered Collector datagram exchange described in the protocol (alternatively, Router i is the Entrance Router for the CServ Intranetwork Service that the Collector datagram traverses to integrate the estimated performance components along the path or paths), while Router j is the Departure Router for the exchange.

<u>All CServ-enabled Routers Participating in Collector Protocol: Router $r \in N_R$ CSPU</u>

- Create and maintain a CSPU process for maintaining the current value of the following performance estimates and updating the values as new estimates are generated:
    - Last Hop Transmission Reliability
    - Traversal Reliability
    - Mean Traversal Delay
    - Traversal Delay Variance

- During a particular learning interval indexed τ_i:
  - Maintain a count of the total arriving datagrams in any class (best effort and CServ, including Collector datagrams) on all input interfaces, N_r
  - Maintain a count of the total arriving datagrams in any class on all input interfaces that pass physical and data link layer integrity checks (along with any forward error correction) and are not dropped due to corruption, X_r
  - Maintain a count of the total datagrams in any class dequeued from output buffers on all output interfaces for transmission, D_r
  - Maintain a count of the total datagrams in any class enqueued in input buffers on all input interfaces, E_r
  - For each datagram in any class enqueued in an input buffer of any input interface, tag it with a metadata timestamp representing the current time (this may be done by the NPU for switch scheduling and optimization purposes)
  - For each datagram $x \in \{1,2,\dots,D_r\}$ in any class dequeued from an output buffer of any output interface, compute the traversal delay value for that datagram, t_x, as the difference between the current time and the metadata timestamp
  - Maintain running estimates of Mean Traversal Delay and Traversal Delay Variance using cumulative moving average and online estimator of population variance of the forms described in Eq. (4.2) and Eq. (4.3) and the $t_x, x \in \{1,2,\dots,D_r\}$ values
- At the end of each learning interval indexed τ_i:
  - Finalize the running estimates of Mean Traversal Delay and Traversal Delay Variance, updating the current value maintained by the CSPU
  - Generate a new estimate for Last Hop Transmission Reliability as X_r/N_r, updating the current value maintained by the CSPU
  - Generate a new current estimate for Traversal Reliability as D_r/E_r, updating the current value maintained by the CSPU
  - Reset the counts for the next learning interval (i.e. $X_r = 0$, $N_r = 0$, $D_r = 0$, $E_r = 0$)
  - Reinitialize the running estimates of Mean Traversal Delay and Traversal Delay Variance

- Upon the successful arrival of a Collector datagram, perform the following "collection" processing:
    - If $r = i$ (this is Entrance Router for the Collector exchange, or for the CServ Intranetwork Service):
        - Update the Reliability field with the product of its current value (should be 1) and the current Traversal Reliability estimate value
        - Update the Delay Mean field with the sum of its current value (should be 0) and the current Mean Traversal Delay estimate value
        - Update the Delay Variance field with the sum of its current value (should be 0) and the current Traversal Delay Variance estimate value
    - Else if $r = j$ (this is Departure Router for the Collector exchange, or for the CServ Intranetwork Service):
        - Update the Reliability field with the product of its current value and the current Last Hop Transmission Reliability estimate value
        - Update the Delay Mean field with the sum of its current value and the difference between the current time and the time stored in the Timestamp field (the last hop's realized transmission and propagation delay)
    - Otherwise ($r \neq i$ and $r \neq j$, this is an intermediate router for the Collector exchange and part of the CServ Intranetwork Service path):
        - Update the Reliability field with the product of its current value, the current Last Hop Transmission Reliability estimate value, and the current Traversal Reliability estimate value
        - Update the Delay Mean field with the sum of its current value, the difference between the current time and the time stored in the Timestamp field (the last hop's realized transmission and propagation delay), and the current Mean Traversal Delay estimate value
        - Update the Delay Variance field with the sum of its current value and the current Traversal Delay Variance estimate value
- Upon dequeueing a Collector datagram from the output buffer of an output interface for transmission to the next router in the CServ Intranetwork Service path, update the Timestamp field with the value that represents the current time

Entrance Router (Router i) CSPU Responsibilities for Collector Datagram Generation

- Create and maintain a CSPU process for a Collector exchange with each Router $j \in N_R \backslash \{i\}$ using the address set N_R distributed by the SC
    - Terminate any existing Collector exchange CSPU process with Router l if it is removed on latest update of address set N_R distributed by the SC (i.e. $l \notin N_R$)
- Maintain 28-bit incrementing, wrap-around counter for Router j that used as a sequence number
- Generate one Collector datagram destined for Departure Router j during each learning interval of length τ (one per learning interval is the default, but it could be less often if the protocol is modified to reduce the collection rate)
    - Generate a dummy internetwork CServ datagram that mimics a real internetwork CServ datagram with the following specifications:
        - Version – Use the encoding that specifies the CServ Internetwork Service header consistent with a real passive internetwork CServ datagram
        - Next Header – Use the default Next Header code that indicates that the encapsulated payload is the CServ message data (which is a dummy placeholder in this case)
        - Path Length – Encode the decimal value 0 in this field (there is no real CServ Internetwork Service path since this is a dummy internetwork CServ datagram used only for the purpose of the Collector protocol)
        - Payload/Data Length – Use the correct value to describe the size of the dummy CServ message data payload in bytes (default is one kilobyte, but this can vary between implementations depending on the needs and capability of the subnet)
        - Checksum – This field is unnecessary and can be set to all zeros since the CServ Internetwork Service header integrity check is only used at real CServ transaction endpoint hosts and the integrity check for CServ Intranetwork Service is either at the physical or data link layers (depending on the CServ Intranetwork Service implementation)
        - Source Address – Use the address of Router i, the Entrance Router for the CServ Intranetwork Service and the "source" of the Collector datagram exchange

- Destination Address – Use the address of Router j, the Departure Router for the CServ Intranetwork Service and the "destination" of the Collector datagram exchange
- Sequence Number – Use some subnet-specific default value in this field to further indicate that this is a dummy Collector datagram probe in the Collector protocol and not a real passive CServ datagram
- Expiration Time – Use some default value in this field since it is not examined by the Departure Router j
- Authentication Mark – For now, use some default value since this field is not used by Departure Router j (however, if necessary, this could be employed in the future to authenticate the dummy Collector datagram as a legitimate probe generated by Entrance Router i)
- Hop Addresses – There are no additional hop addresses encoded since there is no CServ Internetwork Service path for dummy Collector datagram probes in the Collector protocol and the value encoded in the Path Length field is zero
- CServ Data Payload – Fill the payload with the appropriate number of bytes encoded in the Payload/Data Length field (typically all zeros, but depends on the subnet's implementation of the Collector protocol)

- Prepend a State Measurement Service header for Collector datagrams and set the fields as follows:
  - Version – Use the correct encoding that represents a State Measurement Service header for a Collector datagram (this is the primary indicator to any intermediate router or to Departure Router j that this is a dummy Collector datagram probe)
  - Next Header – Use the encoding that represents the CServ Internetwork Service header for internetwork CServ datagrams
  - Diversity Degree – With the lookup for appropriate CServ Intranetwork Service for the "destination," Departure Router j, encode the correct value here representing the number of diverse paths that are used for the service or, equivalently, the number of Collector datagram replications are needed for the particular Collector exchange

- Sequence Number – Use the next value of the incrementing counter for Router *j* (and then increment the counter state for the CSPU process)
- Timestamp – Mark this field with the current time, although this initialization value is not used during the collection process
- Reliability – Initialize the value of this collection field to the decimal value 1, encoded as a double-precision floating point number, as discussed in Sections 4.3.3 and 4.3.5
- Delay Mean – Initialize the value of this collection field to the decimal value 0, encoded as a double-precision floating point number, as discussed in Sections 4.3.3 and 4.3.5
- Delay Variance – Initialize the value of this collection field to the decimal value 0, encoded as a double-precision floating point number, as discussed in Sections 4.3.3 and 4.3.5

- If the appropriate CServ Intranetwork Service for Departure Router *j* specifies the use of diversity routing over multiple CServ Intranetwork Service paths, replicate the state measurement collector datagram the appropriate number of times (to match the value encoded in the Diversity Degree field) prior to prepending the appropriate CServ Intranetwork Service headers for each
  - Note that this follows the design that each replication should have the same values encoded in the Diversity Degree and Sequence Number fields
- For each replication of the state measurement collector datagram (if any), prepend the appropriate CServ Intranetwork Service header as specified by the service lookup for Departure Router *j*
- Enqueue the dummy Collector datagram probes in the input buffer of random input interfaces (using the CServ priority queue if available, but not necessary) since there is no "arriving" interface
  - As with active dummy CServ datagrams in Learning Sessions, these interfaces should be chosen independently for each transmitted Collector datagram and uniformly over all possible Entrance Router i interface input buffers, although this is just to spread the Collector datagram traffic load rather than to avoid interface-specific measurement dependencies

- Continue with all additional necessary Collector protocol processing as described under All CServ-enabled Routers Participating in Collector Protocol (namely, with the exception of updating collection fields with any last hop transmission performance values)

Departure Router (Router j) CSPU Responsibilities for Collector Datagram Termination

- Create and maintain a CSPU process for a Collector exchange with each Router $i \in N_R \backslash \{j\}$ using the address set N_R distributed by the SC
  - Terminate any existing Collector exchange CSPU process with Router l if it is removed on latest update of address set N_R distributed by the SC (i.e. $l \notin N_R$)
- Maintain 28-bit incrementing, wrap-around counter for Router i that reflects the largest Sequence Number field value seen in the State Measurement Service headers of arriving Collector datagrams from Entrance Router i
- Maintain values for the last reported CServ performance metric estimates (specifically, Reliability, Mean Delay, and Delay Variance) for the CServ Intranetwork Service connecting Entrance Router i to Departure Router j
- Upon arrival of Collector datagram and after necessary Collector protocol processing as described under All CServ-enabled Routers Participating in Collector Protocol (namely, update collection fields with last hop transmission performance values and update datagram arrival counters as appropriate), continue with the following protocol processing steps
  - If the value encoded in Diversity Degree field is the decimal value 1:
    - Extract the finalized collected values in the Sequence Number, Reliability, Delay Mean, and Delay Variance fields, then discard the Collector datagram
    - Verify that the value of the Sequence Number is greater than the counter, and if so:
      - Update the counter to reflect the new value
      - Report the new CServ performance metric estimates for the CServ Intranetwork Service with Entrance Router i to the SC if the new values are different enough from the stored last reported values (see further discussion of this in Section 4.6)

- Update the stored last reported CServ performance metrics for the CServ Intranetwork Service with Entrance Router i if the new collected estimates are reported

- Else if the value encoded in the Diversity Degree field is a decimal value $k > 1$:
  - Extract and store the set of finalized collected values in the Diversity Degree, Sequence Number, Reliability, Delay Mean, and Delay Variance fields, then discard the Collector datagram
  - Verify that the value of the Sequence Number is either greater than or equal to the value of the counter, and if so:
    - If the counter value is strictly *less than* the Sequence Number value, update the counter and discard any stored sets of collected values with an associated Sequence Number less than the new counter value
    - Calculate the Path Delay estimate as described in Eq. (4.9) and store with the extracted set of collected values
    - If there are now k sets of collected values with the same Sequence Number with the addition of this new set (along with the k computed Path Delay estimates), generate CServ Intranetwork Service Reliability, Mean Delay, and Delay Variance estimates using the k sets according to Eq. (4.8) and Eq. (4.10)
    - Report the new CServ performance metric estimates for the CServ Intranetwork Service with Entrance Router i to the SC if the new values are different enough from the stored last reported values (see further discussion of this in Section 4.6)
    - Update the stored last reported CServ performance metrics for the CServ Intranetwork Service with Entrance Router i if the new collected estimates are reported

4.3.7 Analysis of the Collector Protocol Subnet Burden

The Collector protocol exchanges Collector datagrams at regular intervals between each pair of active CServ-enabled routers, both access and gateway routers, in the subnet. It is important to bear in mind that not all active CServ-enabled routers are required to serve as endpoints in Collector datagram

exchanges, only those access routers that provide upstream routing core access to CServ-enabled hosts and active gateway routers. This is an especially important consideration for the subnet implementing the Collector protocol for the State Measurement Service since CServ Intranetwork Service paths must consist exclusively of active CServ-enabled router hops. Thus, it requires a much more ubiquitous adoption of active CServ-enabled routers to support the CServ Intranetwork Service paths and ensure connectivity. Even though the penetration of upgraded routers in the routing core may far exceed the penetration rate in subnets leveraging the Learning Session protocol, this consideration limits the number of Collector exchanges required.

Although the Collector protocol is designed to levy a very light transmission and switching burden compared to the relatively-heavy fixed-rate sessions of the Learning Session protocol, it still imposes some overhead on the subnet routing core's data plane. We shadow the analysis of the Learning Session protocol from Section 4.2.8 here, omitting most of the derivation details since they directly follow the previous results. This enables us to compare the burden of this protocol to that of the Learning Session protocol. As in the previous analysis, we continue to employ the three assumptions first presented in Section 4.1 to ensure the tractability of the analysis on our two "lower" and "upper" bound subnet topologies. We update these assumptions in the context of the Collector protocol as follows:

1. All routers in the subnet routing core topology are either active CServ-enabled access routers participating in the Collector datagram exchange or active CServ-enabled gateway routers participating in the Collector protocol.

2. The CServ Intranetwork Service routing strategy used for the purpose of analysis is that of shortest-path routing, where the shortest path is defined by the fewest number of intermediate router hops in the path.

3. The CServ Intranetwork Service does not employ path diversity between Entrance and Departure routers. There is one unique CServ Intranetwork Service path connecting any pair of active routers in the subnet routing core as described by the second assumption above.

Note that this last assumption implies that our analysis finds the minimum necessary transmission and switching data plane burden of the Collector protocol in the case where all routers in the subnet routing

core topology are endpoint participants in the protocol. With the addition of diversity routing in the CServ Intranetwork Service, the data plane burden of the protocol increases as a function of the degree of diversity employed.

4.3.7.1 Collector Protocol Burden: The Petersen Graph

As for the Learning Session protocol, we consider the Petersen graph for the subnet routing topology as a form of "lower" bound for the analysis of the Collector protocol data plane burden (this was motivated previously in Section 4.1). The Petersen graph topology is shown in Fig. 4-2, where we assume that all vertices are active CServ-enabled routers participating as endpoints in the State Measurement Service Collector protocol (the distinction between access routers and gateway routers is immaterial for the purposes of this analysis).

Based on the modeling assumptions, each router in the routing core generates periodic Collector datagrams to exchange with the other nine routers. Paralleling the Learning Session analysis, we observe that, of these nine periodic exchanges, one-third of them do not traverse any intermediate routers along their CServ Intranetwork Service path, while the other two-thirds traverse exactly one intermediate routers on their CServ Intranetwork Service path. This can be seen readily in Fig. 4-7. Following the derivation and terminology from before, we can conclude that the $n_R(n_R - 1) = 90$ aggregate periodic Collector datagram exchanges in the subnet generate 150 units worth of Collector datagram exchange traffic. By the strong regularity of the topology, the 90 aggregate periodic Collector datagram exchanges are distributed uniformly over the 30 logical directed links. Thus, each logical directed link bears the burden of 5 traffic units worth of the aggregate periodic Collector datagram exchanges.

We use the same notation as in Section 4.2.8.1 with the substitution of "Collector datagram exchange traffic units" for "Learning Session traffic units." Namely, we define the average number of Collector datagram exchange traffic units per logical directed link in the Petersen graph topology as $C^{Petersen}$, where we have that $C^{Petersen} = 5$. As before, this is actually an exact value, not an average, in the case of the Petersen subnet routing core topology. The "average" nomenclature is useful for the analysis of the "upper" bound case next. We also define the average data rate of aggregate Collector datagram exchange traffic carried per logical directed link in the Petersen graph topology as $C_{r-c}^{Petersen}$ (this is also

an exact value, not an average, in the case of the Petersen graph topology). For the purpose of analysis, we assume a uniform size for all Collector datagrams. This is a very reasonable assumption as all Collector datagrams are dummy active probes generated specifically for the purpose of the protocol, and thus the generation process can employ a standard dummy CServ message payload such that the datagrams are all the same size. Using similar terminology as before, we define the size of the CServ Collector datagram as b_c [bits/datagram]. With the Collector protocol rate λ_C [Collector datagrams/second], we further define the Collector protocol data rate as R_C [bits/second], where we have that:

$$R_C = \lambda_C \times b_c.$$

Employ the Collector protocol default (and maximum) of one Collector datagram exchange per learning interval τ, we have that $\lambda_C = 1/\tau$ (as explained in Section 4.3.4). With this, we can further specify the Collector protocol data rate as follows:

$$R_C = \frac{b_c}{\tau}. \tag{4.11}$$

With these definitions, we have that the aggregate Collector protocol data rate that traverses each logical directed link in a Petersen graph routing core topology is:

$$\begin{aligned} C_{r-c}^{Petersen} &= C^{Petersen} \times R_C \\ &= C^{Petersen} \times \frac{b_c}{\tau} \\ &= \frac{5b_c}{\tau}. \end{aligned} \tag{4.12}$$

To get a sense of the aggregate traffic burden of the protocol, we need reasonable values for b_c and τ. Considering the analysis of Chapter 3 and the very small difference between the State Measurement Service header overhead for Learning Session datagrams versus Collector datagrams (a difference of only 24 octets or 192 bits) relative to the CServ message payload size of one kilobyte or 8,000 bits, we continue to use an approximate value of $b_c = b_s = 12{,}000$ [bits]. And as before, the choice of τ can be

viewed as the granularity at which the protocol can generate new CServ performance metric estimates and react to changes in the subnet state. Refer to the discussion of the subnet's "coherence time" in Section 4.2.8.1 for more consideration. For the purpose of analysis and comparison with the previous results, we continue to use $\tau = 1$ [second]. With these parameter choices and the expression derived in Eq. (4.12), we have:

$$C_{r-c}^{Petersen} = \frac{5 \times 12{,}000}{1} = 60 \text{ kbps.}$$

This is precisely a reduction of two orders of magnitude in the state measurement service protocol transmission burden compared to $C_{r-s}^{Petersen}$ using the same parameters, a difference which results from the two orders of magnitude difference in the Collector protocol rate compared to the Learning Session rate. This is a much more reasonable state measurement protocol transmission burden for subnet's with less transmission capacity, such as edge wireless subnets operating in challenged environments. We note that the transmission burden of the Collector protocol can be reduced even further by reducing the Collector protocol rate (exchanging Collector datagrams only every few learning intervals) or increasing the learning interval in more stable subnets.

As before, we can also consider the data plane switching burden of the Collector protocol, or the aggregate rate of Collector datagram traffic that each router in the subnet routing core topology must switch to maintain the exchange of the Collectors. Using the same argument as presented in Section 4.2.8.1, a total of 15 traffic units of Collector protocol traffic need to be switched by any given router in the Petersen graph routing core topology. With the same Collector protocol data rate from above, we conclude that the switching throughput for each router in the subnet routing core must be 180 kbps just to support the State Measurement Service Collector protocol. Considering the "10% rule" of Section 4.2.8.1, this implies that the subnet routing core would require active CServ-enabled routers with switching throughputs of at least 1.8 Mbps (a very reasonable switching capacity for modern routers, considering some typical industry performance values [76]), a reduction of two orders of magnitude compared to the Learning Session protocol case. However, in the case of the Collector protocol, the State Measurement Service does not subsume real CServ datagram traffic. From that point of view, the "10% rule" should be reduced (implying the need for larger switching capacity) since the headroom now must accommodate both best effort traffic *and* the live CServ datagrams with critical messages. The

minimal state measurement service transmission and switching burden of the Collector protocol leaves plenty of opportunity for dimensioning both the subnet's transmission and switching facilities to allow for significant CServ traffic and best effort traffic rates.

4.3.7.2 Collector Protocol Burden: The Line Network Graph

As with the Learning Session protocol data plane transmission and switching burden analysis, we use the line network graph as an "upper" bound to contrast the optimistic analysis of the Petersen graph as the subnet's routing core topology. The subnet line network topology is depicted in Fig. 4-1, where we assume that all vertices are active CServ-enabled routers participating as endpoints in the State Measurement Service Collector protocol (again, the distinction between access routers and gateway routers is immaterial for the purposes of this analysis). Further description of the basic properties of the graph was presented in Section 4.2.8.2.

Our modeling assumptions imply that each router in the routing core generates periodic Collector datagrams to exchange with the other nine routers in the subnet routing core topology. If we considering the paths of the $n_R(n_R - 1) = 90$ aggregate periodic Collector datagram exchanges in the subnet, we see the same distribution as with the Learning Session protocol analysis as illustrated in Fig. 4-8. The logical directed links between Router 5 and Router 6 bear the heaviest transmission burden in terms of aggregate Collector datagram exchange traffic, each carrying Collector datagrams from a total of 25 exchanges. Just as in the Learning Session analysis, the average Collector datagram exchange traverses 11/3 hops and thus generates 11/3 traffic units worth of transmission burden on the line graph routing core topology.

Mimicking the nomenclature of Section 4.2.8.2, we define the average number of Collector datagram exchange traffic units per logical directed link in the line network graph topology as C^{Line}, and we have that $C^{Line} = 55/3$ from before. Considering, additionally, the worst-case logical directed links in the topology (the links connecting Router 5 and Router 6), we define the maximum number of Collector datagram exchange traffic units per logical directed link in the line network graph topology as $C^{Line-max}$. From Section 4.2.8.2, we also know that $C^{Line-max} = 25$. We denote the equivalent average and maximum Collector protocol data rate for the aggregate Collector datagram exchange

traffic carried per logical directed link in the line network graph topology as C_{r-c}^{Line} and $C_{r-c}^{Line-max}$, respectively.

Following the analyses of Sections 4.2.8.2 and 4.3.7.1, we have the following expressions for C_{r-c}^{Line} and $C_{r-c}^{Line-max}$:

$$\begin{aligned} C_{r-c}^{Line} &= C^{Line} \times R_C \\ &= C^{Line} \times \frac{b_c}{\tau} \\ &= \frac{55 b_c}{3\tau}; \end{aligned} \tag{4.13}$$

$$\begin{aligned} C_{r-c}^{Line-max} &= C^{Line-max} \times R_C \\ &= C^{Line-max} \times \frac{b_c}{\tau} \\ &= \frac{25 b_c}{\tau}. \end{aligned} \tag{4.14}$$

With these results and the same parameter choices (and approximate Collector datagram size) as used in Section 4.3.7.1, we have:

$$C_{r-c}^{Line} = \frac{55 \times 12{,}000}{3 \times 1} = 220 \text{ kbps};$$
$$C_{r-c}^{Line-max} = \frac{25 \times 12{,}000}{1} = 300 \text{ kbps}.$$

Again, we observe a reduction in the data plane transmission burden by two orders of magnitude compared to the Learning Session protocol used for the same subnet routing core topology. This difference is a result of the two orders of magnitude reduction in the Collector protocol rate compared to the Learning Session rate, all other parameters equal. The worst-case links have transmission burdens under 1 Mbps, indicating the Collector protocol is much better suited to implement the State Measurement Service for challenged subnets with minimal transmission capacity and uncontrollable routing core topologies (for example, an infrastructureless multi-hop wireless network). As discussed at the end of Section 4.3.7.1, there are several opportunities to reduce the data plane transmission burden

further using a more minimalist version of the Collector protocol which employs less frequent Collector datagram exchanges between CServ Intranetwork Service endpoints.

Finally, we consider the data plane switching burden of the Collector protocol for the line network graph routing core topology, our "upper" bound topology. Following the analysis of Section 4.2.8.2, we focus on the worst-case routers for this discussion, specifically Router 5 and Router 6 in Fig. 4-8. For each of these routers, there are a total of 49 traffic units of Collector datagram exchange traffic that must be switched to maintain the periodic subnet routing core exchanges. With the same parameters as before, we use the Collector protocol data rate to discover that the switching throughput for these worst-case active CServ-enabled routers must be 588 kbps to support only the Collector protocol exchanges. Although this worst-case requires slightly more than three times more switching bandwidth than the "lower" bound Petersen graph routing core case, this still represents a very reasonable switching capacity for modern routers even after applying the "10% rule" (which implies that the router switching throughputs should be at least 5.88 Mbps to allow headroom for the service of best effort traffic). We note once again, however, that the Collector protocol does not carry live CServ datagram traffic; rather, it uses active dummy probes exclusively. So the "10% rule" should be adjusted slightly to account for the shift of the real passive CServ traffic from the Learning Sessions to the transmission and switching facilities overhead dimensioned by this informal guideline.

4.4 Comparison of State Measurement Protocols

In this section, we summarize the qualitative and quantitative analyses of the two proposed State Measurement Service protocols from Sections 4.2-4.3 before presenting their suggested use.

The Learning Session protocol was motivated by the desire to learn high-fidelity CServ performance metrics for CServ Intranetwork Service paths that very closely represent the true experience of a real internetwork CServ datagram using the CServ Intranetwork Service to traverse a subnet. To do so, we deploy a fixed-rate session between the CServ Intranetwork Service endpoints, the Entrance and Departure routers, composed of a mixture of real CServ traffic and active dummy probes in the CServ traffic class. Capturing the interplay with other Learning Sessions in the subnet, including the associated congestion loss, critical datagram corruption rate, and queueing delays, these sessions closely estimate the performance of a real CServ datagram using that particular CServ Intranetwork Service in the future

(noting that this requires some level of subnet state coherency, as does most all other aspects of the CServ architecture). These real CServ datagrams are substituted for active dummy probes in the session and thus receive the same general performance experience as estimated, while simultaneously helping to discover any changes in CServ performance metric state for future critical datagrams using the CServ Intranetwork Service. In Chapter 5, it becomes clear that these accurate CServ performance metric estimates enable the MC to more frequently satisfy CServ Requests (CSRs) and provide reliable network service to internetwork CServ transactions.

However, this State Measurement Service design does introduce a feeling of establishing "circuit switched" paths between the CServ Intranetwork Service endpoints, especially since the recommendations are to grant priority switching and transmission for CServ datagrams where possible and to dimension the subnet such that the aggregate Learning Session traffic only accounts for at most 10% of the transmission or switching bandwidth at any routing core link or router. The difference between the Learning Session approach and circuit switching is that there is no explicit resource reservation, either for transmission or switching facilities, along the CServ Intranetwork Service paths (although it is possible that the CServ Intranetwork Service, not the Learning Session protocol, employs resource reservation for virtual circuits, such as with MPLS). During unprecedented "Black Swan" events, transient bursts exceeding the Learning Session rate of real critical message traffic on a CServ Intranetwork Service could affect intersecting Learning Sessions and the subsequent CServ performance metric estimations since those Learning Sessions are not guaranteed any particular share of the network resources. Furthermore, and maybe more importantly, is that the relatively high rate Learning Sessions compared to expected real CServ traffic means that a significant subnet bandwidth is used (although not reserved) regularly to maintain the State Measurement Service process and prepare for normal influxes of real critical message transactions. Table 4.1 summarizes the analytic results from Section 4.2.8 on the data plane burden of Learning Sessions for representative "upper" and "lower" bound core routing topologies.

At the risk of approximating a circuit switched design, the Learning Session protocol requires relatively simple processing to maintain the State Measurement Service process. The Entrance Router is required to do little more than to generate dummy active Learning Session datagram probes to maintain the required Learning Session rate with its Learning Session protocol peers in the subnet, substituting the dummy probes with real traversing internetwork CServ datagrams when they arrive. Additionally, the

CServ performance state estimators at the Departure Router have simple forms as introduced in Section 4.2.6. Meanwhile, intermediate routers along the CServ Intranetwork Service paths connecting Entrance and Departure Routers have no protocol responsibilities. The use of the Learning Session protocol is obscured from these routers, as the Learning Session datagrams are encapsulated in the native protocol datagrams for CServ Intranetwork Service. From this point of view, subnet routers that are not active gateways nor access routers providing upstream access for CServ-enabled end hosts do not need to be upgraded or run CServ-related processes related to the Learning Session protocol. These can be legacy routers that only understand the routing and forwarding rules for CServ Intranetwork Service (such as IPv6 routing and forwarding), greatly reducing the implementation costs of the Learning Session protocol. CServ Intranetwork Service paths can be routed through intermediate routers without CServ-enabled capabilities, which improves routing core connectivity in subnets that cannot deploy active CServ-enabled routers universally. The Learning Session design does require some prearrangement of protocol-related parameters between the active routers participating in the protocol (such as the Learning Session rate and learning interval, see Section 4.2.5), but these parameters can be synced by the SC and distributed as part of the CServ association list.

To mitigate the cumbersome data plane transmission and switching burden of the Learning Session protocol, we introduced an alternative approach to the State Measurement Service, the Collector protocol. This approach substitutes CServ-specific performance tracking responsibilities at all routers in the set of CServ Intranetwork Service paths for the relatively high rate sessions that shoulder the workload in the Learning Session protocol. Each router that participates in CServ Intranetwork Service is required to run CServ processes related to the Collector protocol, estimating a set of performance metrics related to the reliability and delay incurred by traversing the router. To do so, these routers leverage whatever traffic they have access to without the introduction of any significant set of passive probes, meaning that the performance estimates are primarily generated from best effort traffic. Infrequent Collector datagrams are exchanged between CServ Intranetwork Service endpoints (i.e. Entrance and Departure Routers) to aggregate the performance estimates along the CServ Intranetwork Service paths logically connecting the endpoints. This protocol also reduces the parameter dimensionality under the control of the subnet administrator. The only significant parameter of the protocol is the learning interval, which does not even require strict synchronization between protocol participants. The Collector protocol can perform even if constituent routers in the CServ Intranetwork Service path use different learning interval lengths to generate new performance estimates. Although

we choose a default for the frequency of Collector datagram exchanges between CServ Intranetwork Service endpoints, the rate can be reduced if necessary and also does not need to be strictly agreed upon by all participants.

This protocol very nearly eliminates the addition of data plane transmission and switching overhead for the purpose of the State Measurement Service (see the analytic results from Section 4.3.7 summarized in Table 4.1), but it does so with several compromises. First and foremost, the estimated performance at each router does not represent the ground truth for the experience of CServ datagrams. Since CServ datagrams receive priority queueing and switching when available, the statistics computed from the nominal best effort traffic represents a very conservative performance estimate. In general, the experience of a real CServ datagram traversing the router will outperform the reported reliability and delay metrics using this approach. As we see in Chapter 5, these loose resulting CServ performance metric estimates from the Collector protocol make it more challenging for the MC to satisfy the service demands of "tight" CSRs riding the edge of the network capabilities, leading to a higher likelihood of internetwork service denial.

Additionally, the Collector protocol approach requires much more complex subnet router processing. The responsibilities specific to the Entrance Router are not much more significant than in the Learning Session case, but the Departure Router must generate intermediate calculations and perform comparisons over multiple Collector datagrams to determine a final CServ performance metric estimate when diversity routing is leveraged for the CServ Intranetwork Service. More importantly, all routers participating in the Collector protocol must maintain CServ-specific running estimates of performance over all arriving and traversing datagrams, refreshing these estimates each learning interval. This requires that all routers participating in the Collector protocol be upgraded with a CSPU for CServ-proprietary processes. This implies that the deployment of the Collector protocol is a more expensive endeavor than that of the Learning Session protocol. If a subnet administrator cannot afford to upgrade all routers in the routing core, this impacts the richness of the CServ Intranetwork Service connectivity graph structure and the opportunities for robust traffic engineering.

The preceding discussion is summarized in Table 4.2. The recommendation for the subnet routing core is to use the Learning Session protocol where the subnet data plane capabilities render this feasible, such as for a subnet with a robust fiber optic backbone. The precise CServ datagram performance estimation

makes this the more attractive option. However, for subnets that cannot dimension the data plane facilities to accommodate the Learning Session burden or operate over physical transmission modalities in situations that restrict their physical capabilities (such as edge wireless or satellite subnets), the Collector protocol can be deployed as an alternative approach. The loss of CServ performance metric estimation fidelity is not ideal, but this protocol ensures that the State Measurement Service does not impede subnet operation or the simultaneous support of best effort traffic.

As one more alternative option, it is possible to implement the Learning Session protocol with payload-less active probe datagrams in the Learning Session. By removing the default one kilobyte dummy CServ message payload from these probes, the Learning Session rate is reduced by a multiplicative factor of approximately 1/3 (the CServ message payload cannot be removed from any passive real CServ datagrams that contribute to the Learning Session). This option may help bridge the gap for subnets that cannot handle the transmission and switching data plane burden of the Learning Session protocol. However, the subnet administrator must bear in mind that this approach impacts the integrity of the CServ performance metric estimates; the shorter CServ datagrams in the Learning Session means that there is less tolerance for the influx of real passive CServ datagram traffic, and the shorter datagrams experience and estimate reduced transmission delay and possibly different loss statistics. It may be an alluring option to implement the Learning Session protocol with payload-less dummy probe datagrams, but the operator should also consider the Collector protocol as a practical alternative. Future work could focus on the comparative estimation performance of these approaches in real subnet deployments or simulations to better characterize this trade-off.

The opportunity exists to consider the hybrid adoption of both Learning Session and Collector protocols within one subnet; the designs of these protocols do not preclude the simultaneous use of the other. The subnet administrator then has the flexibility to deploy a Learning Session or Collector datagram exchange selectively for each CServ Intranetwork Service supported by the subnet. However, this quickly increases the administration and management complexity of the State Measurement Service. For example, consider Fig. 4-12 which illustrates the input processing alone for an arriving CServ datagram at a router in a subnet implementing a hybrid of the two State Measurement Services. This processing path is significantly streamlined if either the Learning Session or Collector protocol branches are removed. This hybrid opportunity is not further explored in this dissertation, but it could be considered in the future if this tactic seems most appropriate for a particular subnet.

Table 4.1: This table compares the data plane burden of the two proposed State Measurement Service protocols, the Learning Session protocol and the Collector protocol, for representative "lower" and "upper" bound subnet core routing topologies. All values are in kilobits per second (kbps).

Subnet Routing Core Topology	Date Plane Burden	Learning Session Protocol	Collector Protocol
"Lower" Bound: Petersen Graph	Per-link Transmission	6000	60
	Per-router Switching	18000	180
"Upper" Bound: Line Network Graph	Average Link Transmission	22000	220
	Worst-case Link Transmission	30000	300
	Worst-case Router Switching	58800	588

Table 4.2: This table summarizes the qualitative comparison between the two proposed State Measurement Service protocols, the Learning Session protocol and the Collector protocol. Those categories where the characterization of one protocol is clearly preferred is coded green, while the corresponding protocol characterization is coded red.

Learning Session Protocol	Comparison Category	Collector Protocol
Higher	**Data Plane Burden**	Lower
Higher	**CServ Performance Metric Estimation Accuracy**	Lower
CServ	**Traffic Used for Estimation**	CServ + Best Effort
Approx. up to Learning Session traffic rates	**Robustness Against Real CServ Message Fluctuations**	Approx. up to nominal Best Effort traffic rates
Approx. same	**Entrance Router Responsibilities**	Approx. same
Slightly fewer	**Departure Router Responsibilities**	Slightly more
None	**Additional Intermediate Router Responsibilities**	Some
Lower	**CServ Router Complexity**	Higher
Less pervasive	**Required CServ Router Deployment Density**	More pervasive
Required	**Parameter Synchronization**	Not required

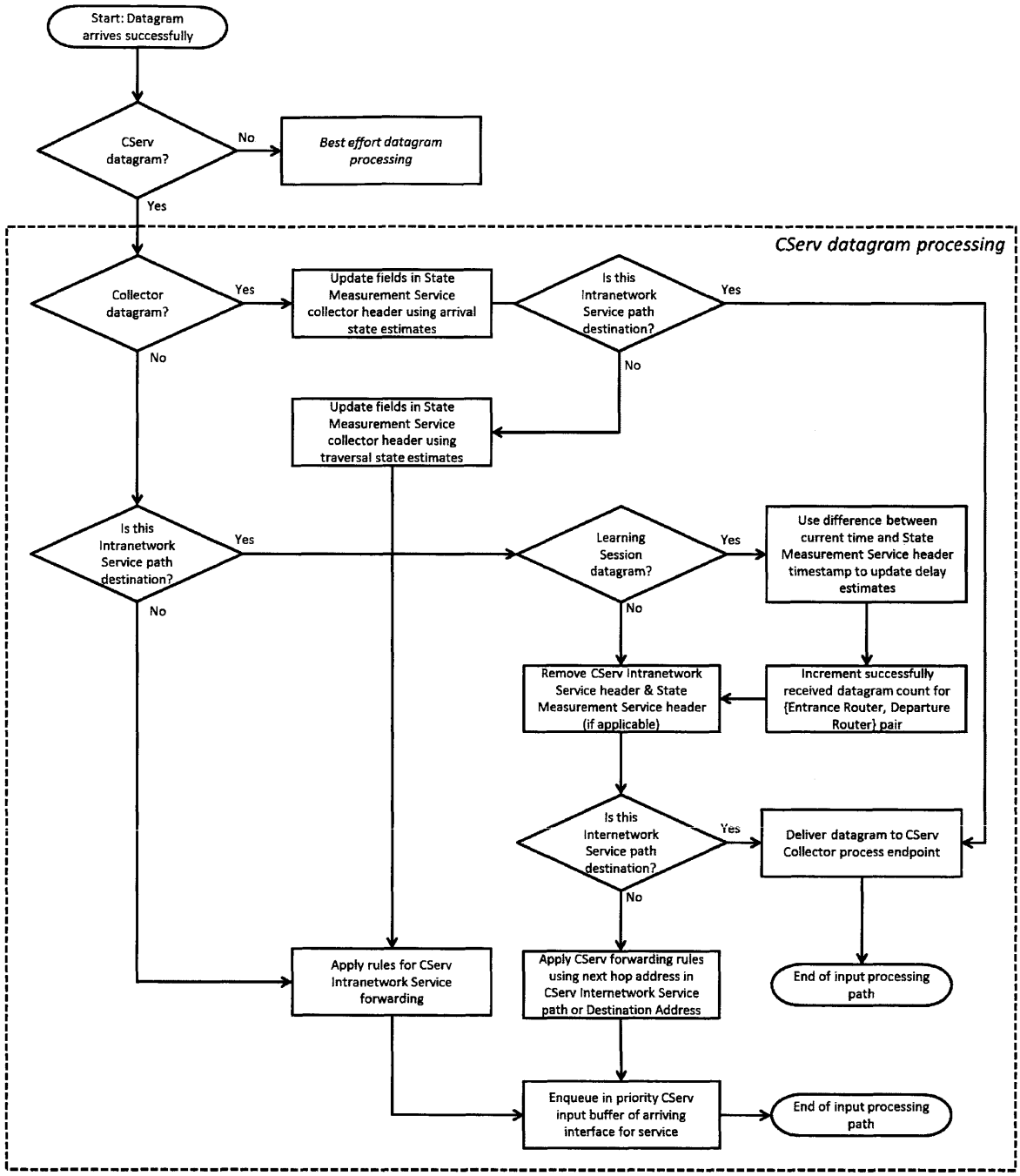


Fig. 4-12: This flowchart depicts the notional input processing at an active CServ-enabled router in a subnet that has deployed a hybrid State Measurement Service using both the Learning Session protocol and the Collector protocol.

