Increasing Quality of Service Using Transport Layer Coding Over Parallel Heterogeneous Networks

by

Jason M. Cloud

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

June 2016

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
April 13, 2016

Signature redacted

Certified by .....
Muriel Médard
Cecil H. Green Professor in Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Increasing Quality of Service Using Transport Layer Coding Over Parallel Heterogeneous Networks

by

Jason M. Cloud

Submitted to the Department of Electrical Engineering and Computer Science
on April 13, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

A shift in the way the Internet is accessed and the type of traffic generated has been underway for the last decade and will continue into the foreseeable future. Furthermore, these shifts are expected to place considerable strain on the ability of existing transport layer protocols to meet users' quality of service demands. Internet traffic is being generated by an increasing number of inexpensive, low-power, mobile devices resulting in an increased incidence of packet losses, unreliable network connections, and path instability. In addition, the majority of traffic, such as streaming video, is fast becoming more delay sensitive driving changes throughout the network. This thesis explores methods to improve end-to-end performance through the use of network and erasure coding techniques applied at the transport layer. These techniques will help provide path diversity and connection resiliency in the form of multi-path transport, in addition to increasing throughput and decreasing in-order delivery delay in the presence of network disruptions and packet erasures.

The potential gains of a network coded transport layer are first introduced using two protocols: Coded TCP (CTCP) and Multi-Path TCP with Network Coding (MPTCP/NC). These two approaches help illustrate different methods to implement network coding to help overcome packet losses and diversify network connections over multiple paths. Analytic and experimental results are provided that show a network coded transport layer can significantly improve throughput in challenged networks; but more importantly, these results hint at the possible gains for application layer quality of service. Observing the performance of HTTP requests and streaming video, the potential to decrease the in-order delivery delay is highlighted.

Based off of the observations outlined above, the second half of the thesis explores different code constructions that reduce in-order delivery delay in multiple path networks. A generation-based systematic code construction, similar to the one used in CTCP, is modeled and analyzed. Of particular note is the inherent trade-off between increasing the generation size to reduce the probability of decode errors resulting in retransmissions and keeping the generation size small enough so that coding delay is minimized. Furthermore, the trade-off between delay and efficiency is explored to help

quantify the cost of reducing delay so that a user's quality of service constraint is met. Finally, a multiple path streaming code construction that removes the artificial constraints imposed by partitioning packets into generations is explored. Comparisons between the two constructions help highlight the importance of feedback in reducing delay, as well as the advantages and disadvantages of one type of code construction over the other.

Thesis Supervisor: Muriel Médard
Title: Cecil H. Green Professor in Electrical Engineering and Computer Science

To my wife Jenny

and my children Jackson and Elliana

Acknowledgements

The pursuit of my Ph.D at MIT has been one of the most challenging, but rewarding experiences of my life. Both the faculty and my fellow students have helped me push the boundaries of my knowledge past what I thought was possible. I will always be grateful for their patience, dedication, and support over the past several years.

I first would like to express my sincerest gratitude to my advisor Prof. Muriel Médard. Her guidance, support, and expertise has been invaluable to both my professional and personal development. The advice and encouragement she has provided me during my studies, in addition to the freedom to explore new ideas and topics of interest, has helped me grow as a researcher. It has truly been an honor and privilege working with you.

I would like to thank Prof. Vincent Chan, Prof. Douglas Leith, and Dr. Aradhana Narula-Tam for your guidance and advice. I am grateful for their time reading my various papers, helping push the boundaries of my research, and serving on my dissertation committee. Their advice, insight, and suggestions have helped improve both the content of this thesis and the quality of my research.

Next, I would like to thank my friends and officemates at MIT. In particular, I would like to thank Weifei Zeng, Arman Rezaee, and Flavio du Pin Calmon. Their help and support throughout my graduate studies has been instrumental to my success. I would also like to thank Ali ParandehGheibi, Kerim Fouli, Jinfeng Du, Minji Kim, Ulric Ferner, and Dan Whisman for all of the discussions regarding research; and Michael Lewy for helping me navigate Muriel's confusing schedule. In addition, I would like to thank Georgios Angelopoulos, Soheil Feizi, Vitaly Abdrashitov, Ali Makhdoumi, Salmon Salamatian, Ahmad Beirami, Daniel Lucani, Shirley Shi, Dave Adams, Tong Wang, and everyone else that I have had the pleasure of working with over the last several years.

Finally, I would like to thank my family for their love, support, patience, and sacrifice. Most of all, I am deeply indebted to my wife Jenny. Her willingness to commute across the country, support me in the pursuit of my dreams, and her uncon-

ditional love have been my beacon of light. I am also indebted to my son Jack and daughter Ellie for reminding me everyday that life is supposed to be fun and warming my heart every time I look at you. I couldn't have done it without the three of you.

The work in this thesis was partially supported by the Assistant Secretary of Defense (ASD R&E) under contract no. FA8721-05-C-0002, the Air Force Office of Scientific Research (AFOSR) under contract no. FA9550-09-1-0364 and FA9550-14-1-0052, the Army Research Office under contract no. W911NF-10-1-0430, the Office of Navy Research (ONR) under contract no. N00014-13-1-0774, and the National Science Foundation (NSF) under grant DGE-0801525. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

Contents

List of Figures

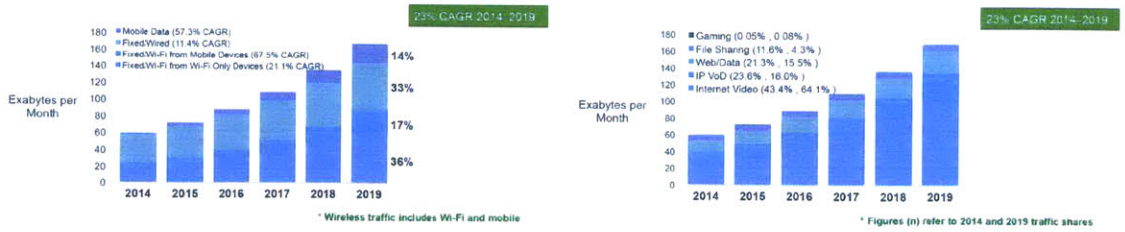
Chapter 1

Introduction

The methods in which end users access the Internet, in addition to the type of content they are primarily consuming, drive exploration of new approaches to improve quality of service (QoS). The need for these new approaches is further amplified by the growth in mobile device use, extending network coverage to rural areas and developing countries, and the explosion of connected devices. While new physical and data link layer technologies are necessary, new techniques to improve end-to-end performance are also required. This thesis focuses on improving end-to-end performance by considering erasure and network coding at the transport layer, which helps increase throughput and decrease in-order delay.

The shift towards wireless Internet access that has been on-going for the last decade and is forecasted to continue into the future is one of the primary motivations for the research contained in this thesis. In 2014, 46% of all IP traffic originated from wireless or mobile devices and this is expected to grow to 67% of all IP traffic by 2019 [1]. This trend, shown in Figure 1-1a, presents a major challenge to the existing transport layer. As more and more traffic is generated by inexpensive, low-power, mobile devices, an increased incidence of packet losses, unreliable network connections, and path instability are expected to place considerable strain on the ability of existing transport layer protocols (i.e., TCP, SCTP, etc.) to meet users' QoS demands.

Furthermore, the traffic that is being produced by an increasing percentage of



(a) Distribution of IP Traffic by Access Type (b) Global IP Traffic by Application Category

Figure 1-1: Wireless and time-sensitive global IP traffic trends projected by Cisco [1]

wireless devices is becoming more delay sensitive. This is illustrated in Figure 1-1b. The figure shows that the amount of delay sensitive traffic, such as Internet video, VoD, VoIP, video-streamed gaming, etc., represents the majority of Internet traffic today and it is expected to range from 80% to 90% of all Internet traffic by 2019 [1]. With an increase in delay sensitive traffic, an increase in user expectations is also occurring. This is shown by [2] where the rate of Internet video viewer abandonment is compared with the startup delay, which is a common technique to overcome the limitations of TCP and provide a high QoS during playback. Unfortunately, an increase in startup delay to help avoid interruptions also increases the rate of viewer dissatisfaction. This is shown Figure 1-2 where the rate of viewer abandonment for Internet video increases significantly after just a three second startup delay.

Existing transport layer protocols, primarily TCP, are unable to cope with the increase in packet loss rates and more stringent QoS demands illustrated by the above trends. Transport layer development traditionally assumed that the lower layers (i.e., the link and physical layers) provided an error-free medium in which to communicate. As a result, packet losses are treated as congestion events and the transport layer uses these events as a signal to decrease the transmission rate. This assumption no longer holds in wireless and mobile environments. Whether communication is over a non-stationary satellite link with a bit error rate (BER) in excess of 10^{-6} (equivalent to a packet loss rate of greater than 1.2% at the transport layer) [4] or a WiFi client experiencing a 5% packet loss rate while connected to a public hotspot [3], congestion events can no longer be considered the primary cause of dropped packets;

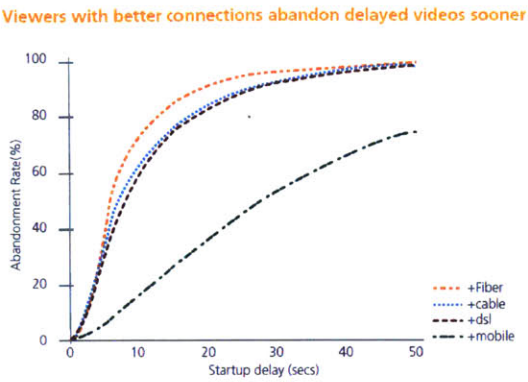


Figure 1-2: Internet video viewer abandonment as a function of startup delay [2]

and continuing to treat packet losses as a signal for congestion, resulting in inadvertent rate throttling, is no longer the appropriate strategy.

Another issue, noted above, that drives the search for alternatives to TCP is the impact of packet losses on delay. TCP provides reliable transport ensuring that the application layer receives an error-free data stream. To provide this reliability, an automatic repeat request (ARQ) mechanism is usually employed where a packet is retransmitted upon notification of its loss. In networks with low packet erasure rates or small round-trip times (RTTs), ARQ is usually sufficient to provide the required QoS guarantees requested by end users. However, frequent retransmissions or large RTTs can create significant interruptions due to head-of-line blocking. This not only increases the possibility of large startup delays that also increases user abandonment as show above, but it also increases the probability of stalling, or buffer under-runs, during playback that also negatively impacts QoS. As a result, new solutions are required to help mitigate the effects of transport layer packet losses on upper layer performance.

Unreliable network connections and path instability are also issues that plague the transport layer in the mobile environment. Whether a network connection is transient or the path between two communicating nodes changes mid-session, current transport layer protocols generally have a difficult time adapting. In addition, most mobile devices have multiple wireless radios (consider a standard smart phone that typically has a 3G/4G radio and a WiFi radio). Current transport protocols are

unable to handle simultaneous communications over multiple heterogeneous networks, and significant gains in both performance and cost reduction are possible though proper diversification. The advent of Multi-Path TCP [5] into consumer products [6] is a first step to providing this flexibility, but improvements in packet management and throughput/delay performance are possible [7].

This thesis will help address these issues through the use of network and erasure coding techniques incorporated into the transport layer. Random linear network coding (RLNC) [8] will be used as the primary workhorse. While RLNC has its limitations, it provides the flexibility to generate codes that help recover from packet losses, provide stability in the presence of different network conditions, and can help to diversify communications across many heterogeneous networks. In addition, RLNC enables implementation of future network coding strategies with minimal impact to network coded transport layers. The remainder of the thesis will explore the benefits of both single-path and multi-path network coded transport layers. This includes the study of methods to improve throughput, as well as methods to decrease in-order delivery delay.

1.1 Related Work

Providing reliable data transport for challenging environments has been a topic of study since the late 1990's [9, 10, 11]. End-to-end solutions typically involve tuning TCP so that high packet loss rates and/or long RTTs do not negatively impact performance. Two prominent versions that perform well in disadvantaged, heterogeneous networks are TCP Cubic [12] and TCP Hybla [13]. Cubic, designed for high speed networks, and Hybla, designed for heterogeneous networks, use a congestion window algorithm that increases the congestion window size ($cwnd$) independently from the RTT. This makes either version useful in environments with high delay. Unlike TCP Cubic, Hybla was developed to also reduce the impact of multiple losses, inappropriate timeouts, and correlated losses. When compared with each other, studies have shown that Hybla performs better than Cubic under high packet loss rates while the

reverse is true under low packet loss rates [14]. Regardless, both experience severe performance degradation under packet loss rates greater than 2%.

In lieu of changes to TCP, performance enhancing proxies (PEPs) are another common approach to increase performance over problem links or networks (e.g., satellite links). Consider a PEP that is located at the gateway to some proprietary network or problem link and a TCP flow that is traversing this network. A TCP session is typically terminated at the PEP, a protocol specifically designed for the network is used for data transport, and a new TCP session is setup on the other side of the network to complete the connection. This implementation poses two issues. First, the cost of implementing a PEP at the network gateway may be high. Second, the termination of TCP sessions at the PEP violates the end-to-end semantics of TCP. One such example is the inability to use IPSEC since the encryption of TCP headers is usually incompatible with PEP deployment [10].

Recently, there has been a resurgence of interest in the use of coding at the transport layer to help overcome issues related to unreliable networks. A more recent protocol, Loss-Tolerant TCP (LT-TCP) [15, 16, 17], combines Reed-Solomon (RS) coding with TCP to overcome packet losses. However, it requires the use of explicit congestion control (ECN) to aid in determining if packet erasures are a result of congestion or poor link characteristics. There are two issues with this proposal. First, the use of ECN to identify congestion is problematic. While ECN is widely implemented, it is rarely turned on or used. In fact, a recent study identified that over 80% of the internet servers tested were not ECN capable [18]. The use of an RS code can also be problematic. LT-TCP separates the packets to be sent into fixed-length blocks and an RS code is used to generate coded packets that are used to help recover from losses. If enough packets (either the original or coded packets) are lost, the sender must regenerate a new set of coded packets and retransmit the entire block. This can result in the degradation of throughput, which can be otherwise avoided. TCP with Network Coding (TCP/NC) [19] uses network coding to overcome this issue, but implementation of the protocol can result in unfairness with other versions of TCP and the code window management can be challenging. TCP/NC implements

network coding below the transport layer to help provide the redundancy needed to avoid duplicate acknowledgements and time-outs that cause TCP to close it's congestion window. Since the redundancy is added on-top of the congestion window, TCP/NC may inadvertently overwhelm the network with too many packets causing congestion and unfairness.

With respect to the topic of multi-path communication using transport layer solutions, a significant amount of research has taken place in recent years. For example, Multi-Path TCP (MPTCP) is a new protocol that has just seen it's first adoption by a commercial product (e.g., Apple's iPhone [6]). MPTCP allows for a single session to be distributed over multiple paths. This adds both redundancy and allows for improved throughput by combining all available network resources. It provides this multi-path support by scheduling packet transmissions and retransmissions through the use of a somewhat complex management scheme. It also provides connection management support that helps to add and delete paths as they become available and unavailable respectively. As paths become available, MPTCP adds TCP sub-flows to the connection, where each sub-flow behaves in the same manner as a standard TCP flow. While the use of TCP ensures fairness with other TCP flows (whether they are or are not part of another MPTCP connection), the performance of TCP over lossy networks (e.g., wireless networks) is known to be poor [20]. As mentioned above, network coding is one possible solution that both reduces the need for a complex management scheme and can help increase the performance of transport layer protocols over both heterogeneous and lossy networks.

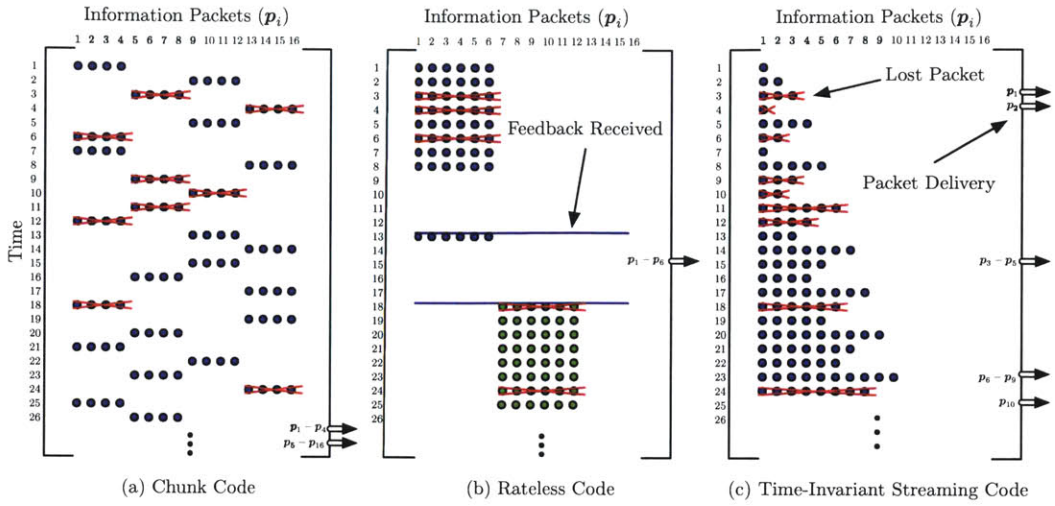
Several suggestions on how to incorporate network coding with MPTCP have been proposed. Gheorgiu et al. [21] propose a protocol called CoMP that uses network coding for multi-path transmission that incorporates only some aspects of TCP. Proposals by [22] and [23] suggests methods that incorporate TCP/NC and adds a multi-path scheduler below the TCP, network coding, and IP layers. Unfortunately, this negates the congestion control benefits of TCP over single paths. Finally, ParandehGheibi et al. [24], and implemented by [25] in OPNET, provide a sub-flow selection control policy for network coded packets over heterogeneous networks that optimizes the

trade-offs between the network usage costs and the Quality of user Experience (QoE) for media-streaming applications.

Each of the protocols and proposals above have generally focused on increasing throughput over unreliable networks, but another important benefit provided by coding at the transport layer is the potential to decrease delivery delay and jitter. This is especially important since there has been an increase in the use of TCP for applications such as video streaming and voice over IP. Without coding, TCP versions such as Cubic, Hybla, Reno, etc. use automatic repeat requests (ARQ) to recover from packet losses. This ARQ mechanism can create head-of-line blocking issues, which can cause unacceptable video playback and interruptions given enough lost packets [26, 27].

A number of groups have approached this problem, in addition to related problems, using various code constructions and techniques. In general, the majority of these works can be summarized by Figure 1-3. The rows and columns of each matrix in the figure indicate the time and the specific information/uncoded packets, $\boldsymbol{p}_i$, that need to be transmitted respectively. The composition of the transmitted packet is shown by the dots in each row, different color dots indicate different generations, horizontal lines show the time when feedback about a specific generation is obtained, and the red crosses show lost packets. Furthermore, the time packets are delivered, in-order, to the client application is shown by the double arrows on the right-hand side of each matrix.

The coding delay of chunked and overlapping chunked codes [28], network coding in time-division duplexing (TDD) channels [29, 30, 31], and network coding in line networks where coding also occurs at intermediate nodes [32] is well understood. In addition, a non-asymptotic analysis of the delay distributions of RLNC [33] and various multicast scenarios [34, 35, 36] have also been investigated. Research looking at the in-order delivery delay in uncoded systems is provided in [37] and [38]; while [39], [40], [41], and [42] consider the in-order delivery delay of non-systematic code schemes. Unfortunately, packet delivery to the upper layers in non-systematic code constructions is dependent on the successful reception of enough coded packets within



(a) Chunk Code (b) Rateless Code (c) Time-Invariant Streaming Code

Figure 1-3: Coding matrices for various schemes assuming an identical loss pattern and a feedback delay of 4 time-slots

each block. Once enough packets have been received, the block can be decoded and all packets contained within the block can be delivered in-order. This results in distinct decode/delivery events that increase jitter and maybe unsuitable for some time sensitive applications such as voice or real-time video. Furthermore, these non-systematic schemes may not be the optimum strategy in networks with a long RTT. Figure 1-3(a) provides an example of a chunked code similar to those used in [28], while the majority of the other cited work uses variations of the scheme shown in Figure 1-3(b).

The research closest to the work presented in this thesis is [43], [44], [45], [46], [47], and [48]. Bounds on the expected in-order delay and a study of the rate/delay trade-offs using a time-invariant coding scheme is provided in [43] and [45] where feedback is assumed to be instantaneous, provided in a block-wise manner, or not available at all. A generalized example of their coding scheme is shown in Figure 1-3(c). While their analysis provides insight into the benefits of coding for streaming applications, their model is similar to a half-duplex communication channel where the sender transmits a finite block of information and then waits for the receiver to send feedback. Unfortunately, it is unclear if their analysis can be extended to full-duplex

channels or models where feedback does not provide complete information about the receiver's state-space.

ParandehGheibi et al. [46, 47] approached this problem using a block, or generation, based network code for streaming applications in peer-to-peer (P2P) systems. Their work focused on determining the minimum number of coded packets that must be buffered by the receiver before playback can begin while ensuring no interruptions occur before the end of the file is reached. While their approach is significant for ensuring a high quality of user experience for streaming applications that can buffer packets before playback begins, it provides little insight into the behavior of individual packet deliveries to higher layers in applications where delay and jitter is a concern. Finally, the work in [48] considers the in-order delivery delay of online network coding where feedback determines the source packets used to generate coded packets. However, they only provide experimental results and do not attempt an analysis.

A systematic network code (i.e., uncoded packets are first sent to a receiver followed immediately by linear combinations of those same packets) has the potential to mitigate the problems mentioned above, but little research has been done in terms of characterizing the delay of such schemes. [49] proposes such a scheme, but only studies the receiver's ability to decode the required packets. [50] and [51] both propose systematic network coding schemes that XOR packets together and investigate the delay-throughput trade-offs; but the use of a small field size when generating coded packets ignores the possibility of receiving multiple network coded packets consisting of linear combinations of the same uncoded packets. It is expected that increasing the field size used to generate coded packets will eliminate this problem, while also helping to reduce the in-order delivery delay of transmitted packets.

1.2 Thesis Contributions and Outline

Design of the transport layer is a complex problem. From the study of how congestion control affects overall network stability to the impacts transport layer implementations have on application layer performance, the breadth of topics that should be

considered is large and cannot all be covered in a single thesis. Therefore, this thesis will focus on applications of network coding for reliable transport. This includes both a study of the throughput gains afforded by network coding, methods to extend single-path transport layers to multi-path scenarios, and an evaluation of code constructions that reduce overall in-order delivery delay.

The motivations for the work presented throughout the thesis is provided in Chapter 2 where an overview of a prototype coded transport layer called Coded TCP (CTCP) [3, 52] is provided. CTCP combines error-correction coding, in terms of a systematic network code, and congestion control to protect against both packet losses and congestion collapse. Both the code construction and congestion control are described in detail, and extensive experimental results are discussed. These results show significant gains over a wide range of network conditions, while remaining friendly with other versions of TCP. However, the most significant contribution of this chapter is the identification of the gains network coding has on the in-order delivery delay for packets in transport. This is shown using experimental measurements of application layer performance. In particular, the completion time of HTTP requests and the number of buffer under-runs experienced during the playback of a streamed video. The gains shown cannot be fully explained solely by the throughput increases provided by both the modifications to congestion control and employment of network coding. Rather, these gains can only be described by the ability of network coding to quickly recover from packet losses without incurring the added delay resulting from the retransmission of lost packets.

Before exploring the delay gains in greater detail, Chapter 3 discusses the application of a coded transport layer for multi-path scenarios. First, experimental results measuring a potential multi-path scenario is provided to help guide future multi-path transport layer designs. In particular, these measurements collected simultaneous packet traces over three parallel heterogeneous networks; and they help illustrate the inherent challenges and potential benefits of communicating over each concurrently. A multi-path protocol called Multi-Path TCP with Network Coding (MPTCP/NC) [7] is then proposed that is intended to effectively utilize all available paths. Unlike

CTCP, MPTCP/NC does not replace the transport layer. Instead, it incapsulates the existing one by inserting two coding layers, or shims. A coding layer placed above TCP adds connection resiliency and path diversity, while a coding layer placed below TCP is used to insert redundancy to help overcome packet losses. While not a practical approach for a variety of reasons outlined later, this setup is used to help provide an analysis of the attainable throughput. Numerical results using the experimental measurements are presented showing the benefits of using network coding for multi-path transport. A discussion is also provided concerning extensions of CTCP for multi-path transport outlining the benefits of replacing the transport layer rather than supplementing it.

The focus of Chapter 4 is on quantifying the non-throughput gains shown in Chapter 2. Specifically, the in-order delivery delay of packets transported over a multi-path network using a systematic generation-based coding scheme similar to the one employed by CTCP is modeled and analyzed [53]. Packets are partitioned into generations of size k packets and systematically transmitted over an available network with redundancy added on a generation-by-generation basis. Feedback is then provided to the source indicating whether or not the sink was able to decode the generation. If it was not, the source retransmits additional degrees of freedom ($dofs$) to ensure the sink's ability to decode each and every generation. The analysis and numerical results help illustrate several important trade-offs. First, the proper selection of k is necessary to minimize the in-order delivery delay for a given code rate. Increasing k results in a decreased decode error probability which reduces the number of retransmissions that are necessary. Unfortunately, a large k also results in large coding delays that increase the overall in-order delivery delay. On the other hand, a small k decreases the coding delay; but it also increases the probability that a generation cannot be decoded resulting in an increased number of retransmissions. This increases the overall delay due to head-of-line blocking created by previously transmitted generations. Therefore, the analysis helps find the proper generation size that minimizes the in-order delivery delay for a given set of networks. Second, decreasing the in-order delivery delay has a cost. This cost is quantified in terms of

the efficiency, or the ratio between the number of packets to be transferred and the number of packets received by the sink. As one might expect, decreasing the delay also decreases the efficiency. The analysis and numerical results help show how much the efficiency must be reduced to meet a given users' QoS constraints.

One of the benefits of the generation-based scheme discussed in Chapter 4 is the ease with which recoding can occur lower within the network stack or the network itself. By coding on a generation-by-generation basis, simple approaches such as prepending a coding coefficient vector to each transmitted coded packet allows network nodes to easily recode whenever it is deemed useful. However, partitioning packets into generations limits the usefulness of a coded packet (e.g., a coded packet can only help recover from a packet loss within a single generation). Chapter 5 shows the benefits of removing this constraint. A streaming code construction is presented for multi-path transmission and an analysis of the in-order delivery delay is provided. Numerical and simulated results are provided that show this streaming code construction can achieve lower in-order delay than the generation-based coding scheme discussed earlier. Furthermore, these results help show the importance of feedback by providing an unfair comparison of an open-loop streaming code with a closed-loop generation-based code.

Finally, Chapter 6 summarizes the contributions of this thesis. In particular, a number of issues that must be considered when implementing any of the protocols or coding schemes described earlier are discussed and a number of directions for future exploration are provided. The discussion provided within this chapter is not meant to be a comprehensive list of all of the benefits and drawbacks of the items outlined in this thesis. Rather, it is meant to provide both intuition and "rules of thumb" that should be taken into account.

Chapter 2

Transport Layer Coding

2.1 Introduction

Existing transport layer protocols, such as TCP, generally provide good performance over well behaved networks where packet erasures are a result of queue overflow, round-trip times (RTT) are small, etc. However, these solutions rarely have the ability to operate over the wide range of conditions that currently exist today without extensive tuning. Whether a client is operating over a network with or without packet erasures while experiencing low or high round-trip times (RTT), the transport layer should adapt to maximize the client's performance.

The widespread adoption of wireless technologies to provide connectivity at the network edge is one motivator for addressing this problem. Both interference and poor channels in unlicensed and white-space bands are major contributors to the increase in packet erasures at the transport layer. While new physical and link layer technologies are helping to alleviate this problem, finding a solution at the transport layer helps to ensure backward compatibility with legacy equipment. This is especially important in networks that employ 802.11 wireless links since an estimated 1.2 billion 802.11 devices have been shipped to date [54]. Replacing these devices so that new physical and link layer technologies can be incorporated is largely impractical due to the costs involved.

Satellite networks provide a second motivator for this problem. The combination

of high packet loss rates (PER) and long RTT's seriously degrades the performance of existing transport layer solutions (namely TCP). A variety of solutions have been proposed over the years to help overcome these challenges. These range from modifications to TCP's congestion control algorithm to implementing performance enhancing proxies (PEPs). Each solution usually has its own drawbacks. In the case of modified TCP protocols, adoption is prevented due to the specialized nature of the protocol and issues related to fairness with other TCP variants. In the case of PEPs, increased hardware costs and issues regarding end-to-end semantics is an issue.

Transport layer coding is one method to help resolve these issues. The use of random linear coding at the transport layer aids in overcoming packet erasures without the need for retransmissions. As a result, transport layer coding can help increase throughput, decrease delay and jitter, etc. This chapter will give an overview of a transport layer coding approach called Coded TCP (CTCP) [3], and help motivate the remainder of the thesis.

A Note About the Contributors for the Content Contained in this Chapter

A number of people were instrumental in the development of CTCP, and the content contained in this chapter would not be possible without their contributions. Early contributions to the protocol's development were made by Minji Kim, Ali ParandehGheibi, and Leonardo Urbina, while Douglas Leith and myself focused on the later stages of development. The results presented within this chapter should be considered joint work by the people mentioned above.

2.2 Goals of a Coded Transport Layer

The transport layer, and in particular TCP, plays a critical role in contemporary networks. It provides a number of critical services, but chief among them are the services that provide reliability, flow control, and congestion avoidance. However, the

implementation of these three services often assume a well behaved network where packet erasures are due to queue overflows (i.e., packet erasures are a direct result of congestion) and round-trip times are relatively small. Networks where neither of these conditions are met typically experience degraded performance as a direct result of the transport layer's inability to adapt to the different environments.

When changing the transport layer in an existing network, the ultimate goal is to "do no harm." In other words, any changes should not have a noticeable impact on other services within the network or require network-wide changes. Therefore any coding solution must be transparent to the network as a whole. With this in mind, any protocol that incorporates coding should meet, at a minimum, the following goals with regard to backward compatibility:

- Require no changes or updates to either lower or upper layers upon implementation; and
- Have no impact on the performance of parallel network flows, whether coded or non-coded.

Meeting the first goal requires that no changes are made to the underlying transport layer packet structure. As an example, the TCP header and payload size should be consistent whether or not coding is used. Meeting the second goal forces coded flows to be fair with other coded flows and friendly with non-coded flows. This goal also ensures that additional overhead created as a result of a change only effects the modified flow. Therefore, any flow that incorporates coding must ensure high efficiency is maintained.

Any changes that are made should also be value-added in the sense that the transport layer should provided better quality of service to higher layers resulting in the following additional goals:

- Provide high throughput under both independent and identically distributed (i.i.d.) and correlated packet losses. These losses may caused by:
  - Channels with high interference or fading;

  - Hidden terminals in wireless networks; and
  - Any additional known or unknown cause of packet loss not related to congestion.

- Operate under a wide range of network conditions including a wide range of round-trip times and packet erasure rates.
- Provide a better quality of service for higher layer applications than current transport layer protocols.

Coding is ideally suited for meeting the goals in the list above. If enough redundancy is inserted into a transport layer packet stream, any number of packet erasures can be corrected. The challenge presents itself when trying to determine how much redundancy should be inserted and what effects this redundancy will have higher in the network stack. While a stream with a large amount of redundancy is going to be able to correct a large number of erasures, it will also have high overhead that limits goodput. The other negative effect coding can have on the end user's QoS is the introduction of coding delays. Any coding approach must ensure that the end user experiences delays no worse than they would experience when using an unmodified transport layer. Furthermore, any developed solution must be adaptable to the environment in which it is located, meaning that the coding approach needs to be flexible enough to be modified on-the-fly. This also extends to any other changes to the transport layer that need to be made. The remainder of this chapter, and this thesis as a whole, will help quantify these trade-offs and considerations.

2.3 An Overview of Coded TCP (CTCP)

Coded TCP (CTCP) [3] combines error-correction coding, in terms of a systematic network code, and congestion control to protect against both packet losses and congestion collapse. Unlike previous approaches such as TCP with Network Coding (TCP/NC) proposed by Sundararajan et al. [19], CTCP is intended to completely replace TCP rather than insert a "shim" to spoof TCP into behaving a certain way.

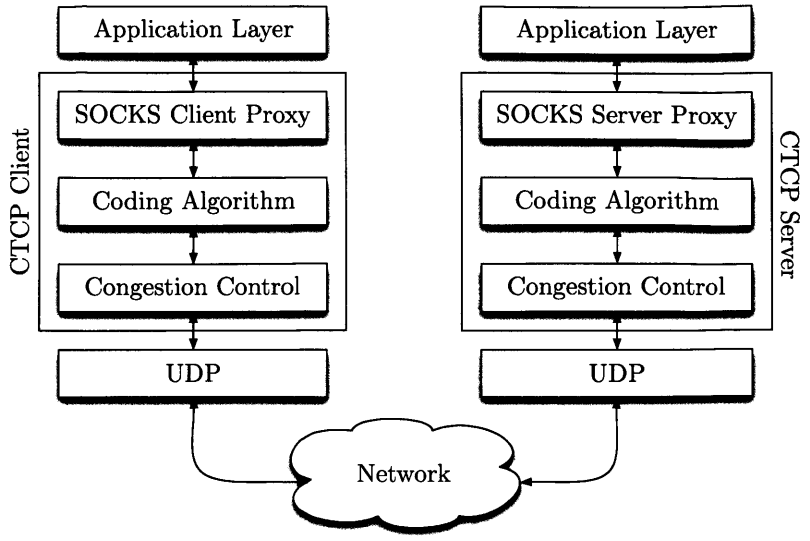


Figure 2-1: CTCP Functional Diagram

This provides CTCP with flexibility in its design and eliminates the need to be subservient to poorly behaving protocols.

A key feature of CTCP is that coding and congestion control are logically decoupled. This removes all dependencies with respect to the specific coding scheme employed. As a result, the congestion control is only concerned with scheduling the transmission of packets, while the coding scheme determines the contents of the packets transmitted (e.g., coded or uncoded).

CTCP is implemented in user space, rather than within the kernel, to help simplify development. A diagram of the major components of the protocol is provided in Figure 2-1. A SOCKS proxy is used to interface with applications. The bit stream from the application layer is first packetized and partitioned into generations. Redundancy is added to each generation using an estimate of the network's packet erasure rate. Each generation is then scheduled for transmission by the congestion control algorithm, and transmitted over the network. An existing transport layer protocol, UDP, is used to do this in order to avoid complications with lower layers and middle-boxes. The following sections provide an in-depth explanation of the two major components of CTCP: the coding algorithm and congestion control.

2.3.1 Code Implementation

Random linear network coding (RLNC) [8] is used to provide packet erasure protection. The gains provided by network coding are twofold: coded packets are used to provide forward error correction; and coded packets are used to simplify feedback and retransmissions, should they be needed.

Consider the transfer of information packets $\boldsymbol{p}_i$, $i = \{1, \ldots, N\}$, between a server and client. These information packets are first partitioned into coding generations $\boldsymbol{G}_j = \left\{\boldsymbol{p}_{(j-1)k+1}, \cdots, \boldsymbol{p}_{\min(jk,N)}\right\}$, $j \in [1, \lceil N/k \rceil]$, of size k packets as they are made available to the CTCP server. Once enough information packets are available for the encoder to fill a generation, random linear combinations of these packets produce coded packets $\boldsymbol{c}_{j,m}$, $m \in [1, k/c - k]$ where $c \in (0, 1]$ is the code rate, i.e.,

$$\boldsymbol{c}_{j,m} = \sum_{x=(j-1)k+1}^{\min(jk,N)} \alpha_{j,m,x} \boldsymbol{p}_x. \tag{2.1}$$

Every coding coefficient $\alpha_{j,m,x} \in \mathbb{F}_{2^8}$ is chosen at random and the information packets are treated as vectors in $\mathbb{F}_{2^8}$. These coded packets are then appended to the end of the generation and transmitted over the network. Once this process completes for $\boldsymbol{G}_j$, the coding window slides to the next generation $\boldsymbol{G}_{j+1}$ and the process repeats without waiting for feedback.

An example of the coding matrix for two generations of size $k = 3$ and code rate $c = 3/4$ is shown in Figure 2-2. The first three transmitted packets are $\boldsymbol{p}_1, \boldsymbol{p}_2$, and $\boldsymbol{p}_3$, and the fourth transmitted packet is $\boldsymbol{c}_{1,1} = \alpha_{1,1,1}\boldsymbol{p}_1 + \alpha_{1,1,2}\boldsymbol{p}_2 + \alpha_{1,1,3}\boldsymbol{p}_3$. Once this first generation has been sent, the second generation consisting of packets $\boldsymbol{p}_4, \boldsymbol{p}_5$, and $\boldsymbol{p}_6$ are transmitted along with the coded packet $\boldsymbol{c}_{2,1} = \alpha_{2,1,4}\boldsymbol{p}_1 + \alpha_{2,1,5}\boldsymbol{p}_5 + \alpha_{2,1,6}\boldsymbol{p}_6$.

After the client has received at least k packets (or degrees of freedom ($dofs$)) from a generation, it is able to decode it (i.e., the client's coding matrix must be full rank for it to be able to decode a generation). If it receives less than k $dofs$, feedback is sent to the server requesting additional $dofs$ from the generation to be transmitted. This process continues until every generation has been successfully decoded.

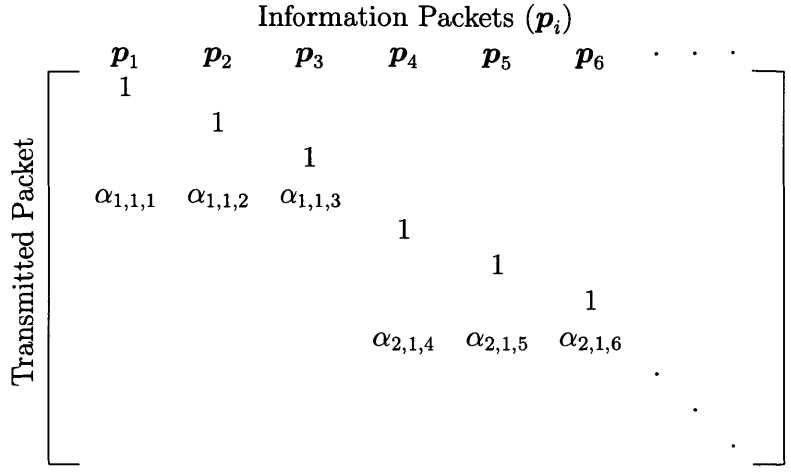


Figure 2-2: Example of the coding matrix used in CTCP when $k = 3$ and $c = 3/4$. The columns represent the information packets that need to be sent, and the rows represent the composition of the packet actually transmitted.

The number of coded packets appended to each generation is dynamically chosen based on an estimate of the packet erasure rate ϵ obtained by the server through the use of feedback. The generation size is statically chosen to be $k = 32$ packets in the results presented throughout this chapter. In truth, the generation size should also be dynamic based on the user's quality of service requirements. However, the choice of k and c is not straight-forward. Incorrectly choosing these two parameters can lead to extremely poor performance, while selecting the correct k and c can greatly improve performance. This will be extensively studied in Chapter 4.

2.3.2 Congestion Control Modifications

A major contributor to CTCP's observed gains, shown later in the chapter, is the congestion window management. Traditional TCP's additive increase, multiplicative decrease (AIMD) congestion control increases the TCP sender's congestion window size *cwnd* by α packets per round-trip time, and it multiplicatively decreases *cwnd* by a back-off factor β on detecting a packet loss. The usual values are $\alpha = 1$ when appropriate byte counting is used, and $\beta = 0.5$. On lossy links, repeated back-offs in response to losses due to noise rather than queue overflow can prevent *cwnd* from

increasing to fill the available link capacity. This behavior is well known and is one of the reasons for TCP's notoriously poor performance over wireless and other error prone links. For example, *cwnd* scales as $\sqrt{3/2\epsilon}$ in [55], where ϵ is the packet erasure rate.

Unfortunately, packet loss is not a reliable indicator of network congestion. An option might be to use delay, rather than loss, as the indicator of congestion. TCP Vegas [56] is an example of a protocol that does this. However, this raises many new issues and purely delay-based congestion control approaches have not been widely adopted in the internet despite being the subject of extensive study. Another option is to use some form of lower layer signaling, such as explicit congestion notification (ECN). Unfortunately, this generally requires network-wide changes. These two considerations motivate the use of hybrid approaches, similar to Compound TCP [57], that make use of both loss and delay information.

The congestion control algorithm in CTCP uses a modified AIMD approach. The congestion window is increased in a similar fashion as H-TCP [58], and it is decreased using a modified multiplicative back-off factor

$$\beta = \frac{RTT_{min}}{RTT}. \tag{2.2}$$

In the above equation, RTT_{min} is the path round-trip propagation delay and RTT is the round-trip time of the last successfully received packet. RTT_{min} is typically estimated as the lowest per packet RTT observed during the lifetime of a connection.

This is similar to the approach considered in [58], which uses $\beta = {}^{RTT_{min}}/_{RTT_{max}}$ with the aim of making TCP throughput performance less sensitive to the level of queue provisioning. On links with only queue overflow losses, equation (2.2) reduces to the approach in [58] since $RTT = RTT_{max}$ (i.e., the link queue is full) when a packet erasure occurs. Assuming that there is a single flow on a link with capacity B at the time of queue overflow, the RTT of the last successfully received packet should be $RTT = 2RTT_{min}$ if a standard bandwidth-delay product worth of queue provisioning is provided (i.e., $RTT \cdot cwnd = 2B$). After applying the back-off factor

according to (2.2), the throughput becomes $\beta \cdot {}^{cwnd}/_{RTT_{min}} = B$ allowing the link queue to empty while maintaining full throughput.

Now assume that the link queue is empty, but a packet erasure is observed. The round-trip time, RTT, of the last successfully received packet in (2.2) should be equal to RTT_{min} (i.e., $RTT = RTT_{min}$) resulting in $\beta = 1$. Therefore, *cwnd* is not decreased, and *cwnd* is able to grow despite the presence of packet loss. Once the link starts to experience queueing delay (i.e., $RTT > RTT_{min}$) which makes $\beta < 1$, *cwnd* will be decreased upon packet loss in an effort to avoid congestion.

2.4 Performance of CTCP Over Emulated Networks

In order to accurately measure CTCP's gains, a simple network was setup and a number of experiments were conducted. This section will provide the results of these experiments and verify that the goals outlined in Section 2.2 were met. Measurements were taken of CTCP's efficiency, fairness, and friendliness, over a large range of emulated network conditions. Furthermore, application layer performance was measured to help illustrate the benefits both the congestion control algorithm and coding have on a user's quality of service.

2.4.1 Experiment and Testbed Setup

An emulated network was setup on a testbed that consists of three commodity servers (Dell Poweredge 850, 3GHz Xeon, Intel 82571EB Gigabit NIC) connected via a router and gigabit switches. Figure 2-3 shows the testbed's configuration. Both the server and client machines were running a Linux 2.6.32.27 kernel, while the router was running FreeBSD 4.11 and `ipfw-dummynet`.

The router was configured with various propagation delays d, packet erasure rates ϵ, queue sizes q and link rates b to emulate a range of network conditions. As indicated in Figure 2-3, packet losses in `dummynet` occur before the rate constraint. This was done so that the bottleneck link capacity b was not artificially reduced. Unless otherwise stated, appropriate byte counting was enabled for standard TCP and every

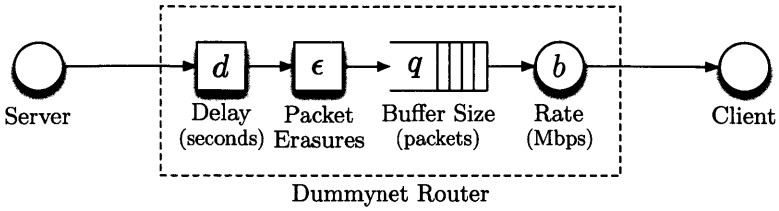


Figure 2-3: Emulated Network Testbed Setup

experiment was run for at least 300 seconds. Data traffic was generated using `rsync` (version 3.0.4), HTTP traffic was generated using `apache2` (version 2.2.8) and `wget` (version 1.10.2), and video traffic was generated using `vlc` at both the server and client (version 0.8.6e as server, version 2.0.4 as client).

As was mentioned earlier, CTCP was implemented in user space as a forward proxy located on the client and a reverse proxy located on the server. The client's request is first directed to the local forward proxy and transmitted to the reverse proxy via CTCP. The reverse proxy then sends the request to the appropriate port on the server. The server's response follows the reverse process. The proxies support the SOCKS protocol and standard tools allow traffic to be transparently redirected via the proxies. In the tests shown below, `proxychains` (version 3.1) was used for this purpose.

2.4.2 Throughput and Efficiency

Experimental measurements showing the efficiency η for CTCP and various other TCP versions are shown in Figure 2-4 where η is defined as:

$$\eta = \frac{\text{Goodput}}{\text{Link Capacity}}. \tag{2.3}$$

Figure 2-4a shows the measured efficiency versus the packet erasure probability ϵ for a 25 Mbps link with a 25 ms RTT, and one bandwidth-delay product of buffering. Baseline data is shown for standard TCP (i.e. TCP SACK/Reno), Cubic TCP (current default on most Linux distributions), H-TCP, TCP Westwood, TCP Veno, and

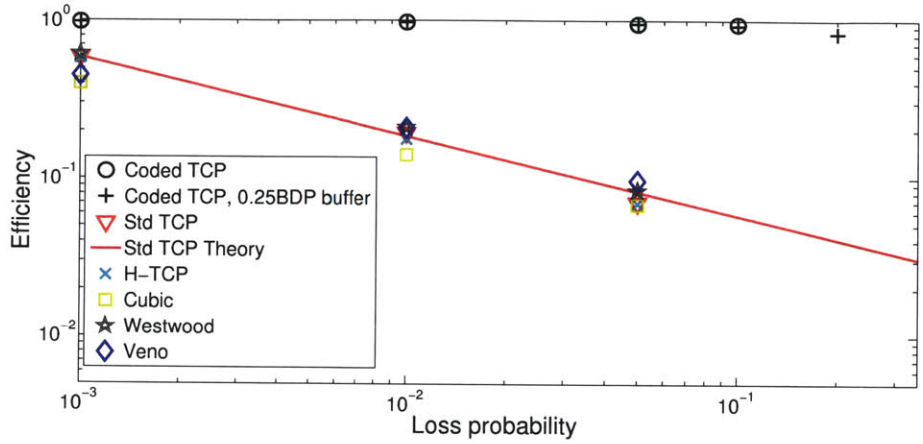
the estimate $\sqrt{3/2\epsilon}$ packets per RTT predicted by the popular Padhye model [55]. Note that the Padhye model provides a good estimate for the performance of standard TCP, in addition to Cubic TCP, H-TCP, TCP Westwood, and TCP Veno. This is as expected since the link bandwidth-delay product of 52 packets lies in the regime where these TCP variants seek to ensure backward compatibility with standard TCP. The figure also shows that standard TCP's achieved goodput decreases rapidly with increasing loss rate. With a packet erasure rate of just 1%, the goodput is only 20% of the link capacity. This feature of standard TCP is well known. However, the efficiency measurements for CTCP, shown in Figure 2-4a and Figure 2-4b, provide evidence that the goodput is $>$ 96% of link capacity for a loss rate of 1%. This is roughly a five-fold increase in goodput compared to standard TCP.

Figure 2-4b shows that CTCP maintains a high efficiency over a large range of link rates, RTT's, and erasure rates. In addition, a theoretical upper bound on the efficiency is shown where it is calculated using the following formula:

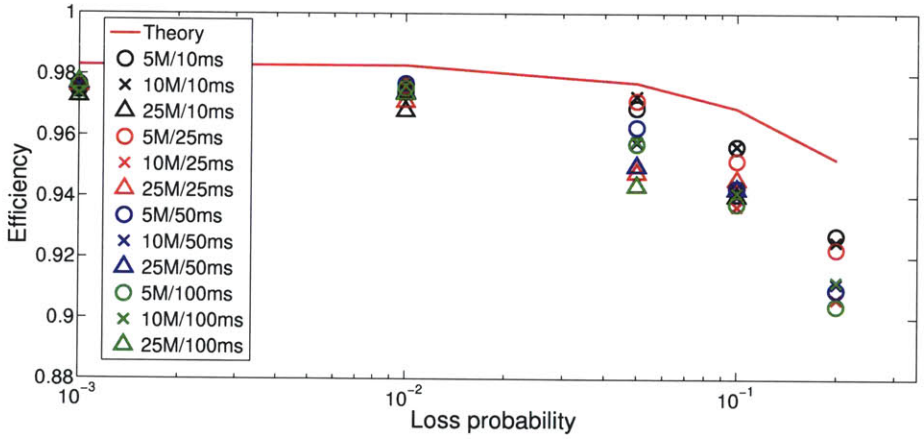
$$\eta = \frac{1}{k}\sum_{i=0}^{n-1}(n-i)\binom{n}{i}\epsilon^i(1-\epsilon)^{k-i}. \tag{2.4}$$

In the above equation, the generation size is $k = 32$, ϵ is the packet erasure probability, and the number of forward-transmitted coded packets transmitted with each generation is $n = \lfloor k/(1-\epsilon) \rfloor - k$ (i.e., the code rate $c = k/(n+k)$). The value determined by (2.4) is the mean number of forward-transmitted coded packets that are unnecessary when there are fewer than n erasures.

Both Figures 2-4a and 2-4b provide evidence that CTCP's efficiency is largely insensitive to both the link rate and the RTT. This is confirmed when considering networks similar to those that traverse satellites where the RTT is in excess of 250 ms. Figure 2-5 shows the goodput over a range of packet erasure rates ϵ and $RTT \geq 500$ ms on a link with a capacity of 10 Mbps. CTCP is able to maintain a large throughput for erasure rates as high as 20%. In fact, measurements indicate that CTCP provides a gain of approximately 21 times that of TCP Hybla [13], a TCP version specifically designed for wireless links, for erasure rates of 20%.