4.5 State Measurement Service Outside the Routing Core

The CServ performance metric characterization of the subnet does not end with the routing core that interconnects the active CServ-enabled access and gateway routers. An end-to-end internetwork path between a CServ-enabled source host and a CServ-enabled destination host requires additional hops in the context of the general network model put forth in Chapter 2. Before the critical message transaction reaches the source subnet's routing core, it must traverse the access network infrastructure that connects the host to the upstream access router. An internetwork CServ datagram later leaves the confines of the source subnet and traverses an external active gateway router to active gateway router hop which connects the adjacent subnets. And when the internetwork CServ datagram finally reaches the destination subnet in the CServ Internetwork Service path, it will traverse the downstream hop from the access router to the destination CServ-enabled host. The CServ performance metric estimation of these hops was not covered in our previous discussion of the State Measurement Service protocols, where the focus has been solely on the subnet routing core. We complete the estimation of all hops in the end-to-end CServ Internetwork Service path by addressing these two additional domains for the State Measurement Service:

1. external gateway-to-gateway CServ performance metrics estimation;
2. and upstream and downstream access network CServ performance metrics estimation.

4.5.1 External Gateway-to-Gateway Performance Estimation

Our coverage of CServ performance metric estimation using the State Measurement Service in the subnet routing core includes the estimation of *internal* gateway router to gateway router performance, or the performance of CServ Intranetwork Service connecting two active gateway routers within the same subnet domain. According to the network model presented in Chapter 2, the purpose of the gateway router is to provide connection to a neighboring subnet, interconnecting their respective routing cores. We refer to the interconnection between gateway routers in neighboring subnets as a *subnet peering connection*, or an *external* gateway-to-gateway connection in the context of this chapter. In the terminology of Border Gateway Protocol [12], the status quo best effort protocol for interconnecting subnet domains in an IP network, these are the edge router pairs that would exchange route advertisements via External BGP (eBGP).

Many external gateway-to-gateway pairs are directly connected without any lower-layer switching technologies in the link span, although sometimes switching fabrics are used (such as at large exchange points of presence). Sometimes peering gateway router pairs are even implemented as virtual routers on the same machine, where the interconnection between them is on-rack or even in software (note that, based on the addressing discussion of Chapter 3, these two virtual routers would have their own hierarchical addresses that reflect their membership to separate subnet domains in the CServ architecture). The physical connection between peering gateway routers is frequently a high-speed optical line or other high-speed wired transmission substrate. If the gateway pair is implemented virtually, the interconnection is likely a high-speed switching fabric or high-throughput software process. For these reasons, the interconnection between peering gateways likely presents relatively stable and reliable behavior in terms of the CServ performance metrics. The primary source of variability is the queueing of datagrams while they await access to the output interface that connects to the external (and possibly virtual) gateway router peer.

The neighboring subnet administrators may choose to implement either the Learning Session or Collector protocol between their active gateway peering points depending on their joint capabilities, both of which are illustrated in Fig. 4-13. Both peering subnet operators must agree on the State Measurement Service protocol, where we designate the Learning Session protocol as the default that both must be capable of deploying. With high-bandwidth interconnections, we further suggest the use of the Learning Session protocol as the State Measurement Service for external gateway-to-gateway connections. This suggestion is based on the discussion of Section 4.4 which identifies the Learning Session protocol as the high-fidelity option for estimating CServ performance metrics. Although a subnet's active gateway router may peer with multiple active gateway routers in a neighboring subnet or active gateway routers in multiple neighboring subnets, the set of these external gateway-to-gateway connections is usually not in any danger of scaling uncontrollably and the interconnection topology is normally point-to-point. This means that each directed interconnection link would generally only need to bear the transmission burden of one Learning Session, which is unlikely to place much strain on the link's transmission bandwidth. And with a limited set of peers, the additional data plane switching burden of external Learning Sessions should not overwhelm the bandwidth of the active gateway router's switching bandwidth. That being said, should the implementation of the Learning Session protocol with external gateway peers overtax the router when combined with the switching burden of internal Learning Sessions, the Collector protocol can serve as an alternative State Measurement Service

option. In either case, as with subnet routing core CServ performance metric estimation, it is the role of the Destination Router in the peering pair (the destination or the Learning Session or the exchanged Collector datagram) to generate the finalized CServ performance estimates and report relevant state updates to its respective SC.

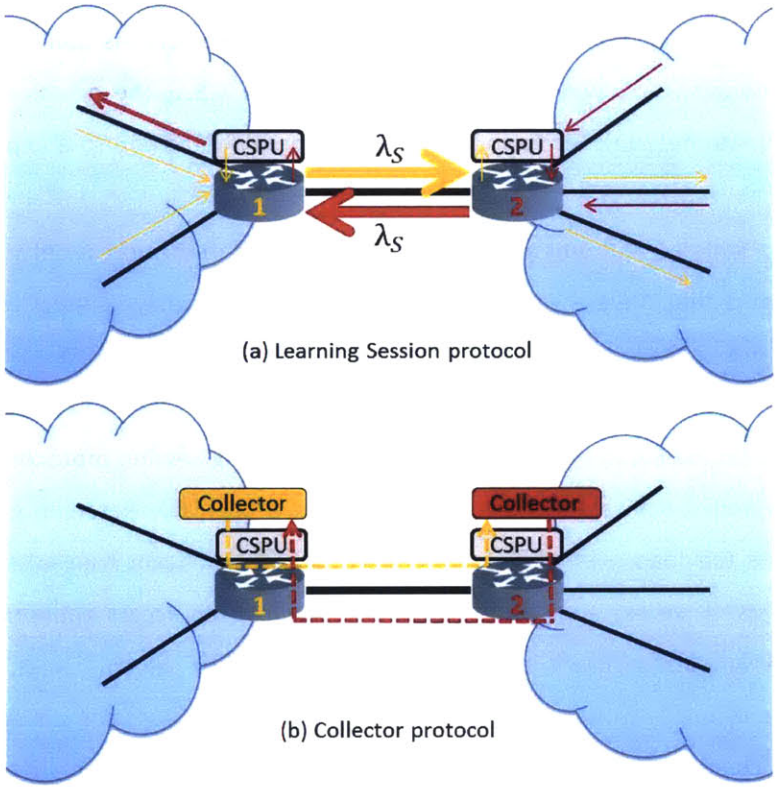


Fig. 4-13: Either the (a) Learning Session protocol or (b) Collector protocol may be implemented as the State Measurement Service between external active gateway router peers based on the joint capabilities of the neighboring subnets.

4.5.2 Access Network Performance Estimation

Although our discussion of the estimation of CServ performance metrics in the subnet routing core includes characterizing the CServ Intranetwork Service paths from active access routers to other active access routers and active gateway routers, it did not comprise the access network hop between the CServ-enabled end host and the active access router itself. As introduced in the network model of Chapter 2, the access network is the local area interconnection between the end host devices and the upstream access router that provides access to the subnet routing core. The technology and topology of access networks may take many forms, from optical rings and trees to an RF base station directly connected to each wireless host device. Additionally, the local infrastructure may include lower-layer devices, such as active switches (such as with Ethernet) or passive splitters and combiners (such as in Passive Optical Networks). Following the coverage of Section 4.5.1, the access network is the last component of CServ Internetwork Service path needed to complete the end-to-end picture.

In deciding upon the State Measurement Service approach suited to the access network, there are three primary considerations that drive the result. First, access networks may connect tens to hundreds of host machines to the routing core via the upstream access router, making scaling considerations essential to the deployment of the State Measurement Service protocol. Recall the analysis of the transmission burden on the subnet routing core using the Learning Session protocol from Section 4.2.8. Even with only ten active devices in the routing core and a favorable Petersen graph topology that efficiently distributes the load over all links, there was still a significant transmission burden on the communication edges. If we extrapolate and scale this result to an access network with hundreds of host devices and likely a less-efficient interconnection topology (for example, links close to the access router root of a tree topology would need to bear many Learning Sessions compared to the links closer to the leaves), it is clear that the Learning Session protocol could impose a restrictive burden on the access network's transmission capabilities. This leads to the second consideration, which is that access networks generally have much less transmission bandwidth than the routing core. Typical access rates for standard access technologies, such as Ethernet, are on the order of hundreds of megabits per second to gigabits per second. However, if these transmission capabilities are shared over hundreds of host devices, we can reduce the bandwidth by multiple orders of magnitude to find the fair-share fraction. From Section 4.2.8, we know that Learning Sessions with reasonable protocol parameters can easily exceed one megabit per second, already consuming much of the share allotted to the host on the more

conservative side of the access technology rate range. Edge wireless access networks typically have even less transmission bandwidth than wired technologies like Ethernet. The third consideration is that, because of the possibly large number of CServ-enabled hosts connected to access networks, we do not want separate CServ performance metric estimates for each pairwise connection between the access router and an individual host. This is a scalability concern for the network as a whole. Even though the total estimated state would only scale linearly with the number of host devices connected to access networks, this could rapidly exceed the total state needed to characterize the CServ Intranetwork Services of the network's subnets. Consider that 90 sets of CServ performance metrics are estimated in a routing core with ten active routers. It would take only 45 CServ-enabled hosts in a subnet to generate the same number CServ performance metric estimates if the design learned the upstream and downstream performance for each host individually. Extending this conclusion to all CServ-enabled hosts in all subnets, the access network CServ performance estimates would quickly dominate the total state the MC must maintain and access.

As a result of the above considerations, we make the following design choices for the State Measurement Service for access networks:

- the Collector protocol is used for the State Measurement Service in access networks with CServ-enabled hosts;
- but the protocol is modified such that CServ performance metric estimates are summarized for upstream and downstream service.

As we have seen, the operation of the Collector protocol generates a minimal amount of protocol traffic, making it well-suited for the limited transmission capacity of access networks. We need to modify the implementation of the Collector protocol, however, because the operation as described for the subnet routing core would generate a pair of upstream and downstream CServ performance metric estimates for each CServ-enabled host in the access network. Instead, we describe how the protocol can be adapted such that one summarized pair of performance estimates are generated per active access router: the *upstream CServ performance metrics* and the *downstream CServ performance metrics*.

The access router is the convergence point for all active CServ-enabled hosts in its attached access network, and thus it is responsible for the generation and reporting of both the upstream and

downstream CServ performance metric estimates. The host machines participate in the modified Collector protocol, but they do not report downstream CServ performance metric estimates directly to the SC. The modified Collector protocol for access networks seeks the worst-case access network connection and uses this to represent the CServ performance metrics of the access network. This characterization is critical to the ability of the MC to compose CServ Internetwork Service for the CServ-enabled hosts in the access network, and thus it is considered in more detail after we provide an overview of the protocol operation.

We assume that the active access router has a set of addresses for the CServ-enabled hosts in its access network. During each learning interval, the access router generates and transmits up to five Collector datagrams downstream to its access network. Two of the Collector datagrams are sent to the hosts that are currently characterized as the "reliability worst-case" hosts, one for downstream performance and one for upstream performance. If one host represents the worst-case for both directions, only one Collector datagram is required. The next two Collector datagrams are sent to the hosts that are currently characterized as the "delay worst-case" hosts, one for downstream performance and one for upstream performance. Again, if one host represents the worst-case for both directions, only one Collector datagram is required. Furthermore, if one of the worst-case hosts is already covered by the reliability worst-case, a second Collector datagram for that host is not needed. The characterization of "delay" in terms of the two CServ performance metric statistics, mean delay and delay variance, is the same as given by Eq. (4.9). The final (and, at a maximum, fifth) Collector datagram is transmitted to a random host in the set of CServ-enabled hosts in the access network, less the up to four worst-case hosts already being probed with Collector datagrams. This host should be chosen independently and uniformly over the set described during each new learning interval. As with the normal Collector protocol, these Collector datagrams are enqueued in the input buffers of random input interfaces of the access router and updated with the appropriate access router traversal statistics. As the Collectors are transmitted to and successfully received by the appropriate active CServ-enabled hosts in the access network, the Transmission and Propagation Delay is calculated and the datagrams are updated with the necessary Last Hop Transmission Reliability estimates at the receiving endpoints to complete the downstream performance estimates.

Active CServ-enabled hosts in the access network do not actively generate and transmit Collector datagrams during each learning interval. Only when a host successfully receives a Collector datagram

from the upstream access router does it generate and transmit a *return Collector datagram* to characterize the upstream access link performance metrics. Specifically, after finalizing the downstream performance estimates, it creates a new return Collector datagram with the downstream performance estimates encapsulated in the datagram as the CServ message payload; this is what distinguishes a return Collector datagram from a typical Collector datagram. The return Collector datagram is then enqueued for priority transmission by the CServ-enabled host and updated with the current estimates of the "traversal" performance metrics. Generally, there should not be much buffer waiting time for transmission unless the CServ-enabled host is currently transmitting a real CServ datagram (the host's transmission of a real CServ transaction should always be prioritized over a return Collector datagram). These "traversal" performance metric estimates for the CServ-enabled host primarily capture channel access waiting times (and possibly ARQ protocol delays) if the access network technology uses a shared communication substrate.

When the return Collector datagram arrives successfully at the access router from the access network (and after being updated with the appropriate delay and reliability estimates for the access network hop), the access network Collector process at the access router's CSPU extracts the two sets of estimates – the downstream estimates from the payload and the upstream estimates from the State Measurement Service header for Collector datagrams. These two sets of collected performance estimates are used to update the downstream CServ performance metric estimates and upstream CServ performance metric estimates, respectively. If the return Collector datagram is from one of the worst-case hosts (either the reliability worst-cases or the delay worst-cases), the current performance metric estimates are updated at the access router CSPU process if the change (either positive or negative) is different enough from the currently stored estimate. To avoid instability, the subnet administrator must choose an appropriate threshold for a metric update, usually as some fixed increment. Otherwise if the return Collector datagram is from a randomly-probed host in the access network, it is only used to update the upstream or downstream performance estimates if it becomes a new worst-case. In other words, it takes over as the new reliability and/or delay worst-case if the estimated upstream or downstream reliability or delay estimates are worse than the currently stored worst-case estimates by the required threshold. In this way, the modified Collector protocol probes for, and tracks, the performance of the worst-case hosts in the access network.

This modified Collector protocol for access networks transmits up to five Collector datagrams in both the downstream and upstream directions during each learning interval. Using the numbers for analysis from Section 4.3.7, this requires at most 60 kbps in the upstream and downstream directions for the State Measurement Service protocol. This represents transmission bandwidth savings over the burden of the standard implementation of the Collector protocol as long as there are more than five active CServ-enabled hosts in the access network. As with the standard Collector protocol, the access network implementation can reduce the transmission frequency of Collector datagrams to every few learning intervals to further reduce the overhead burden if the performance estimates are relatively stable. This avoids frequently probing an access network with no changes in CServ performance metric estimates. Furthermore, the end result of the modified protocol characterizes the CServ performance metric estimates for the access network in terms of only two sets of metrics, the downstream CServ performance metrics and the upstream CServ performance metrics, rather than with a number of sets that scales with the number of active CServ-enabled devices in the access network. These savings come at the expense of slightly more complicated processing at the active access router's CSPU and the worst-case characterization of the access network.

The summarization of upstream and downstream performance into two sets of CServ performance metrics carries the danger that it sacrifices detailed accuracy for the purpose of a low-overhead protocol. The proposed modification to the Collector protocol conservatively searches for a representative worst case among the CServ-enabled hosts connected to the access network and uses the connection to this host to characterize the upstream and/or downstream CServ performance metrics for all hosts in the access network. Although this avoids the use of overly optimistic performance metrics that could lead to misrepresentation of CServ Internetwork Service performance guarantees, this worst-case characterization can paint a pessimistic picture of the access network and potentially challenge the ability of the CServ architecture to discover satisfactory CServ Internetwork Services.

As an example of this, a wireless access network would likely be characterized by the upstream and downstream performance of the device that is furthest from or that has the worst RF channel connection to the access router. Although the difference in delay for this device compared to a device that is closer to the access router may not be substantial compared to the delay of the CServ Internetwork Service path, the datagram transmission loss rate may vary wildly between them. It is worth noting that the use of a well-designed ARQ retransmission algorithm at the data link or media

access control layer should stabilize the reliability between the wireless devices at the expense of increasing the mean delay and delay variability of the hosts. The use of ARQ techniques at the data link or media access control layers is common practice for many wireless access network protocols, such as Wi-Fi [77].

With that example in mind, access networks with CServ-enabled hosts should be designed such that the performance is roughly equalized over all CServ-enabled devices. Although this may be a challenge for some specific access network cases, it should be feasible for most. The general guideline is that CServ-enabled hosts should have favorable connections to the access router such that the access network's estimated upstream and downstream CServ performance metrics are amenable to the successful composition of CServ Internetwork Service paths. This implies that the set of CServ-capable hosts should be "network close" to the access router, increasing the reliability and minimizing the delay, and that any given access router should not be oversubscribed by many CServ-enabled devices. Ideally, an access network providing connection for CServ-enabled hosts should have only a handful of devices connected directly or near directly to the access router without any non-essential (non-active) hosts competing for the access network resources. This access network design and deployment for CServ-enabled devices is critical since an access network hop is part of every valid CServ Internetwork Service path. This is also why multihoming CServ-enabled hosts to multiple access networks (or, better, multiple access networks in different subnets) where possible is strongly recommended. Otherwise, the upstream active access router is the first opportunity for path divergence, and thus the access network hop is the common link between CServ Internetwork Service paths even when internetwork path diversity is employed by the CServ Internetwork Service (see Chapter 5).

As a final note, the need for all active access routers to implement a version of the Collector protocol for access networks may influence the subnet administrator's choice between deploying the Learning Session or Collector protocol as the State Measurement Service in the routing core. In a routing core with a large number of active access routers compared to active gateway routers and strong connectivity, it may be more resource and cost efficient to rely on the Collector protocol as the active access routers are already estimating and tracking their detailed traversal performance metrics.

4.6 CServ Performance Metrics Reporting Responsibilities

Thus far we have focused on the techniques used to learn the performance of CServ Intranetwork Service paths, as well as the external performance between active gateway routers and the upstream and downstream performance of access networks with CServ-enabled endpoints. As introduced in Chapter 2, the purpose of learning how these services perform is to enable the composition of CServ Internetwork Services that satisfy the CServ transaction performance demands of CServ-enable hosts without subnets needing to disclose specific details about their internal topology and state. The State Measurement Service provides the least-invasive way to describe the performance offered by subnet CServ Intranetwork Services, abstracting away the routing protocols, the forwarding methodologies, the traffic load, the subnet topology and capabilities, and the composition of the paths connecting the external gateway interfaces. However, in order to leverage the output of the State Measurement Service (see Chapter 5), the CServ performance estimates need to be reported to the CServ architecture control hierarchy, specifically the subnet's own SC and the logically-centralized global level MC. We discuss this final component of the state learning protocols in this section, highlighting the set of estimated CServ performance metrics that needs to be reported locally to the SC and the subset that needs to be relayed from the SC to the MC.

We begin with a discussion on the conditions under which new CServ performance estimates need to be reported to the control hierarchy. Intuitively, reporting updates every learning interval, particularly if the new estimates are redundant (essentially the same as previously reported values), could overwhelm the off-band control channel and the SC and MC with state updates. Both the Learning Session protocol and the Collector protocol have the capability to generate new CServ performance estimates each learning interval; the Learning Session protocol by definition has the Departure Router endpoints generate new estimates each learning interval, and the Collector protocol transmits a Collector datagram exchange over each CServ Intranetwork Service (or access network) up to once a learning interval and thus can generate new CServ Intranetwork performance estimates with the rate of up to once per interval.

However, the Departure Routers for the CServ Intranetwork Services (or Departure Routers for an external gateway-to-gateway peering connection, or the active access routers) do not report each generated set of CServ performance metric estimates. Rather, these devices track the last reported

values and only report and update that set when the new estimated set is considered different enough from the previously reported set. We do not specify what must be considered "different enough" in this dissertation. This is a decision that can be made by each subnet administrator depending on the relative state stability of their subnet. In general, though, this should be characterized as some deviation in any of the three metrics (reliability, mean delay, or delay variance) by some absolute increment or by some fractional amount from the last reported value. If one estimated metric value meets the reporting criterion, the whole updated set of metrics for that particular CServ performance estimate purview should be reported (and the last reported values updated). These updates are transmitted on the off-band control channel to the local logically-centralized SC, which in turn updates its database of CServ performance estimates for the subnet domain appropriately.

Additionally, the routers should generate updates to transmit to the logically-centralized local SC as a keep-alive mechanism if the estimates of CServ performance metrics are stationary and do not require updates based on changing state. This allows the SC to differentiate between an active router in the subnet that is estimating stationary CServ performance metrics versus an active router that has gone offline or stopped functioning properly. These keep-alive updates can be relatively simple messages transmitted on the control channel. The SC should periodically purge state reported by active router if it has not received a CServ performance metric update or, alternatively, a keep-alive message within the appropriate time-to-live period. This keep-alive or time-to-live period does not need to be the same for all subnets. In subnets with relatively stable topologies and communication substrates (such as a subnet with a fiber optic routing core), this keep-alive interval can be increased compared to a subnet with frequent state changes (such as an edge wireless subnet). In these less-stable subnets, the keep-alive interval should be on the order of the subnet coherency time (the time between impactful subnet state changes) or, at the very least, on the order of strenuous CSR delay requirements (i.e. one second).

Using our representative subnet routing core topologies (the "lower" bound Petersen graph and the "upper" bound line graph network), we present a brief analysis of the estimated performance update rate burden on the control channel. To do so, we define a simplified and unified subnet routing core logical link model. Namely, we model each directed logical link between routers in the subnet routing core as an independent and identically-distributed two-state Markov process. Abstractly, we let the two notional states represent different CServ performance metric sets that capture the contribution of that logical link (this is a tuple of reliability, mean delay, and delay variance). The specific sets of CServ

performance metrics are immaterial for this analysis; we simply assume that if the link transitions form one state to the other, the change in performance is significant enough to require an update to the CServ performance metric estimation of any CServ Intranetwork Service that uses that link. Without loss of generality, we denote the two states for the logical directed link in the model as state "0" and state "1." Next we define the mean time to a link state change, or the *link coherence time*, as T. This Markov process model is depicted in Fig. 4-14.

We define the size of an update message to be b_U [bits]. We do not specify the exact format of an off-band control channel update message (it does not have the same rigid formatting requirements and restrictions as the internetwork and intranetwork CServ datagram, which traverses the data plane). However, we know that at a minimum it bears the 128-bit address of the reporting active CServ-enabled router, the 128-bit address of the Entrance Router for the CServ performance metrics reported (or the address of the external gateway router peer, or a special address that could be used to represent the upstream or downstream access network performance), three 64-bit double-precision floating point numbers for the reliability, mean delay, and delay variance performance estimates, and likely a 64-bit timestamp and an 8-bit type code describing the type of update (keep-alive, internal gateway-to-gateway, external gateway-to-gateway, access network upstream or downstream, access router to gateway router, and others). Together this forms an update message that is a minimum of 520 bits. Although an authentication mark and some integrity check might be deemed necessary in the future, we use $b_U = 520$ bits for this first-cut analysis. Lastly, we define several more variables as follows:

- $B_U^{Petersen} \triangleq$ the average aggregate update data size in bits per state change with the Petersen graph routing core;
- $B_U^{Line} \triangleq$ the average aggregate update data size in bits per state change with the Line network graph routing core;
- $B_U^{Line-max} \triangleq$ the maximum update data size in bits per state change with the Line network graph routing core.

From the analyses of the "lower" bound Petersen graph subnet routing core (see Sections 4.2.8.1 and 4.3.7.1), we know that when the state of one directed link changes, it affects $C^{Petersen} = 5$ CServ Intranetwork Services. This means that $C^{Petersen}$ update messages would be generated and transmitted on the off-band control channel to the SC reflecting this change. Thus, we have that:

$$B_U^{Petersen} = C^{Petersen} \times b_U = 5b_U. \tag{4.15}$$

Similarly from the analyses of the "upper" bound Line network graph subnet routing core (see Sections 4.2.8.2 and 4.3.7.2), we know that when the state of one directed link changes, it affects, on average, $C^{Line} = 55/3$ CServ Intranetwork Services and, at most, $C^{Line-max} = 25$ CServ Intranetwork Services (if the link that changes state is at the center of the topology). Consequently, the average number of update messages generated from a link state change is C^{Line}, and the maximum number of update messages generated from a link state change is $C^{Line-max}$. Thus, we have the following:

$$\begin{aligned} B_U^{Line} &= C^{Line} \times b_U = \frac{55b_U}{3}; \\ B_U^{Line-max} &= C^{Line-max} \times b_U = 25b_U. \end{aligned} \tag{4.16}$$

To complete the analysis of the data rate burden on the off-band control channel, we need to characterize the rate at which there is a link state change under the model considered. We define the number of logical directed links in the Petersen graph topology as $l^{Petersen} = 30$ links, and we define the number of logical directed links in the Line network graph topology as $l^{Line} = 18$ links. We define the merged logical link process rate (by the independence of the link states in the model) in the Petersen graph and the line network graph as $r_U^{Petersen}$ [state changes/second] and r_U^{Line} [state changes/second], respectively. Similarly, we define the subnet coherence time (or the average time until the next state change in the subnet) in the Petersen graph and the line network graph as $T_U^{Petersen}$ [seconds] and T_U^{Line} [seconds], respectively. With the established logical link Markov process model, we have:

$$\begin{aligned} r_U^{Petersen} &= \frac{l^{Petersen}}{T} = \frac{30}{T} = \frac{1}{T_U^{Petersen}}; \\ r_U^{Line} &= \frac{l^{Line}}{T} = \frac{18}{T} = \frac{1}{T_U^{Line}}. \end{aligned} \tag{4.17}$$

We define the average aggregate update data rate transmitted to the SC on the off-band control channel with the Petersen graph subnet routing core and the line network subnet routing core as $R_U^{Petersen}$ [bits/second] and R_U^{Line} [bits/second], respectively. Further, we define the maximum

aggregate update data rate transmitted to the SC on the off-band control channel with the line network subnet routing core (under the case where all link state changes are for the two central links in the topology) as $R_U^{Line-max}$ [bits/second]. With the set of expressions from Eqs. (4.15)-(4.17), we have the following results:

$$\begin{aligned} R_U^{Petersen} &= r_U^{Petersen} \times B_U^{Petersen} = \frac{l^{Petersen} C^{Petersen} b_U}{T}; \\ R_U^{Line} &= r_U^{Line} \times B_U^{Line} = \frac{l^{Line} C^{Line} b_U}{T}; \\ R_U^{Line-max} &= r_U^{Line} \times B_U^{Line-max} = \frac{l^{Line} C^{Line-max} b_U}{T}. \end{aligned} \tag{4.18}$$

We can also characterize the minimum keep-alive update message data rate in this analysis, where we assume that each router in the subnet routing core needs to generate at least one update message per keep-alive interval, at least, to maintain the previously reported state at the SC. To derive the minimum update rate with keep-alive messages only, we assume that the link state in the subnet routing core does not change and that all logical directed links are stationary. If the keep-alive update messages use the same format as the control messages to report updated CServ performance metric state, then each keep-alive message is b_U bits. We note that the aggregate keep-alive update message data rate is independent of the subnet routing core topology, and we define this value as R_K [bits/second]. We define the time between the required generations of periodic keep-alive messages at each active CServ-enabled router as T_K [seconds]. Using the notation from Section 4.1, the number of active CServ-enabled routers generating these keep-alive updates in each subnet routing core is $n_R = 10$. Using this notation, we have the following:

$$R_K = \frac{n_R \times b_U}{T_K}. \tag{4.19}$$

The results from Eq. (4.18) and Eq. (4.19) are compared in Table 4.3 with the approximate update size $b_U = 520$ bits, a logical directed link coherence time of $T = 1$ second, and a periodic keep-alive interval equal to the link coherence time (i.e. $T_K = 1$ second). Under this logical directed link model, the aggregate average control channel rate for the reporting of CServ performance metric updates in the worst-case topology is more than two times that of the representative best-case topology. However,

even the average control channel reporting rate is a manageable 171.6 kbps in the worst-case line network graph topology. And while the maximum aggregate update rate in the worst-case topology increases to three times that of the Petersen graph core routing topology, this is still a very manageable 234 kbps. The minimum keep-alive rate is a fairly negligible 5 kbps. Since the control channel burden of the State Measurement Service is low, this shows that the off-band control channel can be easily designed in a very robust manner with large overhead in most subnet deployments. This analysis of the learning protocol overhead is encouraging as the CServ architecture requires a highly reliable and overprovisioned off-band control channel. However, we must bear in mind that this analysis did not include CServ performance metric updates that would be generated as a result of access network state changes in the subnet or from state changes between external active gateway router peers.

We conclude this section with a discussion of the scale of the CServ performance metric state that the SC needs to maintain as it is reported. We have identified three "types" of CServ performance metric estimates throughout this chapter as we discussed the State Measurement Service protocols. These types are:

1. CServ Intranetwork Service performance estimates between active routers in the subnet routing core (including active access routers and active gateway routers);
2. external peering connection performance estimates between active gateway routers;
3. and upstream/downstream CServ performance estimates for an access network.

If we consider a naïve tabular storage implementation at the SC where each set of CServ performance metrics corresponds to a row, we can model each row as a field-by-field representation of the CServ performance metric update message. Specifically, the row stores the addresses of the two endpoints characterized by the CServ performance metrics (where a special address code can be used to represent the upstream and downstream directions for a particular access network), the triplet of estimated CServ performance metrics (reliability, mean delay, and delay variance), a timestamp indicating either the last time the estimate was updated or the remaining time to live for the row, and last the 8-bit convenience type code that represents the type of entry. With this representation, each row requires 520 bits according to our assessment from earlier in the section. But how many rows do we need the SC to store and maintain in the table? In the following paragraphs, we model the number of rows required and describe the scaling behavior.

Based on the model from Section 4.1, we have that the set of CServ-enabled routers in the subnet routing core is N_R, where all routers in the set are either active access or active gateway routers. Specifically, we stated that $N_R = N_{R-A} \cup N_{R-G}$. Although we proceed with this model, recall that should the routing core contain non-active routers or CServ-enabled core routers that are neither access nor gateway routers, these routers do not participate as endpoints in the State Measurement Service and do not report any performance metric estimates. This can clearly reduce the complexity in the storage requirements at the SC.

The first type of CServ performance metric estimates that are stored at the SC are those that represent the performance of CServ Intranetwork Services connecting the active subnet routers in N_R. Since there is a CServ performance metric estimate set for each CServ Intranetwork Service and there is a CServ Intranetwork Service between each pair of unique routers in N_R, this type requires a total of $n_R(n_R - 1)$ rows in the SC storage table. From this expression, we see that the contribution to the table for this type of CServ performance metric estimate scales polynomially with the number of active CServ-enabled routers in the subnet routing core. Rigorously, this contribution scales as $O({n_R}^2)$.

The second type of CServ performance metric estimates that are maintained at the SC are those that represent the external peering connections between the subnet's active gateway routers and the active gateway routers in adjacent subnets. As discussed in Section 4.5.1, the subnet is responsible for the CServ performance estimates of incoming connections – meaning the CServ performance metrics for CServ datagram transmissions from neighboring subnets that enter the domain of the subnet in question. For the purpose of an analytical model, we assume that each active gateway router has $p \geq 1$ peering connections with active gateway routers in adjacent subnets. Thus this type of CServ performance metric estimates requires a total of pn_{r-g} rows in the SC storage table, and its contribution scales linearly with the number of active gateway routers in the subnet (or, formally, $O(n_{r-g})$). Note that in the worst-case scenario in terms of required storage, the number of peering connections per gateway router could be equal to the number of other subnets in the network (as long as it does not peer with more than one gateway router in any given neighboring subnet).

Finally, the third type of CServ performance metric estimates maintained by the SC are those that summarize the upstream and downstream performance for access networks connected to active access routers. In Section 4.5.2, we described that only two sets of estimates are maintained for each access

network. For the purpose of this analysis, we assume that each active access router in the subnet has $a \geq 1$ connected access networks. With this assumption, this type of CServ performance metric estimates requires a total of $2an_{r-a}$ rows in the SC storage table, and its contribution scales as $O(n_{r-a})$, or linearly with the number of active access routers in the subnet. There is no clear way to bound the value of a, except to note that it is likely limited by the number of interfaces or the switching bandwidth of the access router.

In terms of the notation developed so far, the total number of rows in the CServ performance metrics table at the SC, or w, is given by:

$$w = n_R(n_R - 1) + pn_{r-g} + 2an_{r-a}. \tag{4.20}$$

If the subnet does not contain any active access routers (and thus, is purely a transit subnet in the CServ architecture), then $n_{r-g} = n_R$ and Eq. (4.20) becomes:

$$w = n_R(n_R + p - 1).$$

We can see that the number of rows in the SC table for CServ performance metrics scales as $O({n_R}^2)$ in this case. Alternatively, the subnet could contain the minimum of active gateway routers, or $n_{r-a} = n_R - 1$ and $n_{r-g} = 1$ (without at least one active gateway router, this would be an isolated subnet and have no part in the CServ internetwork architecture). In this case, Eq. (4.20) becomes:

$$w = n_R(n_R + 2a - 1) + (p - 2a).$$

Again, the number of rows in the SC table for CServ performance metrics scales as $O({n_R}^2)$ in this case. From this we conclude that the size of the table that the SC must store and maintain for CServ performance metric estimates scales as $O({n_R}^2)$, where n_R is the number of active CServ-enabled routers in the subnet routing core.

The SC is responsible for reporting all of the row entries in the table, and their respective updates, to the MC except for those CServ performance metric estimates between pairs of active access routers in N_R. The CServ Intranetwork Services that connect two access routers in the same subnet domain are useful only for CServ transactions that do not require internetwork service. If a CSR specifies a destination host in the same subnet as the source host, the SC can resolve the request using the reported CServ Intranetwork Service performance between the respective active access routers in the subnet since it does not require any knowledge of performance metrics in other subnets. Therefore, the number of entries relayed from the SC to the MC is reduced by $n_{r-a}(n_{r-a} - 1)$.

Before moving on, we consider the size of the SC table in bits for some representative numbers. If we denote the size of the table as w_b [bits], we have that $w_b = w \times b_U$ based on the transcription of the update message details into the table. We consider a subnet routing core with $n_R = 10$ active CServ-enabled routers (such as in our "lower" and "upper" case topologies), with $p = 5$ external active peers per active gateway router, and with $a = 5$ access networks per active access router. If we allow for an even mixture of active gateway and access routers ($n_{r-g} = n_{r-a} = 5$), then the SC table that stores CServ performance metric estimates for the subnet has a total of 165 row entries and requires 85.8 kb ($\sim$ 10 kB) of space for storage. This is an easily maintained and indexed table requirement considering that typical home and office computer memory is frequently on the order of several gigabytes. This storage requirement grows quickly to 10,650 row entries needing 5.538 Mb (< 1 MB) of space if $n_R = 100$ under the same conditions, demonstrating the scaling dependency on n_R.

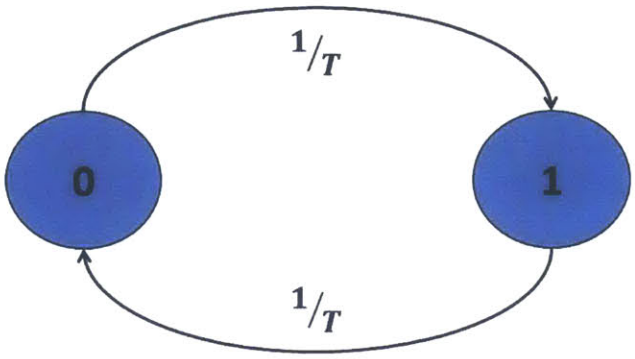


Fig. 4-14: This two-state Markov process is used to model each directed logical link in the subnet core routing topology, where the two states notionally represent different sets of CServ performance metrics that affect the performance estimation of the CServ Intranetwork Service paths traversing the link.

Table 4.3: This table describes the aggregate off-band control channel update message rates required to inform the SC of changes to the CServ Intranetwork Services in the subnet's core routing topology. The results are based on the Markov process link model illustrated in Fig. 4-14.

Subnet Routing Core Topology	Average Update Rate	Maximum Update Rate	Minimum Keep-Alive Update Rate
"Lower" Bound: Petersen Graph	78	78	5.2
"Upper" Bound: Line Network Graph	171.6	234	5.2

4.7 Conclusion

In this chapter, we discussed two possible protocols to implement the State Measurement Service of the CServ architecture. The protocols were described in detail for use in the subnet routing core, and they were ultimately compared through both qualitative and quantitative analyses. This development and comparison produced suggestions for their use based on the capabilities and design of the individual subnet. Later in the chapter, we discussed the use of these protocols to learn the CServ performance metrics outside the subnet routing core, including between external active gateway router connections and within access networks providing access for CServ-enabled hosts.

The State Measurement Service is central to CServ design since it allows for the summarization of subnet internal performance without disclosing sensitive details about the subnet's topology, loading, and policies, among other factors. With the output of the State Measurement Service, namely the estimated CServ performance metrics, the MC can compose a CServ Internetwork Service specific to the requirements of a particular CServ transaction without detailed understanding of or control over the handling of the CServ datagram as it traverses the constituent subnets in the service path or paths. The discovery and composition of CServ Internetwork Service using the product of the State Measurement Service is the third pillar of the CServ architecture and the focus of the next chapter.

Before we move on, it is worth noting one final future possibility for the State Measurement Service. Both proposed approaches to the service in this chapter, the Learning Session protocol and the Collector protocol, rely on the estimation of CServ performance metrics using real traffic. The Learning Session protocol actively generates CServ traffic in order to characterize the performance of the individual CServ datagrams. Alternatively, the Collector datagram protocol uses whatever traffic is available, best effort and CServ datagrams together, to loosely approximate the performance of a real CServ datagram which should generally outperform the resulting estimates. Neither of these protocols is capable of generating CServ performance metric estimates that allow for the unseen or the unexpected. Since "Black Swan" network events that generate large influxes of CServ transactions are a concern for the CServ architecture, an approach that can extrapolate and generate CServ performance estimates for unobserved subnet traffic loads could be greatly useful. An example idea for the approach would be to leverage simple analytic queueing models for the subnet routers. Although most queueing models rely on highly idealized memoryless exponential distributions, there are some general models (such as G/G/1

queueing models and associated queueing time bounds like the exponentially-tight Kingman bound [78]) suited for this purpose. These models can empower a protocol to theoretically extrapolate the delay for higher router loads than actually observed in the network. However, there are many challenges associated with this approach. One problem is that generic queueing models and bounds are unavoidably loose since no assumptions are made on traffic arrival and router processing distributions. Another issue is that these models typically do not account for finite interface buffer space and datagram loss. This type of modeling direction for the State Measurement Service may prove fruitful, but there are many open research problems that must be addressed first.

Chapter 5

CServ Internetwork Service – Discovery and Composition

The operation of the master controller (MC) is the crux of the CServ architecture and CServ Internetwork Service. We can consider that the previous chapters, albeit critical parts of the whole, describe supporting components that bolster the operation of the MC's algorithm that directs individual CServ transactions at the internetwork level based on their specific performance requirements. Through the description of the transmission of CServ datagrams through a series of independently administered subnets in Chapter 2, we illustrated the need for a control entity with a global perspective to choose a sequence of ingress and egress routers to escort the datagram appropriately from source to destination subnet, all the time avoiding fine-grained network control by allowing the constituent subnets to implement their own internal routing and forwarding policies. In Chapter 4, we provided a framework to characterize the CServ Intranetwork Services implemented by these independently administered subnets connecting the ingress and egress routers such that the global control entity does not need detailed knowledge of the internal state of the subnets in order to choose these coarse-grained internetwork paths. To wrap up this architecture description, we now need to describe the use of this captured and reported state summarization for the purpose of discovering and composing the CServ Internetwork Services that become the strings of addresses in the internetwork CServ datagram control headers.

5.1 The CServ Pre-Transaction Control Flow

Here we recall the description of the input and output of the MC with respect to CServ transaction setup from Section 2.3.2, while adding some details concerning the use of the off-band control channel. This is a per-transaction control procedure that precedes the transmission of each critical message using the

CServ data plane, effectively acting as admission control since an internetwork CServ datagram cannot be processed without an explicit CServ Internetwork Service path in the header (see Chapter 3).

To begin a transaction setup, the CServ-enabled host that generates a critical datagram payload for transmission forms a request for a desired level of service from the CServ network. This request is made in the form of a CServ Request (CSR) via the CServ API, which is borne to the logically-centralized MC server by way of the robust off-band control channel. The CSR provides a structured way for the CServ source host to indicate its desired level of service for the transaction, which in turn allows the MC to algorithmically determine the ability of the network to bear the critical message with this level of service. The CSR contains the following information, described in more detail previously in Chapter 2:

- a control message type field;
- the source host address;
- the destination host address;
- the primary CServ performance metric;
- the minimum service reliability value;
- the maximum service delay value;
- the minimum path diversity value;
- a timestamp indicating the time of CSR generation and transmission;
- and a source-host specific sequence number.

Allowing for 128-bit addresses as previously used in the dissertation, 8 bits for the control message type field (indicating that this is a CSR rather than a State Measurement Service report or an association/disassociation message), 64 bits each for the specified performance metric requirements, 1 bit for the indication of the primary performance metric, 7 bits for the minimum path diversity value, 32-bits for the sequence number, and a 64-bit timestamp, this CSR comprises a 496-bit control message. In practice, some additional fields may be included for the purposes of CSR integrity (such as a checksum value) or authenticity (such as a digital signature) verification. For now, we assume that the off-band control channel is heavily overprovisioned and designed for highly-reliable signaling such that the integrity check is not necessary. Furthermore, we assume that host authenticity verification can be completed during CServ device association with the control hierarchy. Although some notion of CSR

authentication should be considered in future work to protect against CSR-based denial-of-service attacks. We note here that even with 1,000 CServ hosts generating a CSR every one second across the network, this comes to less than 500 kbps of CSR traffic directed towards the logically-centralized MC on the aggregate off-band control channel across all subnets, generally a very manageable data rate. CServ hosts, however, are expected to use CServ capabilities sparingly and at rates much less than one CServ transaction per second. A host with high service utilization rate might generate a CServ transaction every minute, whereas a typical CServ endpoint should generate CServ message payloads on the order of one every hour. Thus our numerical example here greatly overestimates the off-band control aggregate CSR data rate for 1,000 CServ-enabled source hosts.

Upon receiving the CSR via the CServ API, the MC executes the Critical Service Discovery and Composition Algorithm (CSDCA) to determine whether or not the network can bear the critical message transaction to its intended CServ-enabled destination host with the requested level of service, and, if so, how exactly that level of service can be achieved. The development and description of this algorithm is the focus of this chapter. As indicated by the algorithm title, this is a two-phase procedure. The *discovery phase* finds end-to-end paths between the source host and destination host at a subnet-granularity. Specifically, other than the access routers that provide connection to the subnet routing cores of the source and destination subnets, all other routers in the paths are active gateway routers at the edge of the subnets transited by the path. As part of this phase, the algorithm computes the CServ performance metrics for these paths. In the second phase, the *composition phase*, the algorithm attempts to create a CServ Internetwork Service solution that meets the demands of the CSR using the paths found during the discovery phase. This solution may be a single path, or it may be a set of subnet-disjoint paths that are used together as a diversity-routed solution. The rationale behind the use of subnet-disjoint paths and controlled diversity routing are discussed in the next section of the chapter. Following the successful execution of both phases, the algorithm output is an explicit internetwork path or set of explicit subnet-disjoint internetwork paths from the source host to the destination host that support the desired service level based on the currently known state information reported by the State Measurement Service. If the execution of the algorithm is unsuccessful in meeting the requested service, the output is a service denial response. (In Chapter 2, we suggested the future inclusion of a service counteroffer if the best composed internetwork service falls slightly short of the requested service level with respect to some CServ performance metric. For the purpose of this chapter, we do not

consider this option further and restrict our discussion to either a service access or service denial output.)

After the completion of the CSDCA process, the final step of the control plane CServ transaction setup involves returning the result of the algorithm to the source host via the CServ API on the off-band control channel. As discussed, this response takes one of two forms: a service access response or a service denial response. A service access grants the source host permission to transmit the CServ payload as part of an internetwork CServ datagram using the discovered CServ Internetwork Service path or set of CServ Internetwork Service paths in the CServ Internetwork Service solution. These paths are a sequence of router addresses, beginning with the upstream active access router in the source subnet (which is particularly useful if the source host is multihomed), continuing with a set of egress and ingress active gateway routers which shepherd the internetwork CServ datagram from subnet to subnet, and finally ending with the upstream active access router in the destination subnet (again, which is useful if the destination host is multihomed). Note that while this prescribes the coarse-grained internetwork path that the internetwork CServ datagram must follow, it does not control the local fine-grained CServ Intranetwork Services that bear the datagram from active router to active router in the path. The size of the service access response is highly variable, depending on the length of CServ Internetwork Service paths and whether or not the solution employs end-to-end diversity routing. It contains the following information:

- a response type field;
- the source host address;
- the destination host address;
- a timestamp indicating the time at which the service access response is invalidated;
- the sequence number from the CSR;
- the number of paths used in the internetwork solution;
- the length of the first CServ Internetwork Service path in terms of the number of addresses;
- the first CServ Internetwork Service path (a sequence of addresses matching the specified length);
- the length of the second CServ Internetwork Service path in terms of the number of addresses;
- the second CServ Internetwork Service path;
- ...

- the length of the last CServ Internetwork Service path (matching the specified number of paths in the solution);
- and the last CServ Internetwork Service path.

We allow 4 bits for the encoding of the response type (access, denial, and space for expansion), 8 bits for the encoding of the number of paths used in the solution, as well as 8 bits for each encoding of the length of a CServ Internetwork Service path. If we consider a service access response employing diversity routing over three CServ Internetwork Service paths, each with two transit subnets (for a total of eight addresses in each explicit CServ Internetwork Service path), then the service access response requires 3.460 kb. The service access response can be significantly larger than the CSR since it may need to encode multiple sequences of 128-bit addresses. Luckily, the off-band control channel communication from the MC to the end hosts is used primarily for the CServ API responses, so the capability should exist to overprovision this channel to support reliable response signaling. For example, if the MC processes 1,000 CSRs per second, the aggregate rate from the MC to the CServ-enabled hosts could be on the order of 3.5 Mbps based on this analysis approximating the service access response size. Again, this is an overstatement, as the submission of CSRs should be for the most critical traffic in the network and not utilized this strenuously. This is all in contrast to the overprovisioning of the control channel from the end hosts and routers to the MC, which must simultaneously bear association and disassociation messages, CServ API CSRs, and CServ performance metric estimate reports from the State Measurement Service.

As with the CSRs, some notion of MC response authentication should be considered for future inclusion in the CServ architecture. This can mitigate exploits of the architecture which involves a malicious actor spoofing the MC service and generating false responses to misdirect or deny critical message transactions. The use of control plane authentication techniques needs to be carefully considered and designed since, in light of the discussions of digital signatures in Chapter 3, it can add a significant overhead to control messages.

We note here that, alternatively, the service denial response is a significantly smaller size than the service access response. It only needs to contain the following information:

- a response type field;

- the source host address;
- and the sequence number from the CSR.

The response type field would be used to identify the control message as a service denial response to the CSR with the provided source-specific sequence number. Using the field sizes previously established, this response requires only 164 bits, which likely represents a negligible contribution to the control channel rate among the service access response control traffic since the network should be dimensioned to avoid these responses to reasonable CSRs.

Once the CServ-enabled source host receives the access response from the MC via the CServ API on the off-band control channel, it prepares the internetwork CServ datagram (or datagrams in the case of end-to-end diversity routing) with the critical message payload and then transmits the datagram (or datagrams) on the CServ data plane. The transmission procedure and the required data plane control information (the CServ Internetwork Service header and CServ Intranetwork Service header) were previously covered in Chapter 3. The hole in the description of the per-transaction control processing thus far is the implementation of the CSDCA, including how it leverages the reports generated by the State Measurement Service for each subnet in the network. In the remainder of this chapter, we complete the picture in the following parts:

- we motivate the use of diversity routing within the CSDCA;
- we discuss the reasoning for subnet-disjoint paths in the diversity-routed solutions;
- we consider the definition of the delay of a path;
- we describe the MC's representation of the internetwork topology and integration of the State Measurement Service reports;
- and we walk-through and characterize the details of CSDCA operation.

5.2 Fundamental Considerations for Internetwork Service

We present several fundamental topics in this section that drive the design of the CSDCA and influence its operation. Namely, we discuss the use of controlled-diversity routing among subnet-disjoint paths, as well as the characterization of the end-to-end CServ performance metrics for a multi-hop path. In doing so, we present some critical assumptions invoked or subsumed by the algorithmic approach of the MC

presented in this chapter. Because of the nature of the assumptions, particularly those that involve statistical independence, they should be regarded with an appropriate level of skepticism and revisited in the context of adversarial action against the CServ architecture. Although the security framework for the CServ architecture is not the focus of this document, we consider the potential collapse of these assumptions and the resulting impact at the end of the chapter.

5.2.1 Controlled Diversity Routing

The CServ architecture employs end-to-end path diversity between the source and destination host endpoints for two purposes:

1. diversity routing over multiple paths improves the end-to-end service reliability over the reliability of the paths individually;
2. and the use of diversity routing over multiple paths provides a level of service survivability against unpredictable failure (or "Black Swan" [52]) events affecting a strict subset of the paths.

If we consider the first statement above, we recognize that the reliability improvement relies on the hidden assumption of statistical independence between the paths used in the diversity solution under nominal conditions. The end-to-end reliability does not improve if the random processes governing the performance of the paths in the diversity solution exhibit perfect correlation. We need to be wary of this requirement; in the next section, we explain the effort made to maintain some level of statistical independence between the paths.

For now, we assume that the end-to-end internetwork paths are statistically independent. For illustration, let us consider a situation where there are k paths in the diversity solution, and that the reliability of the i^{th} path, where $i \in \{1,2,\dots,k\}$, is denoted $0 \leq \rho_i \leq 1$. The reliability of the end-to-end service, ρ, is then given as:

$$\rho = 1 - \prod_{i=1}^{k}(1 - \rho_i). \tag{5.1}$$

If we now assume that $0 < \rho_i < 1 \; \forall \; i \in \{1,2, \ldots, k\}$, then the sequential addition of each path to the set of k paths strictly improves the reliability ρ of the end-to-end service since $0 < (1 - \rho_i) < 1 \; \forall \; i \in \{1,2, \ldots, k\}$, for each path. We note that the addition of a path with a reliability of 0 does not improve the overall service reliability, while the addition of a path with perfect reliability ($\rho_i = 1$) implies perfect service reliability (making any subsequent addition of paths to the set unnecessary).

In the presence of an unprecedented "Black Swan" network even that disrupts a strict subset of the paths in the diversity routed solution, the use of diversity routing provides a notion of end-to-end service survivability. These unpredictable events may include Byzantine failure of routing or transmission components in a path [79]. Because unprecedented events, by definition, have not been previously observed in the operation of the network, their impact cannot be captured by the State Measurement Service and its reported CServ performance metrics. As long as the event does not affect all paths in the diversity routed solution, the service can still provide end-to-end connection between the source and destination host at some reduced reliability level over the unaffected paths. Thus, even if one path can satisfy the required level of reliability, the use of additional path diversity can protect the service against an unexpected failure. This comes with an important caveat that we must bear in mind, however. If the "Black Swan" event is the result of adversarial action, the motivated adversary that intends to disrupt critical communication between a pair of hosts would attack critical components of all possible paths connecting the devices. In this scenario, the use of additional diversity paths does not improve reliability since the adversary seeks to correlate failure modes between all possible internetwork paths. We can view this as a breakdown of the statistical independence assumption. Under coordinated attack, the internetwork paths are likely anything but statistically independent.

In light of these considerations, one might ask why the CServ architecture does not, by default, choose to transit a CServ transaction over all possible internetwork paths connecting the source and destination hosts. This type of operation could be considered flood routing, or simply flooding. Flooding is a datagram transmission paradigm that replicates a datagram on all outgoing router interfaces (other than the one it is received on) with some control techniques to ensure that the datagram instance does not circulate in the network forever. Since no intelligent routing decisions are made based on the datagram destination, this is the antithesis of routing; if the source host is somehow connected to the destination host, the datagram ultimately makes its way to the intended recipient. This approach clearly substitutes protocol overhead for increased network data plane transmission and switching bandwidth

consumption. In [79], reliable flooding techniques are proposed for reliable network service in the presence of constrained Byzantine network faults (without any guarantees on end-to-end datagram delay).

Flooding can be interpreted as the extreme version of diversity routing. It eventually explores all possible paths connecting the endpoints. The use of *controlled diversity routing*, on the other hand, represents the selection of a subset of the paths explored by flooding, where the "controlled" nomenclature highlights the decision to strategically exploit a select the number of paths between the endpoints. Even though the reliability of flooding must be at least as great as the use of constrained diversity routing based loosely on the argument of Eq. (5.1) (although this requires statistical independence between all paths and does not truly apply to flooding due to the correlation induced by the hop-by-hop replication scheme), why do we choose not to blindly flood CServ datagrams?

The objective of the CServ architecture is to discover a CServ Internetwork Service solution that meets the reliability requirement of the CSR, not to transmit each critical message with the maximum possible reliability. Controlled diversity routing represents the degree of risk management required to bear internetwork CServ datagrams at a particular service level without needlessly consuming excess network transmission and switching bandwidth. Our approach uses the State Measurement Service to learn the network state, which is then leveraged to determine the minimal degree of diversity required in the solution to meet the CSR requirement. Flooding's blind reliance on redundancy would significantly reduce the capacity of the network to serve other CServ transactions, and our design aims to maintain high CServ network availability even when many critical messages are generated. Furthermore, the use of flooding makes it difficult to characterize the end-to-end performance of the CServ datagram prior to transmission since this would require characterizing the performance of an exponential number of paths. Even though it introduces the protocol overhead of the CSDCA, our controlled diversity approach enables the generation of *a priori* service guarantees that satisfy the stated requirements of the CSR, a capability aligned with the CServ architecture objectives.

In a simple and conservative analytic example, we illustrate the network transmission bandwidth savings garnered from the deployment of controlled diversity routing over flooding. Consider a network model with n disjoint paths between the source and destination hosts of a CServ transaction, where each path traverses one intermediate network component (such as a router). This physical network model is

illustrated in Fig. 5-1. We assume that all n paths are statistically independent and can be characterized by the same end-to-end reliability $0 \leq \rho \leq 1$ and delay (such as can be learned by the State Measurement Service protocols presented in Chapter 4). Specifically, we assume that the transmission links are lossless and that all possible CServ datagram loss occurs at the unreliable intermediate network component (independently with probability ρ).

Consider a situation where the source host generates a CSR with a reliability requirement of $0 \leq \rho_0 \leq 1$ and some maximum delay requirement that is less than the delay of the paths. We additionally introduce a network fault model that attempts to capture the impact of unprecedented network "Black Swan" failures. Let us assume that each intermediate network component depicted in Fig. 5-1 is affected independently and identically by a Byzantine fault that renders it unable to process CServ datagrams (thus disrupting the path) with probability $0 \leq \omega \leq 1$. This is clearly an abuse of the "Black Swan" concept since we cannot confidently assign a probability distribution to an unprecedented event. However, for the convenience of this analysis and illustration, we allow for this simplified characterization. How many paths are required to probabilistically ensure that the end-to-end service solution meets the minimum reliability requirement of the CSR?

If we deploy flooding as the CServ network service, then we are guaranteed to do the best possible in terms of reliability given the current network state. This does not necessarily mean that the service satisfies the requirement of the CSR. Under some network conditions (or combinations of the parameters ρ and ω), flooding does not provide an end-to-end reliability of ρ_0. With the flooding approach, we do not know *a priori* whether or not the service satisfies the demands of the CServ transaction. Alternatively, we have *a priori* knowledge of the service reliability with the controlled diversity approach (before the introduction of the random Byzantine faults). Under some network state conditions, the use of controlled diversity routing can satisfy the end-to-end reliability requirement while consuming less network transmission bandwidth than flooding. We proceed to show this analytically. For the analysis, we consider the number of path links used by the CServ transaction as the proxy for network transmission bandwidth. As illustrated in Fig. 5-1, each path is considered to have two links. Thus there are $2n$ total links in the network model.

We begin by considering the network transmission bandwidth consumption of the flooding approach, which always transmits the CServ datagram over all n paths. Without any Byzantine faults (i.e. when

$\omega = 0$), the flooding technique uses all n first-hop links and, on average, $n\rho$ of the second-hop links. As ω increases, the probability that the datagram survives the intermediate network device decreases based on the chance that the component is suffering a Byzantine fault. The total expected number of links used by the flooding technique under this model is then $n + n\rho(1 - \omega)$.

If we now consider the use of k-diversity routing, we can form a similar expression for the total expected number of links used by the routing technique. Without any Byzantine faults (i.e. when $\omega = 0$), the approach always uses k of the first-hop links, and, on average, $k\rho$ of the second-hop links. As with the flooding case, the probability that the datagram survives the intermediate network device decreases based on the likelihood that the component is affected by a Byzantine fault. Thus, the total expected number of links used by the flooding technique under this model is $k + k\rho(1 - \omega)$. But what value of k is required to meet the CServ reliability requirement? To solve for the required degree of diversity, we consider the value of k such that the probability that the transmission over at least one of the paths is successful is greater than or equal to ρ_0, the CSR requirement. Doing this and ignoring the integer constraint on k, we get the following:

$$\begin{aligned} & 1 - \left(1 - \rho(1 - \omega)\right)^k \geq \rho_0 \\ & \Rightarrow k \geq \frac{\log(1 - \rho_0)}{\log(1 - \rho(1 - \omega))}. \end{aligned} \tag{5.2}$$

Practically, k must take an integer value since it represents the diversity degree or the number of paths the CServ datagram is transmitted over. Introducing the integer constraint and the physical limits of the network model to Eq. (5.2), we have:

$$k = \min(n, \left\lceil \frac{\log(1 - \rho_0)}{\log(1 - \rho(1 - \omega))} \right\rceil). \tag{5.3}$$

This result indicates that for some parameter values ρ, ρ_0, and ω, the maximum number of paths, or the physical constraint on the level of diversity, impedes the technique from meeting the end-to-end reliability requirement of the CSR. However, in the case where the controlled diversity approach cannot meet the requirement, nor can the use of flooding.

We visualize the results of this analysis in Fig. 5-2. Up to some threshold on the probability of a Byzantine failure on the path, the expected network transmission bandwidth usage of controlled diversity routing outperforms the expected network transmission bandwidth usage of flooding while satisfying the reliability demand of the CSR. The transmission bandwidth savings are maximized when there are no faults, and the gap closes as the likelihood of Byzantine faults on the paths increases. In some scenarios, diversity routing can meet the requirement while using less than half of the network transmission resources used by flooding even when an expected one-third of the paths are affected by Byzantine faults.

The results indicate that while flooding provides survivability against unprecedented "Black Swan" events up to a point without any knowledge of the fault conditions, controlled diversity routing is capable of the same while consuming significantly less network bandwidth. We have demonstrated that survivability against Byzantine failures can be realized without blinding relying on uncontrolled redundancy. This result is actually quite conservative since all n paths in the model are disjoint. In a more realistic scenario, there may be cross-links between the paths that connect the source and destination hosts, and then the flooding approach would consume an even greater proportion of network transmission resources compared to the controlled diversity routing technique.

In the analysis, we adopted an omniscient view that allowed us to choose the degree of diversity optimally to meet the CSR reliability requirement while minimizing the network resources used. In practice, we do not have this information since we cannot *a priori* model the impact of an unprecedented network disruption event, and thus we do not have the necessary information to optimally determine k for survivability purposes. The message remains clear, though; controlled diversity routing has the capability to provide the same survivability property as flooding without the excessive network transmission and switching bandwidth consumption. Additionally, we can choose the optimal degree of diversity in nominal scenarios without "Black Swan" disruption events based on the reports of the CServ State Measurement Service, whereas flooding does not afford the same flexibility.

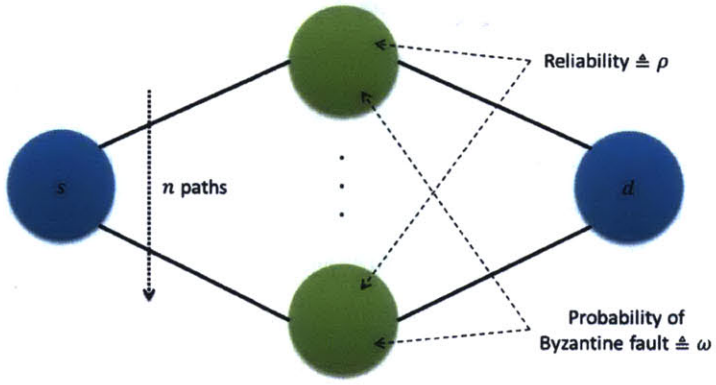


Fig. 5-1: The network model used for the comparison between flooding and controlled diversity routing is depicted here. The reliability of the intermediate network component on each path is independent and identically distributed, as is the probability that the component is affected by a Byzantine failure.

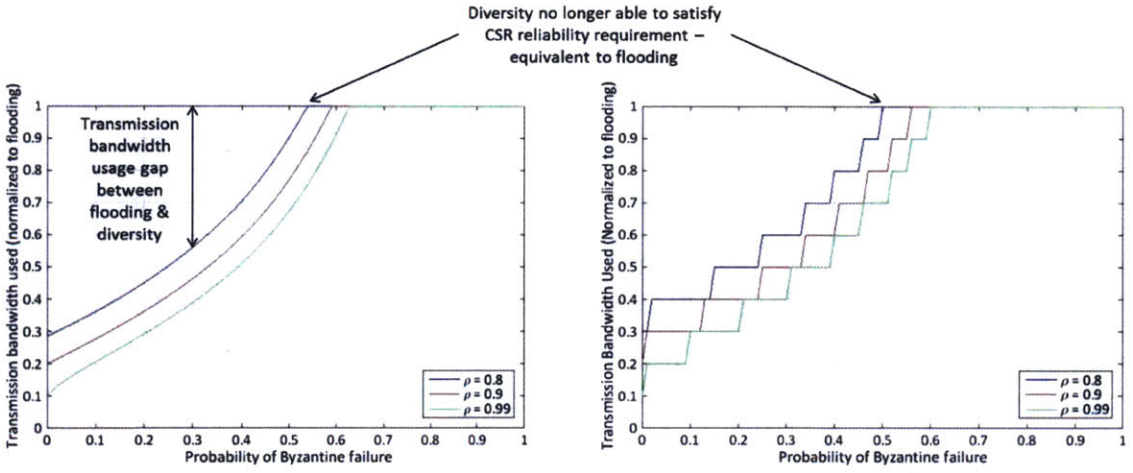


Fig. 5-2: The expected network transmission bandwidth used by controlled diversity routing to achieve an end-to-end reliability of $\rho_0 = 0.99$ is shown, normalized to the expected network transmission bandwidth consumed by the flooding technique. The left plot ignores the integer constraint on the diversity degree, while the right plot does not. The number of paths $n = 10$ in these visualizations.

5.2.2 Subnet-Disjoint Paths

In Section 2.2.1, we introduced the subnet as the central logical participant in the CServ network and global internetwork routing architecture. We stipulated that CServ Internetwork Service paths describe explicit subnet-to-subnet granularity routes where the internal routing and forwarding behavior of the subnet itself is a "black box" to the global network. More formally, CServ Internetwork Service paths specify a sequence of active gateway routers (and access routers at the endpoints), indicating the ingress and egress gateway for each traversed subnet without any further detail of how the datagram is treated within the subnet.

Controlled diversity routing was discussed previously in Section 5.2.1, motivating the design choice to use this technique to improve end-to-end CServ Internetwork Service reliability. In the discussion of diversity routing, we made an assumption on the statistical independence of the paths for the purpose of calculating the composed reliability of the diversity-routed service. Generally, statistical independence assumptions are sticky statements. There is frequently little empirical reason to believe in the independence of processes in the network, as the complex system dynamics tend to couple actions and effects. This is further complicated by the presence of adversarial action; a sophisticated and motivated adversary aims to correlate system failure modes in order to exact the most effective form of network service disruption. As an example, rather than choose to target one possible path, an attacker would want to disrupt all possible internetwork paths connecting a particular source and destination host pair (or source and destination subnet pair) such as to counteract the CServ architecture's use of diversity routing by effectively partitioning the internetwork connection graph.

In our definition of the subnet, we noted that it aptly lends itself to distinguishing between heterogeneous network segments. In a non-exhaustive list of sources of heterogeneity, we highlighted physical communication, hardware/software component, network resource provider and administrator, security protocol, and network management framework heterogeneity. We argued that this heterogeneous characterization, by design, can exhibit a notion of statistical independence between the subnet units. However, the impression of statistical independence does not come easily nor unintentionally. This dissimilitude between subnets needs to be built into the design and differentiation of the subnets from the drawing board (and is, in part, why we do not consider the Internet AS and the subnet equivalent).

As an example of the fragility of the independence assumption between subnets, we considered the infamous "backhoe attenuation" problem. Adjacent fiber optics subnets operated by different administrators, using different switch and router machines and frameworks, and employing different network management and control protocols may share the same physical plant of cables and conduits between points of presence. Should a backhoe unintentionally (or intentionally) uproot the cable conduits during an excavation activity, this could simultaneously disrupt the operation of all of the subnets. Clearly, these subnets do not exhibit the notion of statistical independence that we wish to encapsulate in the subnet; they are highly correlated through their homogeneous fiber physical plant infrastructure. As another example, adjacent infrastructure-based wireless subnets operated by different network providers may use the same edge communication spectrum and protocols. Even though the hardware and software may be highly heterogeneous, the shared wireless media induces strong correlation between the over-the-air hops within these subnets since they can jointly experience the disruption of a RF fade or shadowing event.

Although the notion of statistical independence is difficult to maintain between subnets thanks to the complex interactions of the networked system and the examples of heterogeneity bottlenecks (such as described above), the network planner must be aware of the potential pitfalls and deliberately design and provision the subnets such that they appear "independent enough." Even if the subnets in the CServ internetwork system do not exhibit strict statistical independence, we preserve the desire that neighboring subnets should maintain at least some degree of intentional heterogeneity such that the statistical independence assumption used in the architecture and protocol design does not completely break down and that adversarial action against each requires considerably targeted and individualized effort. As an example, a CServ network should not have three satellite relay subnets using the same band. Or, as another example, disparate fiber optic subnets in the CServ network should use disjoint physical plants with strict geographical separation between the conduits and switching points of presence supporting the transmission fibers of each. This is a challenging requirement of the CServ architecture, but it necessary both for the implementation of highly-reliable end-to-end internetwork service and for some degree of survivability against the unprecedented disruption event.

With this intentional subnet design that renders them "independent enough," we leverage the statistical independence assumption in the composition of end-to-end CServ Internetwork Service using diversity routing over *subnet-disjoint* paths. This means that, given a set of CServ Internetwork Service paths

between a source and destination host used to realize a diversity-routed solution, all transit subnets are unique and used only once in the set. Note that we have specified that "transit subnets" are unique. The source and destination subnet for each CServ Internetwork Service path comprising a diversity-routed service solution may be the same. If the source and destination hosts are single-homed, then the source and destination subnets are logical points of convergence between the paths (they may not be physical points of convergence is the CServ host is connected to several upstream active access routers in the subnet). Ideally, with sufficient subnet-multihoming of both CServ-enabled endpoint hosts, the end-to-end CServ Internetwork Service paths in a diversity-routed solution would be completely disjoint such that all subnets are unique and used only once in the set. For now, however, we do not assume the availability of sufficient multihoming connections to always implement this type of diversity-routed solution between CServ transaction endpoints. The description of the MC's CSDCA describes the discovery of paths that may have common source and/or destination subnets but are otherwise subnet-disjoint.

Why do we require subnet-disjoint paths and not just gateway-disjoint paths? We propose that instead of searching for subnet-disjoint paths, we could instead discover CServ Internetwork Service paths that do not share any routers in the explicit path except possibly for the upstream access router in the source and destination subnets. This opens up the discovery of a larger set of end-to-end paths than if we enforce strict subnet-disjoint paths. Under the search for subnet-disjoint paths, the cardinality of the possible set of paths we can discover is clearly limited by the logical degree of the source and destination subnets (or, in other words, their number of subnet peers). This is not even the minimum on the cardinality of this set; there are internetwork topologies with one bottleneck transit subnet through which all possible CServ Internetwork Service paths between the source and destination subnets are funneled through.

Recall that the CServ architecture has no control over routing and forwarding within any given subnet at the global level; these are local decisions left to the subnet administrator based on their individual policy and business needs. Thus, even if two CServ Internetwork Service paths specify the use of a transit subnet through disjoint sets of ingress and egress gateway routers, there is no knowledge about the CServ Intranetwork Service realization of those logical gateway-to-gateway connections. The CServ Intranetwork Service paths joining the two disjoint pairs of ingress and egress gateway routers for a particular subnet may in fact share the exact same physical transmission links and intermediate routers

in the subnet routing core, such as illustrated in Fig. 5-3. The CServ Intranetwork Services do not have the same required intentionality regarding the notion of appearing "independent enough" through heterogeneity as do the subnets themselves. For this reason, we rely on the more restrictive subnet-disjoint requirement in the discovery of CServ Internetwork Service paths in order to utilize statistical independence assumptions when composing the end-to-end reliability of the service. Consequently, the CServ internetwork graph should be designed with sufficient logical connectivity to allow for the discovery of subnet-disjoint CServ Internetwork Service paths and the composition of adequately reliable CServ Internetwork Services that meet reasonable CSR demands.

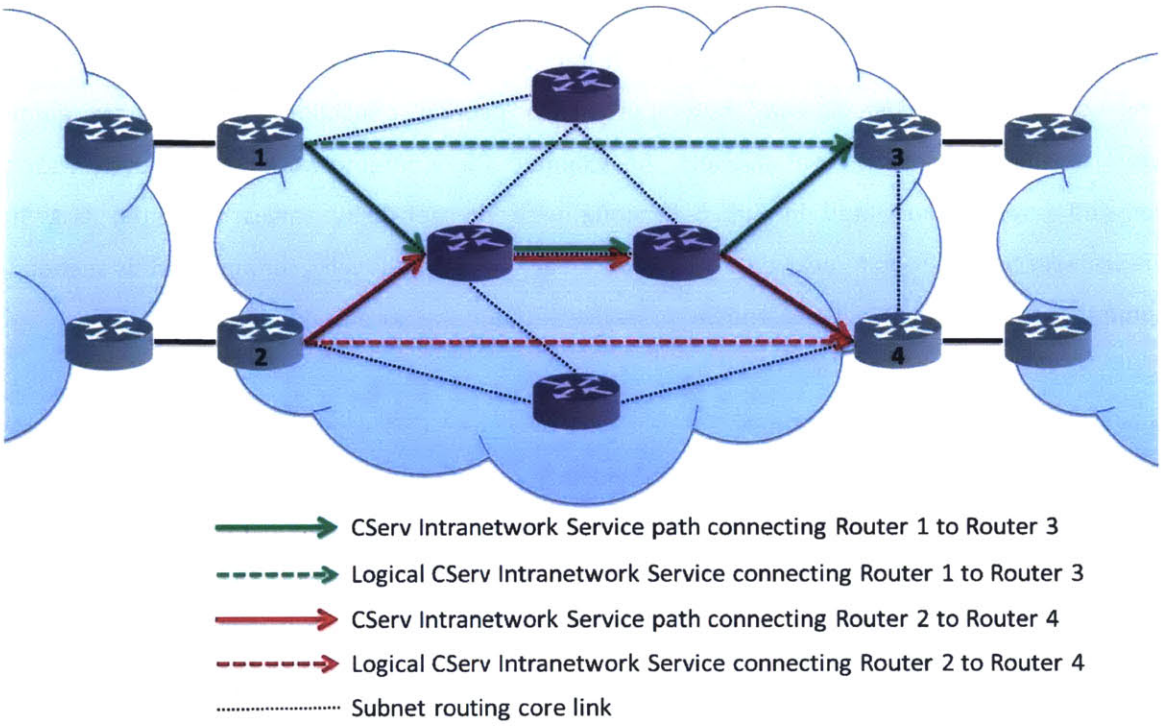


Fig. 5-3: This figure illustrates that gateway-disjoint CServ Internetwork Service paths traversing a particular subnet may be forwarded over the same links and switched by the same routers in the subnet routing core based on the routing and forwarding policy of the CServ Intranetwork Service. For this reason, the CSDCA searches strictly for subnet-disjoint CServ Internetwork Service paths.

5.2.3 The Reliability and Delay of an Internetwork Path

Before we approach the problem of discovering a CServ Internetwork Service path, let us consider how to characterize the reliability and delay of a path. In this section, we assume that we have a CServ Internetwork Service path (the discovery of this and other paths is the topic of the subsequent section), and we assume that each hop of the path is characterized by the reports generated by the State Measurement Service described in Chapter 4. That is, each hop or link between a pair of active CServ-enabled routers in the CServ Internetwork Service path has an associated set of CServ performance metrics, namely the reliability, mean delay, and delay variance, from the most recent State Measurement Service report. Based on the previous discussions of the State Measurement Service, we know that these hops or links in the CServ Internetwork Service path may represent upstream or downstream access network connections, CServ Intranetwork Services presenting logical subnet interior connections between active gateway routers, or exterior peering connections between active gateway routers at the edge of adjacent subnets. An example CServ Internetwork Service path with the associated hops is illustrated in Fig. 5-4, along with the reliability values from the last State Measurement Service report received by the MC for each hop. Additionally, throughout this section, we assume that the hops in a CServ Internetwork Service path are statistically independent.

We begin with the characterization of the reliability of a given CServ Internetwork Service path. As a concrete example, consider the path illustrated in Fig. 5-4, along with the reliability values from the last State Measurement Service report. If the hops in the path are statistically independent, then we can say that the reliability of the path is the product of the reliability values of the hops that form the path. This result is illustrated for the example of Fig. 5-4. Abstractly, consider a CServ Internetwork Service path of $h \in \mathbb{Z}^+$ hops from source host to destination host, comprising upstream and downstream access network connections, the traversal of CServ Intranetwork Services, and connections between external active gateway router peers. We enumerate these hops as hop $i \in \{1,2,\ldots,h\}$. According to the most recent set of State Measurement Service reports, the reliability of hop i is $\rho_i \; \forall \; i \in \{1,2,\ldots,h\}$. Consistent with the probabilistic interpretation of reliability used in this dissertation, we have that $0 \leq \rho_i \leq 1 \; \forall \; i \in \{1,2,\ldots,h\}$. We denote the path reliability as ρ, and we have:

$$\rho = \prod_{i=1}^{h} \rho_i. \tag{5.4}$$

From Eq. (5.4), we note that $0 \leq \rho \leq 1$ thanks to the constraints on the range of the values in the set $\{\rho_i\}_{i=1,2,\ldots,h}$, which is again consistent with our probabilistic interpretation of reliability.

We now consider the concept of the delay of a CServ Internetwork Service path. As an example, imagine the same CServ Internetwork Service path as used in the reliability example above. Based on the most recent State Measurement Service reports received at the MC, there is a mean delay and delay variance value associated with each hop in the path. This is depicted in Fig. 5-5, where the mean delay values are given in milliseconds for illustration. There is an unknown delay distribution associated with each hop in the path, described only by the learned mean and variance values. The objective is to characterize the path delay, which is equivalent to the summation of the random variables that represent the delay of each hop in the path. Thus, the mean delay of the path is the sum of the mean delay values of the hops that form the path. The previous statement does not require the assumption on statistical independence between the hops in the path. However, if we invoke that assumption, we can further state that the delay variance of the path is the sum of the variances of the delay distributions for each hop that forms the path. These results are illustrated for the example in Fig. 5-5.

We have now characterized the path delay distribution for the given CServ Internetwork Service path in terms of its mean delay and delay variance. But, since we do not know anything further about this distribution, how can we say anything about the absolute delay of the path? If we knew the distribution had a hard value as the upper limit for its support, then we could safely use that maximum as the path delay. But we do not know the hard upper limit to its support. And extending that logic to a path delay distribution with support over all positive real values (although this is unrealistic for practical scenarios), the maximum path delay would be infinite and useless for our intentions.

To remedy this situation, we create a probabilistic definition of CServ Internetwork Service path delay. Specifically, we turn to a sharp one-sided tail bound on a probability distribution with a given mean and variance, but which is otherwise unknown. We are interested in a one-sided bound since we are concerned with the positive deviation from the path delay mean, thus allowing the use of a bound that

is tighter than a two-sided bound that also captures the negative deviation from the mean. We leverage Cantelli's Inequality [80] to prove a one-tailed variant of the celebrated Chebyshev's Inequality [81]. Cantelli's Inequality states that, for a real random variable X with finite mean μ and finite variance σ^2, the following holds:

$$\Pr\{X - \mu \geq a\} \leq \frac{\sigma^2}{\sigma^2 + a^2}, \tag{5.5}$$

where $a \geq 0$. With a simple substitution in Eq. (5.5), we find the sharp one-tailed variant of Chebyshev's Inequality with $k > 0$:

$$\Pr\{X - \mu \geq k\sigma\} \leq \frac{1}{1 + k^2}. \tag{5.6}$$

This is the most stringent possible bound for indiscriminate distributions with only known mean and variance. We do not make any further assumptions on the path delay distribution. The use of Eq. (5.6) to characterize CServ Internetwork Service path delay is a conservative approach since the inequality accounts for arbitrarily heavy-tailed path delay distributions.

Let the random variable D with unknown distribution but with known, finite expected value μ and known, finite variance σ^2 represent the delay from the source host to the destination host of a given path. We choose to define the *delay of a CServ Internetwork Service path*, d, as the value such that:

$$\Pr\{D < d\} \geq 0.99. \tag{5.7}$$

This definition of the delay of a CServ Internetwork Service path adopts a value such that we are "two-nines" confident that the realized delay of the path is less than the value. Since the distribution of D, save for its mean and variance, is otherwise unknown, we turn to the one-tailed variant of Chebyshev's Inequality developed in Eq. (5.6). Rearranging Eq. (5.6), we have:

$$\Pr\{X - \mu < k\sigma\} \geq 1 - \frac{1}{1+k^2} = \frac{k^2}{1+k^2} \; \forall \, k > 0. \tag{5.8}$$

Massaging Eq. (5.8) into a form more suited to our purposes and substituting the generic random variable X with our random variable D, we have the following:

$$\Pr\{D < d\} \geq \frac{\left(\frac{d-\mu}{\sigma}\right)^2}{1 + \left(\frac{d-\mu}{\sigma}\right)^2}, d > \mu. \tag{5.9}$$

This result matches the desired form of the definition of delay of a CServ Internetwork Service path from Eq. (5.7). Note that condition that requires that the delay value in the expression must be greater than the path mean. This results from our application of a one-sided inequality. It also provides a handy check condition; if the mean delay of the path is greater than the delay requirement specified by the CSR, we can be sure that the CServ Internetwork Service path does not meet the demands of the source host based on this definition of delay.