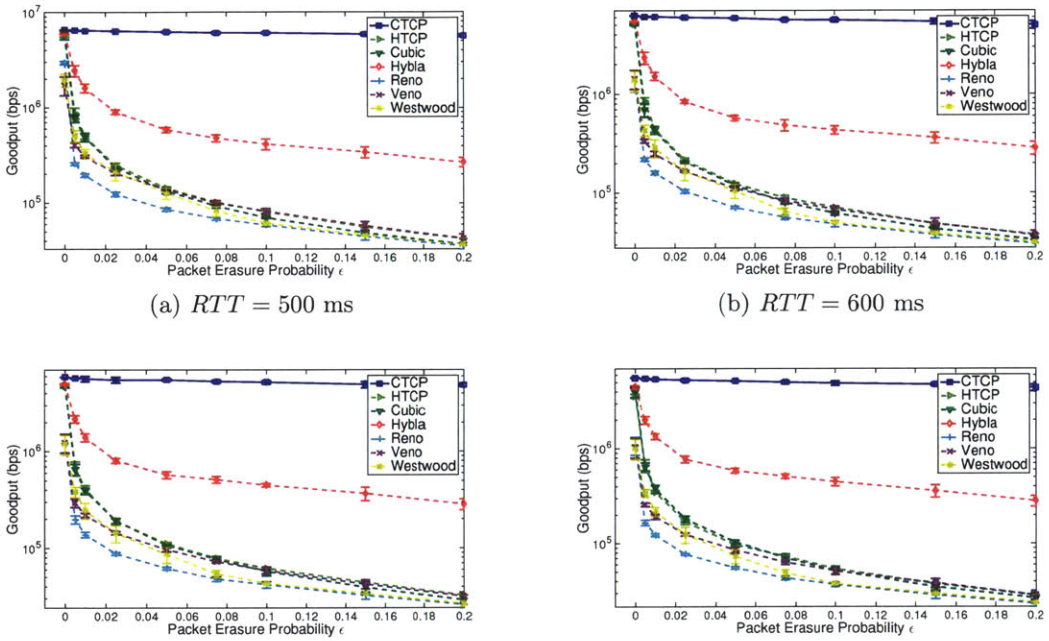


(a) Link 25 Mbps, RTT 20 ms



(b) CTCP

Figure 2-4: Measurements of goodput efficiency against packet loss rate, link rate and RTT taken from [3]. The Theory curve in Figure 2-4a is generated using Equation 2.4.



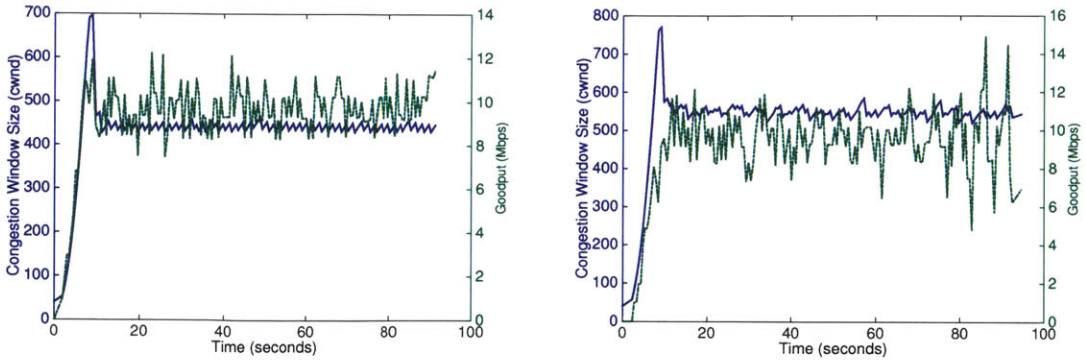
(a) $RTT = 500$ ms

(b) $RTT = 600$ ms

(c) $RTT = 700$ ms

(d) $RTT = 800$ ms

Figure 2-5: Goodput comparison of CTCP and other TCP variants with varying packet erasure probability ϵ and $RTT \geq 500$ over a link with a rate of 10 Mbps. The error bars show plus and minus one standard deviation.



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.2$

Figure 2-6: Congestion window size (*cwnd*) and goodput trace over a 10 Mbps link with a *RTT* of 500 ms. The dotted line shows a 3 second moving average of the goodput and the solid line shows *cwnd*. The mean goodput in (a) and (b) are 9.19 Mbps and 8.92 Mbps respectively.

While the above results provide evidence of high efficiency and large throughput gains, parsing the benefits due to the congestion control modifications from the benefits due to coding is difficult. Figure 2-6 provides a trace of CTCP's congestion window size and goodput over a 10 Mbps link with a *RTT* of 500 ms. Figure 2-6(a) shows CTCP's performance when $\epsilon = 0.05$, and Figure 2-6(b) shows CTCP's performance when $\epsilon = 0.2$. These figures show that *cwnd* remains fairly constant in steady-state enabling CTCP to maintain high throughput regardless of the fact that coding is being used to correct for erasures. This suggests that the majority of the gains in goodput are a direct result of the congestion window modifications. However, the gains due to coding are more subtle and present themselves differently. The "smoothness" of the goodput curves, or lack of periods with low goodput, suggest that coding is helping to reduce end-to-end delay. While these traces do not provide definitive proof that this is the case, Section 2.4.4 will provide additional experimental evidence and Chapter 4 will explore this idea in much greater detail.

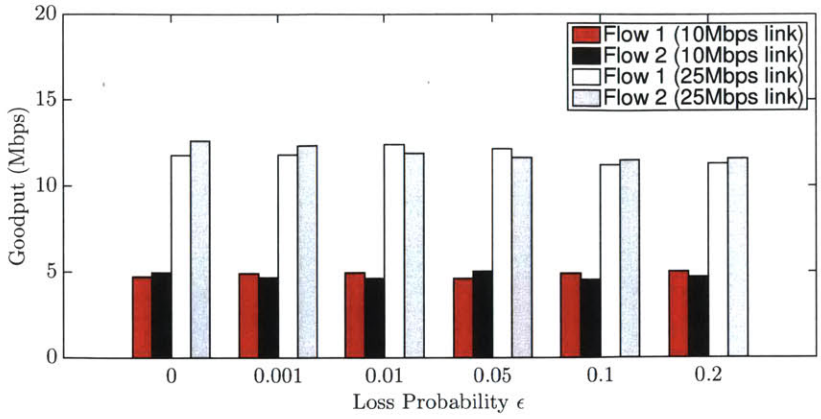


Figure 2-7: Goodput for two CTCP flows sharing a loss-free link

2.4.3 Fairness and Friendliness

A key measure of transport layer performance is the behavior of a flow managed by a given transport layer protocol when it has to compete for resources with other flows. Ideally, each flow obtains its fair share of network capacity. When two or more flows managed by a specific transport layer protocol are competing against each other, it is referred to as fairness. If flows managed by different transport layer protocols are competing against each other, it is referred to as friendliness. Experiments measuring both are provided within this section showing that CTCP behaves well with other CTCP flows, as well as with standard TCP flows.

Fairness is measured by comparing the goodput of two CTCP flows competing for resources on a single link. This is shown in Figure 2-7 for two links as the packet erasure probability is increased. In each experiment, both CTCP flows achieve similar goodput indicating that CTCP exhibits fairness.

Figure 2-8 shows the fraction of link capacity obtained by a single CTCP flow and a single standard TCP flow competing over bandwidth on various error-free links. In this scenario, the only packet losses that occur are due to queue overflows as a result of congestion. The figure demonstrates that CTCP is also friendly with standard TCP over a wide range of RTTs and link rates.

Finally, Figure 2-9 presents goodput data intended to show friendliness between

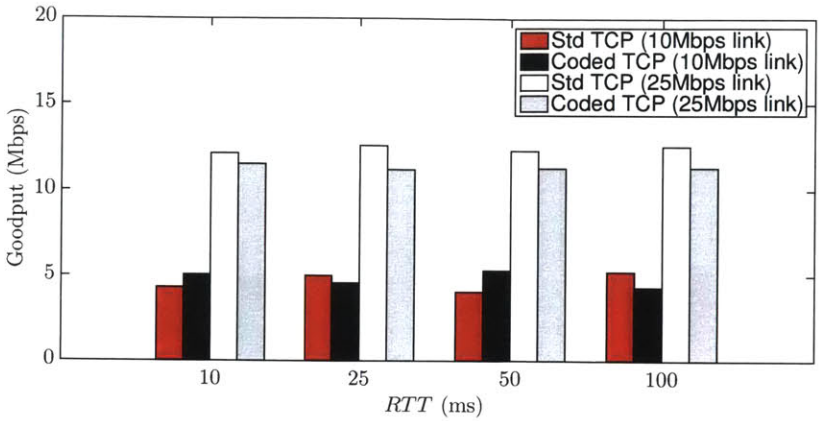
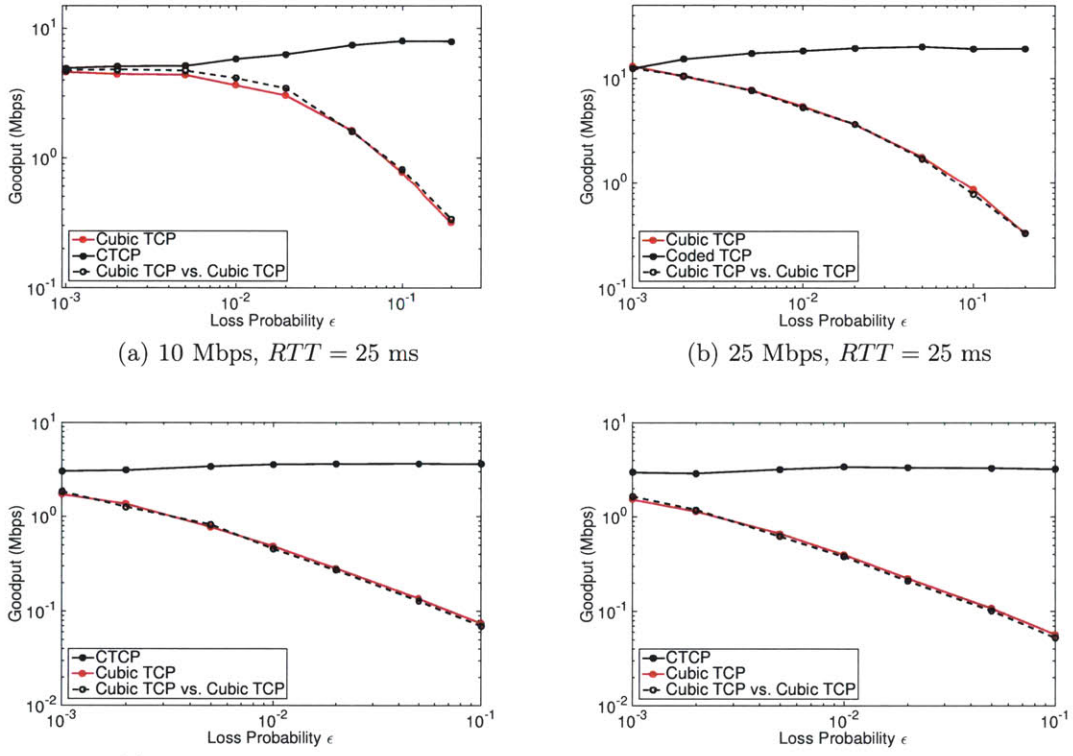


Figure 2-8: Goodput for a standard TCP and a CTCP flow sharing a loss-free link

one CTCP flow and one standard TCP flow sharing a link with packet erasures. Each figure represents two separate experiments. The first experiment is indicated by the single dashed curve, which shows the goodput of a standard TCP flow sharing the same link as another standard TCP flow. Note that the goodput of only one standard TCP flow is shown since both flows achieved nearly identical goodput. The second experiment is indicated by the solid lines which show the goodputs achieved by a single CTCP flow sharing the same link with a single standard TCP flow. The measure of friendliness is how closely the goodput of the standard TCP flows match between the two experiments. In each of the cases shown in Figure 2-9, the dotted line closely matches the red solid line (i.e., the standard TCP goodputs match between the two experiments). This indicates that CTCP is sharing the link capacity fairly when packet erasures are introduced.

Note the goodput shown for the second experiment where a single CTCP flow is sharing a link with a single standard TCP flow. At low loss rates, both CTCP and standard TCP obtain similar goodputs. However as the loss rate increases, the goodput of standard TCP rapidly decreases (as already observed in Figures 2-4a and 2-5). Because standard TCP is misinterpreting packet losses as congestion, it relinquishes link capacity unnecessarily. This leads to CTCP's goodput increasing with the packet loss rate.



(a) 10 Mbps, $RTT = 25$ ms

(b) 25 Mbps, $RTT = 25$ ms

(c) 5 Mbps, $RTT = 500$ ms

(d) 5 Mbps, $RTT = 800$ ms

Figure 2-9: Fairness and friendliness of CTCP over error-prone networks

2.4.4 Application Layer Performance

The performance of a particular transport layer protocol can have serious, non-linear, impacts on upper layer applications. While throughput (or goodput) is usually the primary measure used to measure transport layer performance, the effects of the transport layer on upper layers is possibly a more important measure. This is because the behavior of the transport layer can either improve or adversely affect the end user's quality of service. Two upper layer applications (HTTP and streaming video) are used in this section to measure CTCP's performance.

Figure 2-10 shows completion time measurements for HTTP requests of various file sizes. The measurements were collected using `wget` on the client and `apache2` on the server. The figure shows that the completion times using CTCP are largely insensitive to the packet loss rate. For larger file sizes, the completion times approach the best possible performance indicated by the dashed line. For medium file sizes (e.g., 100 KB) , the completion time is dominated by the slow-start behavior of the congestion control algorithm. However, small file sizes (e.g., 10 KB) do not experience any rate throttling since the amount of data to be sent is smaller than the initial congestion window size. In this regime, CTCP also outperforms standard TCP.

When the link is erasure-fee, both CTCP and TCP achieve similar performance. However, standard TCP's completion time quickly increases with the erasure rate. For a 1 MB connection, the completion time with standard TCP increases from 0.9 seconds to 18.5 seconds as the loss rate increases from 1% to 20% respectively. For a 10 MB connection, the corresponding increase is from 7.1 seconds to 205 seconds. This results in a reduction of more than 20 times (2000%) for a 1 MB connection and by almost 30 times (3000%) for a 10 MB connection.

Figure 2-11 shows the performance of streaming video using both CTCP and standard TCP for transport over a range of packet loss rates on a 25 Mbps link with RTT equal to 10 ms. A 60 second video was streamed from a server over the emulated network to a client. Both used `vlc`. Figure 2-11a plots the measured time for the entire video to complete, which is also commonly referred to as the play-out time.

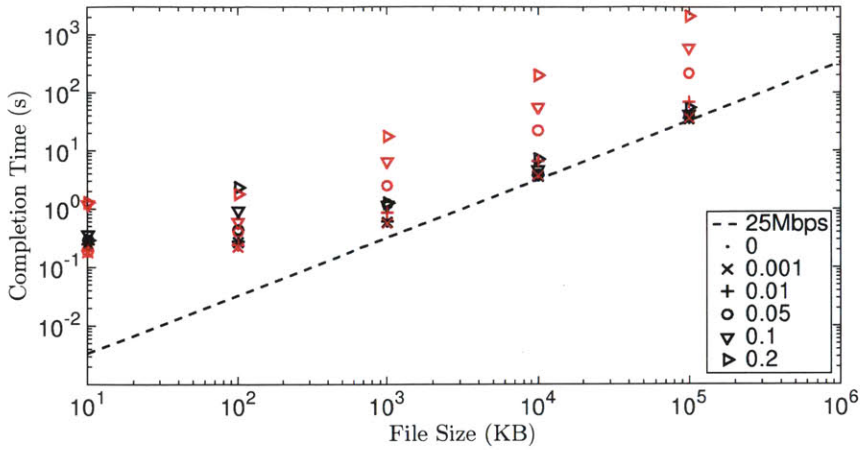
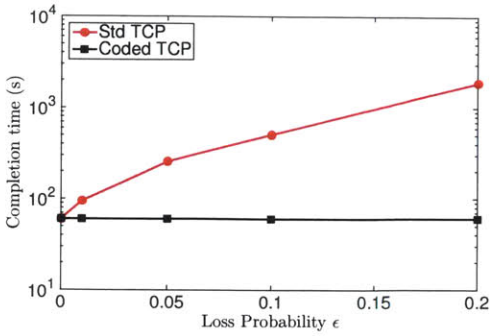


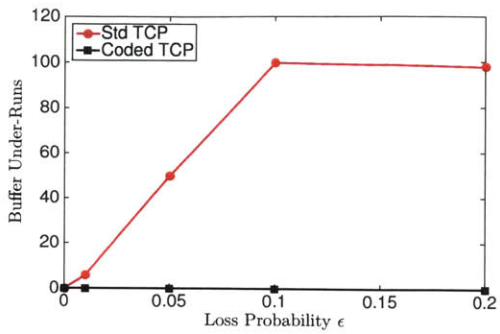
Figure 2-10: Measured HTTP request mean completion time against file size over 25 Mbps link with an RTT of 10 ms. Data is shown for standard TCP (red) and CTCP (black) for a range of loss rates. Error bars are comparable in size to the symbols used in the plot and are omitted.

The play-out time using CTCP is approximately 60 seconds over the entire range of packet erasure rates considered. In contrast, the play-out time using standard TCP increases from 60 seconds to 95 seconds as the erasure rate increases from 0% to 1% respectively. It increases further to 1886 seconds, or 31 minutes, as the loss rate increases to 20%. This is more than 30 times slower than what is achieved by CTCP.

Figure 2-11b shows the number of buffer under-run events experienced by the client. Each buffer under-run is an event that halts the video's playback; and playback is resumed once enough information has been received to play the next block of frames. The figure shows CTCP is able to prevent buffer under-runs for erasure rates as high as 20%. Alternatively, the number of buffer under-runs using standard TCP increases with the erasure rate. For packet erasure probabilities larger than $\epsilon = 0.1$, the number of buffer under-runs plateau at around 100 events. This corresponds to a buffer under-run after each block of frames is received. In terms of user experience, the repeated stalling of the video during playback are indicative of a thoroughly unsatisfactory quality of experience even at a loss rate of 1%.



(a) Completion Time



(b) Buffer Under-Runs

Figure 2-11: Measurements of video streaming performance against loss rate with a 25 Mbps link and a RTT of 10 ms. Data is shown for standard TCP and CTCP. Figure 2-11a shows the running time taken to play a video of nominal duration (60 s); Figure 2-11b shows the number of under-runs of the playout buffer at the client.

2.5 Performance of CTCP Over 802.11 Networks

The measurements shown in previous sections provide insight into CTCP's performance. However, these measurements were conducted within a controlled setting using fixed round-trip times and i.i.d. packet losses. Deployed networks rarely, if ever, experience these conditions. Link capacity can change multiple times over the coarse of a single session, jitter is often an issue, and packet erasures are definitely not i.i.d. Therefore, it is necessary to verify CTCP's performance in conditions that are representative of real networks. This was accomplished using both a wireless 802.11 testbed and public 802.11 hotspots located throughout the greater Boston area.

2.5.1 Networks with Random Noise

The use of unlicensed spectrum in the 2.4 GHz and 5 GHz bands is a major reason for the success of 802.11. Unfortunately, this spectrum is heavily used by everything from cordless phones to microwave ovens. This results in a network with considerable interference that was taken advantage of in the experiments presented below. Each experiment involved an 802.11 b/g wireless client downloading a file from an access point (AP) over a link subjected to interference from a domestic microwave oven

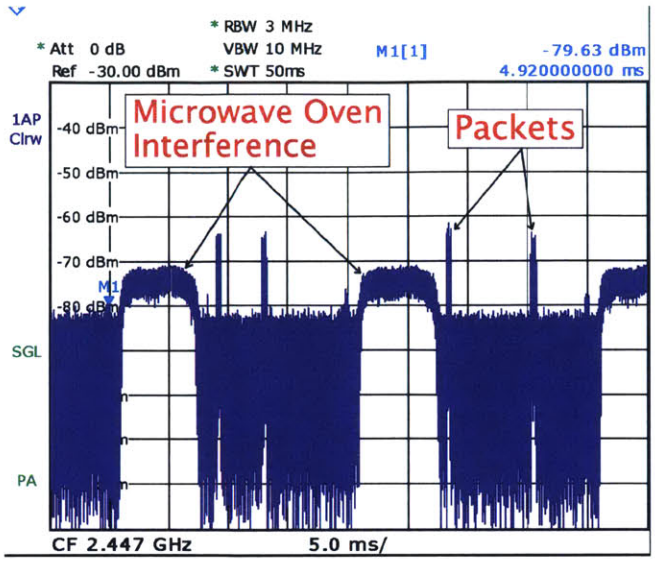


Figure 2-12: Spectrum analyzer screen shot showing interference caused by a microwave oven and transmitted packets sent over Wi-Fi channel 8.

(MWO).

The wireless client and AP were equipped with Atheros 802.11 b/g 5212 chipsets (radio 4.6, MAC 5.9, PHY 4.3 using Linux MadWifi driver version 0.9.4) operating on channel 8. Unless otherwise stated, the default operating system settings were used for all network parameters. A program called `rsync` was used by the client to download a 50 MB file via the AP. The microwave oven used to create interference was a 700 W Tesco MM08 17L, which operates in the 2.4 GHz ISM band with significant overlap ($> 50\%$) with the 802.11's 20 MHz channels 6-13. Its interference is approximately periodic with a period of 20 ms (i.e. 50Hz) and mean pulse width of 9 ms. The pulse width was observed to fluctuate and is due to frequency instability of the MWO cavity magnetron, which is a known effect in MWOs. Figure 2-12 provides a spectrum analyzer screen shot that shows the microwave oven interference, as well as transmissions made over the 802.11 channel 8.

There are two primary reasons for packet losses in this channel. First, the periodic MWO interference caused by the cavity magnetron prevents successful packet transmission, especially for low physical layer (PHY) rates where the duration of a packet transmission is long. This effect was seen for a PHY rate of 1 Mbps were the

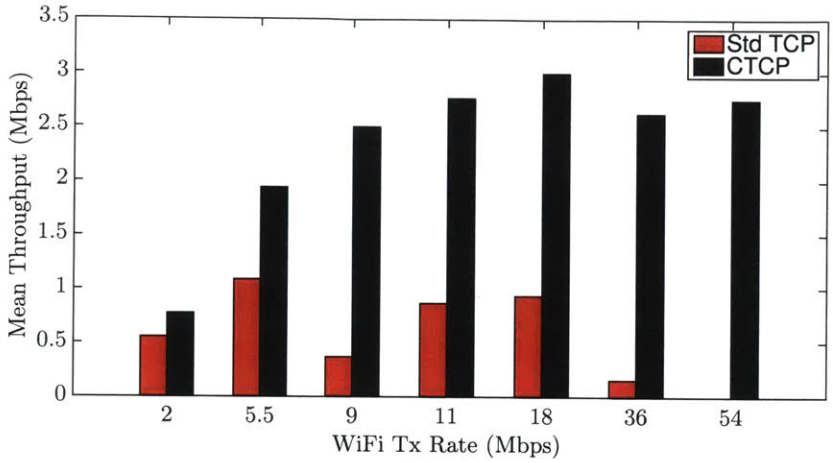


Figure 2-13: Mean throughput versus wireless PHY rate on an 802.11 link with microwave oven interference.

packet loss rate was close to 100%. Second, the 802.11 modulation and coding scheme (MCS) becomes less robust as the PHY rate increases. Since the MWO increases the noise floor, even during periods where there is no cavity magnetron interference, the less robust MCS is unable to cope with the noisy channel making packet losses more likely.

Figure 2-13 presents measurements of the mean throughout achieved over the file download as a function of the PHY rate on the downlink. Data is shown using standard TCP (in this case Cubic TCP, the Linux default variant) and CTCP. The throughput achieved by standard TCP rises slightly as the PHY rate is increased from 2 Mbps to 5.5 Mbps, but then decreases to zero for PHY rates above 36 Mbps. In comparison, CTCP's throughput is approximately double (200%) that of standard TCP at a PHY rate of 5.5 Mbps, more than tripled (300%) at PHY rates of 8, 11 and 18 Mbps, and more than an order of magnitude (1000%) at PHY rates above 18 Mbps. Furthermore, the fluctuations of both TCP and CTCP performance under different link layer coding rates and modulation schemes (indicated by the changes in the 802.11 transmission rate) suggests that CTCP is much more robust to network and link layer changes than TCP.

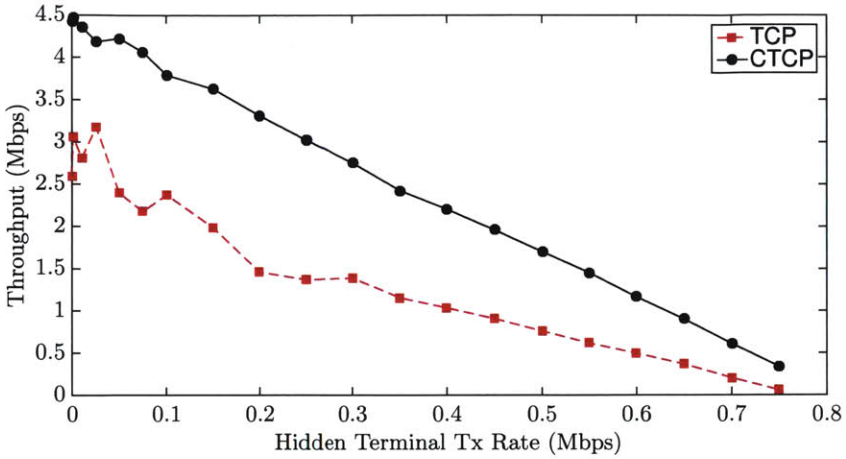


Figure 2-14: Throughput versus intensity of hidden terminal interference when using standard TCP (Cubic TCP) and CTCP over an 802.11 b/g wireless link.

2.5.2 Networks with Hidden Terminals

Using a similar network configuration as the last section, experiments were conducted that measured CTCP's performance when 802.11 is subject to hidden terminal interference. The hidden terminal was created by adding a third station to the network described in Section 2.5.1. Carrier sense on the new terminal's wireless interface card was disabled and 1445 byte UDP packets were transmitted with exponentially distributed inter-arrival times. The transmit rates for both the hidden terminal and AP were set to 11 Mbps. Unless otherwise stated, the default operating system settings are used for all of the remaining network parameters. Figure 2-14 plots the measured throughput of standard TCP and CTCP versus the mean transmit rate of the hidden terminal. CTCP consistently obtains approximately twice (200%) the throughput of standard TCP (Cubic TCP).

2.5.3 Networks "In the Wild"

Finally, the performance of CTCP in a completely uncontrolled environment was measured to determine its effectiveness. Measurements were collected at various public WiFi hotspots in the greater Boston area by downloading a 50 MB file from a server (running Ubuntu 10.04.3 LTS) located on the MIT campus to a laptop (running

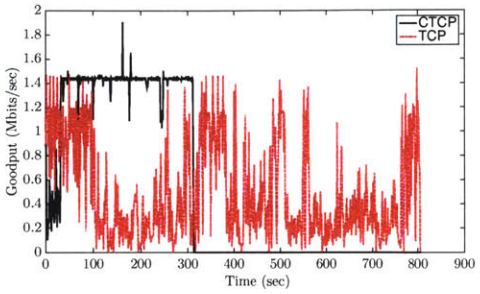
	A	B	C	D	E
CTCP Download Time	313 s	388 s	676 s	292 s	1093 s
TCP Download Time	807 s	1151 s	753 s	391 s	3042 s
Mean PLR	4.28%	5.25%	4.65%	4.56%	2.16%
Mean RTT	54.21 ms	73.51 ms	106.31 ms	50.39 ms	208.94 ms

Table 2.1: Download completion times, the mean packet loss rate (PLR) and the mean RTT for each experiment shown in Figure 2-15.

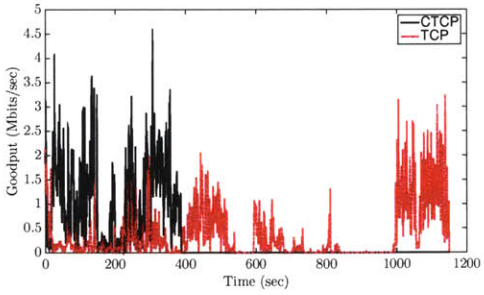
Ubuntu 12.04.1 LTS) located under the public WiFi hotspot. The default operating system settings were used for all network parameters on both the client and server.

Figure 2-15 shows five representative traces of these experiments, along with each traces' mean goodput. Furthermore, Table 2.1 provides the measured download times, mean packet loss rates, and mean round-trip times for each experiment. Each trace represents a different WiFi hotspot, or network, that was chosen because of its location, accessibility, and perceived congestion. For example, each experiment was conducted over a WiFi hotspot located in a shopping center food court, a coffee shop, or a hotel lobby. In Figures 2-15a - 2-15d, the WiFi network spanned a large user area increasing the possibility of hidden terminals. A scan of most networks at the time of the experiment showed greater than 40 active WiFi radios. The only experiment that had a small number of terminals (i.e. five active radios) is shown in Figure 2-15e.

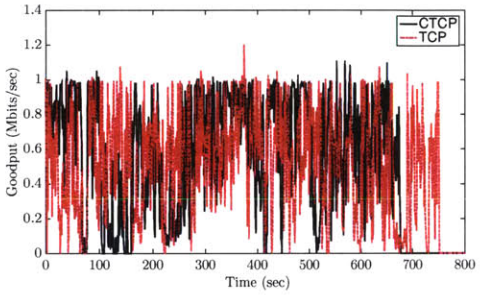
These traces show that CTCP consistently achieves a larger average goodput than standard TCP. Taking the mean goodput over all of the conducted experiments, CTCP achieves a goodput of approximately 750 kbps while standard TCP achieves approximately 300 kbps. An overall gain of approximately 2.5 (250%). Furthermore, it is important to note that standard TCP stalled and had to be restarted twice in the experiment conducted under Hotspot C. CTCP, on the other hand, never stalled nor required a restart. It is important to emphasize the observed loss rates of approximately 4% in each of these experiments, which were unusually high and unexpected. This is directly correlated with CTCP's significant performance gains over standard TCP.



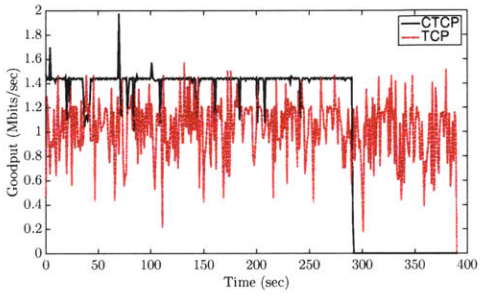
(a) Hotspot A



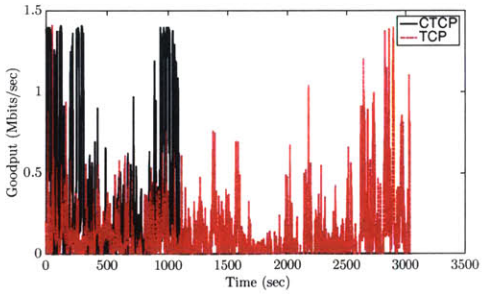
(b) Hotspot B



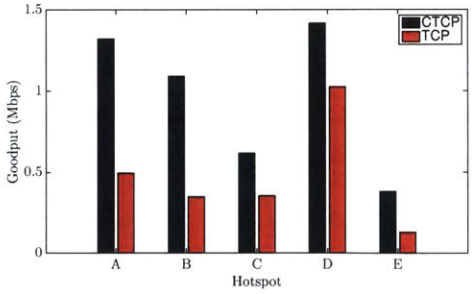
(c) Hotspot C



(d) Hotspot D



(e) Hotspot E



(f) Mean goodput for Hotspots A-E

Figure 2-15: Public WiFi hotspot packet traces

2.6 Conclusions

The measurements presented throughout this chapter show the benefits of using a network coded transport layer to increase both throughput and application layer performance. In fact, CTCP meets all of the goals outlined in Section 2.2. In controlled settings with i.i.d. packet erasures, CTCP is able to outperform standard TCP by a significant margin; and experiments over real networks confirm these gains. The goodput measurements also verified that CTCP provides gains over a wide range of packet loss rates, transmission rates, and round-trip times. In addition, the application layer experiments proved that CTCP can increase upper layer quality of service; and the experimental results helped show that CTCP is both friendly and fair.

While the gains of a coded transport layer shown throughout this chapter are significant, it is difficult to determine their root cause. Both the modified congestion control algorithm and network coding are contributors to the increased performance, and differentiating between the two is a challenge. Regardless, there are indicators pointing to the source for the gains shown. The gains shown for small HTTP request sizes is a clear indicator that network coding is providing a gain. For these small file sizes, the congestion control algorithm has little to no affect on the throughput performance. Therefore, the improvements shown are due to the capability of CTCP to recover from packet erasures quickly. In addition, the gains shown for streaming video are another indicator that network coding is providing a benefit. The reduction in the number of buffer under-runs experienced at high packet erasure rates is too significant to be contributed solely to increased throughput performance due to a modified congestion control algorithm. These gains suggest that network coding is helping to decrease delay. This will be explored further in Chapters 4 and 5.

Finally, the results presented within this chapter focused on single path transport. Therefore, the gains provided by network coding are limited to error correction. The remainder of the thesis will expand on these gains to show how a coded transport layer can be used in a multi-path environment. In particular, the following chapters will help show how network coding can aid in providing connection resiliency, re-

duce management complexity, and decrease in-order delivery delay in networks where multiple paths are available.

Chapter 3

Transport Layer Coding Over Multiple Paths

3.1 Introduction

Simultaneous use of multiple network paths for a single session, or connection, has the potential to increase quality of service, seamlessly offload traffic from expensive networks to cheaper ones, increase session reliability, etc.; yet existing technology does not fully utilize the available resources efficiently to meet these objectives. Instead, only a single network path is preferred while the others are left unused. This poses several issues for disadvantaged, or wireless, clients. Intermittent network connectivity may hinder the ability to communicate effectively, or the exclusive use of a single path may not provide enough resources to meet a user's threshold requirements. Coding at the transport layer can help provide the required tools to diversify a session's traffic over all of the available networks to help solve these problems.

Several real-world examples help to provide the necessary motivations for tackling these problems. Noting that most wireless devices and platforms currently have multiple network interfaces and radios, simultaneous use of these resources can provide many benefits. Consider an aircraft or ship that may have multiple wireless radios, where each independently provides limited bandwidth (see Figure 3-1a). An application needing a high bandwidth connection may require the transport layer to

(a) (b)

Figure 3-1: Simultaneous use of multiple heterogeneous networks can: (a) help to provide an increased quality of service to disadvantaged users, or (b) help to reliably offload traffic from one network to another.

use several of these networks simultaneously to meet the application's requirements. Also consider a mobile client traveling between the coverage area of one network and another (e.g., between a cellular network and WiFi network). Mechanisms exist for a soft handover between coverage areas within the same network (i.e., handover from one cellular tower to another), but a hard handover is typically performed when moving between heterogeneous networks. MPTCP [5] helps perform these handovers without terminating a session, but improvements can be made to increase performance and reliability. Finally, a wireless user may want to offload traffic from an expensive network to another (see Figure 3-1b) in order to reduce usage costs. Existing methods to select the preferred network typically require direct input from the user (e.g., manually turning off or on the WiFi radio). This is, at best, cumbersome and results in inefficiencies.

Transport layer coding can provide methods to handle each of these scenarios; but several challenges exist. Differences in network round-trip times, packet loss rates, bandwidth, coverage areas, etc. present obstacles for ensuring a high quality of service. The first section in this chapter will provide experimental measurements that help to characterize the multi-path environment. Next, a multi-path transport layer protocol called Multi-Path TCP with Network Coding (MPTCP/NC) is introduced

and an analysis of its throughput is provided. This protocol is a first step towards a coded transport layer. While largely impractical to implement, its primary purpose is to help analyze the throughput of a coded transport layer. Finally, the extension of CTCP to the multi-path environment, which is a more practical approach, is discussed.

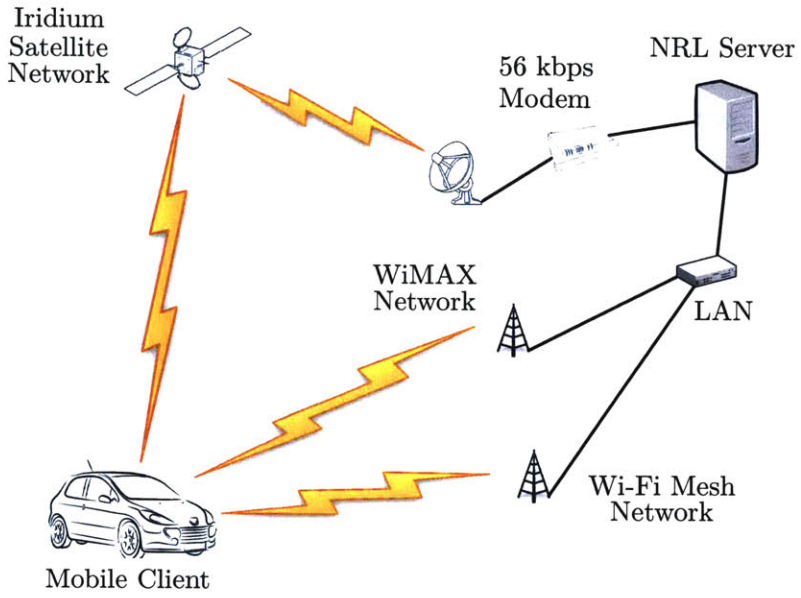
3.2 Characterization of the Multi-Path Environment

Understanding the environment in which a mobile client is expected to operate is critical for developing multi-path techniques. Packet loss, delay, and connectivity can all vary drastically depending on the network used. Simultaneously collecting statistics over multiple heterogeneous networks helps in characterizing the multi-path environment. Using a WiMAX (IEEE 802.16) base station, a WiFi (IEEE 802.11) mesh network, and an Iridium satellite data modem [59], simultaneous network traces were collected between the Network Research Laboratory (NRL), Department of Computer Science, UCLA and a vehicle driving a fixed route around the UCLA campus.

Each experiment sent packets, varying between 64 bytes, 512 bytes , and 1,350 bytes in size, at rates based on the direction of travel. For example, traffic generated by the computer in the NRL and sent to the vehicle, referred to as downlink (D/L) traffic, was sent at rates determined by the individual network (WiMAX: 20 Mbps, WiFi: 20 Mbps, and Iridium: 1 kbps). Traffic generated by the computer in the vehicle and sent to the computer in the NRL, referred to as uplink (U/L) traffic, was also sent at rates determined by the individual network (WiMAX: 1 Mbps, WiFi: 20 Mbps, and Iridium: 1 kbps). In each experiment, only D/L traffic or U/L traffic was generated.

3.2.1 Testbed Configuration

Measurements were taken between a mobile commodity laptop and a fixed server located within the NRL. The server in the NRL was connected to the NRL local area network (LAN), which has gateways to both a WiMAX base-station and the



(a)

Figure 3-2: Multi-Path Experiment Configuration

WiFi mesh network. A 56 kbps modem was used to connect the server to the public switched telephone network (PSTN) in order to utilize the Iridium satellite network. In the vehicle, a single computer with separate WiMAX and WiFi cards, as well as a connection to an Iridium data modem, was used to transmit and receive data. A diagram of the setup is shown in Figure 3-2. UDP network traffic was generated using `iperf` and network traces were collected using `tshark` (a command line version of Wireshark) on both the server and mobile laptop.

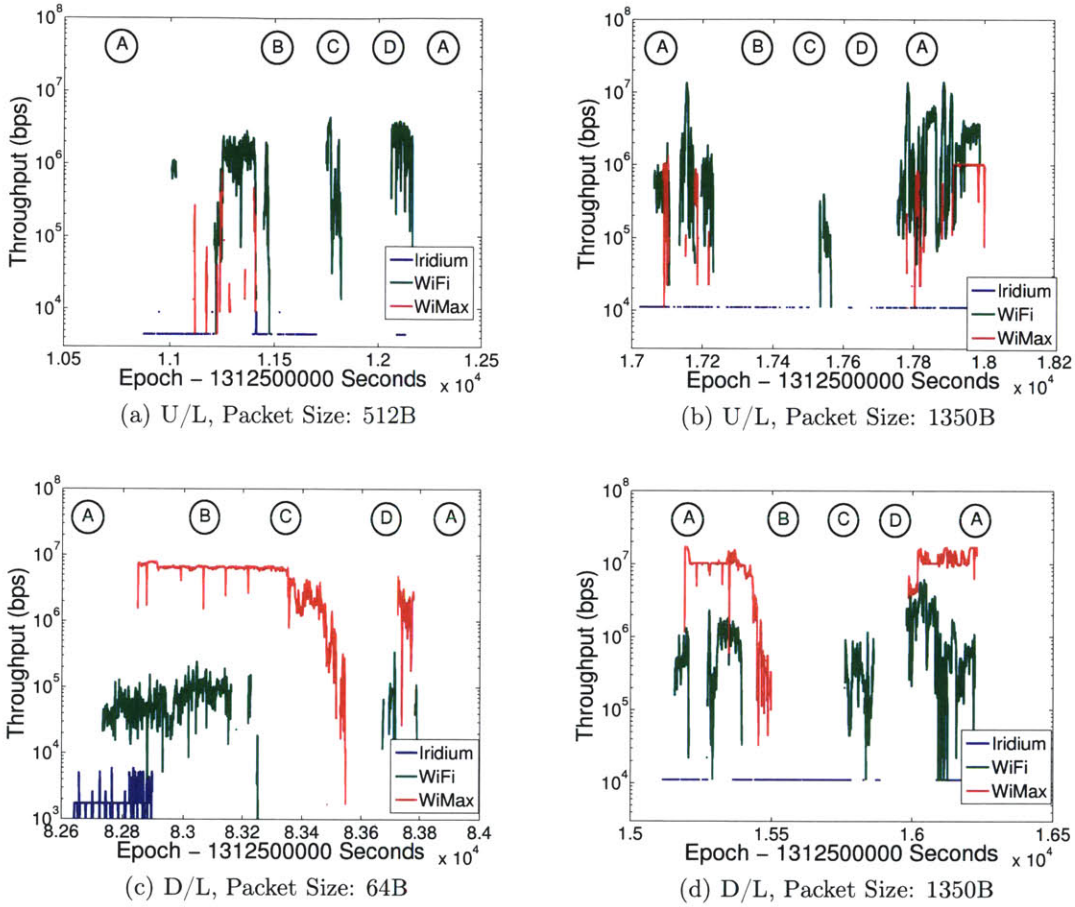
The vehicle containing the mobile computer travelled a fixed route through the UCLA campus. This route was chosen so that the vehicle passed in and out of the coverage areas of all three networks. For example, connections through all three networks were established prior to each experiment. The vehicle would then drop from and reconnect to each of the individual networks, depending on the location of the vehicle and coverage of the specific network, throughout the duration of the experiment. Figure 3-3 provides the vehicle route and placement of the WiMAX and WiFi mesh base stations on the UCLA campus.

Figure 3-3: WiMAX/WiFi Base Station Placement and Vehicle Route

3.2.2 Collected Data

Ten mobile experiments were conducted over a period of five days in August 2011. For each of these experiments, traces were collected and compared over each of the different networks. A sample of the collected traces are shown in Figure 3-4. These traces show the UDP throughput for each network when all traffic is sent either to (D/L) or from (U/L) the vehicle.

The round-trip time (RTT) and packet loss probability ϵ for each network was also collected. The CDFs for both the RTT and packet loss probability during the D/L experiment where 1,350 byte packets were used is shown in Figure 3-5. The RTT was measured using ping messages that were sent throughout the experiment on both the WiFi and WiMAX networks, while ping messages were only sent for approximately 60 seconds at the beginning of the experiment on the Iridium network due to the bandwidth constraints of the network. The packet loss probabilities were determined by comparing the trace files on both the NRL server and the vehicle computer.



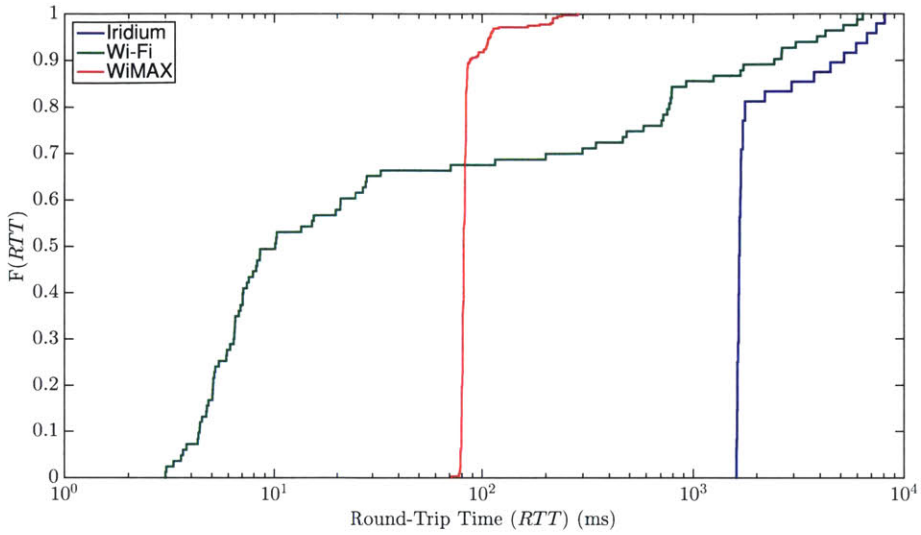
(a) U/L, Packet Size: 512B

(b) U/L, Packet Size: 1350B

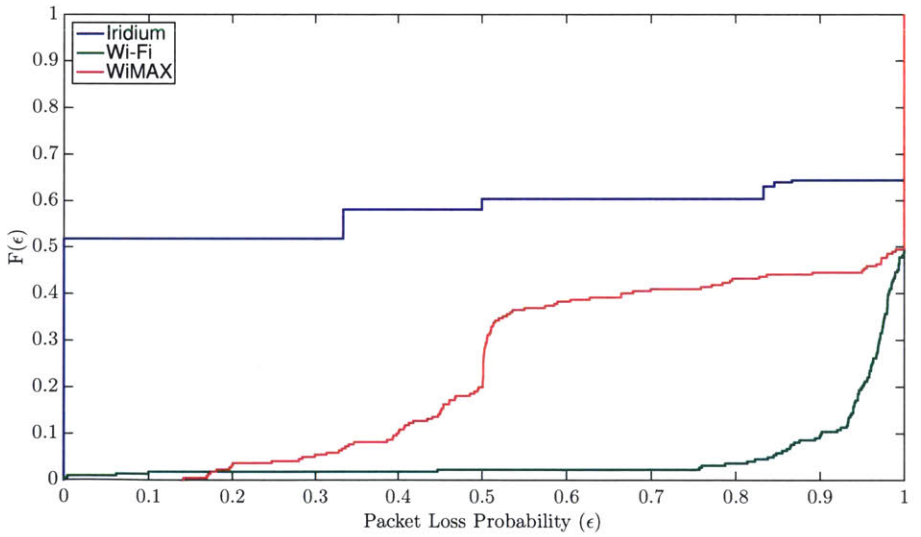
(c) D/L, Packet Size: 64B

(d) D/L, Packet Size: 1350B

Figure 3-4: Sample traces showing the UDP throughput for two U/L and two D/L experiments with varying packet sizes. The labels A, B, C, and D provide the approximate location of the vehicle when compared with Figure 3-3.



(a) RTT CDF



(b) Packet Loss CDF

Figure 3-5: CDFs of the RTT and packet loss probabilities during the D/L experiment using 1,350 byte packets.

3.2.3 Discussions and Comments

Data collected during each of the experiments provides information about the expected environment that mobile users are likely to experience. However, there are caveats concerning the methods in which the data was collected that must be noted. First, the only user on each network was the vehicle; although the WiFi mesh network performance was affected by significant interference from adjacent WiFi networks and the Iridium network was setup over an operational system. As a result, the WiMAX throughput presented in Figure 3-4 is much larger than what would be expected when fully loaded, while the WiFi and Iridium throughput is close to what we would expect in the real-world. The *RTT* shown in Figure 3-5(a) is also affected by this situation. Since only one user has access to the WiMAX network, the *RTT* is fairly consistent throughout each experiment. The WiFi *RTT* is largely affected by contention with adjacent WiFi networks resulting in a wide range of possible *RTT*s, and the Iridium *RTT* is consistent except for periods where it is believe horizontal handoffs between satellites occurred.

Second, the data collection methods were designed so that data could be used to replay each experiment off-line enabling easy evaluation of future protocol designs. This prevented the collection of reliable statistics on the packet loss probabilities. However, Figure 3-5(b) shows the overall reliability of each network. It shows indications that the satellite network provides the most reliability and the WiFi network provides the least. Finally, the use of a modem and the PSTN for the Iridium network (and consequently the low throughput) is due to the Iridium system design. Iridium was developed for world-wide voice communications. Modern satellite systems do provide higher bandwidth, and therefore better performance for packet based communication. Unfortunately, the use of these systems was prohibitively expensive.

Regardless, the traces shown in Figure 3-4 indicate that using a multi-path solution can potentially provide significant performance gains over using only one of the networks exclusively. Throughout each experiment, the vehicle was connected to at least one network the majority of the time; and in many cases, it was connected

to two or more networks. Leveraging this connectivity can help ensure that reliable, continuous data transport is an option in mobile environments. The benefits of leveraging simultaneous networks for data transport will be quantified in the following sections using the collected data. Specifically, packet loss and *RTT* statistics will be used to provide a comparison between the performance of MPTCP and Multi-Path with Network Coding (MPTCP/NC) in multi-path, wireless scenarios.

3.3 A Coded Multi-Path Transport Layer Protocol

Multi-Path TCP with Network Coding (MPTCP/NC) is one approach to enable communications over multiple heterogeneous networks similar to those shown in the previous section. Based on MPTCP [60] that offers many features that enable simultaneous use of heterogeneous networks, MPTCP/NC also uses network coding to help operate on unreliable networks. As noted earlier, it is a first step towards a coded multi-path transport layer and its design is largely impractical. However, the protocol described in this section is used to analyze the throughput that is achievable over multiple paths.