Solving for the right side of the expression in Eq. (5.9), we can find the required condition on the delay of a CServ Internetwork Service path such that $\Pr\{D < d\} \geq y, 0 \leq y \leq 1$:

$$\begin{aligned} &\frac{\left(\frac{d-\mu}{\sigma}\right)^2}{1 + \left(\frac{d-\mu}{\sigma}\right)^2} = y \\ &\Rightarrow d = \mu + \sigma\sqrt{\frac{y}{1-y}} \end{aligned} \tag{5.10}$$

With the result of Eq. (5.10) and $y = 0.99$ (from the definition of delay of a CServ Internetwork Service path), we require that $d \cong \mu + 9.95\sigma$ for $\Pr\{D < d\} \geq 0.99$.

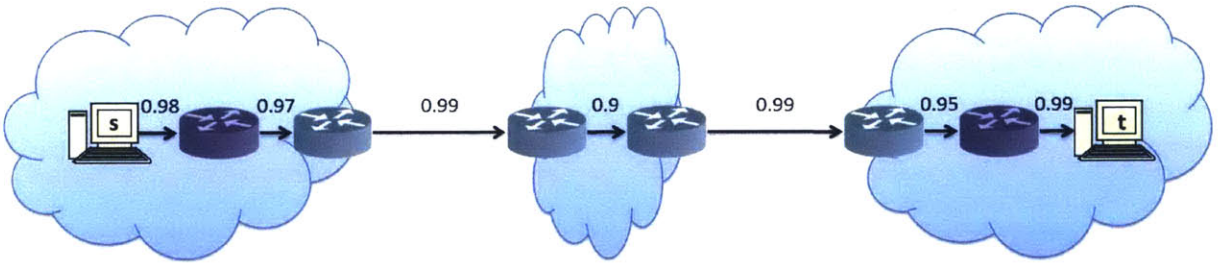
Based on the result above derived from the one-sided variant of Chebyshev's Inequality, we approximate the definition of delay of a CServ Internetwork Service path with finite mean μ and finite variance σ^2 as:

$$d = \mu + 10\sigma. \tag{5.11}$$

This definition ensures that $\Pr\{D < d\} \geq 0.99$ for any arbitrary delay distribution D with the given moments. We note here that equivalent definitions of the delay of a CServ Internetwork Service path with "three-nines" confidence or "four-nines" confidence require approximately 32 and 100 standard deviations above the mean value, respectively. This is generally considered too restrictive for our ultimate goal of discovering CServ Internetwork Service paths that meet stringent delay demands. The definition of Eq. (5.11) is already very conservative since the delay distribution D may be arbitrarily heavy-tailed, as long as it has finite mean and variance. Thus, it encapsulates an additional degree of assurance for better-behaved path delay distributions. This is not overtly captured by the "two-nines" confidence statement.

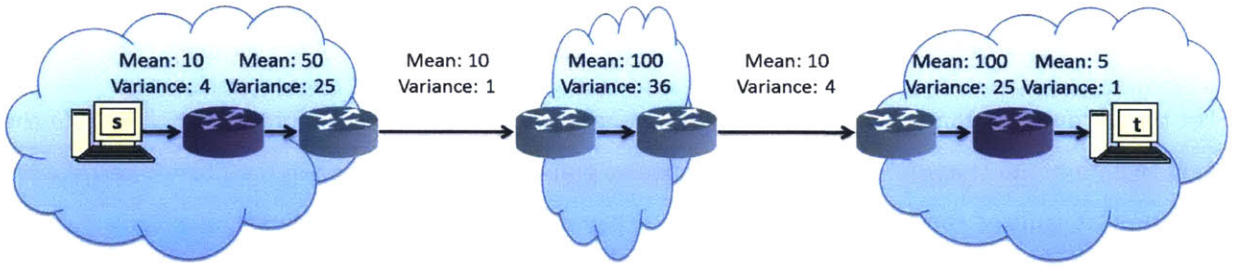
> **Definition 4** – The *delay d of a CServ Internetwork Service path* with aggregate path mean delay μ and aggregate path delay variance σ^2 is given as $d = \mu + 10\sigma$.

This definition is illustrated for the example of Fig. 5-5. As we can see from the example, our definition of delay can significantly extend the value beyond the path mean delay, which is frequently used alone in service-oriented literature to characterize service delay (for instance, [22]).



Path reliability = $0.98 \times 0.97 \times 0.99 \times 0.99 \times 0.9 \times 0.99 \times 0.95 \times 0.99 \cong 0.788623126$

Fig. 5-4: The last reported reliability values at the MC are shown for this example CServ Internetwork Service path. The path reliability is calculated according to the discussion of Section 5.2.3.



Path mean delay $\triangleq \mu = 10 + 50 + 10 + 100 + 10 + 100 + 5 = 285$
Path delay variance $\triangleq \sigma^2 = 4 + 25 + 1 + 36 + 4 + 25 + 1 = 96$
Delay of the CServ Internetwork Service path $\triangleq \mu + 10\sigma \cong 382.98$

Fig. 5-5: The last reported mean delay (in milliseconds) and delay variance values at the MC are shown for this example CServ Internetwork Service path. The path delay is calculated according to the definition of path delay presented in Section 5.2.3.

5.2.4 Finding Shortest Paths with Reliability and Delay

In Section 5.2.3, we presented the reliability and delay of a CServ Internetwork Service path. In doing so, we assumed that we already had the path from source host to destination host. But how do we find the CServ Internetwork Service path in the first place? The CSDCA must first discover a path from the source to destination hosts of a requested CServ transaction before it can compute the path reliability or the path delay; this is the chief objective of the discovery phase of the CSDCA. Recall that it is the primary CServ performance metric specified in the CSR that is used to direct CServ Internetwork Service path discovery. In this section, we present the general algorithm used for path discovery before specializing its use to the cases where either reliability or delay is the primary metric of interest.

For the purposes of this section, we consider a generic weighted network graph $G = (V, E, W)$. In the next section of the chapter, we consider the specific graphical representation of the CServ internetwork structure. It suffices for now that $V = \{1,2, \ldots, v\}$ is the generic graph vertex set, and E is a set of ordered pairs which represent directed edges in the graph. For example, $(1,2)$ indicates a directed edge from vertex 1 to vertex 2. For each edge $e \in E$, there exists an additive and nonnegative real-valued edge weight $w_e \in W$. In the context of a communication network, the vertex set can be considered a set of network devices, including endpoint hosts and routers, with unique addresses (where the enumeration of the vertex is a placeholder for the specific address in the addressing framework). Furthermore, the edges in the edge set can be viewed as logical communication links between the network devices in the vertex set. These are not necessarily physical connections. If the devices in the vertex set are network layer devices (say, for example, devices with IP addresses), there may be lower-layer switches subsumed by the logical edge representations.

Given a digraph G, Dijkstra's Algorithm is a well-known approach to find the "shortest paths" between an input source vertex $s \in V$ and all other vertices in $V \backslash \{s\}$ [50]. This algorithm is a classical instance of dynamic programming that employs memoization to minimize redundant computation [82]. Shortest paths are defined as those with the minimal sum over the edge weights of the edges that form the path. We call these sums the path distances.

Before we describe the mechanics of Dijkstra's Algorithm, we introduce a few conventions and additional notation. For edge weight conventions, we define $w_{e \notin E} = \infty$. In other words, for an ordered

pair that represents an edge not in the edge set, we define the edge weight to be infinite. We can imagine this as adding a directed edge between any given pair of vertices in the vertex set and then setting the edge weight to infinity if the edge was not in the original edge set. This also includes notional "self-edges," such as $(1,1)$. There is no concept of self-connections in the graph, and we can equivalently consider that these are edges with infinite weight. Next, we introduce a $|V| \times 1$ vector Γ that is used in the algorithm to store current estimates of the shortest distances from the source vertex to all other vertices. Specifically, $\Gamma_i, i \in V$ is the current estimate of the shortest distance from the source to vertex i. We also define a $|V| \times 1$ vector H that holds the current previous hop in the path that represents the current estimate of the shortest path from the source to all other vertices. This vector is referred to as the vector of "predecessor pointers." Namely, $H_i, i \in V$ is the previous vertex in the path from the source vertex to vertex i based on the current shortest path estimates. When Dijkstra's Algorithm terminates, the vectors Γ and H no longer represent estimates of the shortest distances and the predecessor pointers for the shortest paths, respectively. Rather, these vectors then hold the actual shortest distances and actual predecessor pointers for the shortest paths from the source vertex to all other vertices. For the purposes of the algorithm description, Q is the set of vertices for which the shortest path and shortest path distance has yet to be finalized. We also require one helper function in the algorithm description. The function $A = adj(i)$ returns a set of vertices $A \subset V$ that are adjacent to vertex i. That is, there is an edge $(i, a) \in E \; \forall \; a \in A$.

Our use of Dijkstra's Algorithm as a subroutine within the CSDCA is not interested in the entire set of shortest paths from a given source vertex, so we can terminate its execution once it discovers a shortest path to the intended destination. That being said, we present Dijkstra's Algorithm here in its most complete form and leave the discussion of CSDCA application-specific optimizations for later.

Dijkstra's Algorithm: $\Gamma, H = Dijkstra(G = (V, E, W), s)$

Inputs: G, s
Outputs: Γ, H
Initialize:

- $\Gamma_j := w_{(s,j)} \; \forall \; j \in V$
- $H_j :=$ undefined $\forall \; j \in V$
- $Q := V \backslash \{s\}$

While $Q \neq \emptyset$:

- **Step 1:** "Find the next closest vertex"
  - Find $i \in Q$ such that $\Gamma_i = \min_{j \in Q} \Gamma_j$
  - Set $Q := Q \backslash \{i\}$

- **Step 2:** "Update the estimates"
    - For each $j \in adj(i)$:
        - $\gamma = \Gamma_i + w_{(i,j)}$
        - If $\gamma < \Gamma_j$:
            - $\Gamma_j := \gamma$
            - $H_j := i$

Return: Γ, H

After the execution of Dijkstra's Algorithm, how do we extract the details of the shortest path from the source vertex to the destination vertex, say vertex $t \in V \backslash \{s\}$. The vector entry Γ_t gives us the distance of the shortest path from the source vertex, but it does not tell us the details of the path itself. That information is encoded in H. Each entry in H represents the immediate predecessor vertex hop in the shortest path from the source vertex. Leveraging the fundamental property that the subpath of a shortest path is itself a shortest path (this invariant is the foundation of Dijkstra's Algorithm), we can work backwards through the entries in H, starting with the destination vertex, to generate the details of the shortest path from the source vertex. To do so, it is useful to define a helper function, *extractshortestpath*. The input of the helper function is the output H from Dijkstra's Algorithm, along with the source and destination vertices $s \neq t \in V$. The output is an ordered h-tuple representing the vertices in the path beginning with s and ending with t, where h is the number of vertices in the shortest path. We let p denote a path data structure with several properties: $p.path$ is the h-tuple array that stores the explicit vertices in the path beginning with s and ending with t, $p.reliability$ is the reliability of the path p, and $p.delay$ is the delay of the path p. The method is easily implemented with a stack, or last in, first out data structure [83]. Consider the following pseudo-code for *extractshortestpath*.

p.path = extractshortestpath(H, s, t)

Inputs: H, s, t
Outputs: $p.path$
Initialize:

- $i := 1$
- $j := t$
- Create empty *stack*
- *stack*.push(t)

While $j \neq s$:

- *stack*.push(H_j)
- $j := H_j$

While *stack* is not empty:

- $p.path_i = stack$.pop
- $i := i + 1$

Return: $p.path$

When *extractshortestpath* returns, the output vector $p.path$ is an ordered tuple that gives the indices (or addresses) of the vertices in the shortest path from the source vertex s to the destination vertex t. This is a fast method since the stack operations can be implemented in amortized $O(1)$ time.

Dijkstra's Algorithm is known to run, with the most naïve implementation, in $O(|V|^2)$ time. Although for sparse graphs (with $|E| \ll |V|^2$), the use of a Fibonacci Heap to carry out the minimum operation in Step 1 improves the worst-case running time to $O(|E| + |V| \log(|V|))$. Since we make no assumptions on the sparsity of the internetwork graph, we use the polynomial running time result to characterize the process. Later in the chapter, it becomes clear that as the workhorse of the discovery phase, Dijkstra's Algorithm plays a key role in the CSDCA. The CSDCA is part of the pre-transmission control overhead for a CServ transaction, and thus the speed at which it is able to execute is critical. The delay incurred by the control overhead, including the CSDCA, has to be accounted for by appropriately reducing the CServ source host's maximum end-to-end CServ transaction delay allowance, which makes it more difficult to satisfy the demands of the CSR. For this reason, we need to consider the running time of not only Dijkstra's Algorithm, but of any additional component of the CSDCA.

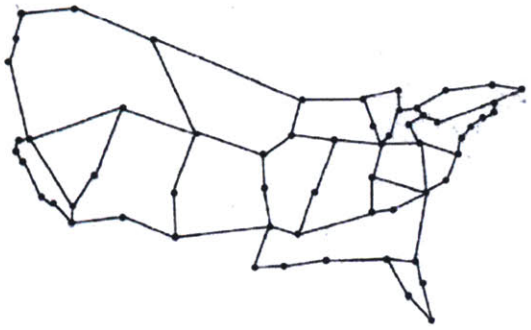
The execution time of an algorithm in absolute time is greatly dependent on the software and hardware used to implement it. For this reason, it is difficult to make statement in the vein of the following: "an instance of Dijkstra's Algorithm on the internetwork graph requires 100 nanoseconds to execute." As with other aspects of the CSDCA, we primarily focus on the asymptotic running time in an effort to ensure that all components execute in no worse than bounded polynomial time. The absolute running time of CSDCA and its subroutines would need to be characterized for the specific MC hardware and software implementation.

For illustration, however, we briefly ballpark the absolute execution time of Dijkstra's Algorithm for both a realistic optical backbone graph and a theoretical worst-case graph using a single high-performance, but commodity, processor. The realistic graph is taken from a U.S. optical backbone network, as depicted in Fig. 5-6 [84], where $|V| = 60$ and $|E| = 77$. Alternatively, a worst-case graph for the

algorithm would be one with the maximum number of edges, or a complete graph. A complete graph with $|V|$ vertices has $|V|(|V|-1)$ edges. The reference commodity processor used for the analysis is the widely-available Intel i7 Extreme Edition 3960X which executes 177.73 GIPS at 3.33 GHz. In the analysis, we count all algorithm operations and, somewhat unsophisticatedly, treat each operation as a single processor instruction in order to develop the absolute running time.

For a naïve implementation of Dijkstra's Algorithm, it takes approximately 64.6 nanoseconds to execute on the example graph of Fig. 5-6 using the reference processor. This is a fairly negligible execution time considering the minimum CSR delay that we anticipate is on the order of a second. Rather than rely on the result from this particular network graph, we additionally consider the execution time on a class of worst-case complete graphs as a function of the cardinality of the vertex set, or $|V|$. The result is shown in Fig. 5-7, which allows us to observe the polynomial growth with the graph size. Even so, with 1000 vertices, our first-cut analysis of the absolute execution time on a commodity processor doesn't exceed 50 microseconds. And we do not expect that the internetwork graph will approach this scale or this connection density for practical deployments of the CServ architecture. The internetwork graph is considered more thoroughly in Section 5.3.

Now with Dijkstra's Algorithm (and the *extractshortestpath* helper algorithm) in hand, we have the necessary tools to find optimal paths on a network graph with nonnegative, additive edge weights. Our goal is to discover CServ Internetwork Service paths from a given source host (or vertex) to a given destination host (or vertex) based on the primary CServ performance metric specified in the CSR, namely reliability or delay. In Section 5.2.3 we described that the reliability of a CServ Internetwork Service path is a *product* of the reliability values, or edge weights, along the path – not a sum of the weights. Furthermore, we discussed that the characterization of the delay of a CServ Internetwork Service path involved summing the mean delay edge weights and the delay variance edge weights along the path *individually*, not as one additive metric on each edge. How exactly can we leverage Dijkstra's Algorithm, then, to find paths in terms of either of these types of metrics? In one case, the answer is simple and straightforward. In the other case, it is not so cut and dry.



Parameter	Value
Number of nodes	60
Number of links	77
Average node degree	2.6
Largest node degree	5
Least node degree	2

Fig. 5-6: This example U.S. optical backbone network is reproduced from [84] and used to characterize the absolute running time of an instance of Dijkstra's Algorithm on a realistic network graph.

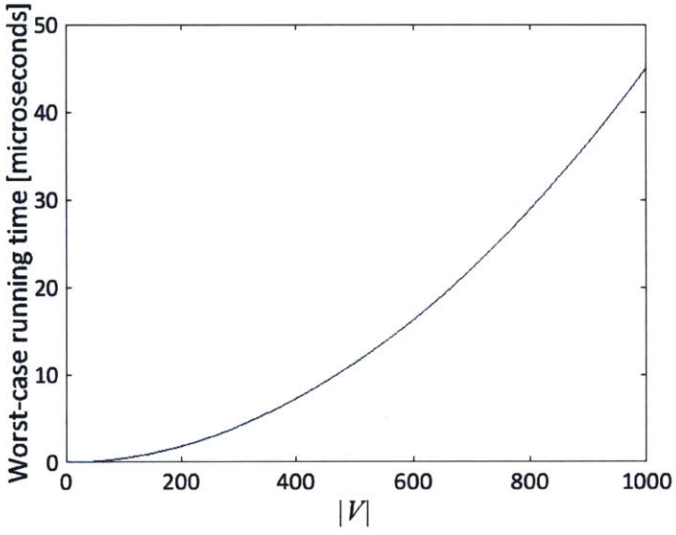


Fig. 5-7: The worst-case execution time of Dijkstra's Algorithm on a complete graph with $|V|$ vertices using the Intel i7 Extreme Edition 3960X reference processor.

5.2.4.1 Dijkstra's Algorithm and Reliability

Let us consider how to use Dijkstra's Algorithm to find a CServ Internetwork Service path when the primary CServ performance metric specified by the CSR is reliability. We continue to use a generic network digraph G, but the additive, nonnegative edge weight set W is left unspecified for the moment. Let us assume, instead, that we have an edge weight set P such that there exists a last reported reliability value $\rho_e \in P$ from the State Measurement Service for each edge $e \in E$, where $0 \leq \rho_e \leq 1$ $\forall\, \rho_e \in P$. Because the reliability of a CServ Internetwork Service path is a product of the reliability of the hops in the path, not the summation of the reliability of the hops, we cannot directly use P as an additive, nonnegative edge weight set. However, there is a convenient mathematical transform that allows us to map the reliability values in the edge weight set P to appropriate values in the nonnegative, additive edge weight set W such that we can use Dijkstra's Algorithm as presented to find a CServ Internetwork Service path with optimal path reliability.

We consider the following transformation of a reliability value $\rho_e \in P$ to a nonnegative, additive edge weight $w_e \in W$:

$$w_e = -\ln(\rho_e). \tag{5.12}$$

The function $f(x) = -\ln(x)$ is monotonically decreasing on the interval $0 < x \leq 1$, as shown in Fig. 5-8. At the boundaries, $-\ln(0) = \infty$ and $-\ln(1) = 0$. Thus, an edge with $\rho_e = 1$ maps to a nonnegative, additive edge weight of 0, the minimal value over the support of the transform function, while an edge with $\rho_e = 0$ (which can equivalently be considered the lack of an edge in terms of reliability) maps to a nonnegative, additive edge weight of infinite value, the largest over the transform function's support. As Dijkstra's Algorithm searches for the shortest path with the minimized sum of its edge weights, it would therefore find the most reliable path using this mapping $P \rightarrow W$.

Additionally, the transform of Eq. (5.12) provides a convenient inversion method to recover the reliability of the discovered CServ Internetwork Service path. Consider the effect of the transform on the expression for the reliability of a CServ Internetwork Service path from Eq. (5.4) using the simple integer enumeration of the hops in the path and letting w represent the transformed sum weight of the path:

$$\rho = \prod_{i=1}^{h} \rho_i$$
$$\Rightarrow w = \sum_{i=1}^{h} w_i = \sum_{i=1}^{h} -\ln(\rho_i). \tag{5.13}$$

Note that for destination vertex $t \in V$, the value $w = \Gamma_t$ from the output of Dijkstra's Algorithm. Considering Eq. (5.13), we note that we can find ρ from w with the following inverse transform:

$$\rho = e^{-w} = e^{-\Gamma_t}. \tag{5.14}$$

After the execution of Dijkstra's Algorithm using the nonnegative, additive edge weight set W generated from the reliability edge weight set P using Eq. (5.12), we can recover the reliability of the shortest (or most reliable) path and the details of the most reliable path itself using the output of the algorithm and Eq. (5.14). This transform and inverse transform process is demonstrated visually for an example CServ Internetwork Service path in Fig. 5-9. The delay of the discovered CServ Internetwork Service path can then be computed using the sums of the last reported mean delay and delay variance performance metrics and Definition 4.

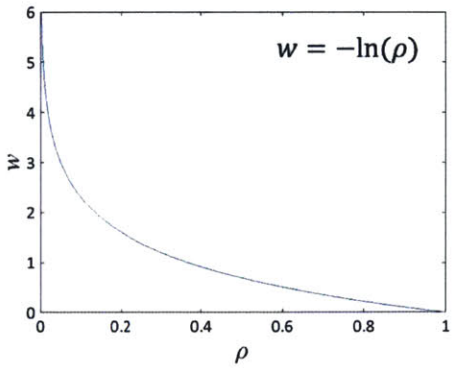


Fig. 5-8: This transform is used to represent the reliability CServ performance metric as a nonnegative, additive graphical edge weight.

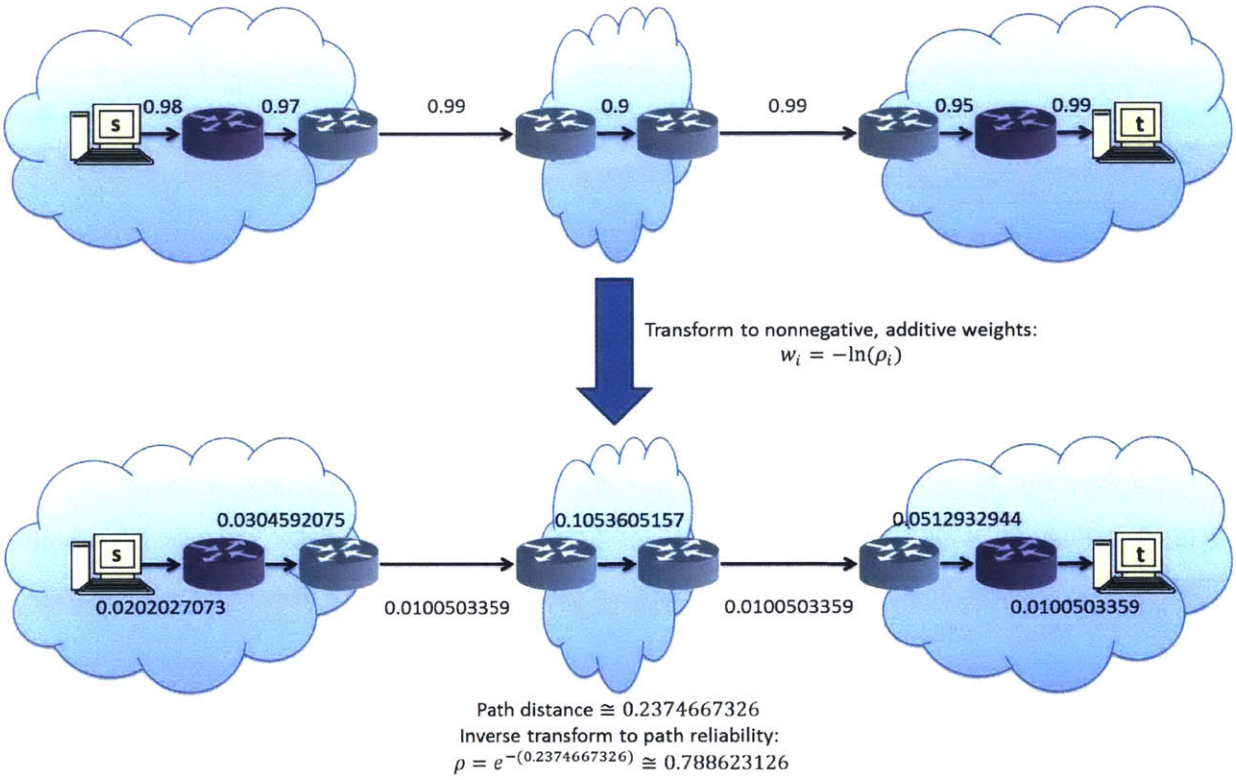


Fig. 5-9: This example illustrates the transformation of reliability edge weights and the inversion to recover the CServ Internetwork Service path reliability from the output of the shortest path algorithm.

5.2.4.2 Dijkstra's Algorithm and Delay

We now wish to use Dijkstra's Algorithm to find a CServ Internetwork Service path when the primary CServ performance metric specified by the CSR is delay. While the algorithmic discovery of the most reliable CServ Internetwork Service path given the internetwork graph and last reported CServ reliability performance metrics only required a weight transform function, the discovery of an optimal delay CServ Internetwork Service path is not so straightforward.

In the terminology of equilibrium theory [85], we can *a priori* assign a total preference order the edges of the internetwork graph in terms of reliability. Consider edge (i,j) and edge (i,k) with their associated last reported reliability metrics from the State Measurement Service, $\rho_{(i,j)}$ and $\rho_{(i,k)}$, respectively. Let $\mathbb{P}$ represent a preference operator on edges, where $(i,j)\ \mathbb{P}\ (i,k)$ reads "edge (i,j) is at least as preferred as edge (i,k)." If $\rho_{(i,j)} \geq \rho_{(i,k)}$, we can say $(i,j)\ \mathbb{P}\ (i,k)$ with respect to reliability. And if $\rho_{(i,j)} < \rho_{(i,k)}$, we can say $(i,k)\ \mathbb{P}\ (i,j)$ with respect to reliability. Note that the tiebreaker between the two values is arbitrary. In this same way, we can create a total preference order using the $\mathbb{P}$ operator for all edges $e \in E$ with respect to their reliability values. This fact underlaid our success with the reliability transform function in Eq. (5.12), mapping reliability values into nonnegative, additive edge weights for the purpose of Dijkstra's Algorithm. The transform preserved the implicit ordering on the edges in terms of reliability. Let us denote the reliability weight transform generically here as $f_\rho(\cdot)$. With this notation, the previous statement says that if $(i,j)\ \mathbb{P}\ (i,k)$ in terms of reliability, then $(i,j)\ \mathbb{P}\ (i,k)$ in terms of Dijkstra's Algorithm after the function $f_\rho(\cdot)$ is applied to the reliability edge weight set P for the edges $e \in E$.

Consider that for each edge in the internetwork graph, there is, along with the reliability values, an associated pair of statistics reflecting the most recently reported mean delay and delay variance from the State Measurement Service. For an edge (i,j), we can represent this pair of mean delay and delay variance statistics as $\left(\mu_{(i,j)}, \sigma^2_{(i,j)}\right)$. Now we consider two edges (i,j) and edge (i,k) with their associated last reported delay statistics metrics from the State Measurement Service. Comparing the mean delays and delay variances pairwise, consider all the following cases:

1. $\mu_{(i,j)} = \mu_{(i,k)}$, any relationship between $\sigma^2_{(i,j)}$ and $\sigma^2_{(i,k)}$;

2. $\sigma^2_{(i,j)} = \sigma^2_{(i,k)}$, any relationship between $\mu_{(i,j)}$ and $\mu_{(i,k)}$;
3. $\mu_{(i,j)} < \mu_{(i,k)}, \sigma^2_{(i,j)} < \sigma^2_{(i,k)}$;
4. $\mu_{(i,j)} > \mu_{(i,k)}, \sigma^2_{(i,j)} > \sigma^2_{(i,k)}$;
5. $\mu_{(i,j)} > \mu_{(i,k)}, \sigma^2_{(i,j)} < \sigma^2_{(i,k)}$;
6. $\mu_{(i,j)} < \mu_{(i,k)}, \sigma^2_{(i,j)} > \sigma^2_{(i,k)}$.

In cases 1-4, we can compare the two edges using the preference operator $\mathbb{P}$. Consider case 1 first. If the mean delays are equal, then we prefer the edge with the smaller delay variance (with an arbitrary tie-breaker). In case 2 where the delay variances are equal, we prefer the edge with the smaller mean delay value (again with an arbitrary tie-breaker). Case 3 allows us to clearly state $(i,j)\ \mathbb{P}\ (i,k)$, while $(i,k)\ \mathbb{P}\ (i,j)$ for case 4. But what about cases 5 and 6? In terms of the ultimate path delay as defined in Definition 4, how do we know *a priori* if we prefer an edge with greater mean delay and smaller delay variance or an edge with smaller mean delay and greater delay variance? The answer is that we do not. Pairs of mean and variance values do not form a well-order. And because of this, we cannot generally assign a total preference order for the edges $e \in E$ with respect to their delay statistics. Consequently, there is no deterministic function that allows us to transform these statistic pairs into nonnegative, additive edge weights that preserve any preference ordering.

Abstractly, given a generic network digraph G and a source and destination vertex pair $s \neq t \in V$, imagine that we can enumerate all possible paths between the two endpoints and calculate their delay according to Definition 4. We index these paths $i \in \{1,2,\ldots,z\}$ and denote their delays as $\{d_i\}_{i\in\{1,2,\ldots,z\}}$. For simplicity, we assume that these paths can be indexed such that $d_1 \leq d_2 \leq \cdots \leq d_z$. Let $\mathbb{H}$ represent a preference operator on paths, where $i\ \mathbb{H}\ j$ reads "path i is at least as preferred as path j." In terms of delay according to Definition 4, we can say that $1\ \mathbb{H}\ 2\ \mathbb{H}\ \ldots\ \mathbb{H}\ z$. Ideally, we would like a method to develop a nonnegative, additive edge weight for G, accounting for the contributions of both the mean delay and delay variance statistics, such that this total preference ordering is preserved. In other words, if the sum of the nonnegative, additive edge weights for path i is denoted φ_i, we want $\varphi_1 \leq \varphi_2 \leq \cdots \leq \varphi_z$. If this is possible, then the use of Dijkstra's Algorithm is guaranteed to find the path with the optimal delay according to Definition 4, or path 1 in this example. Unfortunately, because of the result introduced above for the edges that makes it impossible to *a priori* determine a total preference order, it is not possible to preserve this path ordering with a deterministic transform.

Additionally, the definition of the delay of a CServ Internetwork Service path cannot be decomposed into additive edge-by-edge weights since the presence of the standard deviation in the expression of Eq. (5.11) couples the delay variance terms from each hop in the path.

If we cannot rely on Dijkstra's Algorithm as a subroutine to discover a CServ Internetwork Service path with optimal delay, how do we discover such a path? The only known solution at this time is to enumerate all possible paths between the source and destination vertices, compute the delay for each path according to Definition 4, and then choose the path with the minimum delay. Unfortunately, for a generic graph, there may be $\sim|V|!$ paths between a given vertex pair. More formally, in [86], it was shown that the enumeration of all possible paths between a pair of vertices in a graph belongs to the class of computationally equivalent $\#P$-complete counting problems that are at least as difficult as NP-complete problems. The proposed use of Dijkstra's Algorithm is for its polynomial-time execution; we do not wish to fall back on the brute-force enumeration of exponentially-many paths for the discovery phase of the CSDCA, as this is highly likely to bust any reasonable CSR maximum delay requirement.

Before becoming completely flummoxed, we consider the true objective of the CSDCA. The goal of the algorithm is to discover paths that satisfy the requirements of the CSR, not necessarily to discover and present the *optimal* CServ Internetwork Service paths as the service solution. So the fact that we do not have a computationally efficient method to guarantee discovery of the CServ Internetwork Service path from a source vertex to a destination vertex with the minimal delay is not automatically an issue for the CServ architecture and the CSDCA. If we discover a suboptimal path that satisfies the delay requirement of the CSR, then we have a perfectly valid path to include as the presented service solution or part of the service solution when employing subnet-disjoint diversity routing.

We propose the following approach to this problem. We create a heuristic nonnegative, additive edge weight from the last reported mean delay and delay variance CServ performance metrics for each edge in the internetwork graph. With these heuristic edge weights, we discover paths in the CSDCA, when delay is specified as the primary CServ performance metric in the CSR, with Dijkstra's Algorithm. And to validate the choice of heuristic as a reasonable one (although surely not the only one), we run random network simulations and benchmark its performance against the status quo approach in the literature (which relies purely on mean delay values as naturally nonnegative, additive edge weights without transform).

Leveraging the definition of delay of a CServ Internetwork Service path, we suggest to probabilistically bound the delay of each edge in G in the same way as the heuristic nonnegative, additive edge weight. This bound gives us the delay of the edge based on the reports of the State Measurement Service such that we are sure the actual realized delay of that edge is less than the weight value with "two-nines" probability. We then treat these probabilistic maximum delay values as additive weights such that the path distance for a given CServ Internetwork Service path is the sum of the weights of the edges in the path.

Let us formally describe the transformation, which we call the *Cantelli transform*. We continue to use the generic network digraph G, and we momentarily specify the values in the additive, nonnegative edge weight set W. We assume that we have an edge weight set M such that there exists a last reported mean delay value $\mu_e \in M$ from the State Measurement Service for each edge $e \in E$, where $\mu_e > 0 \; \forall \; \mu_e \in M$. Furthermore, we assume that we also have an edge weight set Υ such that there exists a last reported delay variance value $\sigma_e^2 \in \Upsilon$ from the State Measurement Service for each edge $e \in E$, where $\sigma_e^2 \geq 0 \; \forall \; \sigma_e^2 \in \Upsilon$. We propose the use of the following transformation of a mean delay value $\mu_e \in M$ and a delay variance value $\sigma_e^2 \in \Upsilon$ to a nonnegative, additive edge weight $w_e \in W$ such that the random delay of the edge with unknown distribution other than the finite mean and variance is less than the nonnegative, additive edge weight with probability 0.99:

$$w_e = \mu_e + 10\sqrt{\sigma_e^2}. \tag{5.15}$$

The rationale for this transform was developed previously in Section 5.2.3. As $\mu_e > 0$ and $\sigma_e^2 \geq 0$, $w_e > 0$, satisfying the nonnegative requirement of an edge weight in the set W. As Dijkstra's Algorithm searches for a path based on delay as the primary CServ performance metric, it uses the mapping $\{M, \Upsilon\} \rightarrow W$ as specified in Eq. (5.15).

We note here that, unlike the transform for reliability discussed in Section 5.2.4.1, the Cantelli transform of Eq. (5.15) does not provide a convenient method to recover the delay of the discovered CServ Internetwork Service path. After the algorithm executes, the path distance value Γ_t from the output for destination vertex $t \in V$ cannot be inverted directly to find the path delay since the end-to-end delay metric is not decomposable. Instead, the resulting discovered CServ Internetwork Service path, mined

from H with *extractshortestpath*, is used to compute the delay of the path d using the path sum mean μ and sum variance σ^2 from the original edge delay statistics, $\{M, Y\}$, and Eq. (5.11). The Cantelli transform process is demonstrated visually for an example CServ Internetwork Service path in Fig. 5-10.

The only useful thing that we can prove from the Cantelli transform is that the resulting $\Gamma_t \geq d$. This follows from the subadditive property of the square root function. Consider a discovered CServ Internetwork Service path from source vertex $s \in V$ to destination vertex $t \in V$ where we enumerate the hops of the path as $i = \{1,2, \ldots, h\}$. Based on the transform of Eq. (5.15), we have the following path distance in terms of the reported delay statistics for each hop of the path:

$$\Gamma_t = 10\left(\sum_{i=1}^{h}\sqrt{\sigma_i^2}\right) + \sum_{i=1}^{h}\mu_i. \tag{5.16}$$

Alternatively, using the result of Eq. (5.11), we have the following form for the delay of the CServ Internetwork Service path in terms of the reported delay statistics for each hop of the path:

$$d = 10\left(\sqrt{\sum_{i=1}^{h}\sigma_i^2}\right) + \sum_{i=1}^{h}\mu_i. \tag{5.17}$$

The function $f(x) = x^a$ is subadditive for $0 < a < 1$ (and specifically, for $a = 1/2$ for our purposes here), meaning that $f(b + c) \leq f(b) + f(c)$. Applying the subadditive property to Eqs. (5.16)-(5.17), we have:

$$\left(\sum_{i=1}^{h}\sqrt{\sigma_i^2}\right) \geq \left(\sqrt{\sum_{i=1}^{h}\sigma_i^2}\right),$$

and the stated result that $\Gamma_t \geq d$ follows. What does this result mean for the CSDCA? If the path distance of the discovered CServ Internetwork Service path using the heuristic nonnegative, additive edge weight satisfies the delay requirement of the CSR, then the actual delay of that CServ Internetwork

Service path certainly does. From that point of view, we do not need to compute the actual delay of the CServ Internetwork Service path explicitly. However, if the path distance of the discovered path using the Cantelli transform exceeds the delay requirement of the CSR, then we still need to compute the actual delay of the CServ Internetwork Service path since it may or may not satisfy the demands of the CSR requirement.

The question remains as to the effectiveness of the Cantelli transform. How often does it discover the actual optimal delay CServ Internetwork Service path? Does it outperform the status quo which employs only the mean delay values as edge weights? And, most importantly, how frequently does it fail to find a CServ Internetwork Service path that meets the delay requirement of the CSR when there exists such a path? To answer these questions, we create a random network simulation to study the performance of the proposed heuristic.

We generate random internetwork topologies of the form of Fig. 5-11. The variable of the experiment is the number of candidate subnet-disjoint CServ Internetwork Service paths from a given source to a given destination. The integer number of edges or hops in each candidate path is chosen independently from a fixed uniform distribution with support over the range [3,5]. Once a random internetwork topology is generated, we draw the delay statistics for each edge, namely the mean delay and delay standard deviation, independently from two separate uniform distributions. For the results presented, the mean and standard deviation of the mean delay generating distribution are 100 and 10 milliseconds, respectively. And the mean and standard deviation of the delay standard deviation generating distribution are 10 and 5 milliseconds, respectively.

Once the random network instance is fully characterized, we compute the nonnegative, additive edge weights using the Cantelli transform as previously described and leverage Dijkstra's Algorithm to discover the shortest path based on these weights. The resulting path with the minimal path distance is compared to the actual minimum delay path based on the definition of the delay of a CServ Internetwork Service path in order to calculate the frequency with which the optimal CServ Internetwork Service path is discovered over 10,000 random internetwork realizations per independent parameter value. The results of the simulation are depicted in Fig. 5-12.

The results indicate that the heuristic approach clearly outperforms the status quo technique. The difference in their performance diverges as the number of candidate subnet-disjoint CServ Internetwork Service paths increases, although they both eventually stabilize. At the point when the performance begins to find a steady state, the path discovery approach using mean values as the edge weights finds the optimal delay path less than 30% of the time, whereas the Cantelli transform heuristic does so more than 80% of the time. It would seem that the suggested heuristic is a serviceable method to find CServ Internetwork Service paths with reasonable delay, even if it misses the path with the minimum delay 13% of the time when there are ten candidate subnet-disjoint paths (which may be near the maximum for realistic CServ network deployments) in our simulation. Similar comparative performance results were observed for other generating distributions and simulation parameters.