MPTCP/NC adds two network coding layers to the traditional network protocol stack. The MPTCP/NC layer, placed between the application layer and the transport layer, offers a single interface to the application layer while providing path management and packet injection into n TCP sub-flows. Network coding at this layer helps to provide path redundancy and simplifies packet scheduling across each of the flows by eliminating the need to track each individual packet. The Fast-TCP/NC layer is inserted directly below the TCP layer and provides many of the benefits described in [19], but with several modifications to make it more efficient when used with MPTCP/NC. Unlike the MPTCP/NC layer, network coding in the Fast-TCP/NC layer is used to add redundancy to the flow to help recover from packet losses encountered during transmission. Figure 3-6 provides a high level overview of the protocol implementation. Subsequent sub-sections will describe both of the new layers in more detail.

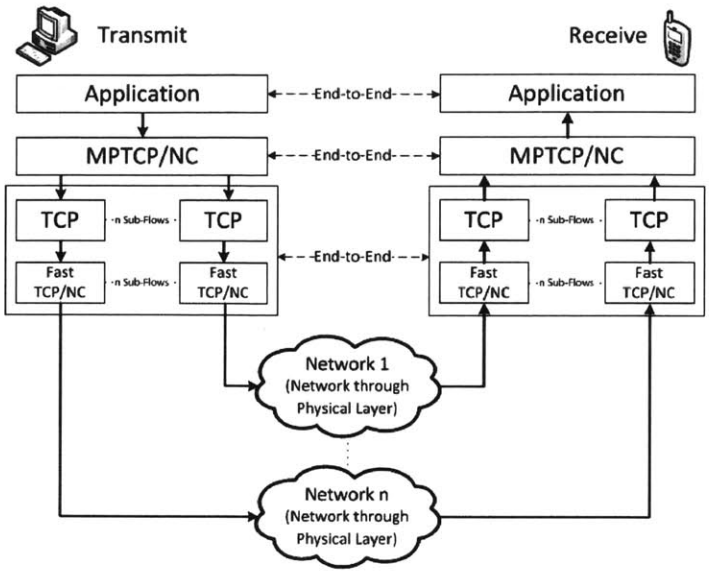


Figure 3-6: MPTCP/NC Protocol

3.3.1 Description of the MPTCP/NC Protocol

The MPTCP/NC protocol adds two layers to the conventional protocol stack. The MPTCP/NC layer sits directly above TCP, or the transport layer, while the FAST-TCP/NC layer is inserted immediately below the transport layer. Instead of seeing multiple TCP links, the application layer interfaces with only one instance of MPTCP/NC. The term *connection* refers to the overlay link at the MPTCP/NC layer between the server and client, and it may include multiple TCP sub-flows at the transport layer. The initiation and management of multiple TCP sub-flows is assumed to be identical to that of MPTCP, since it is assumed that MPTCP's connection management works the same when network coding is added. Details of MPTCP's connection management can be found in [60]. For the remainder of the chapter, it is assumed that two or more TCP sub-flows are properly established and managed under the same connection.

Once each sub-flow is established, the application layer pushes data packets to the MPTCP/NC layer. These packets are coded together using RLNC and then linearly independent coded packets are injected into a TCP sub-flow when possible. Since each injected packet is a linearly independent coded packet, the need to assign specific

packets to a sub-flow is no longer necessary. As the TCP sub-flows transmit packets over the network, the number and indices of uncoded packets that the MPTCP/NC layer codes together is adapted to ensure that new degrees of freedom are always sent.

The transport layer TCP protocol functions normally, as all the operations at MPTCP/NC and Fast-TCP/NC are transparent to the TCP protocol. However, it should be noted that all of the packets in the TCP window are coded packets. These packets are then pushed to the Fast-TCP/NC layer, which also maintains a coding window equal to the size of the TCP congestion window, similar to TCP/NC [19]. For each packet TCP sends to the Fast-TCP/NC layer, the latter produces R coded packets that are random linear combinations of all packets in its coding window. This R is referred to as the redundancy and is theoretically equal to one divided by one minus the packet loss rate. These coded packets are finally pushed to the network layer and transmitted over the network. In particular, the packets that are sent to the network layer are random linear combinations of the encoded packets from MPTCP/NC (i.e., the original packets have been encoded twice by the sender).

If the twice-encoded packets are received by the client, the Fast-TCP/NC layer immediately sends an acknowledgement to the same layer at the server. It then passes the received packets upward to TCP without attempting to decode. All packets delivered by any TCP sub-flow will eventually reach the client's MPTCP/NC layer. The MPTCP/NC layer performs Gaussian Jordan elimination progressively as it receives new packets and delivers information packets to the client's application layer whenever a decoding event occurs.

The original MPTCP protocol provides a connection-level acknowledgement (DATA ACK). In MPTCP/NC, this DATA ACK is used to acknowledge when a new degree of freedom (dof) was received. This new dof must be a packet that is linearly independent from all previously received packets. Once the MPTCP/NC layer collects enough linearly independent packets, it is able to solve a linear system of received packets. The decoded packets are then delivered to the application level.

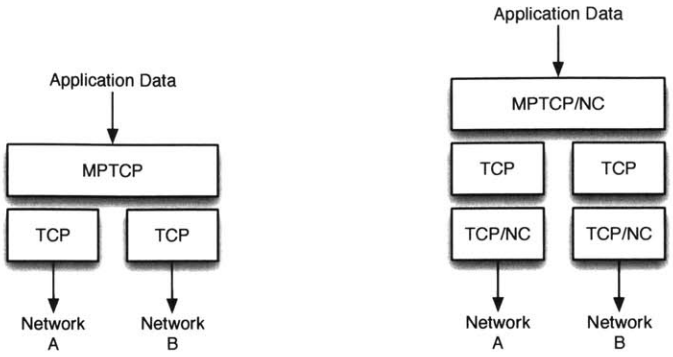
3.3.1.1 The Fast-TCP/NC Layer

The Fast-TCP/NC layer is inserted directly below the transport layer and functions in a similar manner as the TCP/NC layer proposed in [19]. On the server side, the coding window management and encoding operations are exactly identical to TCP/NC (except the encoding is done on the previously coded packets from MPTCP/NC layer). The difference between Fast-TCP/NC and the original TCP/NC lies in the acknowledgment mechanism. In Fast-TCP/NC, there is no *seen* packet concept and the client's Fast-TCP/NC layer does not perform Gaussian Jordan elimination. The Fast-TCP/NC layer sends an acknowledgement whenever a packet is received regardless of whether or not it is linearly independent with previously received packets. This differs from the original TCP/NC that sends an ACK if and only if the packet provides a new dof.

These modifications were made because a TCP sub-flow does not have global knowledge of the linear space spanned by packets received by all of the sub-flows. Therefore, a linear dependency check is not possible at the Fast-TCP/NC layer. Instead, Fast-TCP/NC delivers packets up to the MPTCP/NC layer where linear dependency checks and decoding occur. Furthermore, the second encoding that took place at the server's Fast-TCP/NC layer results in just another random linearly encoded packet given that the coefficients in the packet header are properly organized and adjusted. These packets can be decoded at the MPTCP/NC layer without intermediate decoding at the lower layer.

3.3.1.2 The MPTCP/NC Layer

The MPTCP/NC layer sits between the traditional application layer and the transport layer. This layer maintains a coding window that decides the number of information packets to be coded together. Once a coded packet is produced by the server's MPTCP/NC layer, it is pushed a TCP sub-flow for transmission. Upon reception by the client's MPTCP/NC layer, it is checked to see if it is linearly independent. If it is, a DATA ACK is sent back to the server. If not, the client drops the packet without



(a) MPTCP (b) MPTCP/NC

Figure 3-7: Assumed network stack configuration for both MPTCP and MPTCP/NC.

acknowledging it. When the server's MPTCP/NC layer receives a DATA ACK, it adjusts the coding window accordingly. It should be noted that the design of the coding window is very flexible since it is not constrained by TCP congestion window, which is the case for the original TCP/NC protocol.

3.3.2 Analytical Performance of MPTCP/NC

Approaches similar to that of [55] and [61] are used to provide a mean-field approximation of the throughput for both MPTCP and MPTCP/NC. The MPTCP analysis will assume the standard implementation as shown in Figure 3-7a and defined by [60]. The MPTCP/NC analysis will assume two layers of network coding. The MPTCP/NC layer shown in Figure 3-7b provides a first layer of network coding before packets are injected into a TCP sub-flow, and the TCP/NC layer provides a second layer of network coding, similar to [19], in order to overcome random packet losses due to lossy networks.

The analysis for MPTCP, which is provided in Section 3.3.2.1, will use the model presented by [55] and assume that perfect packet scheduling across each TCP sub-flow takes place. Once the analytical throughput for each individual sub-flow is determined, the results can be summed to determine MPTCP's overall throughput. In reality, perfect scheduling is not possible due to packet losses, termination of a

specific sub-flow, etc. This results in the need to collect feedback regarding which packets were lost, retransmit each lost packet on a second (or third) TCP sub-flow, and verify receipt of that packet by the client. This process significantly decreases the efficiency of MPTCP by both lowering the throughput and increasing the transport time. With this in mind, the analytical results presented later will over estimate the performance of MPTCP.

In the case of MPTCP/NC, network coding can be used to aid in the sub-flow scheduling problem by eliminating the need to track specific packets sent over the network. With respect to the analysis, it is assumed that network coding is performed prior to a packet's injection into a sub-flow. If the coding operations are carried out properly, the server will be successful in transferring data over multiple sub-flows without the need to track individual packets through the multiple networks. Not only does this significantly decrease the complexity of the protocol, but also provides greater freedom for determining how to allocate packets among the collection of sub-flows. It is also assumed that a second layer of network coding occurs below TCP and redundant packets are transmitted to overcome random packet losses. In general, the number of transmitted packets for every dof sent should be $R \geq 1/1-\epsilon$ where R is the redundancy and ϵ is the packet loss probability of the network path.

Finally, it is assumed that both protocols use a TCP Reno style of congestion control on each sub-flow. This assumption keeps the results presented here in line with those presented by [55] and also simplifies the analysis for MPTCP/NC. Because network coding is performed below TCP on each sub-flow, network coding eliminates the need to consider the effects of triple-duplicates on TCP's window size. A more detailed discussion will be provided in subsequent sections.

3.3.2.1 MPTCP Analytical Throughput

The analytical throughput for MPTCP follows directly from equation (32) in [55] where the throughput $B(\epsilon)$ of a single TCP connection is determined. The existing analysis can be extended to the multi-path case by taking the calculated $B_j(\epsilon_j)$ for each sub-flow, $j = \{1, \ldots, n\}$, and summing them together to form the MPTCP

throughput:

$$B(p_1, \ldots, p_n) = \sum_{j=1}^{n} B_j(p_j). \tag{3.1}$$

As noted earlier, this does not take into account the inefficiencies introduced by the MPTCP layer and will over-estimate the achievable MPTCP throughput. It also ignores any rate limiting that MPTCP may do in order to be fair with single TCP flows.

3.3.2.2 Preliminaries

Two metrics are used to develop a mean-field approximation of MPTCP/NC's throughput: the average throughput $\mathcal{T}$, and the expected MPTCP/NC congestion window evolution $\mathbb{E}[W]$. MPTCP/NC's behavior is modeled in terms of rounds. The natural choice for determining the duration of a round is to use the RTT from the server to the client (i.e., $t_{rnd} = RTT$). While this works if there is a single TCP connection, each sub-flow is expected to have different round trip times making it difficult to determine which RTT to use. This is accounted for by setting the duration of each round, t_{rnd}, equal to the greatest common divisor (GCD) of the sub-flows' RTTs assuming that each sub-flows' RTT is an integer. Figure 3-8 provides an illustration of this concept. The top portion of the figure shows the first flow's congestion window size over time while the bottom portion shows the second flow's congestion window size over time. Since $2RTT_1 = RTT_2$, the round duration is $t_{rnd} = RTT_1$.

3.3.2.3 Sub-Flow Analysis

Consider the most basic implementation of TCP and initially assume that a round's duration is equal to the RTT of sub-flow $j = \{1, \ldots, n\}$. Assume that the congestion window size $W_i^{(j)}$ of sub-flow j during round i is determined by the number of acknowledgements $a_{i-1}^{(j)}$ indicating successfully transmitted packets obtained during round $i - 1$:

$$W_i^{(j)} = W_{i-1}^{(j)} + \frac{a_{i-1}^{(j)}}{W_{i-1}^{(j)}}. \tag{3.2}$$

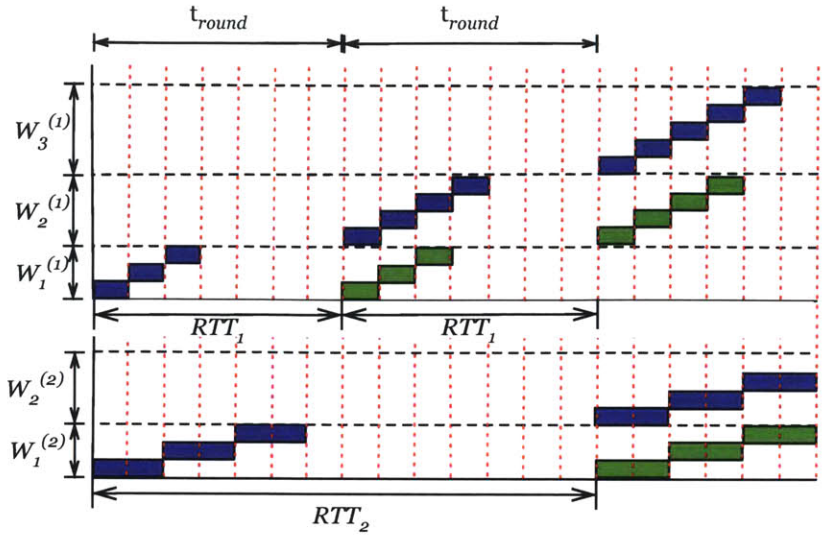


Figure 3-8: MPTCP/NC round duration used for two sub-flows. The blue blocks indicate packets and the green blocks indicate acknowledgements.

This concept is also shown in Figure 3-8 where the congestion window size of each sub-flow grows as a function of the number of acknowledgements received. Now assume that R_j linearly independent network coded packets are sent for each uncoded packet contained the TCP congestion window. Also assume i.i.d. packet losses with a packet loss rate of ϵ_j. Taking the expectation of the window size, $\mathbb{E}[W_i^{(j)}]$, the following is obtained:

$$\mathbb{E}\left[W_i^{(j)}\right] = \mathbb{E}\left[W_{i-1}^{(j)}\right] + \min\left(1, (1-\epsilon_j)\, R_j\right) \tag{3.3}$$

$$= \mathbb{E}\left[W_1^{(j)}\right] + (i-1)\min\left(1, (1-\epsilon_j)\, R_j\right), \tag{3.4}$$

where the minimization is required because the window size can only increase by a maximum of one packet per round.

Since the throughput $\mathcal{T}_i^{(j)}$ per round is related to the number of packets sent in that round,

$$\mathcal{T}_i^{(j)} = \frac{\mathbb{E}\left[W_i^{(j)}\right]}{RTT_j}\min\left(1, (1-\epsilon_j)\, R_j\right). \tag{3.5}$$

The minimization in this equation is necessary to account for packets that are received that do not deliver new degrees of freedom. Since the TCP/NC layer codes all packets within the TCP congestion window, delivered packets 1 through $W_i^{(j)}$ contain new degrees of freedom. If more than $W_i^{(j)}$ packets are received in the round, the MPTCP/NC layer disregards them since they contain no new information.

The above analysis assumed that the round-trip times for each sub-flow were the same. Because this is not necessarily the case, an adjustment to (3.4) is needed to account for the shorter round durations. This is accomplished by defining $\alpha_j = {}^{RTT_j}/_{t_{rnd}}$ and substituting $\lceil {}^{i}/_{\alpha_j} \rceil$ for i,

$$\begin{aligned} \mathbb{E}\left[W^{(j)}_{\lceil i/\alpha_j \rceil}\right] &= \mathbb{E}\left[W_1^{(j)}\right] + (\lceil {}^{i}/_{\alpha_j} \rceil - 1) \min\left(1, (1-\epsilon_j) R_j\right) \\ &= \gamma_i^{(j)}. \end{aligned} \tag{3.6}$$

The throughput for each TCP sub-flow j then becomes

$$\mathcal{T}_i^{(j)} = {}^{\gamma_i^{(j)}}/_{(\alpha_j \cdot t_{rnd})} \min\left(1, (1-\epsilon_j) R_j\right). \tag{3.7}$$

If a large enough redundancy factor R_j is assumed, $\mathcal{T}_i^{(j)}$ can be reduced further. Assume $R_j > {}^{1}/_{(1-p_j)}$, the instantaneous throughput becomes

$$\mathcal{T}_i^{(j)} = \frac{1}{\alpha_j \cdot t_{rnd}} \left(\mathbb{E}\left[W_1^{(j)}\right] + \lceil {}^{i}/_{\alpha_j} \rceil - 1\right). \tag{3.8}$$

Finally, TCP typically has a fixed maximum congestion window size. Taking this into account, the number of packets sent in each RTT is upper-bounded by TCP's maximum congestion window size, $W_{\max}^{(j)}$. Equation (3.8) then becomes

$$\mathcal{T}_i^{(j)} = \frac{1}{\alpha_j \cdot t_{rnd}} \left(\min\left(W_{\max}^{(j)}, \mathbb{E}\left[W_1^{(j)}\right] + \lceil {}^{i}/_{\alpha_j} \rceil - 1\right)\right). \tag{3.9}$$

The model used in the above analysis makes several assumptions that, in practice, should be considered. First, it is assumed that packet losses are i.i.d. with loss

probability ϵ_i. Therefore, the analysis does not account for correlated packet losses due to congestion and other factors. Second, the redundancy R_j is assumed to be sufficiently large enough to ignore the possibility of time-outs. While the probability of a time-out decreases with increasing R_j, time-outs still occur in practice and the impact of each time-out on the throughput is significant (i.e., the congestion window size is reset to $\mathbb{E}[W_1^{(j)}]$). Specifically, a time-out occurs when the sum of received acknowledgements over two rounds, i and $i+1$, is less than the window size during round i with probability $\mathcal{P}r\left(a_i + a_{i+1} < W_i\right)$. Section 3.3.2.5 will provide a method to address this issue. Third, the RTT_j remains constant. In practice, this is not true, and implementations of TCP generally use an averaged round-trip time often referred to as the "smoothed" round-trip time $SRTT$.

3.3.2.4 Window Evolution and End-to-End Throughput

Using the above results, the average end-to-end MPTCP/NC throughput over k rounds is determined using a round duration of t_{rnd} and defining $\alpha_j = {}^{RTT_j}/_{t_{rnd}}$,

$$\mathcal{T}(k) = \frac{1}{k}\sum_{i=1}^{k}\mathcal{T}_i = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{n}\mathcal{T}_i^{(j)}. \tag{3.10}$$

Assuming that ${}^{k}/_{\alpha_j} \in \mathbb{Z}$, $\gamma_i^{(j)} \leq W_{max}^{(j)}$, and relaxing $\lceil {}^{i}/_{\alpha_j} \rceil$ so that it is ${}^{i}/_{\alpha_j}$ for all j,

$$\mathcal{T}(k) = \frac{1}{k}\sum_{j=1}^{n}\left(\frac{1}{\alpha_j \cdot t_{rnd}}\sum_{i=1}^{k}\gamma_i^{(j)}\right) \tag{3.11}$$

$$= \frac{1}{t_{rnd}}\sum_{j=1}^{n}\left(\frac{1}{\alpha_j}\mathbb{E}\left[W_1^{(j)}\right] + \frac{k+1}{2\alpha_j^2} - \frac{1}{\alpha_j}\right). \tag{3.12}$$

If ${}^{k}/_{\alpha_j} \notin \mathbb{Z}, \forall j$, the above equation will contain additional terms that contain packets sent in the rounds from $\lfloor {}^{k}/_{\alpha_j} \rfloor$ to ${}^{k}/_{\alpha_j}$. Furthermore, the relaxation of $\lceil {}^{i}/_{\alpha_j} \rceil$ to ${}^{i}/_{\alpha_j}$ decreases the throughput since (3.12) is no longer accounting for $\lceil {}^{i}/_{\alpha_j} \rceil - {}^{i}/_{\alpha_j}$ packets sent per round. As k grows, these approximations have less of an effect on the throughput.

Finally, the maximum window size of each sub-flow $W_{\max}^{(j)}$ is taken into account.

Define

$$r^{(j)} = \alpha_j \left(W_{\max}^{(j)} - \mathbb{E}\left[W_1^{(j)} \right] \right). \tag{3.13}$$

Using equation (3.12) and assuming that $R_j > 1/(1-p_j)$, the average end-to-end throughput $\mathcal{T}_{e2e}$, in packets per second is:

$$\mathcal{T}_{e2e}(k) = \sum_{j=1}^{n} \mathcal{T}_{e2e}^{(j)}(k), \tag{3.14}$$

where

$$\mathcal{T}_{e2e}^{(j)}(k) = \begin{cases} \frac{1}{\alpha_j \cdot t_{rnd}} \left(\mathbb{E}\left[W_1^{(j)} \right] + \frac{k+1}{2\alpha_j} - 1 \right) & \text{for } k \leq r^{(j)}, \\ \frac{\rho^{(j)}}{\alpha_j \cdot k \cdot t_{rnd}} & \text{for } k > r^{(j)}, \end{cases} \tag{3.15}$$

and

$$\rho^{(j)} = r^{(j)} \mathbb{E}\left[W_1^{(j)} \right] + \frac{r^{(j)} \left(r^{(j)} + 1 - 2\alpha_j \right)}{2\alpha_j} + W_{\max}^{(j)} \left(k - r^{(j)} \right). \tag{3.16}$$

It should be noted that as $k \to \infty$ for $R_j > 1/(1-p_j)$, the average end-to-end throughput becomes

$$\lim_{k \to \infty} T_{e2e}(k) = \sum_{j=1}^{n} W_{\max}^{(j)} / (\alpha_j \cdot t_{tnd}).$$

.

3.3.2.5 Markov Chain Model

The above analysis provides a closed-form solution for MPTCP/NC's end-to-end throughput, but it did not address TCP time-outs. To provide this level of fidelity, a Markov chain model is required to determine MPTCP/NC's throughput. Figure 3-9 shows an example of the Markov chain for a single Fast-TCP/NC sub-flow with $R = 1$ (i.e., no redundant packets are sent over the network). The value represented in the top half of each state represents the congestion control window size (i.e., see equation (3.2)), and the value in the lower half of each state represents the number of packets lost up to and including the transition into the state. Because it is assumed that no redundant packets are sent to make-up for lost packets, transitions between states tend to propagate down within the chain. Once the number of lost packets equals the

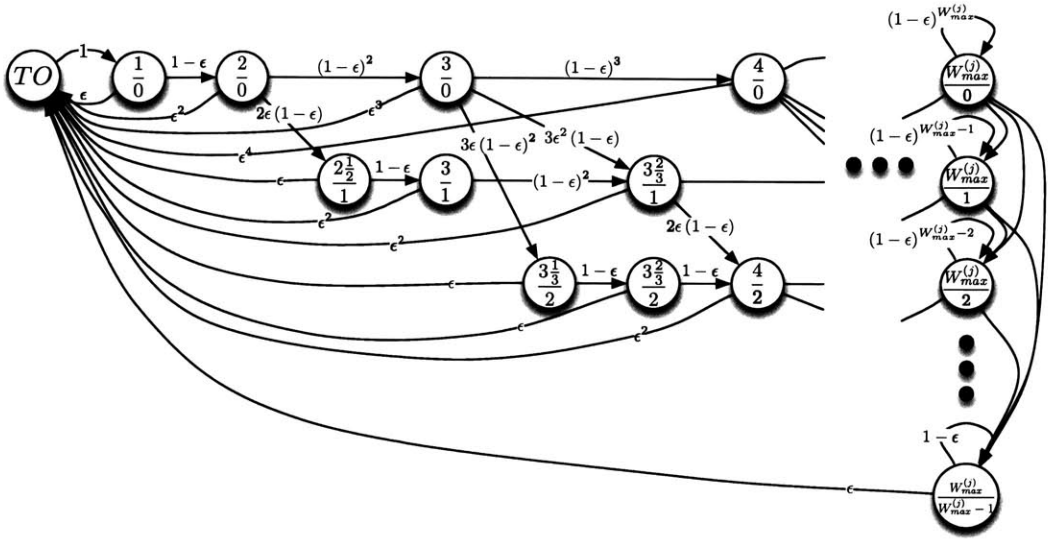


Figure 3-9: Markov Chain Model for MPTCP/NC

congestion window size (i.e., $\lfloor W^{(j)} \rfloor$), TCP will time-out and the congestion window closes.