But as we discussed previously, it is not really the frequency at which the algorithm fails to find the optimum path in terms of delay that matters; it is the frequency at which the algorithm fails to find a path that meets the CSR delay requirement when that path exists in the network. Using the same simulation framework, we introduce the CSR delay requirement as a variable and quantify the frequency with which Dijkstra's Algorithm with the Cantelli transform misses the discovery of a CServ Internetwork Service path that satisfies the delay requirement of the CSR when it exists. As before, 10,000 random network instances are used to characterize the performance for each tested value of the CSR delay requirement. The number of candidate subnet-disjoint paths in each simulated network is 10, and the integer number of hops per candidate path is again pulled independently from a uniform distribution over the support [3,5]. The delay statistics generating distributions are left just as before. The results of this experiment are visualized in Fig. 5-13.

Interestingly, there is a narrow window of opportunity as a function of the CSR delay requirement where the algorithm may fail to find an existing path that meets the demand. This period occurs near the tail of the observed "solution existence phase transition," after which paths that satisfy the CSR delay requirement always exist. At its peak, the frequency of missing existing paths that satisfy the requirement is only around 2%. However, once the CSR delay requirement reaches the point at which solutions are almost always available, the heuristic algorithm no longer misses existing solutions. Rather, there is an approximate steady-state distribution over finding the optimal delay path or finding a path with delay that meets the CSR requirement but is suboptimal. We note here that the steady state frequency of discovering optimal delay paths matches the results previously observed in Fig. 5-12 when

the number of candidate CServ Internetwork Service paths is also 10. As the CSR delay requirement continues to increase, it becomes increasingly trivial for the algorithm to discover a path that represents a solution as long as the random network parameters are held constant.

The bottom-line message developed by this simulation is that there is negligible risk of overlooking valid paths in the operational regime where CServ Internetwork Service path solutions are readily available for most network instances, say greater than 99% of the time. Thus, the strategy for CSDCA success during the discovery phase when delay is the primary CServ performance metric is to avoid CSR generation with delay requirements riding the ragged edge of solution availability. It is in this regime that there is the greatest risk of missing existing path solutions. As long as CSR delay demands fall in the regime where CServ Internetwork Service path solutions are readily available, the heuristic algorithm using the Cantelli transform should find a path that meets the requirements even if it is not a path with optimal delay. The CServ source host should generate CSRs with the maximum tolerable delay for its critical messaging application. If the delay requirement is artificially tightened by the service requestor, there is a risk of service access denial when the CSDCA cannot discover (or compose) CServ Internetwork Service that satisfies the overly restrictive requirements.

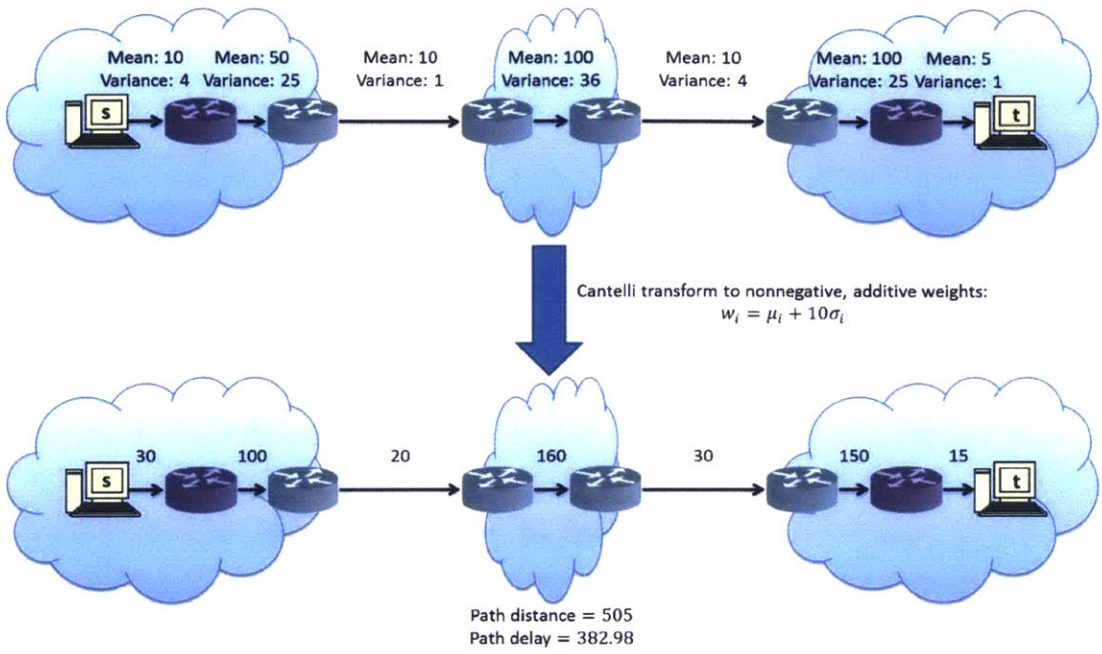


Fig. 5-10: This example illustrates the Cantelli transformation of delay statistic edge weights to nonnegative, additive edge weights for Dijkstra's Algorithm, as well as the difference between the resulting path distance output of the algorithm and the CServ Internetwork Service path delay.

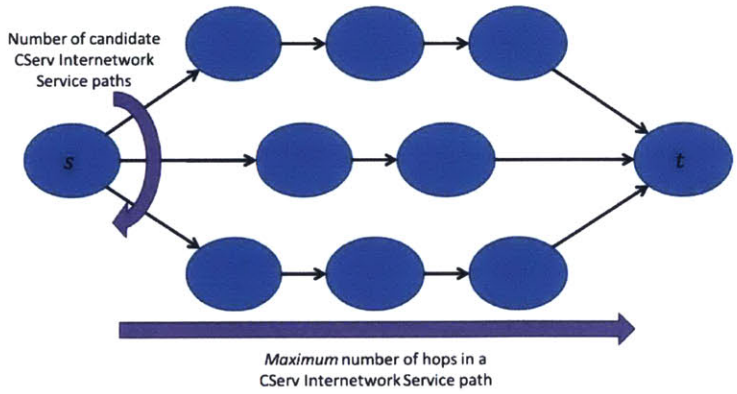


Fig. 5-11: Random internetwork topologies for the simulation are generated using a particular number of candidate CServ Internetwork Service paths and random number of hops per path drawn from a uniform distribution with a particular maximum value.

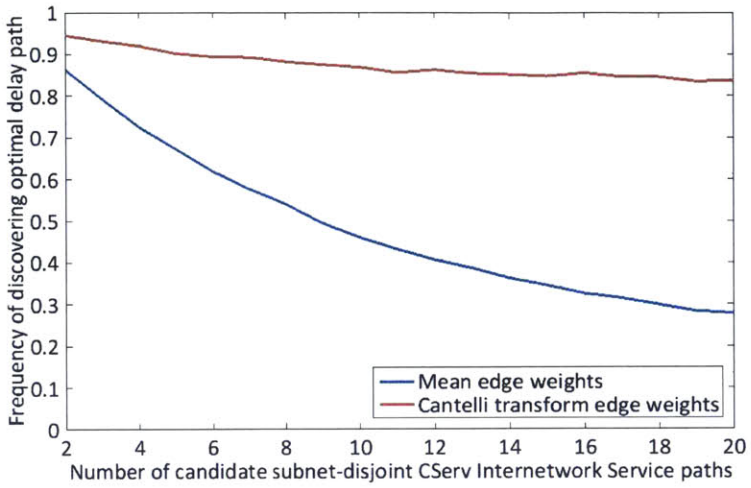


Fig. 5-12: These random simulation results show the frequency at which Dijkstra's Algorithm discovers the optimal delay CServ Internetwork Service path using the Cantelli transform versus the mean delay values as edge weights.

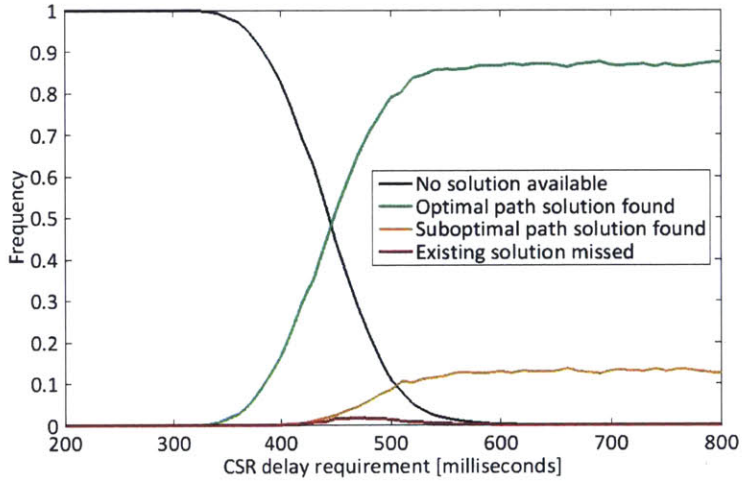


Fig. 5-13: These random network simulations show the frequency with which Dijkstra's Algorithm, using the Cantelli transform to generate nonnegative, additive edge weights, misses the discovery of an existing CServ Internetwork Service path solution that meets the CSR delay requirement.

5.3 The Graphical Internetwork Representation

In the previous sections, we referred to a generic network graph of vertices and edges without context. We are now prepared to discuss the actual CServ internetwork structure at the MC's global level and its graphical representation. The importance of this internetwork graph is that it is the primary input to the CSDCA, along with the information from the CServ source host-generated CSR. The constant maintenance and availability of this graph at the MC is fundamental to the operation of the CServ per-transaction, pre-transmission control overhead.

We briefly recall the types of reports generated by the State Measurement Service running in each subnet that are propagated to the MC. After the SC filters the reports that are relevant only to intranetwork service and intranetwork CServ transactions, the remaining CServ performance metric estimation reports used by the MC are:

1. internal active gateway router to active gateway router performance measurements;
2. external active gateway router to internal active gateway router peering connection performance measurements;
3. internal active access router to active gateway router, and vice versa, performance measurements;
4. and each active access router's access network upstream and downstream performance measurements.

Throughout the remainder of this chapter, we use the following notation when analyzing data size and algorithmic complexity. This terminology differs slightly from that of Chapter 4 where the focus was on the individual subnet. First, we assume that there are n_s subnets that form the network, the addresses of which are members of the set N_s. In each subnet, there are n_g active CServ-enabled gateway routers and n_a access routers. The assumption on the same number of active routers per subnet (and same fractional division between active access and gateway routers) is for analytic convenience; the actual number of active access and gateway routers would likely vary from subnet to subnet based on the subnet's topology, structure, function, communication technology, and network operator. Also for the purpose of an analytical model, we assume that each active gateway router has $p \geq 1$ peering connections with active gateway routers in adjacent subnets and that each active access router in the

subnet has only one connected access network. Applying this terminology to the types of State Measurement Service reports generated and transmitted to the MC from each subnet, we have that:

1. $n_g(n_g - 1)$ internal active gateway router to active gateway router performance metric entries;
2. pn_g external active gateway router to internal active gateway router performance metric entries;
3. $2n_g n_a$ performance metric entries between internal active gateway routers and active access routers;
4. and $2n_a$ active access router upstream and downstream performance metric entries;

are stored, and updated by future reports, at the MC.

As we present the discussion of the internetwork graph and the CSDCA, we rely on the recommendation from Section 3.1.1 regarding the use of hierarchical addressing in the CServ network. Specifically, we assume that given the address for a particular CServ-enabled host, the MC can quickly resolve the address of the upstream active access router that provides connection to the subnet routing core and the address of that subnet using simple bitmask operations on the host address. In other words, the endpoint host address contains the addresses of the upstream active access router and its subnet. And likewise, the upstream access router's address contains the address of its subnet.

As one final prerequisite for the discussion, we assume the existence of a database at the MC that contains the identities of the participants in CServ operation. The population and maintenance of this database is the result of the association-disassociation protocol that we have previously referred to, although its implementation has not been explicitly covered in this dissertation. The concept is that CServ-enabled host devices and active CServ-enabled routers need to inform the MC (and their respective SCs) when joining or leaving the network. A CServ-enabled endpoint that has not associated with CServ cannot request service until it has done so. Furthermore, an active CServ-capable router cannot be part of the CServ network or internetwork graph until it has associated. The MC maintains a data structure with the addresses of associated devices, which can be implemented in hierarchical fashion in line with the hierarchical addressing structure. The most important aspect of this protocol is that a multihomed CServ-enabled host with multiple addresses (corresponding to multiple upstream access routers and, possibly, multiple subnets) needs to associate in order to bind these addresses as

one device. In this way, when a service request is made with one of those addresses as the source or destination of the CServ transaction, the MC can map that address to the set of addresses representing the same device in order to fully leverage the multihomed connections for the discovery and composition of CServ Internetwork Service. This is considered in more detail later in this section and in the next section covering the CSDCA.

5.3.1 The "Natural" Representation

In Section 2.2.1, we introduced the subnet as the logical participant in the CServ global internetwork routing architecture. CServ Internetwork Service paths for critical datagrams follow explicit subnet-to-subnet granularity routes, where the internal routing behavior of the subnet (the CServ Intranetwork Service for that subnet) appears to be a "black box" to the rest of the global network. This lead us to illustrate the network at a global abstraction level as shown in Fig. 5-14 (a simple internetwork example with only four subnets), highlighting the role of the subnet as the central logical participant for CServ Internetwork Service determination.

This type of representation seems most "natural" given our description of the CServ architecture. The source host is part of a source subnet, the destination host (of an internetwork CServ transaction) is part of a destination subnet, and the role of the MC is to discover paths of transit subnets that, together, get the critical message from the source subnet to the destination subnet with a particular level of reliability and within some probabilistically guaranteed amount of time. The type of representation of Fig. 5-14 not only represents the subnets participating in the CServ architecture at the global level, but it specifies the logical peering relationships between them. An edge in this graph connecting two subnet vertices tells us that there is at least one peering connection between active gateway routers in the subnets. Thus, following the connectivity of this graph, we can say that there exists a path between a pair of source and destination subnets. It even empowers us to search for the number of subnet-disjoint paths available between a particular pair.

But this is not enough for the purposes of the CServ architecture. First, consider that the edges only tell us about logical peering connections – that there is *at least one* pair of peering active gateway routers between those neighboring subnets. If there are several pairs of peering active gateway routers that comprise that logical peering connection, then the discovery and specification of a CServ Internetwork

Service path may need to distinguish between the use of one or the other. For example, imagine the scenario where one physical peering connection between two neighboring subnets is perfectly reliable and has almost zero delay according to the reports of the State Measurement Service, while a second physical peering connection loses every other transiting datagram and has high delay variability. The discovery of a CServ Internetwork Service path that includes that sequence of neighboring subnets in the path would clearly want to explicitly specify the use of the first physical peering connection. And second, consider that the performance experienced during the transit of a particular subnet by an internetwork CServ datagram could (and probably does) depend on the physical pair of active gateway routers used as the ingress and egress points for that subnet. The State Measurement Service used in a particular subnet learns the CServ performance metrics between physical active gateway router pairs, not between sets of gateway routers that together represent logical peering points with an adjacent subnet. An understanding of the ingress and egress logical peering connections would not provide this desired granularity of insight into the CServ Internetwork Service path.

So although the subnet is the central participant in the internetwork at the global level, particularly in the determination of diversity-routed service that uses subnet-disjoint paths, the emphasis is on the fact that it is the central *logical* participant. The objective of the CSDCA is to discover subnet-to-subnet granularity paths, but it must do so in terms of the physical active routers that participate in the service path in order to have a complete understanding over the performance of a particular path and control over the composition of service using subnet-disjoint CServ Internetwork Service paths. Therefore, the "natural" representation depicted in Fig. 5-14 is not the representation of the internetwork topology maintained by the MC. In the next section, we formalize the actual CServ internetwork graphical representation.

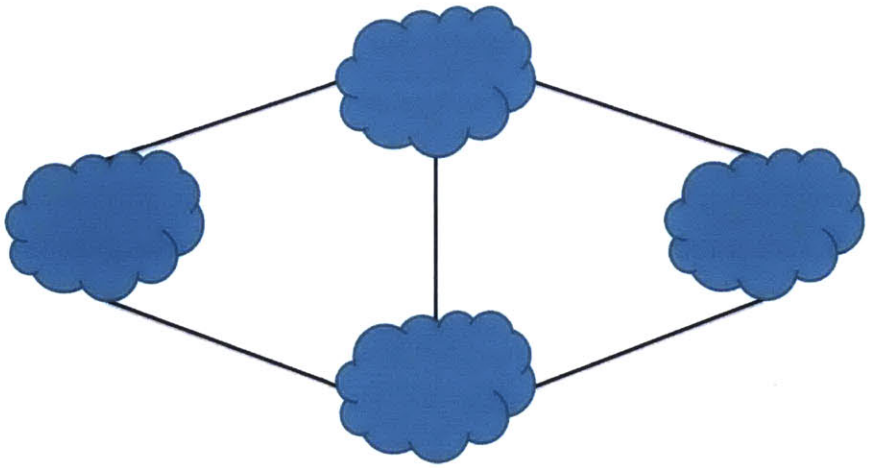


Fig. 5-14: This generic example demonstrates the "natural" representation of the CServ internetwork structure with the subnet as the central participant. In this representation, the vertices signify subnets while the edges represent logical peering connections between adjacent, or neighboring, subnets.

5.3.2 The Common Internetwork Representation

The discussion of Section 5.3.1 motivates the actual internetwork graphical representation maintained by the MC. In this part, we describe the "common" representation. This is the version of the internetwork graph that the MC holds, updating as new State Measurement Service reports arrive, and provides as the baseline for all transaction specific CSRs. In the next section, we discuss the transaction-specific graphical modifications to the common baseline necessary to discover and compose CServ Internetwork Service particular to the source and destinations host of the transaction requested through the CSR. The reliance upon the *common internetwork graph* framework, however, saves the MC from constructing a separate internetwork representation for each incoming CSR, thus reducing the controller workload and complexity.

The common internetwork graphical representation is predicated on the active gateway routers that form the CServ network. Except at the ends of the CServ Internetwork Service path for the source and destination subnets, all addresses in the path string are those of active gateway routers. These routers serve as the interfaces that allow for the use of a subnet as a transit subnet in a CServ Internetwork Service path. If we abstractly consider the subnet as a router in a "subnet-to-subnet" routing core, the active gateway routers within the subnet are the "ports" for that router. The CServ Intranetwork Services within the subnet, the details of which are unknown to the MC, connect the active gateway router "ports" internally and form a "switching fabric." These internal connections between active gateway routers are represented logically by the State Measurement Service reports from internal active gateway router to active gateway router. The physical external peering connections between active gateway routers in neighboring subnets, on the other hand, form the connections between routers in this theoretical notion of the "subnet-to-subnet" routing core. These external connections are also represented by the State Measurement Service reports, but they are physical peering connections rather than logical representations of a connection.

Removing this illustrative "router" abstraction, the vertices in the common internetwork graphical representation are the active gateway routers in the network. This means that there are $n_s \times n_g$ vertices in the graph according to our analytic notation. The edges between these vertices represent one of the following:

1. logical internal active gateway router to active gateway router connections formed by the CServ Intranetwork Services within the subnet;
2. or physical external peering connections between active gateway routers in neighboring subnets.

The active gateway routers that belong to the routing core of the same subnet, then, are connected as complete graphs (or full meshes) in the common internetwork graphical representation. Consistent with our discussion of the State Measurement Service reports, there are $n_g(n_g - 1)$ directed edges internal to each subnet within the representation and a total of $n_s \times n_g(n_g - 1)$ of these types of edges in the whole common internetwork graph. We can consider that the last reported CServ performance metrics for these logical internal connections are the set of edge weights in the graph. The connections between these local complete graphs of active gateway routers are the physical peering connections between active gateway router pairs in neighboring subnets. With the assumption that each active gateway router has p external active gateway router peers, there are a total of $p \times n_g \times n_s$ of these types of directed edges in the common internetwork graph. Again, the last reported CServ performance metrics for these physical external connections are the set of edge weights for these edges in the graph. In the end, we have a common internetwork graph with complete subgraph cliques of densely connected vertices, each representing a subnet in the CServ architecture. The connections between the cliques, however, are comparatively sparse.

We have specified that active gateway routers within a subnet should be connected as a full mesh in this common internetwork graphical representation. What if there is a pair of active gateway routers within a subnet that actually does not communicate or with a last reported State Measurement Service entry that has exceeded its time to live without a keep alive or update report? This should be a rare situation, but we would not want this pair of routers used within a CServ Internetwork Service path. Although we show the connection graphically, this lack of a connection can be encompassed in the edge weights applied to the edge between them. Specifically, the reliability of the edge can be set to zero, while the mean delay and delay variance (or, equivalently, the Cantelli transform value) set to infinity. With these modifications, the edge connecting this pair in the representation effectively disappears from the graph.

Note that only *active* gateway routers are represented in the common internetwork graph; non-active gateway routers are not vertices in this representation. Since non-active gateway routers cannot process

internetwork CServ datagrams and their associated control header format described in Chapter 3, they cannot be hops along the CServ Internetwork Service path. As the common internetwork graph is the baseline for the discovery and composition of CServ Internetwork Services using the CSDCA, we do not want any routers in this representation that cannot participate in the discovered output paths.

An example common internetwork graphical representation corresponding to the "natural" representation of Fig. 5-14 is shown in Fig. 5-15. Note that this is only one of many possible networks that would correspond to the same "natural" representation of Fig. 5-14. In Fig. 5-15, logical connections representing CServ Intranetwork Service internally between active gateway routers are shown as dotted edges. Alternatively, physical external peering connections between active gateway routers in neighboring subnets are illustrated as solid edges. This figure drives home the fact that a logical peering connection between adjacent subnets might actually subsume multiple physical peering connections between sets of active gateway routers in both subnets.

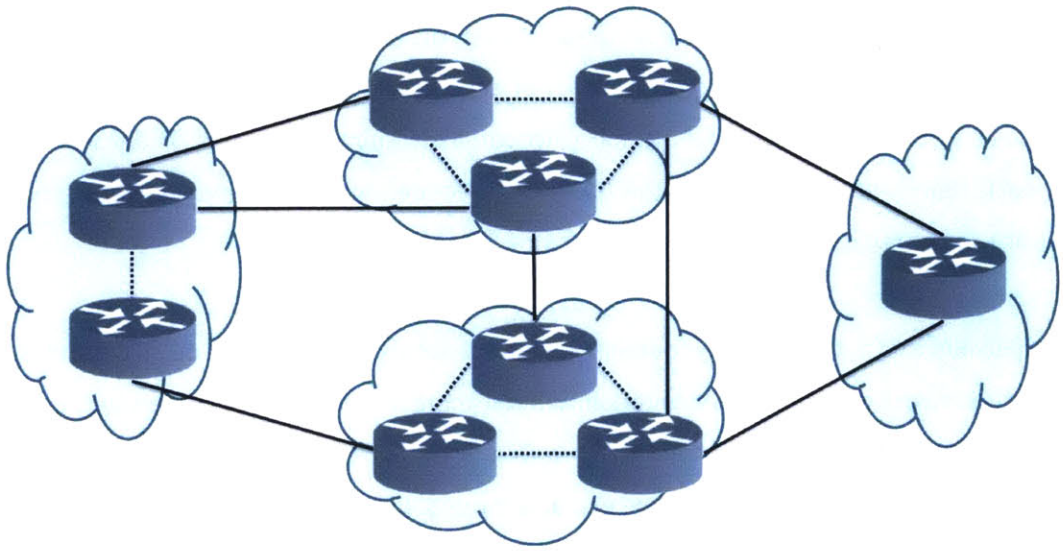


Fig. 5-15: This shows one possible common internetwork graphical representation maintained by the MC that corresponds to the "natural" representation of Fig. 5-14. The dotted edges are logical internal connections implemented by local CServ Intranetwork Services, while the solid edges are physical external peering connections between active gateway routers in neighboring subnets.

5.3.2.1 The Common Internetwork Cost-Adjacency Matrix

The description and illustration of the internetwork topology as a graph in Section 5.3.2 is didactic, but we need a convenient data structure to summarize this information for the purposes of the CSDCA. In this section, we describe the actual data structure maintained by the MC to encode the common internetwork graph. Specifically, we use a cost-adjacency matrix.

We begin by defining a general cost-adjacency matrix, and we then specialize it for the purposes of the common internetwork representation. A cost-adjacency matrix A is a square matrix, say $n \times n$. Each individual element of this square matrix, $a_{ij}\ \forall\ i,j \in \{1,2,\dots,n\}$ represents both the existence or non-existence of a logical communication connection from device i to device j and the weight assigned to that connection. Depending on the type of weight used and its numerical support, the non-existence of a logical communication connection might either be represented as 0 or ∞. For now, we say that $a_{ij} = \infty$ if there is no logical communication connection from device i to device j. Similarly, by convention, we let $a_{ii} = \infty$. Otherwise, if there is a logical communication connection from device i to device j, the element a_{ij} is set to the value of the weight of that communication link. Given a cost-adjacency matrix A, we can interpret this visually as a directed graph of n vertices with a directed edge from vertex i to vertex j whenever $a_{ij} \neq \infty$.

We apply this cost-adjacency matrix representation to our common internetwork graph. For notational simplicity, we let $n = n_s \times n_g$. We define a three-dimensional cost-adjacency matrix C with the dimensions $n \times n \times 4$ for the common internetwork graph. Each "plane" of the matrix, $c_{ijk}\ \forall\ i,j \in \{1,2,\dots,n\}$ and some $k \in \{1,2,3,4\}$, is itself independently a cost-adjacency matrix. The need for four "planes" comes from the set of different CServ performance metrics, namely the reliability, the mean delay, and the delay variance. Why then are there four planes? The fourth is maintained for convenient access to the Cantelli transform introduced in Section 5.2.4.2. We still need the original reported mean delay and delay variance CServ performance metrics; we cannot discard these reported values once the Cantelli transform values are calculated and updated. But we note that the maintenance of the mean delay values, the delay variance values, and the Cantelli transform values is redundant. Any one of them can be recovered from the other two. If necessary, the cost-adjacency matrix C can be reduced to $n \times n \times 3$ to save space at the MC by only keeping two of these three values. For now, however, we assume that all three values are held.

Let us formalize the contents of each plane of the cost-adjacency matrix C. In this description, we use the notation $i \rightarrow j$ to mean that router i has a directed edge to router j in the common internetwork graph. This edge may be either a logical connection resulting from CServ Intranetwork Service connecting two active gateway routers within a subnet, or this link may be a physical connection resulting from a peering connection between active gateway routers in adjacent subnets. Similarly, we use the notation $i \nrightarrow j$ to mean that router i does not have a directed edge to router j in the common internetwork graph. Throughout this description, we enumerate the active gateway routers in the common internetwork graph from the set $\{1,2,\ldots,n = n_s \times n_g\}$ for notational convenience, but we assume that the MC can index these routers and the entries of the cost-adjacency matrix by the actual active gateway router addresses in the CServ framework.

First, we consider the "reliability" plane of the cost-adjacency matrix C. We call this the $k = 1$ plane. If $i \rightarrow j \; \forall \; i \neq j \in \{1,2,\ldots,n\}$, c_{ij1} is set to the negative logarithm of the last reported reliability value from the State Measurement Service for the connection from active gateway router i to active gateway router j. We pre-calculate and maintain the transform introduced in Eq. (5.12) since this is the input used for CServ Internetwork Service path discovery (and the original reported reliability value can always be recovered using the inverse transform). So if the last reported reliability value from the State Measurement Service for the connection from active gateway router i to active gateway router j is ρ_{ij}, then $c_{ij1} = -\ln\{\rho_{ij}\}$. Otherwise, we set $c_{ij1} = \infty$ if $i \nrightarrow j$. Note that $c_{ij1} = \infty$ can be interpreted as a reported reliability of zero between active gateway router i and active gateway router j. By our convention, we also have that $c_{ii1} = \infty \; \forall \; i \in \{1,2,\ldots,n\}$.

Second, we consider the "mean delay" plane and the "delay variance" planes of the cost-adjacency matrix C. We call these the $k = 2$ and $k = 3$ planes, respectively. If $i \rightarrow j \; \forall \; i \neq j \in \{1,2,\ldots,n\}$, c_{ij2} is set to the mean delay value from the last State Measurement Service report for the connection from active gateway router i to active gateway router j. Likewise, if $i \rightarrow j \; \forall \; i \neq j \in \{1,2,\ldots,n\}$, c_{ij3} is set to the delay variance value from the last State Measurement Service report for the connection from active gateway router i to active gateway router j. Specifically, if the last reported mean delay and delay variance values from the State Measurement Service for the connection from active gateway router i to active gateway router j are μ_{ij} and σ_{ij}^2, respectively, then $c_{ij2} = \mu_{ij}$ and $c_{ij3} = \sigma_{ij}^2$. If $i \nrightarrow j$ and there

are no received reports regarding the delay statistics between these routers, then we set $c_{ij2} = c_{ij3} = \infty$. By convention, we also set $c_{ii2} = c_{ii3} = \infty \; \forall \; i \in \{1,2,\ldots,n\}$.

Lastly, we consider the redundant "Cantelli transform" plane that is pre-computed and maintained by the MC for convenience, which we call the $k = 4$ plane. Since the Cantelli transform values are used for CServ Internetwork Service path discovery, the immediate availability of these values, which are updated each time a new State Measurement Service report updates the mean delay and delay variance values, saves the MC from computing this plane after receiving the CSR (and becoming part of the pre-transmission control overhead delay of the CServ architecture). The mean delay and delay variance planes are both kept in the cost-adjacency matrix C, however, because these raw values may be needed to compute the delay of a CServ Internetwork Service path once it has been discovered by the Cantelli transform heuristic. If $i \rightarrow j \; \forall \; i \neq j \in \{1,2,\ldots,n\}$, c_{ij4} is set to the Cantelli transform value calculated from the last reported mean delay and delay variance values from the State Measurement Service for the connection from active gateway router i to active gateway router j. Specifically, if the last reported mean delay and delay variance values from the State Measurement Service for the connection from active gateway router i to active gateway router j are μ_{ij} and σ_{ij}^2, respectively, then $c_{ij4} = \mu_{ij} + 10\sqrt{\sigma_{ij}^2}$ according to Eq. (5.15). If $i \nrightarrow j$ and there are no received reports regarding the delay statistics between these routers, then we set $c_{ij4} = \infty$. By convention, we also have $c_{ii4} = \infty \; \forall \; i \in \{1,2,\ldots,n\}$.

If all fields in the cost-adjacency matrix are represented as 64-bit double-precision floating point numbers, what is the size requirement of the $n \times n \times 4$ common internetwork graph cost-adjacency matrix C? There are $4n^2$ entries in the entire matrix, and thus its persistent storage requires at least $256n^2$ bits of memory at the MC. If we consider a CServ internetwork example with $n_s = 100$ subnets and $n_g = 10$ active gateway routers per subnet (so $n = 1000$), then the common internetwork graph representation C needs 256 Mb (or 32 MB) of storage. The minimum necessary memory for the common internetwork representation grows asymptotically as $O(n^2)$ or $O({n_s}^2{n_g}^2)$.

A small example of this cost-adjacency matrix representation of a common internetwork graph is shown in Fig. 5-16.

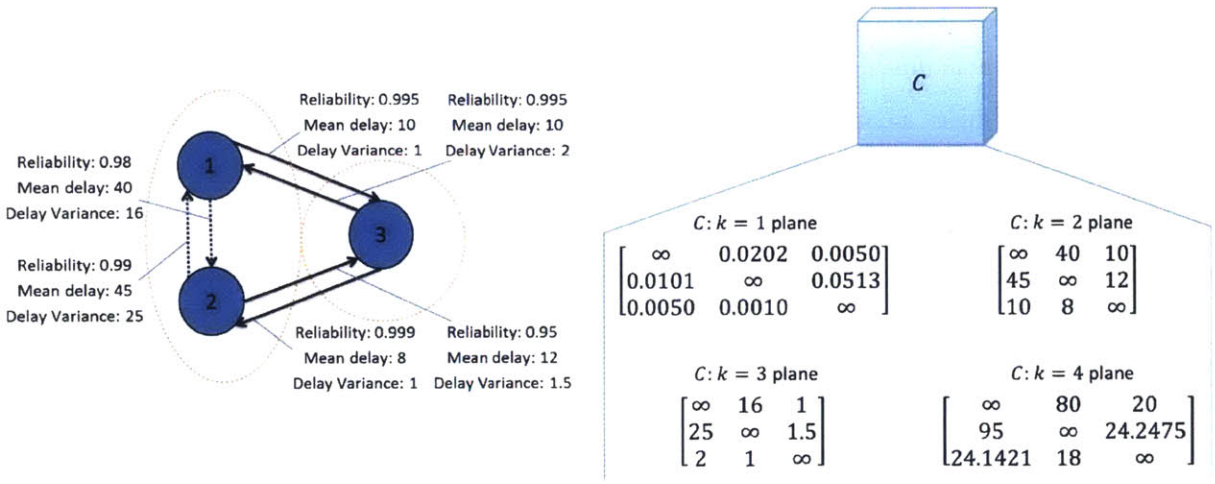


Fig. 5-16: The equivalent cost-adjacency matrix representation of a small toy common internetwork graph is shown. The cost-adjacency matrix C is maintained and updated by State Measurement Service reports at the MC for use in the CSDCA upon arrival of a CSR.

5.3.3 The Transaction-specific Internetwork Representation

The common internetwork representation is the structural foundation for the CSDCA maintained by the MC. CServ Internetwork Service paths consist primarily of a series of active gateway routers that serve to direct the CServ datagram from subnet to subnet on its journey from source subnet to destination subnet. However, the initial and ultimate hops are not yet part of this representation. Before the CServ datagram departs the source subnet via an active egress gateway router, the critical message must first find its way to the subnet's routing core. And once the CServ datagram reaches the ultimate subnet – the destination subnet – it needs to find its way to the correct active access router in the subnet's routing core that provides upstream access for the destination host of the transaction.

In some scenarios, this may be a simple default scenario. Consider the situation in which the source and destination hosts are both single-homed devices in their respective subnets. Under this condition, there is no concern as to which upstream access router the CServ datagram needs to make its way to – there is only one option in each subnet. The only modeling aspect necessary for the CSDCA that is not part of the common internetwork representation, then, is the learned and reported CServ performance metrics that connect the source host to the source subnet's egress gateway router, and the destination host to the destination subnet's ingress gateway router.

Alternatively, consider the scenario in which either or both the source and destination hosts are multihomed. The host may either have upstream access to the source subnet's routing core by way of multiple access routers in different access networks, or the host device may even be part of multiple subnets with upstream access to the routing core in each source subnet via a different upstream active access router. It is for this reason that CServ Internetwork Service paths contain the active access routers in the explicit source subnet to destination subnet path information – to specify the correct upstream active access router providing the desired level of performance in the case that there are multiple options. In either case, we need the necessary connection information and last reported CServ performance metrics from the State Measurement Service in order to complete the end-to-end picture that allows the CSDCA to quantify the end-to-end performance of the requested CServ transaction. This information is not part of the common internetwork representation.

Together, these scenarios motivate the need for a new CServ internetwork representation, one that is specific to each particular CServ transaction request. This representation includes the endpoint host devices of the requested transaction as well as their upstream connectivity to the routing core already captured by the common internetwork representation. We call this representation the *transaction-specific internetwork representation*. It is a simple transformation from the common internetwork representation baseline that serves as input to the CSDCA based on a particular CSR, and it depends on the source and destination hosts of the requested transaction as well as some State Measurement Service information stored and maintained by the MC outside of the common internetwork representation's cost-adjacency matrix.

Upon arrival of a CSR that identifies a particular unicast source and destination host pair, the transaction-specific internetwork representation accounts for the following outstanding questions:

1. What is the source subnet or subnets in the CServ internetwork?
2. What is the upstream access router connectivity of the source host? Is this device a member of multiple subnets? Is this device multihomed within the same source subnet?
3. What are the upstream CServ performance metrics for the access network or networks? What are the last reported CServ performance metrics between the upstream access router and the egress gateway routers of the source subnet or subnets?
4. What is the destination subnet or subnets in the CServ internetwork?
5. What is the upstream access router connectivity of the destination host? Is this device a member of multiple subnets? Is this device multihomed within the same destination subnet?
6. What are the downstream CServ performance metrics for the destination access network or networks? What are the last reported CServ performance metrics between the ingress gateway routers of the destination subnet or subnets and the upstream access router?

Although it initially seems like a lot, the answers to most of these questions are contained within the reports of the State Measurement Service to the MC from the constituent SCs, as described previously in Chapter 4. The additional information needed comes from the association-disassociation protocol that we have alluded to. In the following sections, we first describe the information that the MC needs access to in order to prepare transaction-specific internetwork representation, and then we present the

modifications necessary to the common internetwork representation's cost-adjacency matrix to create the necessary cost-adjacency matrix for the transaction-specific internetwork representation.

5.3.3.1 The Necessary Additional State Information

In this section, we discuss the additional state required to transform the common internetwork representation into a transaction-specific internetwork representation. This state information is the product of the State Measurement Service reports and the association-disassociation protocol, the outputs of which are already stored and maintained at the MC upon the arrival of a CSR. The information available when the CSR arrives is the information used in the creation of the transaction-specific internetwork representation, and thus the information used in the discovery of CServ Internetwork Service paths and the composition of CServ Internetwork Service.

The CSR specifies two fundamental addresses: the endpoints of the CServ transaction. We represent the specified source address from the CSR as a_s, where this denotes the hierarchical address (possibly an IPv6-style address) used to indicate the interface of the source device as discussed in Section 3.1.1. Furthermore, we represent the destination host address specified in the CSR as a_t.

The first step of the state information retrieval process to prepare the transaction-specific internetwork representation is to determine if either of these endpoint devices are multihomed. To do so, the MC needs a data structure that stores the information of the association-disassociation protocol. Namely, when a CServ-enabled device joins the network and intends to use the CServ architecture, it registers or associates itself with the local SC and, in turn, that association message is passed to the MC. If the device is multihomed, either connected to multiple access networks served by multiple upstream active access routers or connected to multiple upstream active access routers in disparate subnets, the device has multiple addresses due to the hierarchical nature of the addressing structure. The association message binds these addresses together as representing one physical device. The multihomed device does not necessarily care which interface the CServ datagram is transmitted on, as long as the service provided over that path meets the requirements of the CSR. Therefore, the MC must maintain a database which stores and, if necessary, updates these address bindings. We call this the *address-binding database*. In general, this is a relatively static database; devices that are part of the CServ network should change infrequently, and their address bindings are unlikely to change at all. If the CSR

indicates a source address a_s that is bound to a physical device with multiple addresses, the result of a lookup from this database keyed with source host address a_s should return the set of any other interface addresses bound to the same device, say A_{s-b} (which would be empty if the source host is single-homed). We represent this address binding lookup as $a_s \xrightarrow{addr} A_{s-b}$. Together, we represent the set of source addresses, which cannot be empty, as $A_s = \{a_s\} \cup A_{s-b}$. Regardless of the address $a_s \in A_s$ used to perform the address binding lookup, the resulting set A_s should be the same. This is an important consistency property required by the address-binding database. This lookup process can also be used to determine that the requesting device is actually associated with CServ; if not, the MC can reject the CSR and return a service denial message.