To determine the throughput, let $\mathcal{S}$ be the set of all states in the Markov chain, p_{ik} be the probability of transitioning from state $i \in \mathcal{S}$ to $k \in \mathcal{S}$, $[P]$ be the transition matrix of the Markov chain, and $[P^n]$ be the nth power of $[P]$. Because the Markov chain is both ergodic and has a finite number of states, $\lim_{n\to\infty}[P^n] = \underline{e}\underline{\pi}$ where $\underline{\pi}$ is the steady-state vector of the Markov chain and $\underline{e} = (1, \ldots, 1)^T$. Defining N_t to be the number of packets successfully received by the client within the time interval $(0, t]$, the steady-state throughput is:

$$\mathcal{T}^{(j)} = \lim_{t\to\infty} \frac{N_t}{t} \tag{3.17}$$

$$= \frac{\mathbb{E}[N]}{\mathbb{E}[T]} \tag{3.18}$$

$$= \frac{\sum_{i\in\mathcal{S}} \pi_i \sum_{k\in\mathcal{S}} n_{ik} p_{ik}}{\sum_{i\in\mathcal{S}} \pi_i \sum_{k\in\mathcal{S}} r_{ik} p_{ik}}, \tag{3.19}$$

where r_{ik} is the time it takes to successfully send n_{ik} packets from state i to state k.

For all $(i, j) \neq (TO, (1, 0))$, the time spent sending packets from i to k is $r_{ik} = RTT_j$, and $r_{TO,(1,0)} = \mathbb{E}[T_o]$ where $\mathbb{E}[T_o]$ is the expected time-out duration. Following

the TCP Reno model presented in [55], the total duration of a time-out period is dependent on the sequence of back-to-back time-outs. For every subsequent time-out, the time-out duration doubles until a duration of $64T_o$ is reached. Therefore, a sequence of k back-to-back timeouts has duration:

$$t_o(k) = \begin{cases} (2^k - 1)T_o & \text{for } k \leq 6, \\ (63 + 64(k-6))T_o & \text{for } k > 6. \end{cases} \tag{3.20}$$

Taking the expectation of $t_o(k)$,

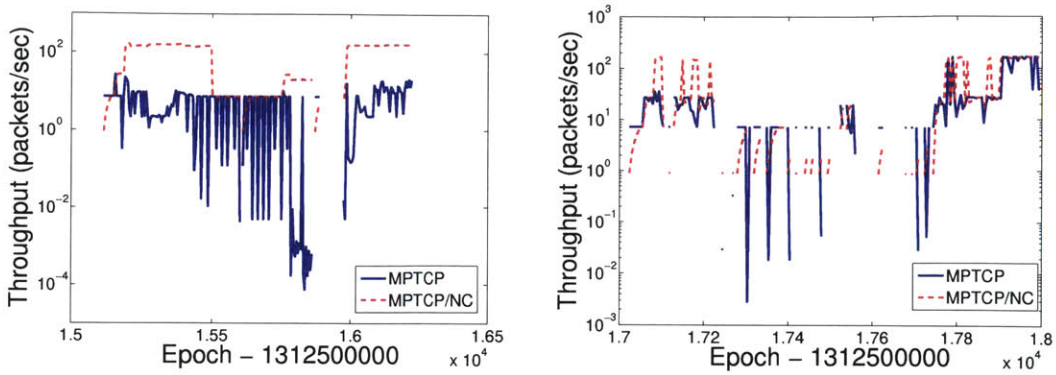
$$\mathbb{E}[T_o] = \sum_{k=1}^{\infty} t_o(k) p_K(k) \tag{3.21}$$

$$= \left(\frac{1 + \sum_{k=1}^{6} 2^{k-1} p^k}{1-p} \right) T_o. \tag{3.22}$$

Inserting equation (3.22) into equations (3.19) and (3.14), the average throughput of MPTCP/NC can be determined.

3.3.3 MPTCP and MPTCP/NC Performance using Empirical Data

This section compares the theoretical throughput of MPTCP given in equation (3.1), with the theoretical throughput of MPTCP/NC given in equation (3.14). Figure 3-10 shows the performance of both protocols using the data presented in Section 3.2 as a baseline. The maximum window size for each TCP sub-flow was set to $W_{\max}^{(j)} = 12$; and a mean RTT, based off of empirical data, was used for each network where $RTT_{\text{Iridium}} = 1.653$s, $RTT_{\text{Wi-Fi}} = 0.607$s, and $RTT_{\text{WiMAX}} = 0.087$s. Empirical packet loss data, averaged over 5 seconds, on each separate path from two of the experiments was used as a baseline for determining both throughputs. It was assumed that the network capacity for each network was large enough to send $W_{\max}^{(j)}$ packets in the case of MPTCP and $R_j W_{\max}^{(j)}$ packets in the case of MPTCP/NC where R_j is assumed to be large enough so that time-outs are very unlikely (i.e., R_j is much larger than



(a) D/L, Packet Size: 1350B

(b) U/L, Packet Size: 1350B

Figure 3-10: Comparison of the theoretical MPTCP and MPTCP/NC throughput using the data presented in Section 3.2.

the 5 second average of the packet loss probability). In addition, Figure 3-10 uses the mean-field approximations developed in the last section and does not show a simulated behavior of each protocol.

The figures show that MPTCP/NC provides a better throughput throughout the "simulated" experiment than MPTCP. While MPTCP is severely hindered by high packet losses as a result of poor channel conditions, MPTCP/NC is able to mask the majority of packet losses and maintain a high throughput. Scheduling of packets on each sub-flow is also easier with MPTCP/NC than with MPTCP due to the network coding operations performed immediately below the application layer. The throughput shown for MPTCP assumes that there is perfect scheduling among the sub-flows with no need to retransmit a packet on more than one sub-flow. This provides a best-case scenario for the achievable throughput. This assumption is not made in MPTCP/NC because each packet transmitted on a sub-flow is viewed as a degree of freedom. If a packet is lost, any packet sent on a different sub-flow can be used in the lost packet's place.

3.4 Extending Coded TCP to Multi-Path Environments

As mentioned earlier within the chapter, MPTCP/NC may not be the best approach for implementing a coded multi-path transport protocol. One of the main reasons for this is the continued use of standard TCP. The Fast-TCP/NC layer is essentially "spoofing" TCP into believing that the underlying network sub-flow is error-free. Instead, an approach similar to CTCP should be used where the congestion control is compatible with coded packet streams. Additionally, it may be preferable not to restrict the information available to each sub-flow when generating redundancy. Redundancy generated in a MPTCP/NC sub-flow only contains linear combinations of packets injected into that sub-flow. This approach is nice if a distinct separation between network layers (i.e., the session and transport layers) is required, but it may also artificially restrict the usefulness of redundant packets when attempting to recover from erasures. An alternative approach that will be assumed in the remainder of the thesis is to allow each sub-flow complete access to all information packets when generating redundancy. This removes any restrictions on the content of each coded packet and allows for much simpler coding schemes and analysis.

With the above considerations in mind, CTCP can be extended to multi-path environments with minimal changes to the protocol described in Chapter 2. Similar to MPTCP/NC, a CTCP sub-flow is setup for each available network path. This allows the congestion control to manage each path separately ensuring fairness with other flows, and redundancy to be injected based on each paths' individual packet loss rates. Unlike MPTCP/NC, each sub-flow has knowledge of every information packet that needs to be delivered. Each sub-flow must then maintain a separate coding window that is used to generate coded packets. This is illustrated in Figure 3-11 for two CTCP sub-flows. CTCP sub-flow A generates redundancy using coding window A while CTCP sub-flow B generates redundancy using coding window B. The major challenge with this setup arises when determining how each sub-flows' coding window should be managed. Chapters 4 and 5 will provide two different approaches that can

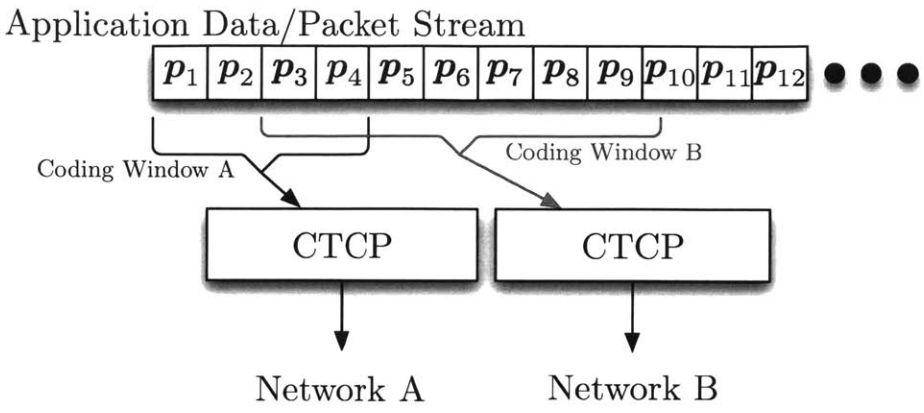


Figure 3-11: Possible Extension of CTCP to Multi-Path Environments

be used. The benefits the two approaches presented is that they both provide excellent erasure correction while maintaining low in-order delivery delay. This is something that MPTCP/NC cannot guarantee because of the multiple coding layers.

3.5 Conclusions

This chapter focused on using network coding for multi-path transport. The experimental measurements provided an example real-world scenario of an environment where exploiting multiple parallel heterogeneous networks is beneficial. The collected data showed a highly dynamic environment with transient connectivity, unpredictable round-trip times, and varying packet loss rates. In order to operate efficiently within this environment, a protocol called MPTCP/NC was introduced that uses two layers of network coding to provide both connection resiliency and error correction. Finally, an analysis of the protocol's throughput was provided; and a fusion of the experimental measurements with the analysis showed the achievable gains of a network coded multi-path transport layer.

While the proposed protocol improves throughput over parallel heterogeneous networks, its actual implementation may not be as advantageous as other approaches. An alternative method is discussed that is an extension of CTCP to the multi-path environment. In this alternative approach, each sub-flow maintains a separate coding

window that operates over a shared collection of information, or application layer, packets. This has many benefits including the added flexibility in code construction, which is the topic for the remainder of the thesis.

Chapter 4

In-Order Delivery Delay for Multi-Path Generation-Based Codes

4.1 Introduction

The results presented in Chapters 2 and 3 showed that transport layer coding can improve throughput over both a single network or multiple parallel networks. However, not all of the gains observed can be fully explained by an increase in throughput alone. A good example are the application layer gains shown in Section 2.4.4. The decreased completion time for small HTTP requests (e.g., 1 kB) and the reduction of streaming video buffer under-runs from approximately 100 to zero are two experiments that specifically highlight gains that are not directly related to throughput. Rather, these gains are most likely attributed to decreases in packet in-order delivery delay. This chapter focuses on quantifying these gains for coding schemes that only code over a single partition of information packets, or a generation, at a time. This is very similar to the type of coding scheme that was used in CTCP, and is a natural approach when considering block codes or rateless codes that require a fixed number of input symbols.

In particular, this chapter explores the use of a systematic random linear network code (RLNC), in conjunction with a coded generalization of SR-ARQ, to help reduce the time needed to recover from losses. Redundancy is added to the original data

stream by injecting coded packets at key locations helping to reduce delay by overcoming packet losses and limiting the number of required retransmissions. Feedback and coded retransmissions are also used to ensure reliability in the event that the client cannot decode a generation.

An example is provided in Figure 4-1. Each column represents an information packet $\boldsymbol{p}_i$ that must be sent, while each row represents the composition of a packet transmitted in the specified time-slot. For instance, the packet transmitted in time-slot 3 is a linear combination of information packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_3$. The double-arrows to the right of the matrix show when each information packet is delivered, in-order, to the application layer. The colors indicate specific generations, and the colored horizontal lines show when feedback about the a given generation is received by the server. As an example, the first generation represented by the blue dots and lines is first transmitted in the first four time-slots. The first two information packets, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, are received by the client and can be immediately delivered. However, both packet $\boldsymbol{p}_3$ and the coded packet comprising of a linear combination of packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_3$ are lost. Since the client cannot decode the generation and recover $\boldsymbol{p}_3$, feedback is sent back to the server requesting additional degrees of freedom ($dofs$) to be sent. This feedback is obtained by the server in time-slot 8 and an additional coded packet from the first generation is retransmitted. Once enough $dofs$ from the first generation are received by the client, the generation can be decoded and all lost information packets can be delivered in-order. This occurs for the first generation in time-slot 16.

The example highlights the first major cause of delay: head-of-line blocking. A reliable transport layer ensures that packets are delivered in-order. Therefore, packet delivery is halted whenever a packet loss occurs; and it does not start again until the loss is corrected. Protocols like TCP typically use selective repeat automatic-repeat-request (SR-ARQ) to recover from any packet erasure that occur. If a packet loss is observed, the client requests that the lost packet be resent. While SR-ARQ helps to ensure high efficiency, one problem with it is that packet recovery due to a loss can take on the order of a round-trip time (RTT) or more [37]. When the RTT (or

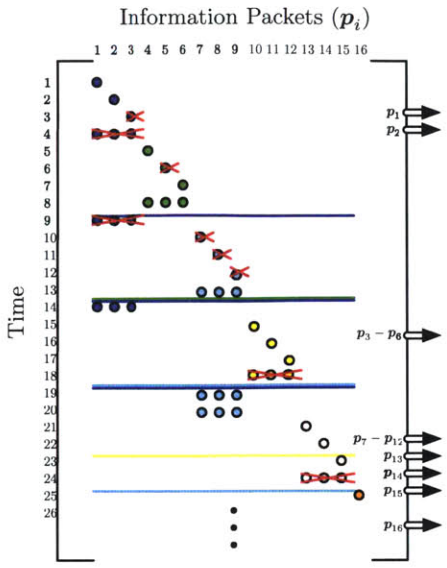


Figure 4-1: An example of the systematic code with feedback considered within this chapter.

more precisely the bandwidth-delay product (BDP)) is very small and feedback is close to being instantaneous, SR-ARQ provides near optimal in-order delivery delay. Unfortunately, feedback is often delayed due to queuing and large physical distances between a client and server. This can have major implications for applications that require reliable delivery with constraints on the time between the transmission and in-order delivery of a packet. As a result, alternatives to SR-ARQ are necessary to meet these constraints.

The second major cause of delay is related to coding. As the generation size k increases, the delay due to coding also increases. This is similar to the head-of-line blocking issue discussed above, but it is on a generation-by-generation basis. As an example, assume that the first packet within a generation is lost and enough redundancy was added to the generation so that it can be decoded without requiring retransmissions (i.e., the lost packet is recovered once enough $dofs$ have been received which occurs near the end of the generation). If k is small, the lost packet will be recovered fairly quickly. However, the time it takes to recover the packet increases as k increases. Unfortunately, the probability of requiring retransmissions also decreases

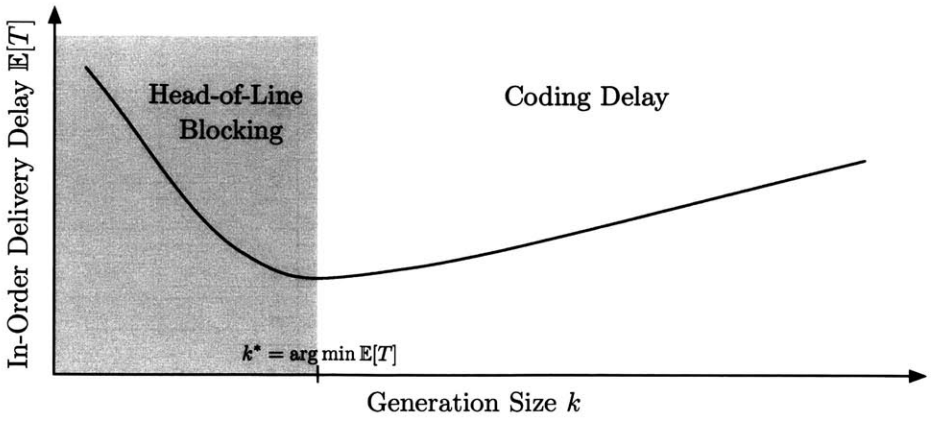


Figure 4-2: The trade-off between decreasing the probability of retransmissions and minimizing the generation size k.

with k. This results in an inherent trade-off that is depicted in Figure 4-2. An excessive amount of retransmissions increase the delay if the generation size is too small, while a generation that is too large increases the delay due to coding.

This chapter will answer the following two questions: how large should each generation be so that the in-order delivery delay is minimized; and how much redundancy should be added to meet a user's requested QoS. These answers will be provided through an analysis of the in-order delivery delay as a function of the generation size and redundancy, where the in-order delivery delay is formally defined as:

Definition 4.1. The in-order delivery delay T is the difference between the time an information packet is first transmitted and the time that the same packet is delivered in-order.

The expected delay $\mathbb{E}[\mathrm{T}]$ and the jitter σ_{T} (i.e., the standard deviation of T) will be the primary metrics of performance. However, decreasing $\mathbb{E}[\mathrm{T}]$ and σ_{T} comes at a price. This price is captured in terms of efficiency η defined by the following:

Definition 4.2. The efficiency η is the ratio between the total number of information packets, or degrees of freedom ($dofs$), transmitted during a network session and the expected number of packets received by the sink.

Numerical results will also be used to help determine the cost (in terms of rate) of reducing the delay and as a tool to help determine the appropriate coding window size for a given network path/link.

4.2 Multi-Path Generation-Based Coding Algorithm

The problem of supporting a single network session operating over multiple paths can be addressed without the use of coding; however, the intent of this chapter is to show that coding can reduce in-order delivery delay more so than conventional uncoded approaches. This section will introduce a simple network coding algorithm to show these gains, without any guarantee regarding its optimality.

Information packets $\boldsymbol{p}_i$, $i = 1, 2, \ldots$, are first partitioned into coding generations $\boldsymbol{G}_{q,j}$, which represents the jth generation transmitted on path $q \in \mathcal{P}$, as they are made available to the source's transport layer. It is assumed that the generation size, k_q, can vary depending on the path, but the size of each generation transmitted on a particular path is constant. Furthermore, packets are inserted into a generation based on the first non-busy network path found. Once enough information packets are available to the encoder to fill an entire generation, random linear combinations of these packets are used to produce coded packets

$$\boldsymbol{c}_{q,j,m} = \sum_{\boldsymbol{p}_i \in \boldsymbol{G}_{q,j}} \alpha_{q,j,m,i} \boldsymbol{p}_i, \tag{4.1}$$

where $m \in [1, {}^{k_q}/_{c_q}\,(1 - c_q)]$, $c_q \leq 1 - \epsilon_q$ is the code rate used on path q, the coding coefficients $\alpha_{q,j,m,i} \in \mathbb{F}_p$ are chosen at random, and each packet $\boldsymbol{p}_i$ is treated as a vector in $\mathbb{F}_p$. Both the information and coded packets are then systematically transmitted. Once a generation has completed transmission, a new generation is produced and the process repeats without waiting for feedback.

Figure 4-3 provides an example of this process using two paths. Packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_3$ are placed in $\boldsymbol{G}_{1,1}$, while packets $\boldsymbol{p}_4$ and $\boldsymbol{p}_5$ are placed in $\boldsymbol{G}_{2,1}$. A single coded packet is then appended to the end of each generation to provide redundancy against packet

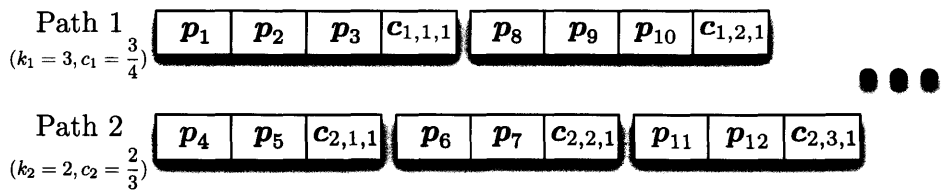


Figure 4-3: Example of the process used to partition packets into generations. Both network paths transmit packets at the same rate; however, the generation size and code rate for each path differ.

erasures. Both generations (including the coded packets) are transmitted in parallel. Once $\boldsymbol{G}_{2,1}$ completes, $\boldsymbol{G}_{2,2}$ (containing packets $\boldsymbol{p}_6$ and $\boldsymbol{p}_7$) is formed and transmitted over the second path. While $\boldsymbol{G}_{2,2}$ is in the process of being transmitted, the first path becomes available allowing generation $\boldsymbol{G}_{1,2}$ (containing packets $\boldsymbol{p}_8$ through $\boldsymbol{p}_{10}$) to be formed and transmitted. This process continues until the entire packet stream or file is partitioned and transmitted.

The sink is able to recover an entire generation $\boldsymbol{G}_{q,j}$ if k_q or more degrees of freedom ($dofs$) from that generation are successfully received. If only $k_q - l$, $l \in (0, k_q]$, packets from $\boldsymbol{G}_{q,j}$ are received, a decoding error occurs and feedback requesting these l additional $dofs$ is sent on the shortest path $\hat{q}$. Upon reception of this feedback, the source retransmits $l/c_{\hat{q}}$ additional coded packets (or $dofs$) from $\boldsymbol{G}_{q,j}$ on path $\hat{q}$. This retransmission process continues for every generation until at least k_q $dofs$ for each $\boldsymbol{G}_{q,j}$ has been obtained by the sink. Once a generation has been successfully decoded, all information packets are buffered at the sink until they can be delivered in-order. This process naturally leads to the concept of *rounds*.

Definition 4.3. The ith round for a single generation begins with the transmission of l_{i-1}/c_q $dofs$ where l_{i-1} is the number of $dofs$ required by the sink at the conclusion of round $i-1$. The ith round ends when feedback is obtained regarding the number of $dofs$ l_i still needed to successfully decode.

Algorithms 1 and 2 provide a concise description of this process. By providing feedback and retransmitting $dofs$ on the shortest path in the manner outlined in both the algorithms, the time it takes to overcome packet losses that cause decode errors

is effectively minimized. As a result, the overall in-order delivery delay will also be reduced.

Algorithm 1: Code Generation

Initialize $i = 1$ and $\boldsymbol{j} = [1, \ldots, 1]^T$ where $\boldsymbol{j} \in \mathbb{R}^{|\mathcal{P}|}$
while *Information packets to send* **do**
  $q \leftarrow$First non-busy path available
  for $m \leftarrow i$ **to** $k_q + i - 1$ **do**
    Add $\boldsymbol{p}_m$ to $\boldsymbol{G}_{q,j_q}$
  for $m \leftarrow 1$ **to** ${}^{k_q}/_{c_q}(1 - c_q)$ **do**
    Add $\boldsymbol{c}_{q,j_q,m} = \sum_{n=i}^{k_q+i-1} \alpha_{q,j_q,m,n}\boldsymbol{p}_n$ to $\boldsymbol{G}_{q,j_q}$
  Begin transmission of $\boldsymbol{G}_{q,j_q}$ and continue
  $j_q \leftarrow j_q + 1$
  $i \leftarrow i + k_q$

Algorithm 2: Retransmission

ACK from $\boldsymbol{G}_{q,j_q}$ received
if *No packets from* $\boldsymbol{G}_{q,j_q}$ *in-flight on either path* q *or* $\hat{q}$ *and* $l > 0$ **then**
  for each $m \in [1, {}^{l}/_{c_{\hat{q}}}]$ **do**
    Transmit $\boldsymbol{c}_{q,j_q,m} = \sum_{\boldsymbol{p}_i \in \boldsymbol{G}_{q,j_q}} \alpha_{q,j_q,m,i}\boldsymbol{p}_i$ on path $\hat{q}$

4.3 System Model

The scenario where a single source and sink are connected via multiple disjoint network paths is considered. A single network session between the two is produced and communication occurs in parallel over each of these paths. While the source and sink may coordinate transmission of information across paths, the underlying networks, or paths $q \in \mathcal{P}$, are treated independently (i.e., the behavior of path q does not affect the behavior of path $q' \neq q$). Delayed feedback between the sink and source is assumed to be available. This feedback is sent over each path and contains information regarding the state of sink. Furthermore, it is assumed that this feedback allows the source to make an accurate estimation of the packet erasure probability ϵ_q, which is assumed to be independent and identically distributed (i.i.d.) on each path q.

Time is slotted where each time-slot's duration, t_q seconds, is determined using the transmission rate $Rate_q$ bits/second on path q (i.e., $t_q = {}^1/_{Rate_q}$). Note that t_q may not be the same as $t_{q'}$ for $q \neq q'$. Once a packet is transmitted, it takes d_q seconds to propagate through the network. Throughout this chapter, the delay on the shortest (or fastest) path $\hat{q} \in \mathcal{P}$ and the longest (or slowest) path $\check{q} \in \mathcal{P}$ will be significant. Both of these paths are formally defined as:

$$\hat{q} = \arg\min_{q \in \mathcal{P}} d_q \tag{4.2}$$

and

$$\check{q} = \arg\max_{q \in \mathcal{P}} d_q \tag{4.3}$$

respectively. Therefore, the round-trip time for a packet transmitted in round 1 on path q is $t_q + d_q + d_{\hat{q}}$ where it is assumed that the size of an acknowledgement is sufficiently small enough to be ignored. Since all retransmissions occur on path $\hat{q}$, the round-trip time for all packets in subsequent rounds is approximately $2d_{\hat{q}}$. Using this model, the entire system can be fully described by the tuple (ϵ_q, t_q, d_q) for each path $q \in \mathcal{P}$.

4.4 Model and Analytical Assumptions

Before proceeding, several simplifying assumptions must be made in order to keep the analysis tractable due to the inherent complexity of the underlying stochastic process. First, the time it takes for the source to schedule retransmissions on path $\hat{q}$ is ignored. Generations requiring additional $dofs$ to decode are given priority by the source so that retransmissions occur immediately after feedback is obtained indicating additional $dofs$ are needed. The second assumption ignores the time time it takes to retransmit these $dofs$. For example, the packet transmission time on path $\hat{q}$ is $t_{\hat{q}}$ seconds. Assuming l dof retransmissions are needed, the additional ${}^{lt_{\hat{q}}}/_{c_{\hat{q}}}$ seconds needed to transmit these packets are not taken into account. Third, the number of previously transmitted generations b_q that can cause head-of-line blocking is limited

to the number of generations that can be transmitted in $2d_{\hat{q}}$ seconds. Assume that a generation is received that was first transmitted on path q. The number of generations that can cause head-of-line blocking is limited to $b_q = \sum_{q' \in \mathcal{P}} b_{q,q'}$, where $b_{q,q'}$ is the number of generations on path q' that can prevent a generation originally sent on path q from being delivered. Formally, $b_{q,q'}$ is defined as:

$$b_{q,q'} = \begin{cases} \left\lceil \frac{2d_{\hat{q}}c_q}{k_q t_q} \right\rceil - 1 & \text{for } q = q', \\ \left\lceil \frac{2d_{\hat{q}}c_q}{k_q t_q} \right\rceil - \left\lceil \left\lceil \frac{2d_{\hat{q}}c_q}{k_q t_q} \right\rceil - \frac{c_q\left(2d_{\hat{q}} - d_{q'} + d_q - 1\right)}{k_q t_q} \right\rceil & \text{for } q \neq q', d_q > d_{q'}, \\ \left\lceil \frac{2d_{\hat{q}}c_q}{k_q t_q} \right\rceil + \left\lfloor \frac{c_q\left(2d_{\hat{q}} + d_{q'} - d_q - 1\right)}{k_q t_q} - \left\lfloor \frac{2d_{\hat{q}}c_q}{k_q t_q} \right\rfloor \right\rfloor & \text{for } q \neq q', d_q \leq d_{q'}. \end{cases} \tag{4.4}$$

It should be noted that b_q is dependent on the path q. Figure 4-4 provides and example consisting of two paths. The number of previously transmitted generations that can cause head-of-line blocking for generation $\boldsymbol{G}_{1,7}$ on path $\hat{q} = 1$ is limited to $b_{1,1} = 5$, while the number of generations on path $\check{q} = 2$ is limited to $b_{1,2} = 7$. Similarly, the number of previous generations that can block generation $\boldsymbol{G}_{2,7}$ from being delivered is limited to $b_{2,1} = 3$ and $b_{2,2} = 5$ on paths $\hat{q} = 1$ and $\check{q} = 2$ respectively.

The first two assumptions ensure feedback is acted upon immediately removing any delay resulting from packet scheduling and retransmission. The third assumption removes the possibility that generations transmitted in earlier rounds cause head-of-line blocking, thereby decreasing the overall delay. These assumptions have a large effect on the deference between the true and analytical delay when the probability of decoding a generation in any given round is small. However, this is not the regime of interest since the primary objective of transport layer coding is to reduce both decode errors and retransmissions.

Several additional assumptions are needed that do not explicitly affect the delay, but they help simplify the modeling and analysis. First, all packets within a generation are available to the transport layer without delay (i.e., an infinite packet source is assumed). Second, feedback regarding the outcome of any transmitted generation is

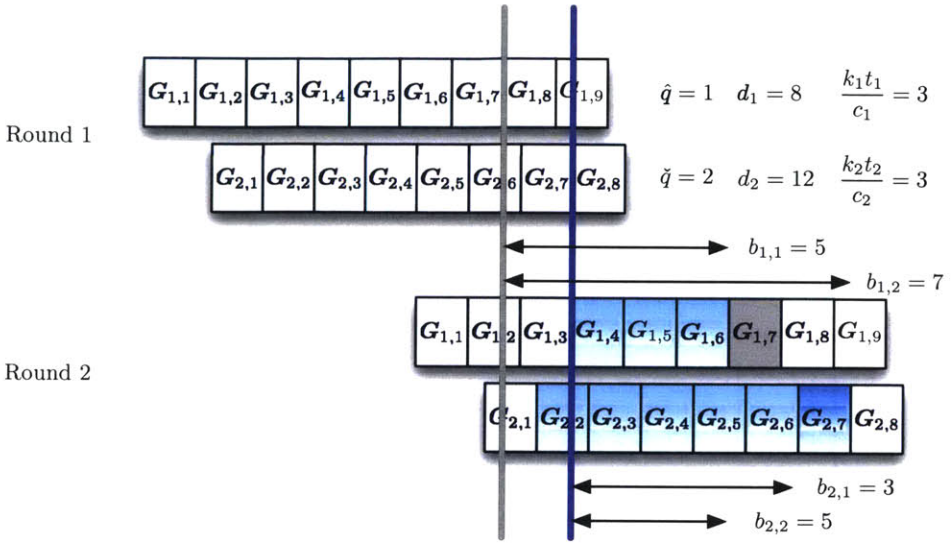


Figure 4-4: Example of the number of previously transmitted generation that can create head-of-line blocking assuming two paths. The figure shows the reception times of each generation over two rounds.

only provided after the entire generation has been sent. In other words, the feedback delay must be larger than the time it takes to transmit an entire generation on the slowest path, i.e.,

$$2d_{\hat{q}} > \max_{j \in \mathcal{P}} k_j t_j / c_j, \tag{4.5}$$

where $\hat{q}$ is defined in (4.2), t_j is the time-slot size on path j, k_j is the generation size, and c_j is the code rate. Without this assumption, feedback regarding the outcome of packets within a generation will be received by the source prior to the reception of coded packets by the sink. Therefore, selective repeat ARQ (SR-ARQ) can be used without a large impact to the performance. Third, the following assumption is made regarding the time it takes to transmit an entire generation on each path:

$$k_i t_i / c_i = k_j t_j / c_j, \text{ for all } i, j \in \mathcal{P}. \tag{4.6}$$

This assumption states that the time it takes to transmit a single generation is the same regardless of path and it eliminates the added complication of generation sequencing among all of the paths. Fourth, the source services longer paths prior to

shorter paths. For example if $d_i > d_j$ and both are not busy, packets $\boldsymbol{p}_m, \cdots, \boldsymbol{p}_{m+k_i}$ are partitioned into a generation that will be transmitted on path i and packets $\boldsymbol{p}_{m+k_i+1}, \cdots, \boldsymbol{p}_{m+k_i+k_j}$ are partitioned into a generation that will be transmitted on path j. Finally, a unique longest path $\check{q} \in \mathcal{P}$ is assumed since the longest path dictates the delivery of packets. A unique longest path helps simplify the analysis by avoiding sequencing issues.

4.5 Required Probability Models and Distributions

A number of random variables are required to accurately describe the delivery process. These random variables can be broken up into two categories. The first category involves variables that describe the rounds that a generation is delivered. In general, these variables give a coarse idea of the delay experienced by an information packet. The second category of variable helps provide a more accurate idea of the delay given a generation can be delivered in a specific round.

4.5.1 Primary Models and Distributions

Before these random variables are formally defined, it is important to note that all k_q/c_q packets contained in a generation $\boldsymbol{G}_{q,j}$ are first transmitted on path q, while any retransmissions are made on path $\hat{q}$. Specifically, erasures occur with probability ϵ_q in round 1 and probability $\epsilon_{\hat{q}}$ for all subsequent rounds. This process can be described using the Markov chain shown in Figure 4-5 where the initial state at time 0 is S and a generation is successfully decoded once the trapping state 0 is entered for the first time.

Formally, define X_{r-1} to be the state of the chain in round $r > 0$ and $[M_q] \in \mathbb{R}^{(k_q+1)\times(k_q+1)}$ to be the chain's transition probability matrix. Let the state X_0 after

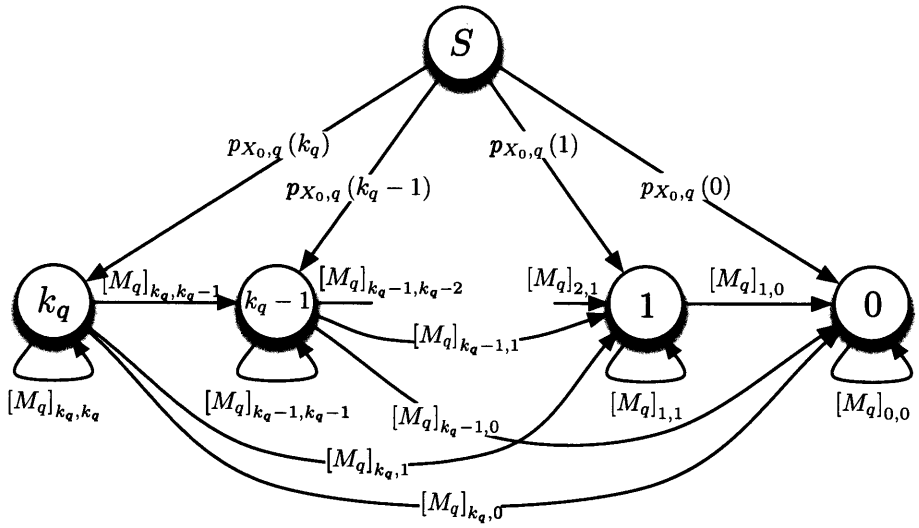


Figure 4-5: Markov chain model that describes the process of delivering a single generation.

round $r = 1$ be determined according to the distribution:

$$p_{X_0,q}(x_0) = \begin{cases} B\left(k_q/c_q, k_q - x_0, \epsilon_q\right) & \text{for } x_0 \in (0, k_q], \\ \sum_{m=k_q}^{k_q/c_q} B\left(k_q/c_q, m, \epsilon_q\right) & \text{for } x_0 = 0, \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$

where $B(n,k,p) = \binom{n}{k}(1-p)^k p^{n-k}$. Also define the elements of $[M_q]$ as follows:

$$[M_q]_{ij} = \begin{cases} B\left(i/c_{\hat{q}}, i-j, \epsilon_{\hat{q}}\right) & \text{for } i \in [1, k_q], j \in (0, i], \\ \sum_{m=i}^{i/c_{\hat{q}}} B\left(i/c_{\hat{q}}, m, \epsilon_{\hat{q}}\right) & \text{for } i \in [1, k_q], j = 0, \\ 1 & \text{for } i = 0, j = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

It follows that for $j \in [0, k_q]$ and integer $r > 0$, the probability of being in state X_r

in round r is:

$$p_{X_r,q}(x_r) = \begin{cases} p_{X_0,q}(x_r) & \text{for } r = 1, \\ \sum_{x_0=0}^{k_q} p_{X_0,q}(x_0) \left[M_q^{r-1}\right]_{x_0 x_r} & \text{for } r > 1, \\ 0 & \text{otherwise,} \end{cases} \tag{4.9}$$

for $x_0, x_r \in [0, k_q]$ and $\left[M_q^0\right]_{ij} = 0$ for all i, j. Note that state 0 is a trapping state; and the probability of being in state 0 on or before round r is $p_{X_r,q}(0)$.

Within the model, a generation is successfully decoded when state $X_r = 0$ is first entered. Define Y_q to be the round when this occurs (i.e., $X_{Y_q} = 0$). This variable has the following distribution:

$$p_{Y_q}(y) = \begin{cases} p_{X_0,q}(0) & \text{for } y = 1, \\ p_{X_y,q}(0) - p_{X_{y-1},q}(0) & \text{for } y \geq 2, \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

Next define $Z_{q,q'}$, $q, q' \in \mathcal{P}$, to be the number of transmission rounds required to successfully transfer all $b_{q,q'}$ generations first transmitted on path q'. The distribution of $Z_{q,q'}$ is provided by the following lemma.

Lemma 4.4. *Let $b_{q,q'}$ independent processes, each defined by (4.7) and (4.8), start at the same time on path q'. The probability that all processes complete in less than or equal to z rounds with at least one process completing in round z is*

$$p_{Z_{q,q'}}(z) = \begin{cases} p_{X_0,q'}^{b_{q,q'}}(0) & \text{for } z = 1, \\ p_{X_z,q'}^{b_{q,q'}}(0) - p_{X_{z-1},q'}^{b_{q,q'}}(0) & \text{for } z \geq 2, \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

Proof. Let $f(z) = 1 - \sum_{j=1}^{z-1} p_{Y_{q'}}(j)$. The probability of $b_{q,q'}$ independent processes completing in less than or equal to z rounds with at least one process completing in

round z is:

$$Pr\left\{Z_{q,q'}=z\right\} = \sum_{i=1}^{b_{q,q'}} B\left(b_{q,q'},i,f\left(z\right)\right)\left(\frac{p_{Y_{q'}}\left(z\right)}{f\left(z\right)}\right)^{i} \tag{4.12}$$

$$= \sum_{i=1}^{b_{q,q'}} \binom{b_{q,q'}}{i}\left(p_{Y_{q'}}\left(z\right)\right)^{i}\left(\sum_{j=1}^{z-1} p_{Y_{q'}}\left(j\right)\right)^{b_{q,q'}-i}. \tag{4.13}$$

For $z=1$, (4.13) becomes $Pr\left\{Z_{q,q'}=1\right\}=\left(p_{X_0,q'}\left(0\right)\right)^{b_{q,q'}}$. For $z\geq 2$, (4.13) becomes:

$$Pr\left\{Z_{q,q'}=z\right\} = \sum_{i=1}^{b_{q,q'}} \binom{b_{q,q'}}{i}\left(p_{Y_{q'}}\left(z\right)\right)^{i}\left(\sum_{x_0=0}^{k_{q'}} p_{X_0,q'}\left(x_0\right)\left[M_{q'}^{z-2}\right]_{x_0 0}\right)^{b_{q,q'}-i} \tag{4.14}$$

$$= \left(\sum_{x_0=0}^{k_{q'}} p_{X_0,q'}\left(x_0\right)\left[M_{q'}^{z-1}\right]_{x_0 0}\right)^{b_{q,q'}} - \left(\sum_{x_0=0}^{k_{q'}} p_{X_0,q'}\left(x_0\right)\left[M_{q'}^{z-2}\right]_{x_0 0}\right)^{b_{q,q'}}. \tag{4.15}$$

□

The distribution on $Z_{q,q'}$ can now be used to determine the total number of rounds it takes to transfer all b_q generations that can cause head-of-line blocking. Define this random variable as

$$Z_q = \max_{q'\in\mathcal{P}} Z_{q,q'}. \tag{4.16}$$

The distribution on Z_q is then defined by the following lemma:

Lemma 4.5. *Let $Z_q=\max_{q'\in\mathcal{P}} Z_{q,q'}$. The distribution on Z_q is:*

$$p_{Z_q}\left(z\right)=\begin{cases}\prod_{q'\in\mathcal{P}} p_{X_0,q'}^{b_{q,q'}}\left(0\right) & \text{for } z=1,\\ \prod_{q'\in\mathcal{P}} p_{X_z,q'}^{b_{q,q'}}\left(0\right)-\prod_{q'\in\mathcal{P}} p_{X_{z-1},q'}^{b_{q,q'}}\left(0\right) & \text{for } z\geq 2,\\ 0 & \text{otherwise.}\end{cases} \tag{4.17}$$

Proof. For $Z_{q,q'}=1$, the CDF is $Pr\left\{Z_{q,q'}=1\right\}=p_{X_0,q'}^{b_{q'}}\left(0\right)$. For $Z_{q,q'}\geq 2$, the CDF

becomes:

$$
\begin{aligned}
&Pr\left\{Z_{q,q'} \leq z\right\} \\
&\qquad = \sum_{i=1}^{z} p_{z_{q,q'}}(i) && (4.18)\\
&\qquad = p_{X_{0,q'}}^{b_{q,q'}}(0) + \sum_{i=2}^{z} \left(p_{X_i,q'}^{b_{q,q'}}(0) - p_{X_{i-1},q'}^{b_{q,q'}}(0)\right) && (4.19)\\
&\qquad = p_{X_z,q'}^{b_{q,q'}}(0). && (4.20)
\end{aligned}
$$

Let $Z_q = \max_{q' \in \mathcal{P}} Z_{q,q'}$. Its CDF is:

$$
\begin{aligned}
Pr\left\{Z_q \leq z\right\} &= Pr\left\{Z_{q,1} \leq z, \ldots, Z_{q,|\mathcal{P}|} \leq z\right\} && (4.21)\\
&= Pr\left\{Z_{q,1} \leq z\right\} \cdots Pr\left\{Z_{q,|\mathcal{P}|} \leq z\right\} && (4.22)
\end{aligned}
$$

When $Z_q = 1$, $Pr\left\{Z_q = 1\right\} = \prod_{q' \in \mathcal{P}} p_{X_{0,q'}}^{b_{q,q'}}(0)$. For $Z_q \geq 2$, $Pr\left\{Z_q \leq z\right\} = \prod_{q' \in \mathcal{P}} p_{X_z,q'}^{b_{q,q'}}(0)$. This results in the PDF $p_{Z_q}(z) = Pr\left\{Z_q \leq z\right\} - Pr\left\{Z_q \leq z-1\right\}$. □

4.5.2 Secondary Models and Distributions

Both Y_q and Z_q will be used to determine whether or not head-of-line blocking is preventing packet delivery of a given generation. Both of these variables provide a coarse idea of the delay. To provide a more accurate estimate of the delay, additional random variables are needed. These variables will help describe the location of the packet or generation that is preventing delivery within a given round.

Define the first of these variables, S_q, to be the location of the first packet erasure within a single generation. This variable is important for the following reason. Assume that $Y_q \geq Z_q$. The first S_q packets within a generation can be delivered in round Z_q experiencing a delay on the order of Z_q rounds. However, the remaining $k_q - S_q$ packets within the generation cannot be delivered until the entire generation is decoded in round Y_q. As a result, these packets will experience a delay on the order of Y_q round. An example is shown in Figure 4-6. The generation consists of packets

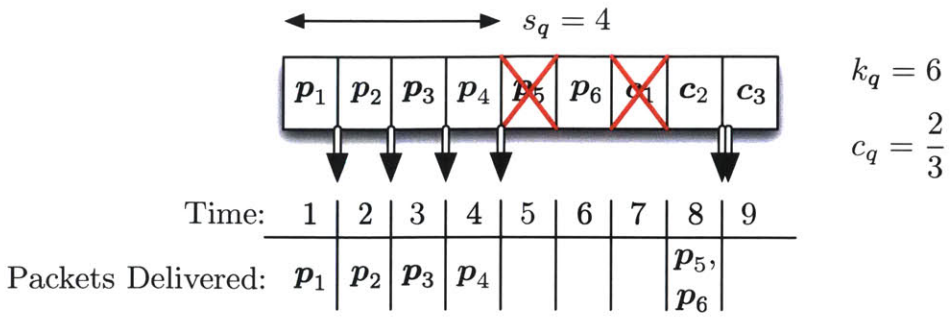


Figure 4-6: Example of the in-order delivery of packets within a single generation where coded packets help recover from packet erasures.

$\boldsymbol{p}_1$ through $\boldsymbol{p}_6$. The first packet erasure occurs when $\boldsymbol{p}_5$ is lost. As a result, the first four packets (i.e., $s_q = 4$) can be delivered immediately upon reception. In order to recover from the erasure, a single *dof* is required and provided by packet $\boldsymbol{c}_2$. Once this is obtained, the entire generation is decoded allowing $\boldsymbol{p}_5$ to be recovered and delivered.

The distribution on S_q given Y_q is:

$$p_{S_q|Y_q}(s|y) = \begin{cases} \epsilon_q (1-\epsilon_q)^s & \text{for } s \in [0, k_q - 1], y = 1, \\ (1-\epsilon_q)^s & \text{for } s = k_q, y = 1, \\ \frac{\epsilon_q (1-\epsilon_q)^s}{1-(1-\epsilon_q)^{k_q}} & \text{for } s \in [0, k_q - 1], y \neq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.23}$$

For $Y_q = 1$, a packet loss may or may not occur. However for $Y_q > 1$, a packet loss is known to have occurred. This results in the normalized geometric distribution shown above. It is important to note that S_q is determined during the first transmission round. Therefore, it is dependent on the path $q \in \mathcal{P}$ on which the generation was originally sent. Within the subsequent analysis, the first three moments of S_q will be of particular interest. These are given by the following lemma.

Lemma 4.6. *Define* $\overline{s^i_{1,q}} = \mathbb{E}\left[S^i_q | Y_q = 1\right]$ *and* $\overline{s^i_{2,q}} = \mathbb{E}\left[S^i_q | Y_q \neq 1\right]$. *Then given*

$Y_q = y$, *the first three moments of* S_q *are*

$$\overline{s^1_{1,q}} = \frac{1-\epsilon_q}{\epsilon_q}\left(1-(1-\epsilon_q)^{k_q}\right), \tag{4.24}$$

$$\overline{s^2_{1,q}} = \frac{2(1-\epsilon_q)}{\epsilon_q^2}\left(1-(k_q\epsilon_q+1)(1-\epsilon_q)^{k_q}\right) - \overline{s^1_{1,q}}, \tag{4.25}$$

$$\begin{aligned}\overline{s^3_{1,q}} = {} & \frac{6(1-\epsilon_q)^3}{\epsilon_q^3}\left(1-(k_q\epsilon_q+1)(1-\epsilon_q)^{k_q}\right) + (4-3\epsilon_q)\,\overline{s^1_{1,q}} + 3(1-\epsilon_q)\,\overline{s^2_{1,q}} \\ & - \frac{3k_q}{\epsilon_q}(k_q+1)(1-\epsilon_q)^{k_q+1},\end{aligned} \tag{4.26}$$

and

$$\overline{s^i_{2,q}} = \frac{\overline{s^i_{1,q}} - k_q^i(1-\epsilon_q)^{k_q}}{1-(1-\epsilon_q)^{k_q}}, \tag{4.27}$$

for $i = 1, 2, 3$.

Proof. Define the moment generating function of S_q when $Y = 1$ to be

$$\begin{aligned} M_{S_q|Y_q}(t) &= \mathbb{E}[e^{tS_q}|Y_q = 1] && (4.28)\\ &= \frac{\epsilon_q\left(1-e^{k_q t}(1-\epsilon_q)^{k_q}\right)}{1-e^t+\epsilon_q e^t} + e^{k_q t}(1-\epsilon_q)^{k_q}. && (4.29)\end{aligned}$$

The first, second, and third moments of S_q when $Y_q = 1$ are then $\delta/\delta t M_{S_q|Y_q}(0)$, $\delta^2/\delta t^2 M_{S_q|Y_q}(0)$, and $\delta^3/\delta t^3 M_{S_q|Y_q}(0)$ respectively. For $Y_q \neq 1$, scale the above expectations by subtracting the term $k_q^i(1-\epsilon_q)^{k_q}$ from each of the moments above and dividing by $1-(1-\epsilon_q)^{k_q}$. □

The second random variable that helps describe the location of the head-of-line blocking packet or generation is $V_{q,q'}$. This variable describes the position of the last received generation originally sent on path $q' \in \mathcal{P}$ preventing delivery in round $Z_{q,q'}$. The following lemma helps define the distribution on $V_{q,q'}$ given $Z_{q,q'} > 1$.