In the exact same way, the CSR indicates a destination address a_t. The source device for the CServ datagram exchange does not care which interface the destination host receives the datagram on, as long as the CServ Internetwork Service satisfies the performance demands of the CSR. Should the destination host device be multihomed, we want to know the set of hierarchical addresses that are bound to the same physical device. The address-binding database lookup procedure is repeated for the specified destination host address, $a_t \xrightarrow{addr} A_{t-b}$, returning the set of any other interface addresses bound to the same physical device through the association protocol, A_{t-b}. Together, the set of destination addresses, which cannot be empty, as $A_t = \{a_t\} \cup A_{t-b}$, and we require the same consistency property in the lookup process as described for the CServ transaction source device. Again, the address binding lookup process can be used to determine that the specified destination host device is actually associated with CServ; if there are no entries for the device in the database, the MC can reject the CSR as an impossible transaction and return a service denial message.

The second step of the state information retrieval process necessary to prepare the transaction-specific internetwork representation is to find the upstream active access router for each interface address in both sets, A_s and A_t, along with the necessary last reported CServ performance metrics from the State Measurement Service. We begin the discussion with the source host and its associated set of source host addresses, A_s. With the hierarchical addressing framework, we assume that there is a unique upstream active access router for each address in the set A_s (in other words, active access routers belong to one subnet routing core and are not themselves multihomed). In the case that there are multiple access networks with the same upstream access router and the source host device is

multihomed to these access networks, the addressing scheme should be designed in a way such that the upstream active access router can be treated as multiple devices since the reported CServ performance metrics of the access networks are likely different. With this assumption, we have that there is a unique upstream active access router with address a_{a-s} for each $a_s \in A_s$. The address a_{a-s} should be trivial to determine with the hierarchical addressing style discussed in Section 3.1.1. Since the interface address contains the address of the less-specific active access router providing connection to the routing core, a simple bitmask operation is capable of recovering a_{a-s} from a_s. We note that if hierarchical addressing is not employed, an additional database for storage and lookup of the mapping from a_s to the appropriate upstream access router address a_{a-s} would be required for this operation. Likewise, the subnet address a_{n-s} for the source host interface and upstream access router can be easily recovered from the source host interface address or upstream access router address using another bitmask operation thanks to the hierarchical addressing scheme where the subnet address is contained in the addresses of each.

With the upstream access router address a_{a-s}, we additionally require the State Measurement Service reported performance information for this router. Namely, we need the upstream access network performance metrics, denoted Ψ_a^{up}, and the reported CServ performance metrics from the access router to each upstream gateway router $a_g \in G_{a_{n-s}}$ in the subnet, denoted Ψ_{a-g}^{up}. In the previous statement, we used $G_{a_{n-s}}$ to denote the set of addresses of active gateway routers in the subnet a_{n-s}. Additionally, CServ performance metric sets Ψ contain the last reported reliability, mean delay, and delay variance values from the State Measurement Service. Ideally, we would like this last reported State Measurement Service performance metric information to be stored in a database that is keyed with the active access router address, a_{a-s}. In other words, a lookup operation in this database using a_{a-s} should return Ψ_a^{up} and $\Psi_{a-g}^{up}\ \forall\ a_g \in G_{a_{n-s}}$. To make this database and lookup more general, it could also return Ψ_a^{down} and $\Psi_{a-g}^{down}\ \forall\ a_g \in G_{a_{n-s}}$, representing the downstream access network performance metrics for the access router a_{a-s} and the last reported CServ performance metrics from each gateway router $a_g \in G_{a_{n-s}}$ in the subnet to access router a_{a-s}. As this is a source subnet, these retrieved downstream metrics are not necessary and could be filtered after the database lookup. We call this database the *subnet internal-performance database*, which is maintained by the MC and updated by reports from the State Measurement Service running in each network subnet. We denote a lookup of performance metrics using the subnet internal-performance database, and keyed by a_{a-s}, as

$a_{a-s} \xrightarrow{perf} \{\Psi_a^{up}, \Psi_a^{down}, \{\Psi_{a-g}^{up}, \Psi_{a-g}^{down}\} \forall\, a_g \in G_{a_{n-s}}\}$. Applying the filter for a source subnet, we effectively have $a_{a-s} \xrightarrow{perf} \{\Psi_a^{up}, \{\Psi_{a-g}^{up}\} \forall\, a_g \in G_{a_{n-s}}\}$.

The same process is required for each upstream active access router a_{a-t} for each $a_t \in A_t$, except that now we are interested in the downstream direction performance metrics as these are active access routers providing upstream access to the routing cores in destination subnets. As before with the hierarchical addressing framework, the addresses of the upstream access router, a_{a-t}, and destination subnet, a_{n-t}, for a particular destination host interface address a_t are trivial to determine with simple bitmask operations. A lookup operation on the *subnet internal-performance database* keyed with the upstream access router address a_{a-t} returns $\{\Psi_a^{up}, \Psi_a^{down}, \{\Psi_{a-g}^{up}, \Psi_{a-g}^{down}\} \forall\, a_g \in G_{a_{n-t}}\}$, or with the notation from before $a_{a-t} \xrightarrow{perf} \{\Psi_a^{up}, \Psi_a^{down}, \{\Psi_{a-g}^{up}, \Psi_{a-g}^{down}\} \forall\, a_g \in G_{a_{n-t}}\}$. In this case, we can filter the retrieved upstream direction performance metrics since a_{a-t} is in the destination subnet, and effectively we have $a_{a-t} \xrightarrow{perf} \{\Psi_a^{down}, \{\Psi_{a-g}^{down}\} \forall\, a_g \in G_{a_{n-t}}\}$.

We summarize the resulting additional state information gathered for a transaction-specific internetwork representation, garnered from lookups using the address-binding and subnet internal-performance databases. The input to this process is a pair of addresses, a_s and a_t, the source and destination host addresses specified in the CSR. The outputs of the process are the following sets of information, with the notation where $a_s \xrightarrow{trans} \{\cdot\}$ represents the transaction-specific state information retrieved using the address a_s from the CSR:

- $a_s \xrightarrow{trans} \{A_s, \{a_{a-s}, a_{n-s}, \Psi_a^{up}, \{\Psi_{a-g}^{up}\} \forall\, a_g \in G_{a_{n-s}}\} \forall\, a_s \in A_s\}$;
- $a_t \xrightarrow{trans} \{A_t, \{a_{a-t}, a_{n-t}, \Psi_a^{down}, \{\Psi_{a-g}^{down}\} \forall\, a_g \in G_{a_{n-t}}\} \forall\, a_t \in A_t\}$.

The speed of these database lookups is of utmost importance since the delay of this state information retrieval is part of the control overhead associated with the pre-transmission process that begins a CServ datagram exchange. The transaction-specific internetwork representation cannot be prepared and maintained prior to the arrival of a CSR since this would require the maintenance and storage of a

separate representation for each possible source and destination host pair in the entire CServ network, most of which would be redundant information.

The design of the databases that are used to store and maintain this specific state information from the State Measurement Service reports and association-disassociation protocol is not covered in this dissertation. These structures should be considered in more detail in the future in order to minimize the control delay associated with the preparation of the transaction-specific representation. In general, we want the lookup and preparation process to be fast with predictable delay. Updates to these data structures do not need to be as quick as the retrieval of information, but they should not be particularly slow and cumbersome since the reports of the State Measurement Service may indicate fluctuating CServ performance metric state during periods of subnet transience. Furthermore, these databases should be accessible to multiple processes simultaneously, since each transaction-specific instance of the CSDCA (and creation of a transaction-specific internetwork representation) needs access to the information. Even without an optimized storage and retrieval framework, we expect the discovery phase to dominate the running time required by the control overhead and CSDCA prior to the transmission of a CServ datagram.

We suggest the consideration of multimaps (or multihash tables) [83] for the address-binding and subnet internal-performance databases. Multimaps are generalizations of hash tables or hash maps, which use hash functions to map single key-value pairs, for the mapping of one key to many values. Since we have a fairly static set of well-known keys in both cases (either the hosts associated with CServ or the active access routers in the network), the choice of perfect hash functions that avoid collisions should be feasible. This allows for very desirable lookup performance properties. Specifically, perfect hashing allows for constant time lookups in the worst case while the required storage space scales only linearly with the number of entries.

5.3.3.2 The Transaction-specific Cost-Adjacency Matrix

With the necessary additional state information ready to go, we are ready to transform the common internetwork representation into the transaction-specific internetwork representation. We first introduce the concept graphically, and then we illustrate the creation of the transaction-specific cost-adjacency matrix from the common cost-adjacency matrix. This transaction-specific cost-adjacency

matrix is the primary input to the CSDCA, along with the requirements from the CSR that initiates the algorithm.

Considering the graphical interpretation of the common internetwork representation, developed in Section 5.3.2, the only subnets in the graph that are modified to generate the transaction-specific internetwork representation are the source and destination subnets (or, more generally, the sets of source and destination subnets in case the source or destination hosts are multihomed). Otherwise, the rest of the common internetwork representation remains unchanged.

At a minimum, the representations of two subnets are modified when the transaction-specific version is created for the CSR of an internetwork CServ transaction: the source subnet and the destination subnet. This is the case where both the source host and destination host specified in the CSR are single-homed. We begin with this discussion, and then we generalize the discussion for the case where one or both of the endpoints hosts are multihomed. As we discuss this case, consider the graphical example of Fig. 5-17; the flow of the illustration shows the transformation of the common internetwork graph to a transaction-specific internetwork graph.

In the first step, additional vertices are added to the graph. One vertex is added to represent the source host, and another to represent the destination host. There is never more than one vertex for each endpoint host, as the CSDCA functions to discover an internetwork path between these two explicit endpoints. If we consider the domains of the subnets, the source host vertex is part of the domain of the one source subnet in the single-homed case, and the destination host vertex is a member of the domain of the one destination subnet. The next vertices appended to the graph represent the upstream access routers that provide connection to the routing cores in the respective source and destination subnets. One vertex is added for the upstream access router for the source host since the source host is single-homed; this vertex can also be considered part of the domain of the one source subnet. And, likewise, one vertex is added for the upstream access router for the destination host since the destination host is also single-homed. Again, we can consider this vertex to be part of the domain of the one destination subnet.

Then in the second step, the new vertices are connected to the baseline common internetwork graph. Looking at the source subnet, the source host vertex is connected with a directed edge to its upstream

access router. This directed edge represents the access network that connects the source host device to its upstream access router. The CServ performance metrics learned for the upstream connection to this access router are applied to this edge. The upstream access router in the source subnet is then connected, with a directed edge, to all active gateway routers that subnet (or to all other vertices in that subnet except for the vertex that represents the source host). These directed edges represent the CServ Intranetwork Services that connect the access router to each active gateway router in the subnet routing core; the State Measurement Service reported CServ performance metrics for these connections are applied to the new edges. Edges are added to connect the new vertices in the destination subnet in nearly the same way; the different is in the orientation of the edges. Directed edges are added from each active gateway router vertex in the destination subnet to the new vertex that represents the upstream access router. These directed edges represent the CServ Intranetwork Services that connect the gateway routers to the access router in the destination subnet routing core, and the learned CServ performance metrics for these services are affixed to the edges. Finally, a directed edge from the new vertex that represents the upstream access router in the destination subnet is connected to the destination host vertex. This directed edge represents the downstream direction for communication on the access network that connects the upstream access router to the destination host device. In the single-homed case, this completes the graph modification process that converts the common internetwork graph to a transaction-specific internetwork graph ready for use with the CSDCA.

Let us now consider a multihomed example to clarify the changes that must be made to accommodate connections to multiple upstream access routers in the same subnet or in disparate subnets. As previously mentioned, there are always exactly two vertices total for the source and destination host devices. Even in the multihomed case, we do not replicate the source or destination vertex. The multiple interfaces with different addresses are captured by the use of more than one edge leaving (or entering) the source (or destination) host vertex. As we discuss this process, refer to the graphical example in Fig. 5-18 that illustrates the case of a multihomed source host with two upstream access routers in two distinct subnets. For each source and destination host interface that represents an upstream access network connection to a distinct access router (or possibly a distinct access network connected to the same upstream access router as another access network interface), a vertex is added to the common internetwork graph. For example, if the source host device is connected to two upstream access routers in the same subnet, two vertices are added to the graph to represent these active access routers. Alternatively, if the source host device is connected to two upstream access routers in distinct subnets

(as shown in Fig. 5-18), two vertices are added to the graph to represent these active access routers. The difference is in how they are connected to the common internetwork graph, as we shall soon see. The same process applies for adding vertices to the graph if the destination host is multihomed to multiple upstream access routers. Once the new vertices have been introduced to represent the source and destination hosts and the set of all upstream access routers, we connect these new vertices to the baseline common internetwork graph as follows. First, directed edges are connected from the source host vertex to each of its upstream access routers. These directed edges represent the upstream connections from the source host to its upstream access routers via the appropriate access network, and the appropriate State Measurement Service reported upstream access network performance metrics are associated with these edges. Then, as in the single-homed case, directed edges are added from each upstream access router to the upstream active gateway routers in the respective source subnets. If all upstream access routers are in the same source subnet, then the directed edges are connected to the same sets of source subnet active gateway router vertices. If the upstream access routers are in different subnets (as in Fig. 5-18), then the directed edges are connected to disparate sets of active gateway router vertices in the different source subnets. These connections all represent the appropriate CServ Intranetwork Services, and the learned CServ performance metrics for these services are associated with the edges.

In general, host devices that use the CServ architecture for critical messaging are unlikely to be multihomed to more than a few upstream access routers via a handful of different access network interfaces. The ability to connect an endpoint host to multiple access networks is an important consideration of the CServ user since it allows for improved end-to-end transaction reliability and survivability against the unexpected, such as network "Black Swan" events that might lead to service disruptions or malicious denial on a particular interface or communication modality. That being said, we are not talking about a large explosion in the graph size through the transformation from the common internetwork graph to the transaction-specific internetwork graph. Most endpoint host devices are expected to be limited to something on the order of three network interfaces. Let us assume that there are n_g active gateway routers in each subnet, and that the source and destination hosts of a CServ transaction are each multihomed to Δ upstream access routers. Transformation from a common internetwork graph to a transaction-specific internetwork graph in this case necessitates the addition of a total of $2(\Delta + 1)$ vertices to the common internetwork graph and $2\Delta(n_g + 1)$ directed edges. If $\Delta = 3$ and $n_g = 10$, this is only eight additional vertices and 66 directed edges.

With this graphical intuition regarding the transaction-specific internetwork representation and the transformation from the common internetwork representation, we consider the cost-adjacency matrix representation of the transaction-specific internetwork topology and connection information. The cost-adjacency matrix form of the common internetwork representation was previously developed in Section 5.3.2.1 as the data structure used to encode the graph. Throughout this section, we consider the example of Fig. 5-19 and Fig. 5-20, which illustrate the transformation of a very simple common internetwork cost-adjacency matrix (specifically, only the $k = 2$ plane of mean delay performance metrics from the generally three-dimensional matrix) to a transaction-specific internetwork cost-adjacency matrix. These figures simultaneously show the equivalent graphical representations of the toy internetwork topology.

We consider the process in two steps. The first step, shown in Fig. 5-19, involves the copying of the original common internetwork cost-adjacency matrix C into a new, larger matrix for the transaction-specific representation, which we denote C_{st}. The baseline common topology serves as the framework for the transaction-specific internetwork cost-adjacency matrix. Since each row (or column) in a plane of the matrix represents a vertex in the equivalent internetwork graph, the square dimensions of the transaction-specific internetwork cost-adjacency matrix are larger than those of the common internetwork cost-adjacency matrix to account for the newly introduced vertices. Recall from Section 5.3.2.1 that the square dimensions of a plane of the common internetwork cost-adjacency matrix were $n \times n$, where $n = n_s \times n_g$. The four planes of the transaction-specific cost-adjacency matrix remain square, but we need to extend the dimensions of each side. Specifically, we know that we are introducing exactly two vertices that represent the source and destination host devices. Additionally, using the notation of Section 5.3.3.1, we are introducing $|A_s| \triangleq n_{a_s}$ vertices for the upstream access routers for the source host and $|A_t| \triangleq n_{a_t}$ vertices for the upstream access routers for the destination host. We note that the cardinalities of both of these sets may be as small as one in the case of single-homed source and destination host devices. The addresses of these access routers in the transaction-specific cost-adjacency matrix representation are part of the lookup results discussed in Section 5.3.3.1. Thus the dimensions of a plane of C_{st} are $\left(n + (2 + n_{a_s} + n_{a_t})\right) \times \left(n + (2 + n_{a_s} + n_{a_t})\right)$. For simplicity, we define $n_{st} \triangleq n + (2 + n_{a_s} + n_{a_t}) = \left(n_s \times n_g\right) + (2 + n_{a_s} + n_{a_t})$. The dimensions of C_{st} are $n_{st} \times n_{st} \times 4$, where we next discuss that the four square planes of the three-dimensional matrix are used in the same way as presented previously for the common internetwork cost-adjacency matrix. Having allocated working space in memory for the transaction-specific CSDCA process and the storage of

this transaction-specific internetwork cost-adjacency matrix, the $n \times n$ per plane entries of the original common internetwork cost-adjacency matrix C are copied into the top left entries of each plane of C_{st}. All other entries of C_{st} in the newly introduced rows and columns are initialized to infinite values.

The second step of the process involves introducing the directed connections between the new vertices and the vertices of the common internetwork graph, as illustrated in Fig. 5-20. As the new entries in C_{st} are all initialized to infinite values, this is done by setting the appropriate new entries in the expanded cost-adjacency matrix to finite values based on the results of the lookups of previous State Measurement Service reports. In Section 5.3.3.1, we discussed the additional state information required to create the transaction-specific cost-adjacency matrix, which was summarized at the end of the section. The CServ performance metric information in these sets is used to populate the new rows and columns of C_{st}. As with the common internetwork cost-adjacency matrix, the $k = 1$ plane is used for the reliability transform values covered in Section 5.2.4.1, the $k = 2$ plane holds the mean delay values, the $k = 3$ plane stores the delay variance values, and the $k = 4$ plane is populated with the Cantelli transform values as discussed in Section 5.2.4.2. These fields should be completed such that the source host vertex has directed connections to its upstream access routers in the appropriate source subnet or subnets, and furthermore the upstream access routers in the source subnet or subnets should have directed connections to all gateway routers in their respective subnet. Likewise, the correct matrix entries should be populated such that there are directed connections from all active gateway routers in the destination subnet or subnets to the appropriate access router or routers in their respective subnets, and then there should be directed connections from each introduced access router in the destination subnet or subnets to the destination host vertex. This process is depicted in Fig. 5-20 using the notation of Section 5.3.3.1 for clarity.

Once the new rows and columns of C_{st} have been populated according to the information retrieved from the address-binding and subnet internal-performance database lookups, the transaction-specific cost-adjacency matrix is ready to be used as an input to the CSDCA. With this data structure as the representation of the entire CServ internetwork topology and its learned and reported CServ performance metrics, we finally present the details of the CServ architecture's control plane workhorse in the next section. This is the algorithm that ultimately determines if the network can support the requested level of service for a particular critical message exchange between CServ users.

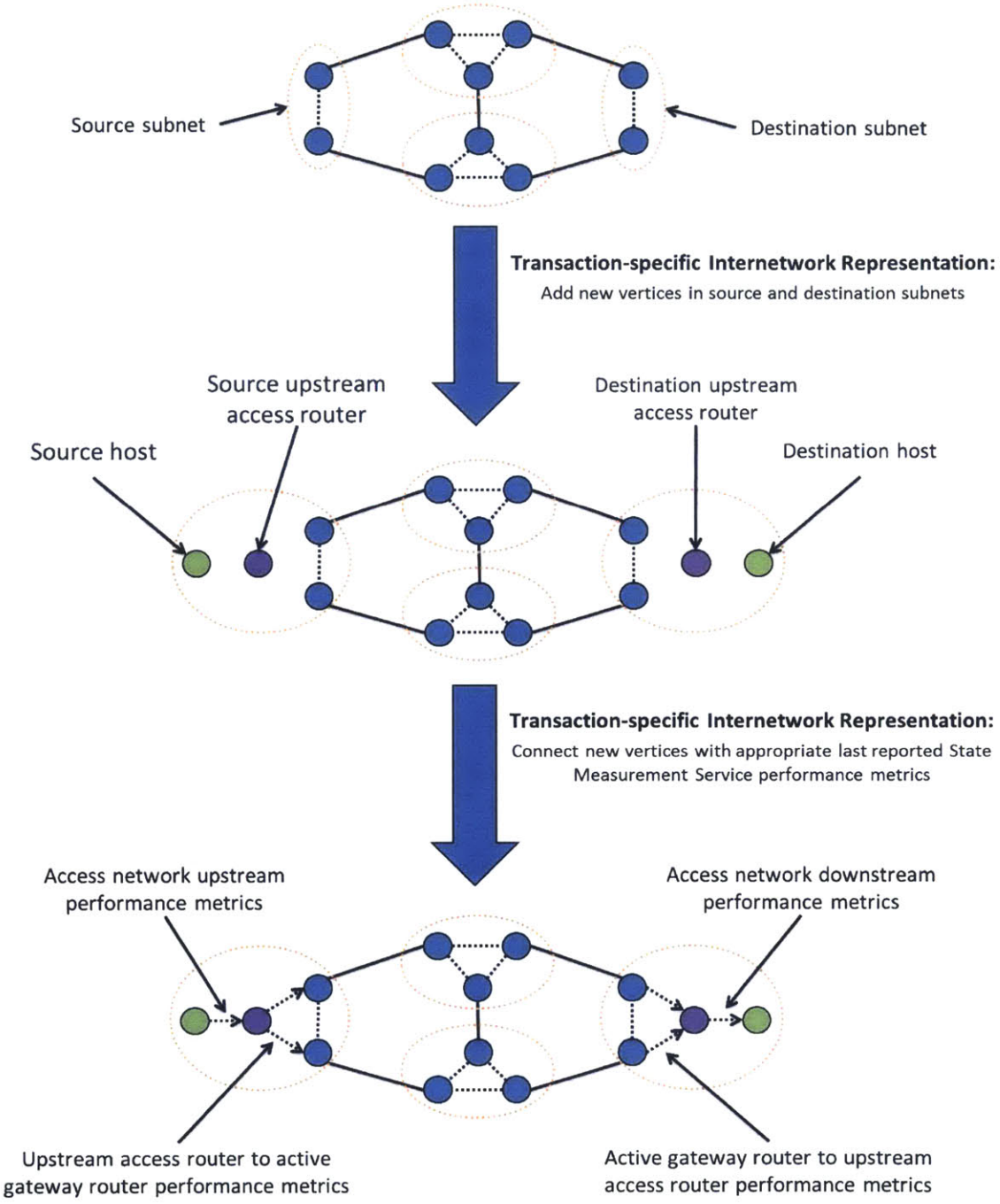


Fig. 5-17: In two steps, this shows the transformation of a simple common internetwork graph to a transaction-specific internetwork graph in response to a CSR when both the source and destination hosts are single-homed devices.

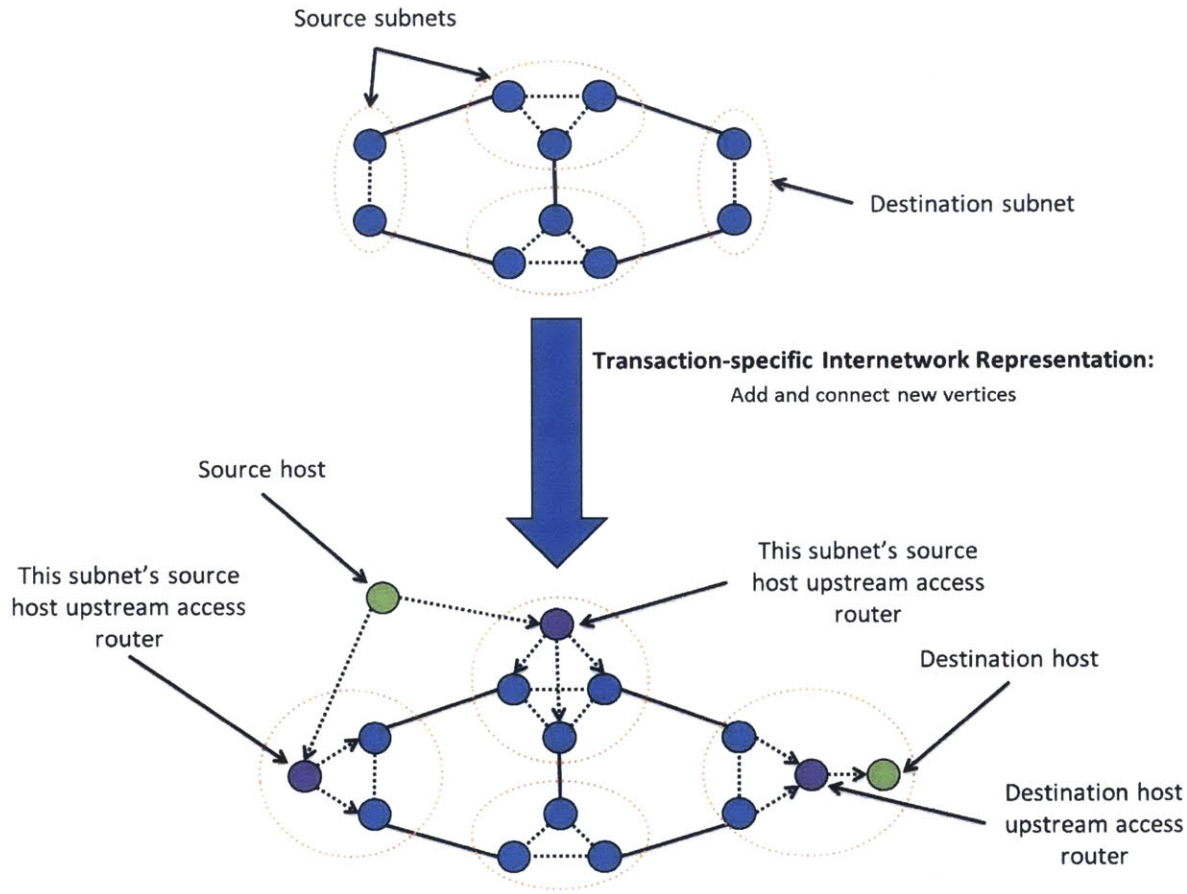


Fig. 5-18: For the same simple common internetwork graph as in Fig. 5-17, this illustrates (in one aggregate step) the transformation to a transaction-specific internetwork graph in response to a CSR when the source host is multihomed to two upstream access routers in distinct source subnets. The destination host is still a single-homed device in this example.

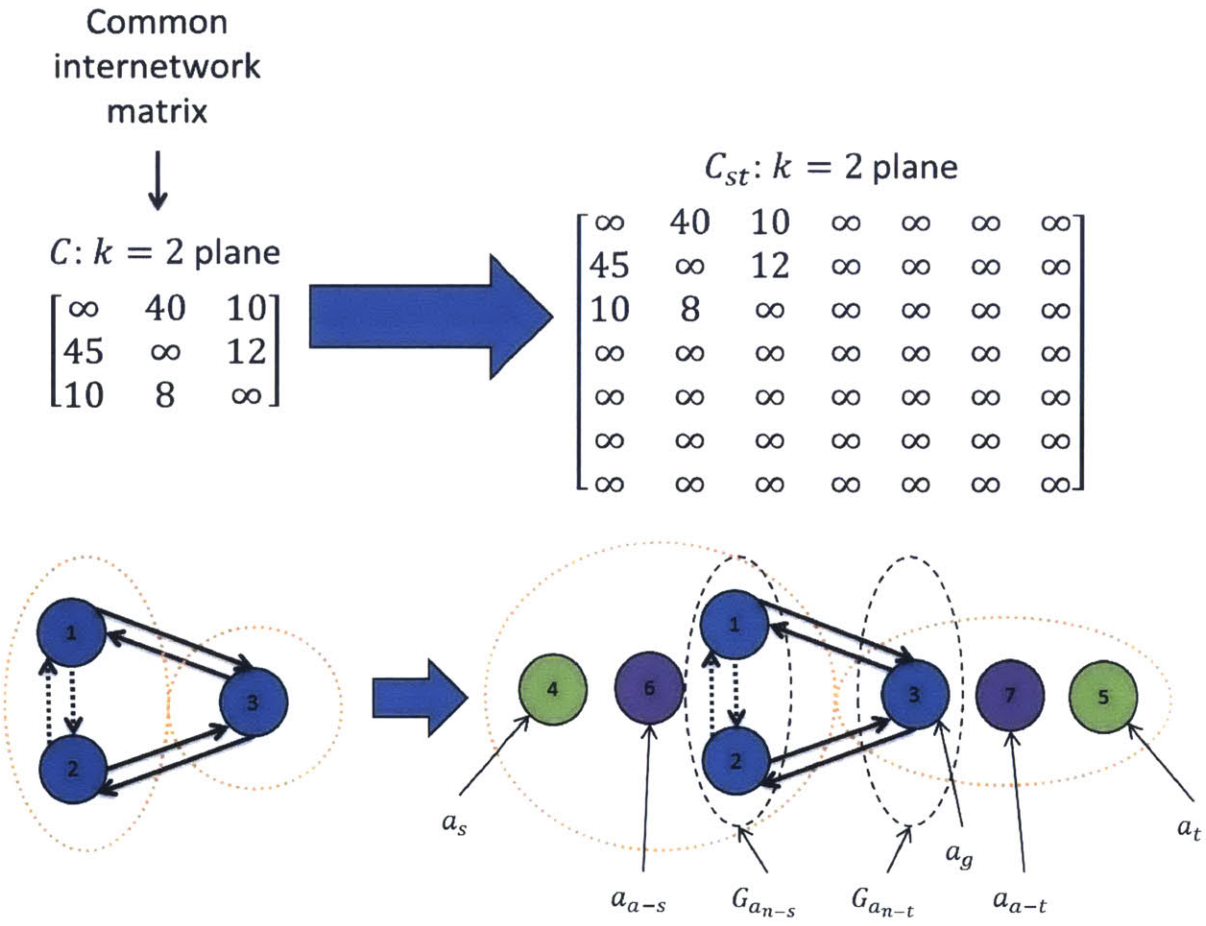


Fig. 5-19: For a very simple common internetwork cost-adjacency matrix C, this shows the first step in the transformation to a transaction-specific internetwork cost-adjacency matrix C_{st} by introducing additional rows and columns for the new graph vertices.

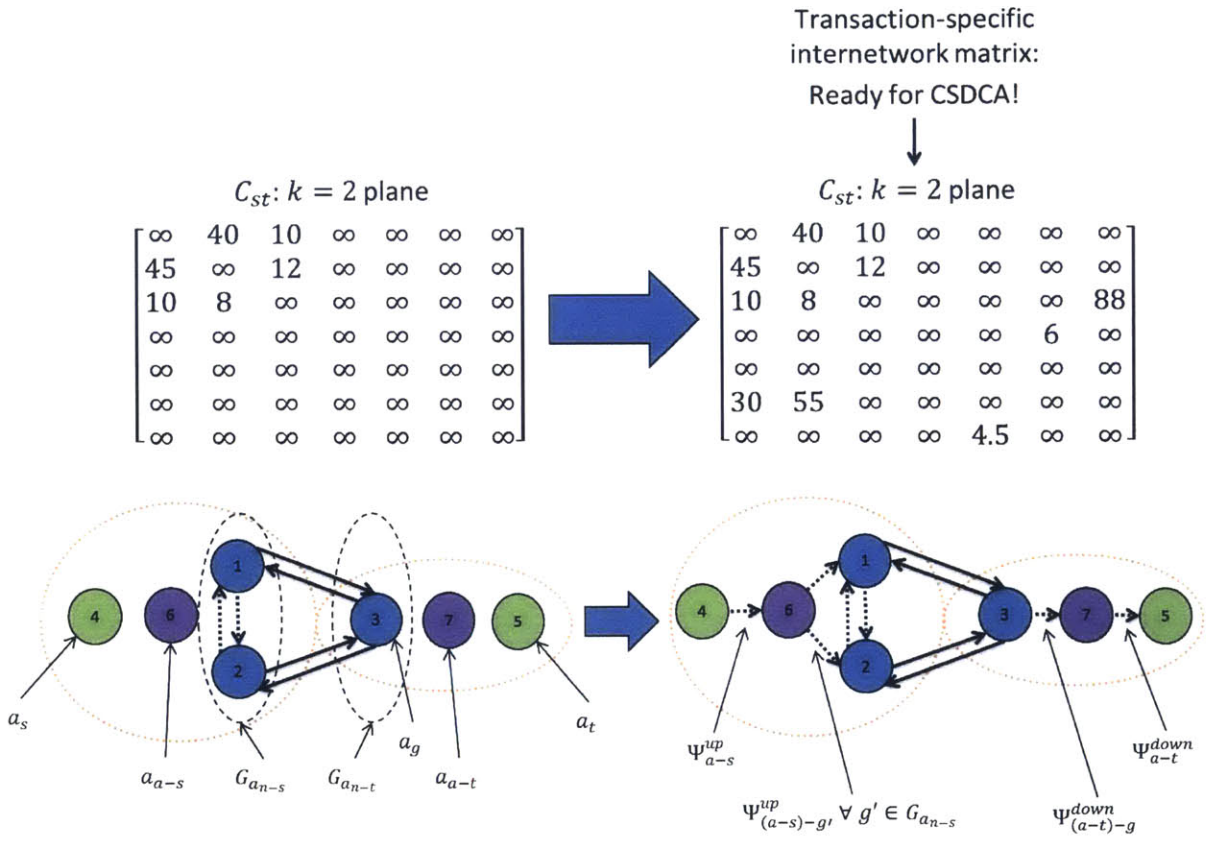


Fig. 5-20: Continuing from Fig. 5-19, this shows step two in the creation of the transaction-specific internetwork cost-adjacency matrix – introducing the last reported CServ performance metrics associated with the additional directed graph edges as finite-valued entries in the new rows and columns of the cost-adjacency matrix.

5.4 The Critical Service Discovery and Composition Algorithm

The CSDCA is the brain of the logically-centralized MC in the CServ architecture. Prior to the transmission of an internetwork CServ datagram, the algorithm performs a per-transaction computation to find a CServ solution which conforms to the demands of the requested service level for the transaction, if possible. This computation involves two distinct phases, as indicated by the algorithm's name. The first phase *discovers* individual subnet-disjoint CServ Internetwork Service paths using the transaction-specific internetwork cost-adjacency matrix (the data structure encoding the graphical representation of the internetwork topology and State Measurement Report performance metrics). And the second phase, if necessary, *composes* the CServ solution, or the CServ Internetwork Service solution, by looking for appropriate diversity combinations of the discovered paths that jointly meet the requirements of the CServ user application as indicated via the CSR using the CServ API.

In this section, we present the algorithm in several steps. In the first part, we use a convenient visualization of the algorithm's operation to help further describe the details and complexities of the procedure. In the second part, we discuss the particulars of a few supporting considerations needed to fully understand the CSDCA; namely, we look at how the delay requirement as indicated by the CServ user is adjusted to allow for the control overhead in the CSDCA execution, how transit subnets from previously discovered CServ Internetwork Service paths are removed from the transaction-specific internetwork cost-adjacency matrix to force the discovery of subnet-disjoint paths, and how individual discovered paths are considered for inclusion in the composed solution. Finally, we put this all together and present the pseudocode describing the operation of both phases of the CSDCA.

5.4.1 Visualizing the Algorithm Phases

In this section, we visualize the operation of the CSDCA. This illustration should help frame the discussion and details to follow. The subsequently discussed visualization is given in Fig. 5-21.

The control overhead for a CServ datagram transaction begins with the user's transmission of a CSR on the robust off-band control channel via the CServ API. As previously discussed, this control message contains, in part:

- the source host address;
- the destination host address;
- the primary CServ performance metric;
- the minimum service reliability value;
- the maximum service delay value;
- the minimum path diversity value;
- and a timestamp indicating the time of CSR generation and transmission.

There are additional fields in the CSR that are not required for the discussion here. This CSR control message traverses the off-band control channel, transiting the local logically-centralized subnet controller (which can intercept the message if the specified destination host address is local to the subnet and, thus, the transaction does not require internetwork service) on its way to the logically-centralized MC. Since the MC is a logically-centralized architecture component (where the logical centralization requires consistency among its stored internetwork state information), the destination controller entity depends on the location of the source host that initiates the CServ transaction. In general, we want minimal delay between the source host and the MC point-of-presence, since this delay is part of the control delay incurred prior to the transmission of the CServ datagram with the critical message.

Upon receipt of the CSR, the MC creates a transaction-specific process for the CSDCA. As previously discussed in Section 5.3, this involves copying the baseline common internetwork cost-adjacency matrix into a new, larger data structure that includes additional topology and state information for the transaction-specific internetwork representation. This process was considered in detail in Section 5.3.3. Once the transaction-specific internetwork representation is prepared in terms of a cost-adjacency matrix, the CSDCA is ready to kick-off.

We can visualize the operation of the CSDCA on a two-axis plot, as shown in Fig. 5-21. The two axes, which form the service plane, represent the CServ performance metrics of interest for the CServ one-shot transaction from source host to destination host: reliability and delay. In the CSR, the source host indicates the primary metric of interest, which we assign to the x-axis of the plane. The secondary matrix is assigned then to the y-axis. In our example, the CSR has indicated that delay is the primary CServ performance metric of interest to the CServ user.

In the CSR, the user indicates the performance requirements for the CServ transaction in terms of the two CServ performance metrics. Namely, the source host application specifies the maximum tolerable delay for the end-to-end transaction and the minimum required level of transmission reliability. Together, these performance metrics allow the CServ user to specify and control the transaction risk; risk management is a fundamental concept to critical messaging over unreliable networks and substrates. The maximum tolerable delay and minimum tolerable reliability can be visualized as user requirement demarcation points on the appropriate axes of the plane (the x-axis and y-axis, respectively, in our example).

The minimum tolerable reliability user requirement forms a permissible reliability range. Namely, if the CServ user specifies that $0 \leq \rho_{min} \leq 1$ is the minimum tolerable reliability for the end-to-end transmission of the internetwork CServ datagram, then the permissible range for the CServ Internetwork Service reliability, ρ_{serv}, is $\rho_{min} \leq \rho_{serv} \leq 1$. Note that the CServ user could specify that $\rho_{min} = 0$, imposing no requirements on the reliability performance of the CServ Internetwork Service. Alternatively, the CServ user could specify $\rho_{min} = 1$, effectively eliminating the permissible reliability range. In this case, it is highly unlikely that the CSDCA would actually discover any CServ Internetwork Service paths that meet the user requirement (and the composition of service from individual CServ Internetwork Service paths with less than perfect reliability would likewise fail).

The maximum tolerable delay value itself does not form the permissible delay range for the CServ Internetwork Service. The user requirement must first be adjusted to account for the delay overhead associated with the control message delay and the CSDCA execution delay. This adjustment is considered in more detail in Section 5.4.2.1. For now, we simply assume that the adjustment can be easily calculated. Once the maximum tolerable delay value has been appropriately reduced to account for the control and algorithm delay, the adjusted maximum tolerable delay value serves as a new demarcation point on the appropriate axis of the service plane, and this value forms the permissible delay range. Specifically, if the adjusted end-to-end transmission delay requirement is $0 \leq d'_{max} \leq \infty$, then the permissible range for the CServ Internetwork Service reliability, d_{serv}, is $0 \leq d_{serv} \leq d'_{max}$. The larger the value of d'_{max}, the more likely that the CSDCA successfully discovers a CServ Internetwork Service solution. The CServ source application can effectively impose no restriction on delay by setting the specified maximum tolerable delay as some excessively large value. Alternatively, as d'_{max} diminishes and approaches zero, the physical constraints of the internetwork topology and performance

may preclude the ability of the algorithm to create a service solution. Whereas diversity routing over subnet-disjoint paths can improve the end-to-end reliability of a composed service solution, diversity routing cannot reduce the delay of the composed service.

Jointly, the two permissible CServ performance metric ranges form the CServ Internetwork Service solution space, or simply CServ solution space. The parameter space simultaneously carved out by the $\rho_{min} \leq \rho_{serv} \leq 1$ and $0 \leq d_{serv} \leq d'_{max}$ constraints on both axes of the service plane show where the CSDCA aims to either discover or compose the CServ Internetwork Service solution. In the visualization of Fig. 5-21, this area is shaded in red. With this framework, we can discuss the two phase operation of the CSDCA.

In the first phase of the CSDCA, the *discovery phase*, the algorithm uses successive iterations of Dijkstra's Algorithm as a subroutine to find individual CServ Internetwork Service paths. The edge weights, or equivalently the plane of the transaction-specific cost-adjacency matrix as described in Section 5.3, used in Dijkstra's Algorithm for the discovery of paths from source host to destination host depends on the CServ performance metric specified as the primary metric of interest in the CSR. If reliability is specified as the primary CServ performance metric, then the reliability transform values are used as the nonnegative, additive edge weights. Otherwise if delay is specified as the primary metric, then the Cantelli transform values are used. In our visualized example, delay has been specified as the primary CServ performance metric, and thus the Cantelli transform values are leveraged directly by Dijkstra's Algorithm to discover CServ Internetwork Service paths.

As each path is discovered by Dijkstra's Algorithm, the transit subnets of that path are deleted from the transaction-specific internetwork representation before the subsequent path discovery iteration of the algorithm. In this way, each path discovered in this phase is guaranteed to be subnet-disjoint, as defined in Section 5.2.2. The source and destination subnets of a particular path are not removed from the internetwork representation for subsequent Dijkstra's Algorithm iterations in case either of the endpoint hosts is single-homed. The algorithmic process for the deletion of transit subnets is further described in Section 5.4.2.2.

The number of paths discovered in the discovery phase of the CSDCA is a function of two parameters, one under the direct control of the MC and the other not. The CSDCA has a parameter common to all

transaction-specific instances of the algorithm, the maximum number of iterations of the discovery phase. We denote this parameter as $p_{max} \in \mathbb{Z}^+$. This can be interpreted as the maximum number of subnet-disjoint CServ Internetwork Service paths that the CSDCA is capable of searching for in the discovery phase of the algorithm before it is forced to move on to the next phase. The p_{max} parameter is necessary to limit the execution time of the CSDCA, which is vital to ensure that the pre-transmission transaction control overhead does not consume the entire time allotted for the transmission of the actual CServ datagram. Although p_{max} must be greater than or equal to one, in practice this parameter should take a greater value (say on the order of at least five). The choice of this value is dependent upon the specific hardware and software implementation of the CSDCA and the complexity of the internetwork topology; a CSDCA implementation that executes very quickly with respect to the CServ Internetwork Service path delays can afford to discover more paths in the discovery phase. The other parameter that determines the number of paths that are discovered in the discovery phase is transaction-dependent and related to the physical connectivity of the internetwork topology. As paths are discovered and transit subnets deleted to force the discovery of subnet-disjoint paths, this may eventually disconnect the source and destination hosts. Once the hosts are discovered disconnected by an iteration of Dijkstra's Algorithm, the discovery phase terminates. We can denote this transaction-dependent number of subnet-disjoint paths available as $p_{poss} \in \mathbb{Z}^+$, where we assume that the original transaction-dependent internetwork graph is connected and thus there is at least one path available. The transaction-dependent value of p_{poss} is not necessarily known beforehand. Therefore, the discovery phase of the CSDCA always attempts to find p_{max} subnet-disjoint CServ Internetwork Service paths. But depending on the internetwork topology, the actual number of paths found during this phase, $p_{discover}$, is given as $p_{discover} = \min\{p_{poss}, p_{max}\}$.

When a CServ Internetwork Service path is found during the discovery phase of the CSDCA, the performance of that path is characterized in terms of both its reliability and its delay (regardless of which CServ performance metric is deemed the primary metric). We denote each discovered path as $p_i, i \in \{1,2,\ldots,p_{discover}\}$. Each of these CServ Internetwork Service paths can then be represented by a pair of CServ performance metrics, $(p_i.reliability, p_i.delay)$, the reliability and delay of the path respectively. With these performance pairs, each path can be represented as a point in the service plane, as illustrated for the four discovered paths in Fig. 5-21. If the edge weights used for path discovery are the reliability transforms, then the reliability of the path can be recovered directly from the Dijkstra's Algorithm-computed path distance using the inverse transform introduced in Section

5.2.4.1. However, if the edge weights used for path discovery are the Cantelli transforms, the delay of the path cannot be retrieved directly from the output of Dijkstra's Algorithm. In either case, the reliability and delay performance of a CServ Internetwork Service path can be computed from the path p_i and the information contained in the transaction-specific internetwork cost-adjacency matrix C_{st} based on the definitions from Section 5.2.3. While the definition of the reliability of a path is straightforward, the definition of the delay of the CServ Internetwork Service path is based on the one-sided variant of the Chebyshev Inequality, derived from the Cantelli Inequality, and the sum of the mean delays and delay variances for the path under consideration.

Once the $p_{discover}$ subnet-disjoint CServ Internetwork Service paths have been found and characterized, the CSDCA proceeds to the second phase: the *composition phase*. The objective of this phase of the algorithm is to search among the discovered paths to find a CServ Internetwork Service that falls in the CServ solution space previously discussed. This could be a trivial task if one of the CServ Internetwork Service paths alone is characterized by a point in the interior of that space (and the CServ source host application does not require a minimum level of diversity greater than one in the solution). Or this could involve testing different combinations of the viable candidate paths to find one that jointly composes a diversity-routed solution in the CServ Internetwork Service solution space.

As a first step to this phase, discovered CServ Internetwork Service paths with delay greater than the adjusted end-to-end transmission delay requirement, d'_{max}, can be removed from the set of viable paths. The delay of a diversity-routed CServ Internetwork Service is defined as the maximum delay of the CServ Internetwork Service paths in the solution. Therefore, the delay of a path cannot be "improved" through the use of subnet-disjoint diversity routing. In the visualized example of Fig. 5-21, both p_3 and p_4 are disqualified immediately from the composition phase because their delay exceeds the adjusted end-to-end transmission delay requirement (although not the maximum tolerable delay initially indicated by the CSR). We denote the number of remaining CServ Internetwork Service paths as $0 \leq p'_{discover} \leq p_{discover}$. These remaining paths are then sorted in order of decreasing reliability.

Assuming that $p'_{discover} > 0$ (otherwise the CSDCA can terminate immediately and return a service denial response), the second step of this phase is to search among the remaining available paths to find a service solution. The objective of the CSDCA is to use the minimum necessary degree of diversity to find a CServ Internetwork Service that satisfies the source host requirements, since this minimizes the

network transmission and switching bandwidth used by the transaction and avoids unnecessary redundancy (see Section 5.2.1). As part of the CSR, the source host application specifies the minimum required degree of diversity. We denote this value as $p_{min} \in \mathbb{Z}^+$. A specified value of $p_{min} = 1$ indicates that the source host application accepts CServ Internetwork Services that do not employ subnet-disjoint diversity routing. Any p_{min} value greater than one necessitates the composition of a diversity-routed solution. If $p_{min} > p'_{discover}$, the CSDCA can terminate and return a service denial response since it cannot find a solution with the requested degree of diversity. Otherwise, if $p_{min} \leq p'_{discover}$, the process continues.

The service composition attempts begins by considering p_{min}-diversity combinations of CServ Internetwork Service paths in the set of remaining viable paths. If $p_{min} = 1$, the minimum reliability path is checked. If the reliability of this path meets the requirement, then a random path from the set of viable paths is chosen as the CServ Internetwork Service solution (we are sure that any individual path meets the requirement if the minimum meets the requirement). If the reliability of the worst path does not meet the minimal reliability requirement, then the maximum reliability path is checked. If the reliability of this path meets the minimal reliability requirement, then this path is chosen as the CServ Internetwork Service solution. Otherwise, if the reliability of the best path does not meet the minimal reliability requirement, the composition process moves to considering 2-diversity combinations of CServ Internetwork Service paths as the CServ solution (if the best path does not meet the minimal reliability requirement, then no other individual path in the set of viable paths can meet the requirement).

If $p_{min} > 1$, the service composition process proceeds in a very similar fashion to the search over single CServ Internetwork Service path solutions. First, the p_{min}-diversity combination of the p_{min} worst paths are checked. How is the reliability of the diversity-routed combination computed? This is covered in Section 5.4.2.3. For now it suffices to say that the reliability of the diversity-routed combination is always greater than the reliability of paths individually. If the diversity-routed combination of the p_{min} worst paths satisfies the minimal reliability requirement, then a random set of p_{min} paths from the $p'_{discover}$ viable paths are chosen as the service solution. If the reliability of the worst set of paths does not meet the minimal reliability requirement, then the diversity-routed reliability of the set of p_{min} best paths is checked. If the diversity-routed reliability of this set of paths meets the minimal reliability requirement, then this set of p_{min} most reliable paths are selected as the diversity-routed solution. Otherwise, if the reliability of the set of the p_{min} best paths does not meet the minimal reliability

requirement, the composition process moves on to considering $(p_{min} + 1)$-diversity combinations of CServ Internetwork Service paths the CServ solution. If the best set of p_{min} paths does not meet the minimal reliability requirement, then no other diversity-routed combination of p_{min} paths can satisfy it either.

This process continues incrementally until all $p'_{discover}$ viable paths are considered simultaneously as a possible diversity-routed solution. If this diversity-routed combination does not meet the minimal reliability requirement, then the CSDCA returns a service denial response. When all $p'_{discover}$ paths are considered together as a diversity-routed solution, there is no need to differentiate between first checking the worst set of $p'_{discover}$ paths and then checking the best set of $p'_{discover}$ paths; there is only one possible set of cardinality $p'_{discover}$.

In our visualized example of Fig. 5-21, $p'_{discover} = 2$. Regardless of whether or not the CSR specified $p_{min} = 1$ or $p_{min} = 2$, the individual candidate CServ Internetwork Service paths p_1 and p_2 do not satisfy the minimal required reliability. As shown in the cartoon, the diversity-routed CServ Internetwork Service using both of these paths does meet the minimal required reliability, and this is the CServ solution that is returned to the CServ source host requestor that originally generated the CSR. The point that represents this diversity-routed solution in the service plane indicates the reliability and delay performance metrics of this composed CServ Internetwork Service, ρ_{serv} and d_{serv} respectively.

Why does the composition phase choose a random set of path as the CServ Internetwork Service when possible? This is to avoid overuse of any given path in the network. Some paths may serve popular candidates for particular source host and destination host combinations. This randomization attempts to distribute CServ transactions over a set of paths where possible, reducing any CServ traffic congestion, rather than to rely on the same paths over and over for each transaction. Since we prefer this randomly-chosen CServ solution, we check the worst set of paths of a given set cardinality before checking the best set of paths of that cardinality in each step of the composition phase. Checking the diversity-routed combination of the best set of paths of a given set cardinality tells us if there exists a solution at that step, whereas checking the diversity-routed combination of the worst set of paths informs us that any diversity-routed combination of paths of that set size meets the reliability requirement of the CSR.

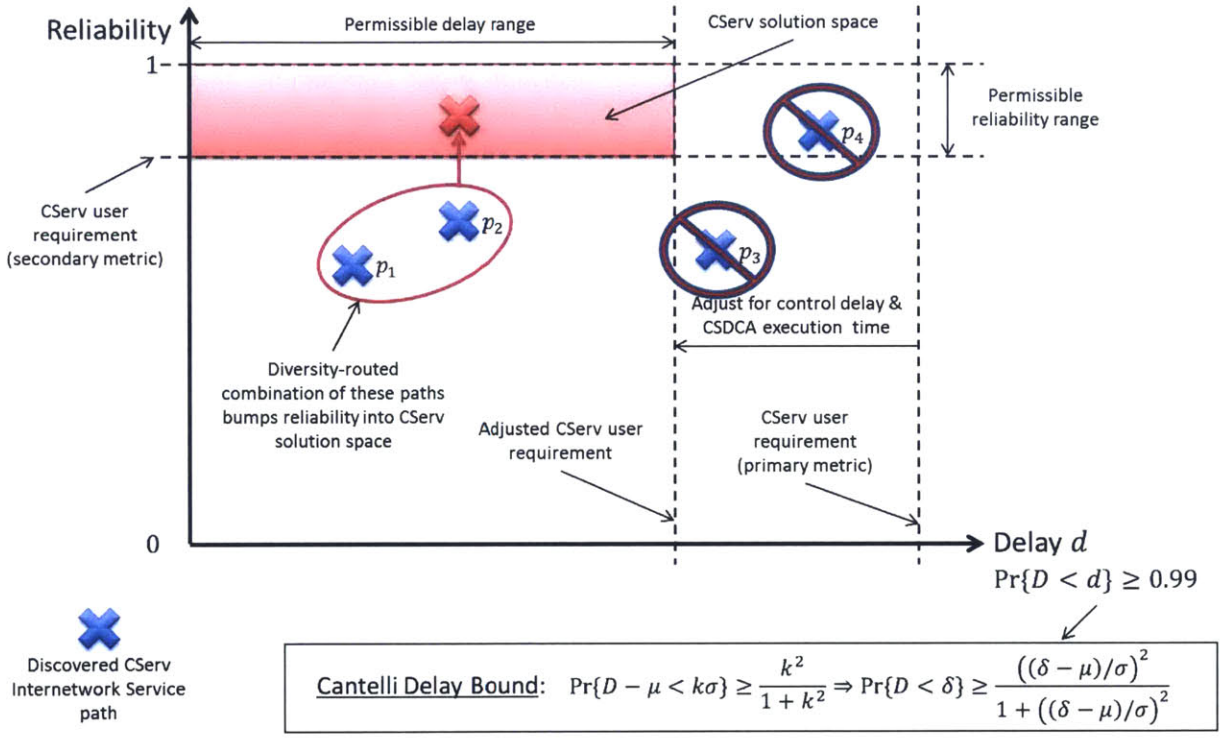


Fig. 5-21: A visualization of the operation of the CSDCA, which discovers CServ Internetwork Service paths before composing a CServ Internetwork Service solution that meets the CSR transaction performance requirements. In this illustration, D denotes the distribution of CServ Internetwork Service path delay, unknown except for its finite mean delay μ and finite delay variance σ^2.

5.4.2 A Few Supporting Considerations for the CSDCA

Before presenting the CSDCA as a complete algorithm, we want to focus on a few components mentioned in Section 5.4.1 in more detail. Specifically, we want to discuss the delay adjustment which is used to compute the new maximum allowable delay for the discovered paths, the method used to remove transit subnets from the transaction-specific internetwork representation in order to discover subnet-disjoint diverse paths, and the process of composing a solution from a set of discovered paths in more precise detail. Along with the previous discussions of path discovery using Dijkstra's Algorithm from Section 5.2.4, this prepares us to put the algorithm phase components together into a full algorithm description in Section 5.4.3.

5.4.2.1 The CServ Request Delay Adjustment

In Section 5.4.1, we mentioned that the user requirement on maximum allowable delay from the CSR must be adjusted to account for the delay overhead associated with the control message transmission and the CSDCA execution delay. Namely, the requested maximum tolerable delay value presented in the CSR is $0 \leq d_{max} \leq \infty$ and the adjusted end-to-end transmission delay requirement is $0 \leq d'_{max} \leq \infty$. The adjustment intends to account for the entire pre-transmission setup overhead process such that the delay experienced by the internetwork CServ datagram (including pre-transmission setup and end-to-end transmission) does not ultimately exceed the original maximum tolerable delay value, d_{max}. This overhead includes the delay of the off-band control channel CSR transmission from the CServ source host to the MC, the execution of the CSDCA algorithm, and the return delay of the off-band control channel service access (or denial) response transmission from the MC to the CServ source host.

We begin by considering the transmission delay of the off-band control channel. When the off-band control channel was introduced in Chapter 2, we stated that the control network topology and communication technology must be designed such that the system is highly reliable and presents predictable and constant delay to the CServ devices. The ultimate goal of the off-band control network and MC design is to present constant two-way signaling delay between CServ users and the control hierarchy with very little, if any, delay jitter. For this reason, the design likely requires heavy overprovisioning such that there is virtually no contention on the off-band control network (and possibly circuit-oriented connections between devices and the control plane entities). In the discussion of State

Measurement Service update reporting and CSR request generation, we have seen that the data rate burden on this network should be fairly low, making overprovisioning for robustness an achievable goal. Furthermore, the use of a physically-distributed but logically-centralized MC structure allows for the design to place MC entities close to the end-users. This distributes the control network burden over the physically-distributed entities and allows for predictable and minimal signaling delay between the CServ users and the controller points of presence. The actual physical design of the control hierarchy and the off-band control communication network is not the focus of this dissertation; future work should define exactly what topology and technology achieve these requirements of the control plane architecture.

With predictable signaling delay, the delay of the control channel transmissions between the CServ source host and the MC (and vice versa) can be easily computed and compensated for by the adjustment process. The CSR itself includes a generation timestamp that is transmitted with the service request. Upon receipt of the CSR at the MC, the difference between the current time and the timestamp value can be used to calculate the delay of the control channel transmission. With the appropriate off-band control channel design that presents constant and predictable two-way delay between the users and the control hierarchy, this value can be doubled to account for both the realized CSR transmission time and the future service access or denial response transmission time. Additionally, this value may be padded by some fixed amount to allow for any unexpected delay jitter experienced by the return transmission on the off-band control channel. As long as the control network is designed as required, this pad should be unnecessary under most nominal operating conditions.

There are sufficient instances of the CSDCA execution to accurately characterize its execution time; each time the CSDCA runs for a specific transaction, the MC should track its execution time from receipt of the CSR to the issuance of the service access or denial response (and termination of the CSDCA process). This set of execution times can be used to track the worst-case execution time which is used to further adjust the maximum tolerable delay requirement. The MC entity should run each instance of the CSDCA as a virtualized machine process such that the performance of each process is comparable. Since all transactions operate from the same baseline common internetwork graph and with the same hard-coded CSDCA algorithm parameters, there should then be little variability between the CSDCA execution instances. The maximum algorithm execution time should be close to the mean algorithm execution time, making the adjustment using the maximum an acceptable design decision that is not overly conservative.

This maximum CSDCA execution time is added to the two-way control channel transmission delay value computed from the CSR to form a transaction-specific value $0 < d_{adjust} < \infty$. With this value, we have that $d'_{max} = d_{max} - d_{adjust}$. This adjusted maximum tolerable delay value is used to judge the usability of the discovered CServ Internetwork Service paths found during the discovery phase of the CSDCA.

A belt-and-suspenders timestamp is included in the service access response from the MC after the execution of the CSDCA in case any of these delay adjustments should fail to accurately capture the delay of the pre-transmission control overhead. Specifically, this timestamp indicates the time at which the computed CServ Internetwork Service is invalidated. The CSR generation timestamp, the original maximum tolerable delay value d_{max}, and the delay of the CServ Internetwork Service solution can be used to create a timestamp that represents the latest time that a CServ datagram can be transmitted using that solution and still meet the requested service performance requirements. The CSR generation timestamp plus the maximum tolerable delay indicated by the CServ source host describes the latest time at which the internetwork CServ datagram should reach the intended destination host. If the composed CServ Internetwork Service solution delay is subtracted from this value, this gives the latest possible transmission time for the CServ datagram such that the CServ Internetwork Solution can bear the datagram to the destination host by the desired deadline. Therefore, if there are any issues in the off-band control channel transmission or CSDCA execution that generate anomalous and unexpected control overhead delay, this timestamp in the service access response can be used by the CServ source host to invalidate the computed CServ Internetwork Service solution.

We note that this delay requirement adjustment process runs in $O(1)$, or constant, time. The complexity of this process is independent of the number of vertices of the transaction-specific or common internetwork representation.

5.4.2.2 Removing Transit Subnets

During the Discovery phase of the CSDCA, the objective is to find multiple subnet-disjoint diverse CServ Internetwork Service paths between the source host and destination host as motivated in Section 5.2.2. In order to find a subnet-disjoint path as defined in that section via Dijkstra's Algorithm, the transit subnets of the last discovered path must be removed from the transaction-specific internetwork cost-

adjacency matrix representation before the next iteration of Dijkstra's Algorithm. In this section, we discuss the process of excluding transit subnets from subsequently discovered CServ Internetwork Service paths.

Following an instance of Dijkstra's Algorithm and the *extractshortestpath* helper function, the discovered path data structure p contains the explicit sequence of addresses that form the CServ Internetwork Service path in the path attribute $p.path$, starting with the source host address s and ending with the destination host address t. Recall from the discussion of hierarchical addressing in Section 3.1.1 that the addresses in this explicit path describe the subnets in the network that are traversed by the path. These subnet addresses can be extracted from the addresses in the path using a simple bitmask operation that retains the most-significant bits of the address (for example, the most significant 32 bits). The objective of the transit subnet removal process is to ensure that subsequently discovered CServ Internetwork Service paths do not traverse the same transit subnets, excluding the source and destination subnets which can be reused. This process is completed in two steps:

1. First, the unique transit subnet addresses in the last discovered CServ Internetwork Service path p are extracted from the path array attribute $p.path$;
2. Second, the vertices in these subnets that represent active CServ-enabled gateway routers in the internetwork representation are effectively removed from the transaction-specific internetwork cost-adjacency matrix.

We begin with the first step, which involves extracting the unique transit subnet addresses from the last discovered CServ Internetwork Service path using the path array attribute that explicitly describes the sequence of addresses in the path. These are used to form a set of transit subnets to be removed from the transaction-specific internetwork representation. As previously mentioned, this explicit sequence begins with the source host address s and ends with the destination host address t. Using the subnet bitmask operation, the source subnet address and destination subnet address should be gleaned from the first and last entries of the path array $p.path$, namely s and t. These subnet addresses are never added to the set of transit subnets to be removed from the transaction-specific internetwork representation. Next, each additional address in the path array attribute between s and t should be considered one at a time. For each address, the subnet bitmask operation is first applied. After the subnet address is recovered with this operation, the first check is whether or not the subnet address is

the same as either the source or destination subnet address in the path. If it is, the processing of that address stops, and we proceed to the next address in the sequence. There should be an egress active gateway router for the source subnet in the path and an ingress active gateway router for the destination subnet in the path; the subnet addresses for these gateway routers would not be included in the set of subnets to remove from the transaction-specific internetwork representation. Otherwise, if the subnet address is neither the source nor the destination subnet address, we next check to see if the transit subnet address has already been added to the set of transit subnet addresses to remove from the internetwork representation. If it is already in the set, we move on to consider the next address in the path array attribute sequence. If it is not already in the set, we add this transit subnet address to the set of transit subnets to remove from the internetwork representation. Once all addresses between the source and destination addresses in the $p.path$ attribute have been considered, this step ends. This entire step is completed using only the information in the last discovered path data structure, p. There are at most $O(n_s)$ operations to check for the subnet uniqueness of each address in the $p.path$ sequence, where n_s is the number of subnets in the network, and the length of the sequence is dependent upon the number of hops in the CServ Internetwork Service path. In the very worst case, this path could, in theory, contain all vertices in the transaction-specific internetwork representation (this would require that a Hamiltonian path [87] exists on the transaction-specific internetwork graph), or all $n_{st} = (n_s \times n_g) + (2 + n_{a_s} + n_{a_t})$ vertices using the notation from Section 5.3.3.2. This is not the likely case, as no further subnet-disjoint paths could be found since this path would traverse all possible transit subnets. In a more realistic scenario, the number of subnets in the path is unlikely to exceed the diameter of the network in terms of subnet hops. Thus, this step runs in $O(n_{st} \times n_s) = O({n_s}^2 n_g)$ time.

In the next step, we remove vertices representing active gateway routers in these transit subnets from the transaction-specific internetwork graph. To effectively remove a vertex from the graph, we need only set the weights of incoming edges in the cost-adjacency to the infinite value such that these vertices are isolated for the purpose of the shortest path Dijkstra's Algorithm. The weights of outgoing edges from these vertices can also be set to the infinite value, but this is not strictly necessary. Based on the description of the transaction-specific cost-adjacency matrix in Section 5.3.3.2, only the $k = 1$ and $k = 4$ planes of the matrix need to be modified, as these are the planes used in Dijkstra's Algorithm for routing purposes. More specifically, only the plane corresponding to the primary service metric needs to be modified. If reliability is the primary metric used for path discovery, only the $k = 1$ plane needs to be modified to remove transit subnets. Alternatively, if delay is the primary metric used for path discovery,

only the $k = 4$ plane with the Cantelli transform weights needs to be modified to remove transit subnets.

The transit subnet removal procedure operates as follows. For each of the n_{st} vertices in the cost-adjacency matrix, one-by-one, the bitmask operation is applied to reveal the subnet that vertex belongs to (recall that the MC can map indices of the matrix to their hierarchical addresses). The subnet address of the vertex is then checked against the set of transit subnet addresses to be removed from the transaction-specific internetwork representation. If the vertex belongs to one of the addresses in this set, the entire column of n_{st} entries of the appropriate plane of the cost-adjacency matrix (either the $k = 1$ or $k = 4$ plane) is set to ∞. (As noted, the entire row of the same plane for this vertex could also be set to ∞, but this is unnecessary.) Otherwise, the column of the cost-adjacency matrix remains unmodified. After all vertices have been checked, the resulting transaction-specific cost-adjacency matrix representation is ready for the discovery of a subnet-disjoint path during the next instance of Dijkstra's Algorithm. In the most naïve approach to this step, all n_{st} vertices in the graph are checked against the set of transit subnets to remove (note that some vertices do not strictly need to be checked, such as the source and destination host vertices). Each vertex is checked against a set of at most $(n_s - 2)$ addresses in the set of transit subnets to remove (excluding the possibility of the source and destination subnets). If there is a match, n_{st} operations are required to set the entire column of entries in the appropriate plane of the cost-adjacency matrix to ∞. In the worst case scenario, all but six of the n_{st} vertices could match subnet addresses in this set; those six vertices represent the source and destination hosts, their upstream access routers, and the active gateway routers in their subnets. Thus, this step runs in $O({n_{st}}^2) = O({n_s}^2 {n_g}^2)$ time. The entire transit subnet removal process, including both steps, requires worst-case $O({n_{st}}^2) = O({n_s}^2 {n_g}^2)$ time. We note that this is the same running time as the basic implementation of Dijkstra's Algorithm on the transaction-specific internetwork graph, $O({n_{st}}^2)$.

5.4.2.3 Composing a Solution

In Section 5.4.1, we introduced the composition phase of the CSDCA and briefly described its operation. In this part, we motivate its operation and describe the process more generally.

Following the execution of the discovery phase of the CSDCA, we have a set of paths $\mathbb{P}$, where each $p \in \mathbb{P}$ is characterized by three distinct attributes:

1. $p.path$ is the array of addresses that describe the CServ Internetwork Service path from source host to destination host;
2. $p.reliability$ is the computed reliability value of the CServ Internetwork Service path using the definition of Section 5.2.3;
3. and $p.delay$ is the computed delay bound of the CServ Internetwork Service path using the definition of Section 5.2.3 (this could also be the distance of the path using the Cantelli transform nonnegative, additive weights if this distance is less than the adjusted maximum tolerable delay requirement).

The size of the set $\mathbb{P}$ is dependent upon the parameter of the CSDCA discovery phase which specifies the maximum number of subnet-disjoint diverse paths to discover, $p_{max} \in \mathbb{Z}^+$, and the topology of the transaction-specific internetwork graph which may limit the possible number of subnet-disjoint diverse paths between the source and destination host vertices to $p_{poss} \in \mathbb{Z}^+$. The value of p_{poss} depends on the degree of the source and destination subnets, as well as the connection topology between them. Ultimately, the number of discovered paths is $p_{discover} \triangleq \min\{p_{max}, p_{poss}\}$. The cardinality of the set $\mathbb{P}$ is thus $p_{discover}$, or $|\mathbb{P}| = p_{discover}$.

The first step of the composition phase is to exclude any path $p \in \mathbb{P}$ that does not meet the adjusted maximum tolerable delay value, d'_{max}, as presented in Section 5.4.2.1. In other words, for each $p \in \mathbb{P}$, we remove p from $\mathbb{P}$ if $p.delay > d'_{max}$. This step is necessary since the diversity-routed solution using multiple paths cannot improve the delay performance of any given path in the solution; the delay bound of a diversity-routed solution is defined as the maximum delay of the constituent paths in the solution. If the delay of a path exceeds the maximum tolerable delay, it cannot be used as part of the CServ Internetwork Service solution. Following the consideration of each path $p \in \mathbb{P}$, we are left with a set of delay-feasible paths $\mathbb{P}' \subseteq \mathbb{P}$ such that $|\mathbb{P}'| = p'_{discover} \in \mathbb{N}$ and $0 \leq p'_{discover} \leq p_{discover}$. This step runs in $O(1)$, or constant, time with respect to the size of the transaction-specific internetwork graph since the number of paths to consider, $p_{discover}$, is a value that, at its largest, is a constant independent of the size and complexity of the graph. It is for this reason that we set a maximum number of paths to consider in the composition phase as a parameter of the CSDCA. This limits the execution time of the

algorithm both through the number of iterations of the discovery phase and the complexity of the composition phase.

In the second step of the composition phase, we attempt to compose a reliability-feasible solution based on the remaining paths for consideration, $\mathbb{P}'$. As previously discussed in the dissertation, the use of subnet-disjoint diversity routing as a CServ Internetwork Service solution can improve upon the reliability of each individual path. With the fragile assumption of statistical independence between subnet-disjoint CServ Internetwork Service paths, the reliability of a k-diversity routed CServ Internetwork Service, where the reliability of a path $p_i \in \mathbb{P}'$, or $p_i.reliability$, is given as ρ_i, is given in Eq. (5.1). The subsequent inclusion of any path to this diversity routed solution strictly improves upon the reliability of the CServ Internetwork Service as long as the reliability of the newly added path is greater than zero.

We begin by consider how many possible CServ Internetwork Service solutions there are in the set $\mathbb{P}'$. If we were to consider all possible solutions, including individual paths, 2-diversity routed solutions, 3-diversity routed solutions, and so forth, the total number of possible solutions, β, is given as:

$$\beta = \sum_{i=1}^{p'_{discover}} \binom{p'_{discover}}{i} = \sum_{i=1}^{p'_{discover}} \frac{p'_{discover}!}{i!\,(p'_{discover} - i)!}. \tag{5.18}$$

The growth of the number of possible solutions is shown in Fig. 5-22 as we vary the cardinality of $\mathbb{P}'$ on two different scales. As the number of paths to consider grows, the number of possible solutions becomes computationally intractable. Since the number of possible solutions grows exponentially in the size of $\mathbb{P}'$, we want to limit the number of solutions we consider, in addition to our parameter p_{max} which limits the maximum size of $\mathbb{P}'$ through the discovery phase, in order to quickly execute the composition phase of the CSDCA.

In addition to limiting the number of solutions to consider during the composition phase, we have three more concerns that we want to subsume into the design of the composition phase of the CSDCA. First, we want to use the minimum required number of diverse paths in the CServ Internetwork Service solution in order to minimize the amount of network resources used by the transaction and avoid

needless redundancy. Second, we want to respect the minimum required diversity request of the CSR, p_{min}, which specifies the minimum number of subnet-disjoint paths that the CServ source host wants in the composed CServ Internetwork Service solution. Third, and finally, we would like to randomize the solution over the possible paths for consideration in $\mathbb{P}'$ when possible. This helps to avoid using the same CServ Internetwork Service paths in many composed solutions, risking the oversubscription of some network-wide CServ Intranetwork Services with real CServ datagram transactions (consider the Learning Session rate of the State Measurement Service Learning Session protocol presented in Chapter 4). We incorporate all of these considerations in the following description of the service composition process, which seeks to find a diversity-routed set of CServ Internetwork Service paths that, together, are reliability-feasible and meet the minimum required reliability of the transaction's CSR.

The second step of the composition phase proceeds as follows. We begin by sorting the remaining paths in $\mathbb{P}'$ in order of reliability. Let the reliability of path $p_i \in \mathbb{P}'$, or $p_i.reliability$, be denoted as ρ_i. Without loss of generality, we can index the paths in $\mathbb{P}'$ such that $\rho_1 \geq \rho_2 \geq \cdots \geq \rho_{p'_{discover}}$. Since the set of remaining paths is a small data set (at most it contains p_{max} paths), we can use a simple insertion sort [83] which runs, worst-case, in quadratic time with respect to the size of the set $\mathbb{P}'$ (i.e. $O(|\mathbb{P}'|^2)$ running time), and in constant time with respect to the size and complexity of the transaction-specific internetwork graph.

We then initialize the number of paths to consider as a CServ Internetwork Service solution, k, as p_{min}, the minimum required diversity in the solution from the CSR. If $p_{min} > p'_{discover}$, we can immediately give up and return a service denial. Otherwise, we proceed as follows. We first check the diversity-routed reliability of the p_{min} least reliable paths in the set, that is $\left\{p_{p'_{discover}-p_{min}+1}, \ldots, p_{p'_{discover}}\right\}$, using Eq. (5.1). If the reliability of this diversity-routed CServ Internetwork Service solution meets the minimum tolerable reliability requirement of the CSR, ρ_{min}, then we are free to randomly select any p_{min} paths from the set $\mathbb{P}'$ as the CServ Internetwork Service solution. If the p_{min} least reliable paths meet the minimum tolerable reliability requirement, then we know for sure that any p_{min} paths in the set meet the requirement. If the reliability of this diversity-routed CServ Internetwork Service solution does not meet the minimum tolerable reliability requirement of the CSR, we then check the p_{min} most reliable paths in the set $\mathbb{P}'$, that is $\{p_1, \ldots, p_{p_{min}}\}$, using Eq. (5.1). We do this in lieu of checking all possible sets of p_{min} paths from $\mathbb{P}'$ to speed up the composition phase process. If the reliability of this

diversity-routed CServ Internetwork Service solution meets the minimum tolerable reliability requirement of the CSR, ρ_{min}, we choose the p_{min} most reliable paths in the set $\mathbb{P}'$ as the CServ Internetwork Service solution. We are not free to randomly select paths in this case; if the p_{min} most reliable paths meet the minimum tolerable reliability requirement, there is no guarantee that any other set of p_{min} paths meet the requirement. If this combination of the p_{min} most reliable paths does not meet the minimum tolerable reliability requirement, then we increment the number of paths to consider in the CServ Internetwork Service solution, k, such that $k = p_{min} + 1$. The process then proceeds as before, first checking the k least reliable paths from $\mathbb{P}'$, that is $\left\{p_{p'_{discover}-k+1}, \ldots, p_{p'_{discover}}\right\}$, before checking the k most reliable paths from $\mathbb{P}'$, that is $\{p_1, \ldots, p_k\}$, using Eq. (5.1). If a solution that meets the minimum tolerable reliability from the CSR is still not found, the number of paths to consider is further incremented such that $k = p_{min} + 2$. This process continues until a solution is found or until $k = p'_{discover}$. At that point, the $p'_{discover}$-diversity routed solution of all paths in $\mathbb{P}'$ either meets the minimum tolerable reliability requirement from the CSR or it does not. If it does, a solution has been found. Otherwise, the composition phase returns a service denial.

The previously described process is illustrated in Fig. 5-23 for an example where $p'_{discover} = 5$ and $p_{min} = 2$. At each value of k, there are constant number of operations using the aforementioned steps. In the worst case (when $p_{min} = 1$ and there is no solution in the set of paths), this process requires linear running time in the number of paths in $\mathbb{P}'$, or $O(|\mathbb{P}'|)$, and constant running time with respect to the size and complexity of the transaction-specific internetwork graph.

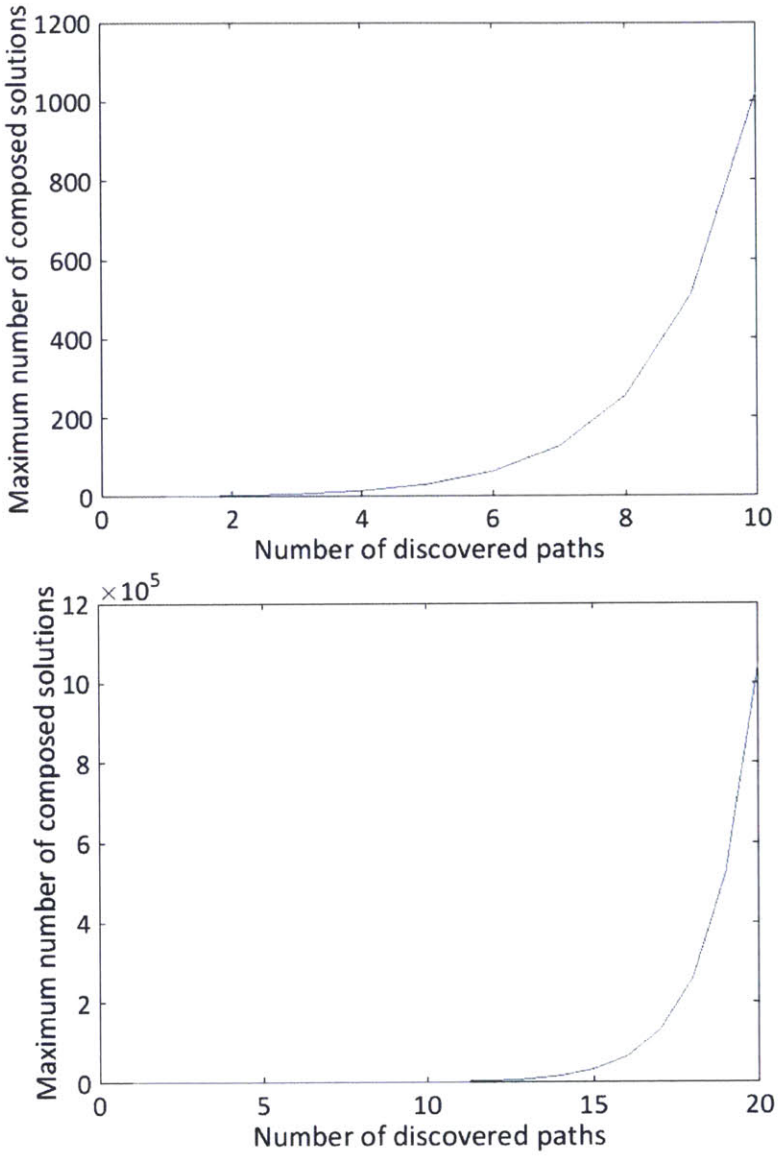


Fig. 5-22: The number of possible exhaustive composed solutions that can be considered during the composition phase as a function of the number of subnet-disjoint CServ Internetwork Service paths found during the discovery phase of the CSDCA.

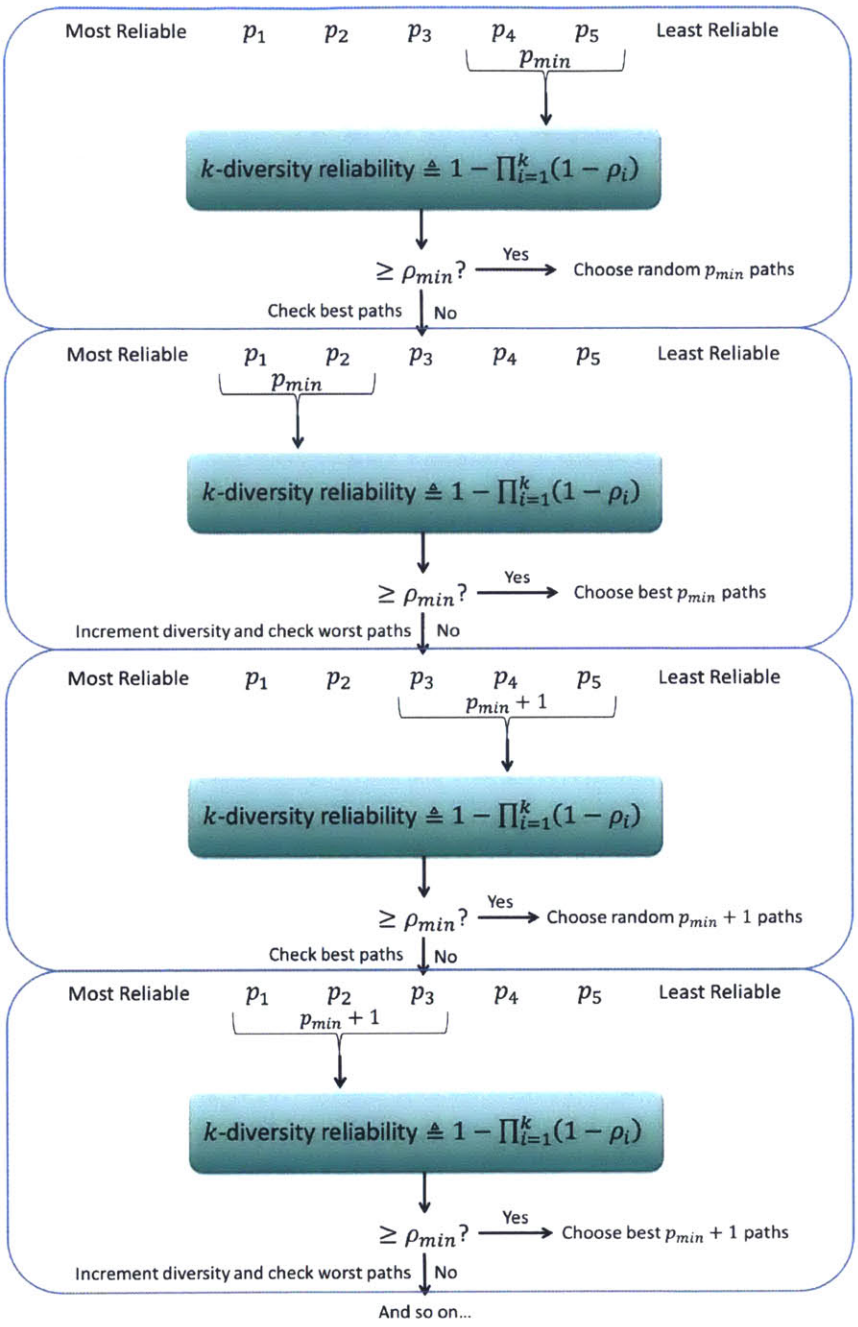


Fig. 5-23: This figure illustrates the steps in the composition phase of the CSDCA after excessive delay paths are excluded and the remaining paths are sorted based on their reliability performance. The depicted example uses $p'_{discover} = 5$ and $p_{min} = 2$.

5.4.3 The Algorithm

With the necessary foundations established, we bring together the parts of the CSDCA into a holistic picture in this section with an accompanying block diagram description in Fig. 5-24. This section formalizes the overview presented in Section 5.4.1 with the CSDCA visualization. For each non-trivial component of the algorithm, we provide an asymptotic running time with respect to the size of the transaction-specific internetwork graphical representation (which has n_{st} vertices), the complexity of which dominates the running time of the CSDCA. Following the presentation of the algorithm, we give the overall CSDCA running time and make a few related comments.

The Critical Service Discovery and Composition Algorithm

Discovery Phase

Start:

- The CServ Request arrives at the MC from the CServ source host on the off-band control channel

Initialization:

- The transaction-specific internetwork cost-adjacency matrix C_{st} is prepared from the current common internetwork cost-adjacency matrix and additional stored state information
    - Running time: $O({n_{st}}^2)$
    - Comment: This involves copying and lookups of $n_{st} \times n_{st} \times 4$ matrix fields.
- Initialize discovery phase variables: $i \coloneqq 1, \mathbb{P} \coloneqq \emptyset$

Discovery of a CServ Internetwork Service path:

- Run in instance of Dijkstra's Algorithm on C_{st} using the appropriate source vertex and "plane" of the three-dimensional matrix according to the primary CServ performance metric
    - Running time: $O({n_{st}}^2)$

- Is a path with finite distance to the destination vertex found?
    - No: Go to ***Composition Phase***
    - Yes: Continue
- Use *extractshortestpath* helper method to retrieve the explicit CServ Internetwork Service path information, $p_i.path$, for this discovered path p_i
    - Running time: $O(n_{st})$
    - Comment: This is an absolute worst-case running time if the discovered CServ Internetwork Service path traverses, on the order of, all vertices (assuming that a Hamiltonian path exists); typically, the length of the path is realistically limited by the transaction-specific internetwork graph diameter.
- Compute $p_i.reliability$ and $p_i.delay$ from the output of Dijkstra's Algorithm, the explicit path information, and the transaction-specific cost-adjacency matrix
    - Running time: $O(n_{st})$
    - Comment: Refer to the previous comment; worst-case running time is $O(n_{st})$ only when a discovered CServ Internetwork Service path includes, on the order of, all graph vertices.
- Add the newly discovered path to the set of discovered paths: $\mathbb{P} := \mathbb{P} \cup p_i$
- Is $i < p_{max}$? (The parameter that describes the maximum number of paths to discover during this phase of the algorithm)
    - No: Go to ***Composition Phase***
    - Yes: Continue
- Remove transit subnets of path p_i from the transaction-specific internetwork cost-adjacency matrix C_{st}
    - Running time: $O({n_{st}}^2)$
- Increment the counting variable of the Discovery Phase: $i := i + 1$
- Go to **Discovery of a CServ Internetwork Service path**

End of Discovery Phase

Composition Phase

Initialization:

- Compute the adjusted maximum tolerable delay requirement, d'_{max}, from the maximum tolerable delay value presented in the CServ Request
- For each path $p \in \mathbb{P}$, remove p if $p.delay > d'_{max}$, forming $\mathbb{P}' \subseteq \mathbb{P}$
    - Running time: $O(1)$
    - Comment: The maximum number of possible paths in $\mathbb{P}$ for consideration is limited by the p_{max} parameter of the Discovery Phase, not the topology of the transaction-specific internetwork graph.
- Sort paths in $\mathbb{P}'$ according to their reliability attributes
    - Running time: $O(1)$
    - Comment: A simple insertion sort, for example, runs in constant time with respect to the size and complexity of the transaction-specific internetwork graph; an insertion sort would be $O(n^2)$ generally for n elements, but the number of elements in this sorting operation is, at most, some small constant that is not dependent on the size of the transaction-specific internetwork graph.
- Initialize composition phase variables: $k \coloneqq p_{min}$
- Is $k > |\mathbb{P}'|$?
    - Yes: Go to **End service denial**
    - No: Continue

Check for k-diversity solutions:

- Is the reliability of the k-diversity routed solution using the k least reliable paths greater than or equal to the minimum tolerable reliability value from the CServ Request?
    - Yes:
        - Choose k paths at random from $\mathbb{P}'$ as the CServ Internetwork Service solution
        - Go to **End service access**
    - No: Continue

    - Running time: $O(1)$
- Is $k = |\mathbb{P}'|$?
    - Yes: Go to **End service denial**
    - No: Continue
- Is the reliability of the k-diversity routed solution using the k most reliable paths greater than or equal to the minimum tolerable reliability value from the CServ Request?
    - Yes:
        - Choose the k most reliable paths from $\mathbb{P}'$ as the CServ Internetwork Service solution
        - Go to **End service access**
    - No: Continue
    - Running time: $O(1)$
- Increment the diversity variable of the Composition Phase: $k := k + 1$
- Go to **Check for k-diversity solutions**

End service access:

- Return service access response using CServ Internetwork Service solution to the CServ source host on the off-band control channel

End service denial:

- Return a service denial response to the CServ source host on the off-band control channel

End of Discovery Phase

End of The Critical Service Discovery and Composition Algorithm

The overall running time of the CSDCA is dominated by the steps that run in $O({n_{st}}^2)$ time, namely the preparation of the transaction-specific internetwork cost-adjacency matrix, the instance of Dijkstra's Algorithm, and the removal of transit subnets from the cost-adjacency matrix following the discovery of a CServ Internetwork Service path. While the preparation of the cost-adjacency matrix itself occurs only one, the other two components may occur several times depending on the hard-coded value of p_{max} and the connectivity and topology of the transaction-specific internetwork graph. At most, however, a small constant number of iterations of Dijkstra's Algorithm and the removal of transit subnets from the graph occur during the execution of the CSDCA. All parts of the composition phase execute in time that is constant with respect to the size and complexity of the network.

In conclusion, the CSDCA running time is $O({n_{st}}^2)$, where $n_{st} = (n_s \times n_g) + (2 + n_{a_s} + n_{a_t})$ vertices using the notation from Section 5.3.3.2. More specifically, the running time of the CSDCA is quadratic in the number of subnets in the network and the number of active CServ-enabled gateway routers per subnet (the $[2 + n_{a_s} + n_{a_t}]$ term can be treated as a small constant that does not grow with the size or complexity of the internetwork, only the upstream connectivity of the transaction source and destination hosts). Although an optimized implementation of Dijkstra's Algorithm allows it to run in sub-quadratic time, simultaneous running time improvements must be made to the preparation of the transaction-specific cost-adjacency matrix and the process for removing transit subnets from the representation in order to improve upon the overall running time of the CSDCA. And although it is helpful to understand the asymptotic running time behavior, it is more important, for the purpose of adjusting the maximum tolerable delay requirement, to characterize the realized maximum running time of the CSDCA using the specific hardware and software implementation for the network's MC entity.

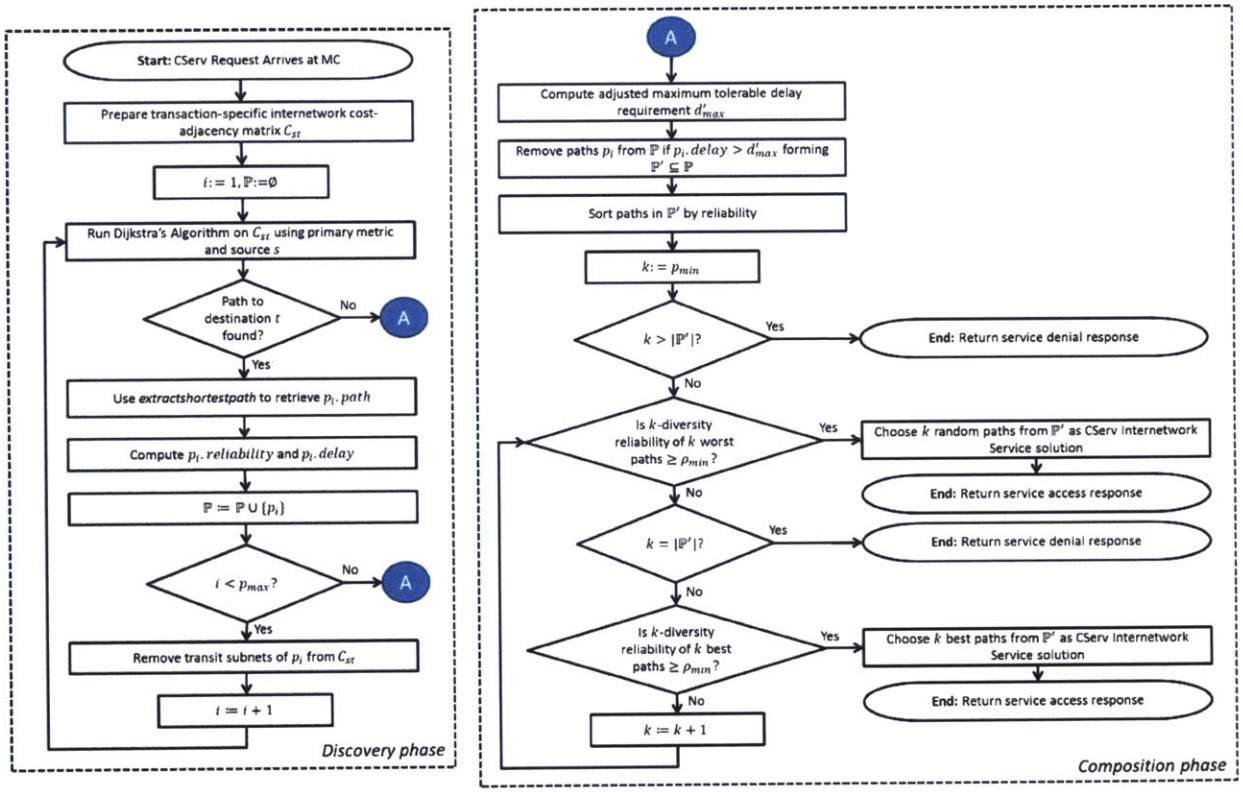


Fig. 5-24: This block diagram outlines the operation of the two phases, the discovery phase and the composition phase, of the CSDCA.

5.5 Conclusion

In this chapter, we described the role of the MC in the CServ architecture's control plane structure. Together, the State Measurement Service and the control hierarchy (SCs and MC) form the control plan for the CServ network. The State Measurement Service empowers the MC to make internetwork service decisions, which was the focus of this part of the dissertation. With the reports from the CServ performance metric learning protocols, we described how the MC generates decisions on the requests for internetwork CServ service. The focus was both on the representation of the global network at the MC and the algorithmic process for determining the availability of an internetwork path or set of paths that meet the requirements of the CServ source data application. In doing so, we defined the precise way that we probabilistically describe the delay of a CServ Internetwork Service path, and we applied this definition to our heuristic path discovery algorithm which precedes the composition of service from the set of discovered paths.

Many of the techniques used in the CSDCA, such as the determination of the reliability and delay of a path and the determination of the reliability of a diversity-routed internetwork solution, rely on the brittle assumption of statistical independent between the network subnets. As motivated in Chapter 2, this assumption is borne of the intentional heterogeneity that distinguishes the disparate CServ subnets. However, even with this intentional design, the fragility of this assumption can be quickly exposed during certain events, particularly those that involve adversarial action. An adversary is likely to attempt to correlate all possible failure modes to exact impact on the operation of the CServ network. More generally, "Black Swan" events in general are likely to impact the CServ network in unforeseen ways that may challenge the assumption on statistical independence. However, this makes the use of subnet-disjoint diversity routing as a CServ Internetwork Service solution even more critical, as it presents some notion of survivability in the face of adversarial or dire situations that would otherwise easily disrupt critical messaging service. It is vital to consider the meaning of the probabilistic guarantees during unexpected, high-impact events and what it means to operate under these network conditions.

Throughout the chapter, we relied on two architectural aspects that have not been the focus of this dissertation and require further attention. First, we assumed that the off-band control network can be designed such that it presents reliable, robust, and predictable signaling delay between the end-user hosts and the control hierarchy (namely, the MC in this chapter). We have motivated the reasoning

behind this architectural requirement and its feasibility. However, the actual control network and communication design should be the focus of future work. Second, we assumed the availability of efficient data structures for the storage of State Measurement Service reports that enable constant time lookups. Without these efficient structures, the running time of the CSDCA could be worse than the presented result in Section 5.4.3. We motivated and proposed multimaps (or multihash tables) as a promising candidate for this storage structure requirement, but some details must be worked out in order to realize this design. This should also be the focus of future work on the CServ architecture.

Finally, it is worth noting that there are many possible optimizations to the CSDCA that may, or may not, be necessary to improve the realized running time on the MC entity's particular hardware and software configuration. The most fundamental form of the CSDCA is presented in this chapter, but many other forms exist that can meet the same goals. For example, there is no point in discovering five subnet-disjoint CServ Internetwork Service paths if only two are needed to satisfy the demands of the CSR. This approach uses wasteful iterations of Dijkstra's Algorithm. From this point of view, it might be more efficient to interleave the discovery and composition phases appropriately rather than to always execute the two phases in sequence as presented.

Chapter 6

Conclusion

In this thesis, we have addressed the architectural design of a novel internetwork architecture, the Critical Service (CServ) architecture, which provides *a priori* explicit reliability and delay performance guarantees to the end-user application on a transaction-specific basis in order to support mission-critical messaging over unreliable and interconnected subnet systems. This work represents the foundations for the system description; we have highlighted the primary enabling technologies and components, describing their operation, interaction, and performance. However, this internetwork architecture proposal is still in its infancy, and there are many aspects that have not yet been fully fleshed out and require future research and development.

We considered related service-oriented architectures, and we identified their mismatch to the desired goal of providing explicit internetwork guarantees to application-specific transaction needs. This motivated the creation of the CServ architecture. We began with an overview of the system, presenting the primary technologies and components through a transaction walk-through. This discussion started with the presentation of pre-transaction state maintenance, which includes the active learning of subnet performance through the State Measurement Service. The State Measurement Service allows the individual networks to retain autonomous operation, choosing their preferred routing and forwarding policies, while still participating in the CServ network. The consideration of pre-transaction state maintenance necessitated the deployment of an off-band control plane used to collect and aggregate estimated performance state at the local subnet. This subnet controller played a fundamental role in the control plane hierarchy; the subnet controller represents the autonomous local control, relaying only the necessary local performance state information to the global, top-level logically-centralized control plane, the master controller (MC). The MC collects and maintains a subset of the learned and reported network-wide state, facilitating its primary role as the arbitrator for internetwork critical messaging

service. This structure also shields subnets from the need to exchange estimated performance state with their direct neighboring subnet peers.

With the help of the pre-transaction state maintenance, the CServ transaction is ready to commence upon the generation of a critical message payload. The flow was considered step-by-step, beginning with the MC conversation via the CServ API. The source of the critical transaction negotiates service with the logically-centralized MC that acts as a service broker. The source states the minimal requirements of the transaction, and then the MC attempts to compose service based on the most recently reported network-wide, aggregate state in order to meet the demands of the application. Following a successful service access response which describes the explicit internetwork service, the critical message is borne on the CServ data plane from the source host to the destination host using the results from the MC algorithm and the locally offered subnet intranetwork services along the way. In the subsequent chapters of the thesis, we considered several of the CServ systems in more detail, including the execution of data plane control in terms of the CServ datagram header information, possible protocol methods to implement the requirements of the State Measurement Service, the representation of the internetwork topology at the logically-centralized MC, and, the brains of the architecture, the MC algorithm used to discover and compose internetwork service that meets the mission-critical messaging needs.

6.1 Final Thoughts

Throughout the description of the CServ architecture, we have relied upon the statistical independence between subnets of the network (and sometimes even within a subnet, as during the description of the Collector protocol that can be used to realize the State Measurement Service). Although subnets need only look "independent enough" such that there are no strong correlations between them for most of the results to hold to some degree, there is a danger in making this type of modeling assumption. The use of this statistical independence assumption, importantly, is how we compose subnet-disjoint diversity routed solutions during the Critical Service Discovery and Composition Algorithm (CSDCA); if the assumption does not hold, then the composed CServ Internetwork Service does not necessary meet the demands of the source host application. This assumption is also invoked to define the delay of a network path based on its performance statistics. The reliance upon this assumption means that we must be wary of its validity.

In several locations during the dissertation, we alluded to the impact of "Black Swan" network events on the CServ architecture. The concept of the "Black Swan" event was first introduced by statistician Nicholas Taleb in his 2007 book, *The Black Swan: The Impact of the Highly Improbable* [52]. In this book, the black swan is used as a metaphor for unexpected, unforeseen events that have major impact and are often inappropriately rationalized subsequently with the use of retrospection. The very nature of their improbability make these events impossible to assign probabilistic measure to *a priori* – they are considered outside the realm of the human capability to generate models. For this reason, we have made the reader as aware as possible to the potential impact of such events in the context of the CServ architecture. Possible "Black Swan" impacts in the CServ network include the generation of unexpected waves of mission-critical CServ messages beyond the expected peak rate or the simultaneous disruption of many or all transit subnets interconnecting the source and destination subnet of a transaction. While we can conceive of some possible outcomes, it is difficult to predict the "Black Swan" event that might produce these results.

One thing is for certain: the CServ architecture does not explicitly provide opportunity to model or learn what it has not observed. The State Measurement Service, for example, only estimates CServ performance metrics based on the realized performance of past traffic. So when that unforeseen event and impact occurs, there is no real understanding of guaranteed performance in the same vein as previously presented in this work. Furthermore, if the "Black Swan" event is the result of a malicious actor intent on the disruption of critical messaging service, the statistical independence assumption is likely to completely break down; the malicious actor would seek to correlate all possible failure modes in the network in order to exact maximum effect. The best that we can aim for in the design is survivability. Through the use of subnet-disjoint diversity internetwork routing, even if the subnets are not strictly statistically independent, we have gained some notion of survivability against an unexpected high impact event so long as it does not affect all paths or subnets in the solution.

Another consideration that we must bear in mind in the context of the CServ architecture is the limitation imposed by the speed of light. The design makes use of learned and reported state information in order to compose internetwork service that meets the demands of critical applications. Given that this estimated performance information must traverse an off-band control channel from the network entity forming the estimate to the control plane hierarchy, there is a necessary staleness to the information that the MC operates on due to physical propagation times. This is particularly true for a

physically-distributed but logically-centralized implementation of the MC entity, as the implementation of consistency algorithms to synchronize the devices introduces additional delay thanks to the physical separation of the components. Although we have to accept the physical restrictions of acting on network telemetry reports, it is important to comprehend what it means to make guarantees with stale state information. The application that utilizes the CServ architecture for critical messaging should be designed with this in mind. There has been some work to quantify what it means to operate on old state using information theoretic techniques in other architectures [88], and this type of understanding and modeling could, in the future, be applied to the CServ architecture directly. But for now, we stand on the belief that acting on slightly stale state is better for the purpose of high-performance service than acting blindly on no state information at all.

6.2 Future Work

As the CServ architecture is a new internetwork proposal, there are many open research problems that need to be considered and developed before this type of network can be deployed. Throughout the dissertation, we have alerted the reader to some opportunities for additional research beyond what has been presented. We consider some more directions here, although this is by no means an exhaustive list of open problems.

A big "sore thumb" for the CServ architecture is its security architecture, which has not been covered in this work. The CServ proposal is designed to bear the most critical of the critical messages in the network, and it marks these messages using control header information as such. Due to the nature of these mission-critical datagrams, they must immediately be considered as high-value targets for adversarial action. The adversary who wishes to disrupt critical communications would look for manners to divert, disrupt, destroy, or delay CServ traffic. Even though we have allocated a field in the CServ Internetwork Service header for authentication of messages to ensure that a rogue host does not spoof a critical alert or command message, future work needs to consider other aspects of the security framework to further protect the critical messages and the CServ infrastructure.

For example, there should be some technique to positively verify that a CServ datagram has been granted data plane access through the MC. This could be as simple as including a signature field for the MC that can be verified by intermediate active routers in the CServ Internetwork Service path, but the

cryptographic process of signature verification could induce unnecessary or unacceptable delay into the path. At the same time, the contents of the critical message in the CServ datagram should be protected from prying eyes since the CServ datagram header control information identifies it as a critical message and, thus, as a datagram likely to reveal important or sensitive information. Should the encryption of the CServ datagram payload be the role of the network layer? Or should it be left to the application? Datagram encryption for data privacy is included as part of some transport layer protocols, such as Transport Layer Security (TLS) [89]. Since the CServ network layer is designed to absorb some functionality of the transport layer, there is an argument to be made that the CServ network layer should be accountable for protecting the privacy of the transmitted critical message. The inclusion of encryption and decryption processes in the network layer, however, would generate additional CServ Internetwork Service delays and further challenge the CServ architecture to successfully meet the performance demands of CServ applications. The question of where the responsibility for message encryption lies requires further thought and development. It is worth noting, however, that the use of the Learning Session protocol for the State Measurement Service provides a sort of camouflage for the critical message datagram, even if it does not encrypt the payload. Since the protocol stuffs real CServ datagrams into Learning Session flows that are primarily comprised of active dummy datagrams in the CServ traffic class, it is more challenging for an adversary to ascertain the real critical message of interest.

As another example, there should be some method of checking the veracity of State Measurement Service reports. Given the current architecture description, an active router within a subnet could misrepresent estimated CServ Intranetwork Service performance metrics in order to black-hole critical traffic or to divert it to other subnets. With global purview, the MC is in the perfect position to initiate a protocol to check the realized CServ Intranetwork Service performance of a subnet using dummy active probes. It could request that two peering subnets of the subnet in question exchange a dummy CServ datagram that can be used to substantiate the reported CServ performance metrics. However, this introduces additional data plane transmission and switching burden beyond the State Measurement Service and requires more complex processing at the MC.

And as one final example, admission control is a vital concern of the CServ architecture and has not been considered in detail in this thesis. It is important to ensure that CServ traffic does not overburden the network, either benignly or due to adversarial action. If the transient volume of critical datagrams

entering the data plane exceed the dimensioning of the network (if it exceeds Learning Session rate or goes beyond ten percent of the total network traffic, for example), this both jeopardizes the validity of the performance guarantees of the MC and the ability of the network to simultaneously serve best effort network traffic. In fact, an adversary may wish to exploit this using some sort of (distributed) denial-of-service attack that floods the MC with CServ Requests. Luckily, the MC is again in the perfect position to police network admission for critical messages since it serves as a service computation broker, but the details of these algorithms and policies need to be further developed and analyzed.

Another area we have already alluded to that requires additional research and work is that of the physical control plane, including the controller entities themselves and the off-band control network. Although we have discussed the concept and requirements of the control plane and communication channels, this network requires careful design consideration and dimensioning guidelines. First, the actual physical topology of the network controllers should be considered in more detail. How should the physically-distributed MC entities be allocated? Is it enough to have one or two physical MCs per "metro area," or do we only need one or two for the entire network? This question is intimately coupled with several others, such as the scale and physical distribution of the entire network, the cost of a MC entity, and the performance of the distributed system protocol used to realize state consistency between the entities and create a logically-centralized global control plane. The design of the off-band control channel and network interconnecting the hosts and routers with the control plane also needs further development. Specifically, what kind of communication substrate and protocol best suits the needs of this robust, reliable, and predictable system? And how exactly would this off-band network be implemented separately from the network data plane? Is there a logical control network topology that is simple but efficient enough to distribute the aggregate load of the state measurement service reports, CServ Requests, and service responses?

There is opportunity for additional work on the CServ architecture components covered in this document as well, such as the State Measurement Service and the CSDCA. There may be other protocol ideas that outperform those presented in this dissertation, or there may be useful optimizations to the algorithms as presented here (we have already introduced a straightforward optimization to the MC algorithm that interleaves the discovery and composition phases) that improve their efficiency and execution time. Ultimately, many of these questions may be answered through verification of the CServ architecture concept at scale, using a network testbed framework or, at the very least, some system-

level simulation. This would be a time-consuming and complex task, but this may be the last step after the consideration of other open problems to confirm that the CServ architecture concept can provide accurate-enough internetwork services in a timely fashion to satisfy stringent critical messaging requirements.

Bibliography

[1] L. Roberts, "The Arpanet and computer networks," in *HPW '86 Proceedings of the ACM Conference on the history of personal workstations*, New York, 1986.

[2] Cisco, "The zettabyte era - trends and analysis," 29 May 2013. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html. [Accessed 7 May 2014].

[3] D. E. Comer, Computer Networks and Internets, 5th ed., Upper Saddle River: Pearson Education, 2009.

[4] I. Aldridge, "What is high-frequency trading, afterall?," 8 July 2010. [Online]. Available: http://www.huffingtonpost.com/irene-aldridge/what-is-high-frequency-tr_b_639203.html. [Accessed 1 May 2014].

[5] Federal Energy Regulatory Commission, "Assessment of Demand Response and Advanced Metering," 2008.

[6] D. S. Alberts, J. J. Garstka and F. P. Stein, Network Centric Warfare: Developing and Leveraging Information Superiority, 2nd ed., CCRP Publication Series, 2000.

[7] "World Stats," Miniwatts Marketing Group, November 2015. [Online]. Available: http://www.internetworldstats.com/stats.htm. [Accessed 31 December 2015].

[8] Cisco, "High-performance automated trading network architectures," 2010. [Online]. Available: https://www.cisco.com/web/strategy/docs/finance/c11-600126_wp.pdf. [Accessed 1 May 2014].

[9] A. Haldane, "Patience and finance," 9 September 2010. [Online]. Available: http://www.bis.org/review/r100909e.pdf. [Accessed 1 May 2014].

[10] V. W. S. Chan, *Private communication,* 2014.

[11] J. M. Chapin and V. W. S. Chan, "Ultra high connectivity military networks," *IEEE Military Communications Conference,* pp. 1011-1018, 2010.

[12] Y. Rekhter, T. Li and S. Hares, "A border gateway protocol 4 (BGP-4)," *IETF RFC 4271,* Jan. 2006.

[13] J. Hawkinson and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)," *IETF RFC 1930,* Mar. 1996.

[14] D. McPherson and V. Gill, "BGP MULTI_EXIT_DISC (MED) considerations," *IETF RFC 4451,* Mar. 2006.

[15] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 4th ed., Boston: Pearson Education, 2008.

[16] CCITT, Study Group VII, Draft recommendation X-25, March 1976.

[17] C. Brown and A. Malis, "Multiprotocol Interconnect over Frame Relay," *IETF RFC 2427,* Sep. 1998.

[18] A. Joel, Asynchronous Transfer Mode, IEEE Press, 1993.

[19] E. Rosen, A. Viswanathan and R. Callon, "Multiprotocol Label Switching Architecture," *IETF RFC 3031,* Jan. 2001.

[20] L. Andersson, I. Minei and B. Thomas, "LDP Specification," *IETF RFC 5036,* Oct. 2007.

[21] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," *IETF RFC 3209,* Dec. 2001.

[22] L. Benmohamed, B. Doshi, T. DeSimone and R. Cole, "Inter-domain routing with multi-dimensional QoS requirements," in *IEEE Military Communications Conference*, 2005.

[23] J. Moy, "OSPF Version 2," *IETF RFC 2328,* April 1998.

[24] R. Coltun, D. Ferguson, J. Moy and A. Lindem, "OSPF for IPv6," *IETF RFC 5340,* July 2008.

[25] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," *IETF RFC 1142,* Feb. 1990.

[26] International Organization for Standardization, "Intermediate system to Intermediate system intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473)," *ISO/IEC 10589:2002, Second Edition,* Nov. 2002.

[27] D. Katz, K. Kompella and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2," *IETF RFC 3630,* Sep. 2003.

[28] T. Li and H. Smit, "IS-IS Extensions for Traffic Engineering," *IETF RFC 5305,* Oct. 2008.

[29] J. Postel, "Internet Protocol," *IETF RFC 791,* Sep. 1981.

[30] K. Nichols, S. Blake, F. Baker and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," *IETF RFC 2474,* Dec. 1998.

[31] W. Xia, Y. Wen, C. Heng Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communication Surveys & Tutorials,* vol. 17, no. 1, pp. 27-51, 2015.

[32] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 3, pp. 1617-1634, 2014.

[33] T. Anderson et al., "The NEBULA Future Internet Architecture," in *The Future Internet*, vol. 7858, 2013, pp. 16-26.

[34] J. Chapin and V. Chan, "Architecture concepts for a future heterogeneous, survivable tactical Internet," in *IEEE Military Communications Conference*, 2013.

[35] C. Hornig, "A Standard for the Transmission of IP Datagrams over Ethernet Networks," *IETF RFC 894,* Apr. 1984.

[36] J. Chapin and V. Chan, "Ultra high connectivity military networks," in *IEEE Military Communications Conference*, 2010.

[37] V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications,* vol. 22, no. 5, May 1974.

[38] *Advanced video coding for generic audiovisual services, ITU-T Recommendation H.264,* International Telecommunication Union, 2003.

[39] *Information Technology - Coding of Audio-Visual Objects - Part 10: Advanced Video Coding,* ISO/IEC 14496-10, 2012.

[40] J. Hawkinson and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)," *IETF RFC 1930,* Mar. 1996.

[41] D. Naylor, M. Mukurjee, P. Agyapong, R. Grandi, R. Kang and M. Machado, "XIA: Architecting a More Trustworthy and Evolvable Internet," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 3, pp. 50-57, July 2014.

[42] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 3, pp. 66-73, July 2014.

[43] J. Postel, "Transmission Control Protocol," *IETF RFC 793,* Sep. 1981.

[44] J. Postel, "User Datagram Protocol," *IETF RFC 768,* Aug. 1980.

[45] *IEEE Standard for Ethernet,* IEEE 802.3-2012, 2012.

[46] D. Bertsekas and R. Gallager, Data Networks, 2nd ed., Upper Saddle River: Prentice-Hall, 1992.

[47] Cisco, "Cisco's Space Router Successfully Operates in Orbit," 19 January 2010. [Online]. Available: http://newsroom.cisco.com/press-release-content?articleId=5315645. [Accessed 28 September 2015].

[48] V. Fuller, T. Li, J. Yu and K. Varadhan, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy," *IETF RFC 1519,* September 1993.

[49] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," *IETF RFC 4291,* February 2006.

[50] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik,* vol. 1, pp. 269-271, 1959.

[51] *IEEE Standard for Floating-Point Arithmetic,* IEEE 754-2008, 2008.

[52] N. N. Taleb, The Black Swan: The Impact of the Highly Improbable, New York: Random House, 2010.

[53] L. Smith and L. Ian, "Free Pool of IPv4 Address Space Depleted," Number Resource Organization, 3 February 2011. [Online]. Available: https://www.nro.net/news/ipv4-free-pool-depleted. [Accessed 3 November 2015].

[54] J. Curran, "ARIN IPv4 Free Pool Reaches Zero," American Registry for Internet Numbers, 24 September 2015. [Online]. Available: https://www.arin.net/announcements/2015/20150924.html. [Accessed 3 November 2015].

[55] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *IETF RFC 2460,* Dec. 1998.

[56] P. Mockapetris, "Domain Names - Concepts and Facilities," *IETF RFC 1034,* Nov. 1987.

[57] P. Mockapetris, "Domain Names - Implementation and Specification," *IETF RFC 1035,* Nov. 1987.

[58] T. Narten, G. Huston and L. Roberts, "IPv6 Address Assignment to End Sites," *IETF RFC 6177,* Mar. 2011.

[59] IEEE, "Guidelines for 64-Bit Global Identifier (EUI-64)," *IEEE Standards Association.*

[60] T. Bates, P. Smith and G. Huston, "CIDR Report," 4 November 2015. [Online]. Available: http://www.cidr-report.org/as2.0/. [Accessed 4 November 2015].

[61] Q. Vohra and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space," *IETF RFC 6793,* Dec. 2012.

[62] S. Kawamura and M. Kwashima, "A Recommendation for IPv6 Address Text Representation," *IETF RFC 5952,* Aug. 2010.

[63] C. Partridge and F. Kastenholz, "Technical Criteria for Choosing IP The Next Generation (IPng)," *IETF RFC 1726,* Dec. 1994.

[64] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," in *SIGCOMM '00 Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Stockholm, 2000.

[65] B. Kaliski, "TWIRL and RSA Key Size," 6 May 2003. [Online]. Available: http://www.emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm. [Accessed 5 November 2015].

[66] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory,* vol. 31, no. 4, pp. 469-472, 1985.

[67] Federal Information Processing Standards, "Digital Signature Standard (DSS)," *FIPS PUB 186-4,* Jul. 2013.

[68] E. Barker and A. Roginsky, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," *NIST Special Publication 800-131A, Revision 1 Draft,* Jul. 2015.

[69] B. Davie, A. Charny, J. C. R. Bennett, K. Benson, J. Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)," *IETF RFC 3246,* Mar. 2002.

[70] L. Andersson, I. Minei and B. Thomas, "LDP Specification," *IETF RFC 5036,* Oct. 2007.

[71] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," *IETF RFC 3209,* Dec. 2001.

[72] IEEE, "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifica," *IEEE Standards Association,* IEEE Std 802.11ac-2013, 2013.

[73] J. Mogul and S. Deering, "Path MTU Discovery," *IETF RFC 1191,* Nov. 1990.

[74] J. McCann, S. Deering and J. Mogul, "Path MTU Discovery for IP version 6," *IETF RFC 1981,* Aug. 1996.

[75] K. C. Guan, Cost-Effective Optical Network Architecture - A Joint Optimization of Topology, Switching, Routing and Wavelength Assignment, PhD Dissertation, Massachusetts Institute of Technology, 2007.

[76] Cisco, "Portable Product Sheets – Routing Performance," 3 November 2009. [Online]. Available: http://www.cisco.com/web/partners/downloads/765/tools/quickreference/routerperformance.pdf. [Accessed 3 December 2015].

[77] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,* IEEE 802.11, 2012.

[78] R. Gallager, Stochastic Processes: Theory for Applications, Cambridge: Cambridge University Press, 2013.

[79] R. J. Perlman, "Network layer protocols with Byzantine robustness," PhD dissertation, Massachusetts Institute of Technology, 1988.

[80] F. Cantelli, "Intorno ad un teorema fondamentale della teoria del rischio," *Bollettino dell'Associazione degli Attuari Italiani,* pp. 1-23, 1910.

[81] P. Tchebichef, "Des valeurs moyennes," *Journal de mathématiques pures et appliquées,* vol. 2, no. 12, pp. 177-184, 1867.

[82] M. Sniedovich, "Dijkstra's algorithm revisited: the dynamic programming connexion," *Journal of Control and Cybernetics,* vol. 35, no. 3, pp. 599-620, 2006.

[83] T. Cormen, C. Leiserson, R. Rivest and C. Stein, Introduction to Algorithms, 2nd ed., Cambridge: The MIT Press, 2002.

[84] J. M. Simmons, Optical network design and planning, New York: Springer, 2008.

[85] J. Quirk and R. Saposnik, Introduction to General Equilibrium Theory and Welfare Economics, New York: McGraw-Hill, 1968.

[86] L. G. Valiant, "The Complexity of Enumeration and Reliability Problems," *SIAM J. Comput.,* vol. 8, no. 3, pp. 410-421, 1979.

[87] R. Balakrishnan and K. Ranganathan, A Textbook of Graph Theory, Springer Science & Business Media, 2012.

[88] L. Zhang, Fast Scheduling for Optical Flow Switching, Cambridge: Massachusetts Institute of Technology, SM Thesis, 2010.

[89] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," *IETF RFC 5246,* Aug. 2008.