Lemma 4.7. *Let $b_{q,q'}$ independent processes, each defined by (4.7) and (4.8), start at the same time and all complete in or before round $Z_{q,q'} = z$ with at least one process completing in round $Z_{q,q'} = z$. The probability that the jth process, $j = b_{q,q'} - V_{q,q'}$, is the last to complete is*

$$p_{V_{q,q'}|Z_{q,q'}}(v|z) = \frac{p_{X_z,q'}^{b_{q,q'}-v-1}(0)\, p_{X_{z-1},q'}^{v}(0)\, p_{Y_{q'}}(z)}{p_{Z_{q,q'}}(z)}, \tag{4.30}$$

for $V_{q,q'} = 0, \ldots, b_{q,q'} - 1$. Furthermore, define $\overline{v_{q,q'}^{i}(z)} = \mathbb{E}\left[V_{q,q'}^{i}|Z_{q,q'} = z\right]$. Then

$$\overline{v_{q,q'}^{1}(z)} = \frac{p_{X_{z-1},q'}(0)}{p_{Y_{q'}}(z)} - \frac{b_{q,q'} p_{X_{z-1},q'}^{b_{q,q'}}(0)}{p_{Z_{q,q'}}(z)}, \tag{4.31}$$

and

$$\overline{v_{q,q'}^{2}(z)} = \frac{p_{X_{z-1},q'}(0)}{p_{Y_{q'}}(z)} + \frac{2p_{X_{z-1},q'}^{2}(0)}{p_{Y_{q'}}^{2}(z)} - \frac{b_{q,q'}^{2} p_{X_{z-1},q'}^{b_{q,q'}}(0)}{p_{Z_{q,q'}}(z)} - \frac{2b_{q,q'} p_{X_{z-1},q'}^{b_{q,q'}+1}(0)}{p_{Z_{q,q'}}(z)\, p_{Y_{q'}}(z)}. \tag{4.32}$$

Proof. Define $\beta_z = Pr\{Y_{q'} = z | Y_{q'} \leq z\} = {}^{p_{Y_{q'}}(z)}/_{p_{X_z,q'}(0)}$. The distribution on $V_{q,q'} \in [0, b_{q,q'} - 1]$ for $Z_{q,q'} \geq 2$ is

$$p_{V_{q,q'}|Z_{q,q'}}(v_{q,q'}|z_{q,q'}) = \frac{\beta_z (1-\beta_z)^{v_{q,q'}}}{\sum_{j=0}^{b_{q,q'}-1} \beta_z (1-\beta_z)^{j}} \tag{4.33}$$

$$= \frac{\beta_z (1-\beta_z)^{v_{q,q'}}}{1-(1-\beta_z)^{b_{q,q'}}} \tag{4.34}$$

$$= \frac{p_{X_z,q'}^{b_{q,q'}-v_{q,q'}-1}(0)\, p_{X_{z-1},q'}^{v_{q,q'}}(0)\, p_{Y_q'}(z)}{p_{Z_{q,q'}}(z)}. \tag{4.35}$$

Define the moment generating function of $V_{q,q'}$ given $Z_{q,q'}$ to be

$$M_{V_{q,q'}|Z_{q,q'}}(t) = \mathbb{E}\left[e^{tV_{q,q'}}|Z_{q,q'} = z\right] \tag{4.36}$$

$$= \frac{\beta_z}{1-(1-\beta_z)^{b_{q,q'}}} \left(\frac{1 - e^{b_{q,q'}t}(1-\beta_z)^{b_{q,q'}}}{1 - e^{t}(1-\beta_z)} \right). \tag{4.37}$$

The first two moments of $V_{q,q'}$ given $Z_{q,q'} = z$ are determined by ${}^{\delta}/_{\delta t} M_{V_{q,q'}|Z_{q,q'}}(0)$ and

$\delta^2/\delta t^2 M_{V_{q,q'}|Z_{q,q'}}(0)$. □

Finally, each path may potentially have different delays. To overcome this hurdle, define an auxiliary variable $W_{q,q'}$ for each $q' \in \mathcal{P}$. This variable helps determine the delay a generation causing head-of-line blocking on path q' has on generations originally transmitted on path q. Formally,

$$W_{q,q'} = \begin{cases} d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}}(V_{q,q'}+1) & \text{for } d_{q'} \leq d_q, \\ d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}} V_{q,q'} & \text{for } d_{q'} > d_q, \end{cases} \tag{4.38}$$

where the assumption that $k_q t_q/c_q = k_{q'}t_{q'}/c_{q'}$ is used to simplify the equation. This variable is the difference in time between the delivery of the last generation containing a packet $\boldsymbol{p}_j$, $j < i$, in round Z_q and the time the generation containing packet $\boldsymbol{p}_i$ can be delivered assuming that its generation is also decoded in round Z_q. It is important to note that the values of $W_{q,q'}$ for paths where $Z_{q,q'} < Z_q$ do not need to be considered since these paths are not preventing delivery. Therefore, the path/generation that is causing head-of-line blocking is the one with minimum $W_{q,q'}$ on a path where $Z_{q,q'} = Z_q$. The expectation $\mathbb{E}\left[\min_{q' \in \{\mathcal{P}: Z_{q,q'} = Z_q\}} W_{q,q'} | Z_{q,q'} = Z_q\right]$ is provided by the following lemma.

Lemma 4.8. *Let* $W_{\min,q} = \min_{q' \in \{\mathcal{P}: Z_{q,q'} = Z_q\}} W_{q,q'}$. *Then the following holds:*

$$\mathbb{E}\left[W_{\min,q} | Z_q\right] \leq \max_{q' \in \mathcal{P}} \mathbb{E}\left[d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}(z)} + 1\right)\right]. \tag{4.39}$$

Proof. Label each path $\mathcal{P} = \{1, 2, \ldots, |\mathcal{P}|\}$. The joint probability distribution for the number of rounds each path takes to complete given $Z_q = z_q$ is

$$p_{Z_{q,1},\ldots,Z_{q,|\mathcal{P}|}|Z_q}\left(z_{q,1}, \ldots, z_{q,|\mathcal{P}|} | z_q\right) = \frac{1}{p_{Z_q}(z_q)} \prod_{q' \in \mathcal{P}} p_{Z_{q,q'}}(z_{q,q'}), \tag{4.40}$$

for $q, q' \in \mathcal{P}$, $z_{q,q'} \in [1, z_q]$, and at least one $z_{q,q'} = z_q$. Also define

$$\alpha = \frac{\text{Number of paths with } d_{q'} \leq d_q}{\text{Total number of paths}}, \tag{4.41}$$

where $0 \leq \alpha \leq 1$. The expectation of $W_{\min,q}$ given $Z_q = z_q$ is given by

$$\mathbb{E}\left[W_{\min,q}|Z_q\right] = \sum_{z_{q,1}=1}^{z_q} \cdots \sum_{z_{q,|\mathcal{P}|}=1}^{z_q} p_{Z_{q,1},\ldots,Z_{q,|\mathcal{P}|}|Z_q}\left(z_{q,1},\ldots,z_{q,|\mathcal{P}|}|z_q\right) \cdot \mathbb{E}\left[\min_{q' \in \left\{\mathcal{P}:z_{q,q'}=z_q\right\}} W_{q,q'}|Z_{q,q'} = Z_q\right]. \tag{4.42}$$

Using Jensen's inequality, the expectation in the right-hand side of the above equation becomes

$$\mathbb{E}\left[\min_{q' \in \left\{\mathcal{P}:z_{q,q'}=z_q\right\}} W_{q,q'}|Z_{q,q'} = Z_q\right] \leq \min_{q' \in \left\{\mathcal{P}:z_{q,q'}=z_q\right\}} \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q\right] \tag{4.43}$$

$$\leq \max_{q' \in \left\{\mathcal{P}:z_{q,q'}=z_q\right\}} \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q\right] \tag{4.44}$$

$$\leq \max_{q' \in \mathcal{P}} \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q\right]. \tag{4.45}$$

Inserting (4.45) into (4.42), we obtain the following

$$\mathbb{E}\left[W_{\min,q}|Z_q\right] \leq \sum_{z_{q,1}=1}^{z_q} \cdots \sum_{z_{q,|\mathcal{P}|}=1}^{z_q} p_{Z_{q,1},\ldots,Z_{q,|\mathcal{P}|}|Z_q}\left(z_{q,1},\ldots,z_{q,|\mathcal{P}|}|z_q\right) \cdot \max_{q' \in \mathcal{P}} \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q\right] \tag{4.46}$$

$$= \max_{q' \in \mathcal{P}} \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q\right]. \tag{4.47}$$

Finally, we substitute (4.38) for $W_{q,q'}$:

$$\mathbb{E}\left[W_{\min,q}|Z_q\right] \leq \max_{q' \in \mathcal{P}} \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q\right] \tag{4.48}$$

$$= \max_{q' \in \mathcal{P}} \Big(\alpha \mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q, d_{q'} \leq d_q\right] + (1-\alpha)\,\mathbb{E}\left[W_{q,q'}|Z_{q,q'} = Z_q, d_{q'} > d_q\right] \Big) \tag{4.49}$$

$$= \max_{q' \in \mathcal{P}} \left(d_q - d_{q'} + \frac{k_{q'} t_{q'}}{c_{q'}} \left(\overline{v^1_{q,q'}(z)} + \alpha \right) \right) \tag{4.50}$$

$$\leq \max_{q' \in \mathcal{P}} \left(d_q - d_{q'} + \frac{k_{q'} t_{q'}}{c_{q'}} \left(\overline{v^1_{q,q'}(z)} + 1 \right) \right). \tag{4.51}$$

□

Now that Y_q, Z_q, S_q, $V_{q,q'}$, and $W_{\min,q}$ are defined, they can be used to determine the first two moments of the in-order delivery delay. Additional random variables will be required to do this, but they are relatively unimportant and will be defined when they are needed.

4.6 The In-Order Delivery Delay's First Two Moments

This section's focus is on deriving the expected in-order delivery delay $\mathbb{E}[\mathrm{T}]$ and its variance σ_T^2. This will be accomplished by determining the delay moments of information packets first transmitted on each path separately. Once these moments are established, they will be averaged over all of the paths to find $\mathbb{E}[\mathrm{T}]$. The primary tool that will be used to find each of these moments is the law of total expectation:

$$\mathbb{E}\left[\mathrm{T}_q^i\right] = \mathbb{E}_{Y_q}\left[\mathbb{E}_{Z_q}\left[\mathbb{E}_{\mathrm{T}_q}\left[\mathrm{T}_q^i | Y_q, Z_q\right]\right]\right], \tag{4.52}$$

where $i \geq 1$.

Because of the differences in generation size, code rate, propagation delay, etc. between each path in $\mathcal{P}$, the value of $\mathbb{E}\left[\mathrm{T}_q^i\right]$ will differ. The slowest path $\check{q}$ will create head-of-line blocking and bulk delivery of generations sent on faster paths. Therefore, the delay of packets sent on the slowest path $\check{q}$ and on faster paths must be derived separately to account for these differences.

The remainder of this section will step through the four distinct cases that are possible. Each case is defined by the number of rounds, Y_q, it takes for a generation on path q to be decoded and the number of rounds, Z_q, it takes for all b_q generations that can cause head-of-line blocking to complete. To help simplify notation going forward, define

$$\bar{\tau}_{Y_q,z_q}^i = \mathbb{E}\left[\mathrm{T}_q^i | Y_q, Z_q\right] \tag{4.53}$$

for $i = \{1, 2\}$.

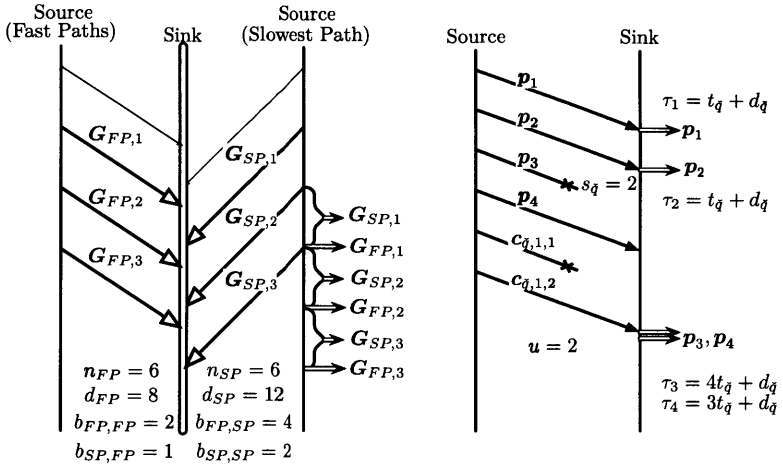
4.6.1 Case 1: $Y_q = Z_q = 1$

Consider a generation containing packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+k_q}$ first transmitted on path q. Assume that this generation completes within the first round of transmission and all generations containing packets $\boldsymbol{p}_j$, $j < i$, also complete in the first round. First, consider packets sent on the slowest path $\check{q}$. There are no information packets causing head-of-line blocking (i.e., all information packets $\boldsymbol{p}_j$, $j < i$, have been delivered). Therefore, all packets received prior to the first loss (i.e., packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+s_{\check{q}}}$) are immediately delivered. Once a packet loss is observed, packets received after the loss (i.e., packets $\boldsymbol{p}_{i+s_{\check{q}}+2}, \ldots \boldsymbol{p}_{i+k_{\check{q}}}$) are buffered until the entire generation can be decoded. Since $Y_q = 1$, this also occurs in the first round.

While packets sent on the slowest path can be immediately delivered upon reception, packets sent on a faster path q (i.e., $d_q > d_{\check{q}}$) are delayed until the slower path completes. This creates bulk packet deliveries. If $d_{\check{q}} - d_q < k_q t_q$, the first s_q packets are delivered once the slowest path's generation is decoded, and the remaining packets are delivered once the generation sent on path q is decoded. Otherwise, all packet in the generation sent on q are delivered together once the slowest path's generation is completed.

An example is given in Figure 4-7. Subfigure (a) shows the transfer of generations over multiple paths from a single source to a single sink. The left side represents a fast path (FP), and the right side represents the slowest path (SP). The single vertical lines represent the single source, while the double vertical line represents the single sink. Subfigure (b) shows the delivery process for packets transmitted on the slowest path. The double arrows to the right of each subfigure represent the time when a specific packet or generation is delivered to the application layer at the sink. Furthermore, the variables listed in the figure are defined as $n_q = {}^{k_q t_q}/_{c_q}$ and the delay of each packet τ_i.

Packets are placed into generations in the following order: $\boldsymbol{G}_{SP,1}$, $\boldsymbol{G}_{FP,1}$, $\boldsymbol{G}_{SP,2}$, $\boldsymbol{G}_{FP,2}$, etc. Packets in $\boldsymbol{G}_{SP,1}, \boldsymbol{G}_{SP,2}$, etc. can be immediately delivered when they are



(a) In-Order Delivery with Multiple Paths

(b) In-Order Delivery for the Slowest Path

Figure 4-7: Case 1: $Y_q = Z_q = 1$. Packets sent on the slowest path can be immediately delivered, while packets sent on faster paths are delayed.

received since there are no generations creating head-of-line blocking. The delay of these packets is shown in Figure 4-7(b) where ${}^{k_{\check{q}}}/_{c_{\check{q}}} = 6$, $k_{\check{q}} = 4$, the number of packets received prior to the first loss is $s_{\check{q}} = 2$, and the number of coded packets needed to recover from the two packet losses is $u = 2$. However, packets sent on faster paths in generations $\boldsymbol{G}_{FP,1}$, $\boldsymbol{G}_{FP,2}$, etc. are delayed due to the slower paths.

The mean delay for packets first transmitted on the slowest path is given by the following:

$$\bar{\tau}_{Y_{\check{q}}=Z_{\check{q}}=1} = \sum_{s=0}^{k_{\check{q}}-1} \left(\left(t_{\check{q}} + d_{\check{q}}\right) \frac{s}{k_{\check{q}}} + \frac{1}{k_{\check{q}}} \sum_{i=0}^{k_{\check{q}}-s-1} \left(\left(k_{\check{q}} - s - i + \mathbb{E}\left[U|S_{\check{q}}\right]\right) t_{\check{q}} + d_{\check{q}} \right) \right) p_{S_{\check{q}}|Y_{\check{q}}}\left(s|1\right)$$
$$+ \left(t_{\check{q}} + d_{\check{q}}\right) p_{S_{\check{q}}|Y_{\check{q}}}\left(k_{\check{q}}|1\right) \tag{4.54}$$

$$= d_{\check{q}} + \frac{t_{\check{q}}}{2k_{\check{q}}} \left(\overline{s_{1,\check{q}}^2} - \left(2k_{\check{q}} - 1\right) \overline{s_{1,\check{q}}^1} + k_{\check{q}}\left(k_{\check{q}} + 1\right) \right.$$
$$\left. + 2 \sum_{s=0}^{k_{\check{q}}-1} \left(k_{\check{q}} - s\right) \mathbb{E}\left[U|S_{\check{q}}\right] p_{S_{\check{q}}|Y_{\check{q}}}\left(s|1\right) \right), \tag{4.55}$$

where the expectation over all $S_{\check{q}}$ and all packets within the generation has been taken, and the definitions of $\overline{s^1_{1,\check{q}}}$ and $\overline{s^2_{1,\check{q}}}$ are given by Lemma 4.6. The expectation $\mathbb{E}[U|S_q]$ in the above equation is the expected number of coded packets needed to recover from all packet erasures occurring in the first $k_{\check{q}}$ packets. Assuming that $\mathbb{E}[U|S_q] = 1$, the above equation can be bounded:

$$\bar{\tau}_{Y_{\check{q}}=Z_{\check{q}}=1} \geq d_{\check{q}} + \frac{t_{\check{q}}}{2k_{\check{q}}}\Bigg(\overline{s^2_{1,\check{q}}} - (2k_{\check{q}}-1)\,\overline{s^1_{1,\check{q}}} + k_{\check{q}}\,(k_{\check{q}}+1) + 2\sum_{s=0}^{k_{\check{q}}-1}(k_{\check{q}}-s)\,p_{S_{\check{q}}|Y_{\check{q}}}(s|1)\Bigg) \tag{4.56}$$

$$= \frac{t_{\check{q}}}{2k_{\check{q}}}\left(\overline{s^2_{1,\check{q}}} - \overline{s^1_{1\check{q}}}\,(2k_{\check{q}}+1) + k_{\check{q}}\,(k_{\check{q}}+3)\right) + d_{\check{q}}. \tag{4.57}$$

The components of the delay are also shown in (4.54). The first term provides the delay when a packet loss is observed within the first $k_{\check{q}}$ packets. Within this term, the first s packets can be immediately delivered (i.e., they each experience a delay of $t_{\check{q}} + d_{\check{q}}$). The packets following the first loss are buffered until the entire generation is decoded incurring a delay of $(k_{\check{q}} - s - i + \mathbb{E}[U|S_{\check{q}}])\,t_{\check{q}} + d_{\check{q}}$. When $s_{\check{q}} < k_{\check{q}}$, the number of coded packets required is at least one (i.e., $\mathbb{E}[U|S_{\check{q}}] \geq 1$) leading to the bound in (4.56). The second term provides the delay of each packet when there are no packet losses within the first $k_{\check{q}}$ packets.

Squaring each of these terms results in the following second moment:

$$\begin{aligned}\bar{\tau}^2_{Y_{\check{q}}=Z_{\check{q}}=1} \geq\ & d^2_{\check{q}} + (k_{\check{q}}+3)\,t_{\check{q}}d_{\check{q}} + \frac{1}{6}\left(2k^2_{\check{q}} + 9k_{\check{q}} + 13\right)t^2_{\check{q}} \\ & - \left(\frac{2k_{\check{q}}+1}{k_{\check{q}}}t_{\check{q}}d_{\check{q}} + \left(k_{\check{q}} + 3 + \frac{7}{6k_{\check{q}}}\right)t^2_{\check{q}}\right)\overline{s^1_{1,\check{q}}} \\ & - \frac{1}{3k_{\check{q}}}t^2_{\check{q}}\overline{s^3_{1,\check{q}}} + \left(\frac{t_{\check{q}}d_{\check{q}}}{k_{\check{q}}} + \left(\frac{2k_{\check{q}}+3}{2k_{\check{q}}}\right)t^2_{\check{q}}\right)\overline{s^2_{1,\check{q}}}.\end{aligned} \tag{4.58}$$

The bound in the above equation is a result of assuming that the number of coded packets U required to decode the generation is limited to one when $s_q < k_q$.

Since the slowest path creates head-of-line blocking, bulk arrivals occur on all of the faster paths. In addition to the transmission and propagation delay experienced

by these packets, the added delay due to waiting for the last generation sent on the slowest path also needs to be taken into account. Therefore, the mean delay for these packets is given by:

$$\bar{\tau}_{Y_q=Z_q=1} = \frac{1}{k_q}\sum_{i=1}^{k_q}\left(\left(\frac{k_q}{c_q} - k_q + i\right)t_q + d_q + (d_{\check{q}} - d_q) - \left(\frac{k_{\check{q}}}{c_{\check{q}}} - k_{\check{q}} - \mathbb{E}\left[U\right]\right)t_{\check{q}}\right) \tag{4.59}$$

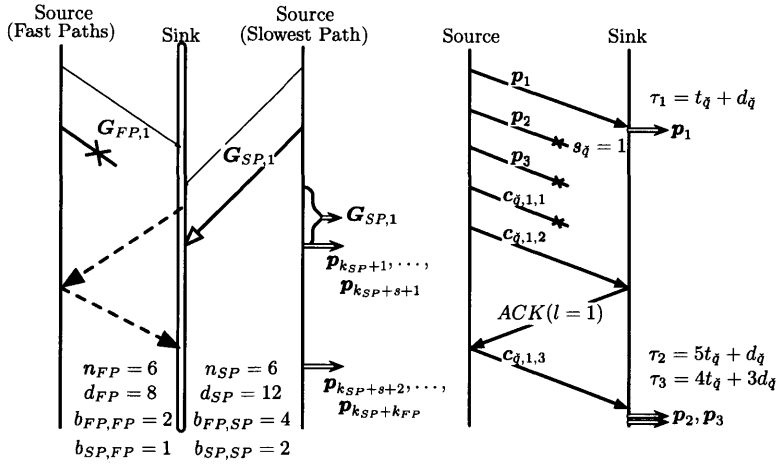
$$\geq t_q\left(\frac{k_q}{c_q} - \frac{k_q}{2} + \frac{1}{2}\right) - t_{\check{q}}\left(\frac{k_{\check{q}}}{c_{\check{q}}} - k_{\check{q}}\right) + d_{\check{q}}. \tag{4.60}$$

The first two terms in (4.59) accounts for the position of each information packet within the generation and the propagation delay, while the last two terms account for the delay due to the slowest path. The bound going from (4.59) to (4.60), as well as in the second moment given below is a result of assuming that $U = 0$.

$$\bar{\tau}^2_{Y_q=Z_q=1} \geq \frac{1}{6}\left(\frac{6k_q}{c_q}\left(\frac{k_q}{c_q}+1\right) - 3k_q\left(\frac{2k_q}{c_q}+1\right) + 2k_q^2 + 1\right)t_q^2 + \left(d_{\check{q}} - \left(\frac{k_{\check{q}}}{c_{\check{q}}} - k_{\check{q}}\right)t_{\check{q}}\right)\left(\frac{2k_q}{c_q} - k_q + 1\right)t_q + \left(d_{\check{q}} - \left(\frac{k_{\check{q}}}{c_{\check{q}}} - k_{\check{q}}\right)t_{\check{q}}\right)^2. \tag{4.61}$$

4.6.2 Case 2: $Y_q > Z_q = 1$

Unlike the first case, the generation containing packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+k_q}$ first transmitted on path q requires retransmissions in order to decode. On the slowest path, information packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{s_{\check{q}}}$ received prior to the first loss can be delivered immediately upon reception. Similarly, packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{s_q}$ sent on faster paths can be delivered as soon as all packets $\boldsymbol{p}_j$, $j < i$, have been delivered. However, packets received after the first loss can only be delivered once the entire generation is decoded (i.e., until at least k_q $dofs$ have been received). Since $Y_q > 1$, this does not occur in the first round. After retransmissions provide the necessary $dofs$, the remaining packets $\boldsymbol{p}_{i+s_q+1}, \ldots, \boldsymbol{p}_{i+k_q}$ can be delivered in-order.



(a) In-Order Delivery with Multiple Paths

(b) In-Order Delivery for the Slowest Path

Figure 4-8: Case 2: $Y_q > Z_q = 1$. Packets received from the slowest path prior to the first packet erasure are immediately delivered, while packets received prior to the first packet loss on faster paths are buffered until all proceeding information packets are delivered. The remaining $k_q - s$ packets are delivered after retransmissions provide enough $dofs$ to decode the generation.

An example is provided in Figure 4-8. Figure 4-8(a) shows the case when $q \neq \check{q}$ where $n_q = {}^{k_q t_q}/_{c_q}$. Packets $\boldsymbol{p}_{k_{SP}+1}, \ldots, \boldsymbol{p}_{k_{SP}+s+1} \in \boldsymbol{G}_{FP,1}$ are buffered until all of the packets in $\boldsymbol{G}_{SP,1}$ are delivered. The remaining $k_q - s$ packets are delivered after retransmissions provide the necessary $dofs$ to decode $\boldsymbol{G}_{FP,1}$. Figure 4-8(b) shows the case when $q = \check{q}$. All packets received prior to the first loss are delivered immediately, while packets received after the loss are buffered until the entire generation is decoded in round $Y_{\check{q}} = 2$. The delay τ_i experienced by each packet is listed next to the time that it is delivered to the application layer in the subfigure.

Taking the expectation over all S_q and all packets within the generation, the expected delay for packets first transmitted on the slowest path $\check{q}$ is provided by

$$\bar{\tau}_{Y_{\check{q}}>Z_{\check{q}}=1} = \sum_{s=0}^{k_{\check{q}}-1}\Bigg((t_{\check{q}}+d_{\check{q}})\frac{s}{k_{\check{q}}} + \frac{1}{k_{\check{q}}}\sum_{i=0}^{k_{\check{q}}-s-1}\Big((k_{\check{q}}-s-i+(k_{\check{q}}/c_{\check{q}}-k_{\check{q}}))\,t_{\check{q}} + d_{\hat{q}} + 2\,(y-1)\,d_{\hat{q}}\Big)\Bigg)p_{S_{\check{q}}|Y_{\check{q}}}(s|y) \tag{4.62}$$

$$= t_{\check{q}}\left(\frac{\overline{s^2_{2,\check{q}}}}{2k_{\check{q}}} + \left(\frac{1}{2k_{\check{q}}}-\frac{1}{c_{\check{q}}}\right)\overline{s^1_{2,\check{q}}} + \frac{k_{\check{q}}}{c_{\check{q}}} - \frac{k_{\check{q}}}{2} + \frac{1}{2}\right) + d_{\check{q}} + 2d_{\hat{q}}\,(y-1)\left(1-\frac{\overline{s^1_{2,\check{q}}}}{k_{\check{q}}}\right), \tag{4.63}$$

where $\overline{s^1_{2,\check{q}}}$ and $\overline{s^2_{2,\check{q}}}$ are given by Lemma 4.6. Similar to the first case, the components of the delay are provided in (4.62). Squaring each of these leads to the following second moment:

$$\begin{aligned}\bar{\tau}^2_{Y_{\check{q}}>Z_{\check{q}}=1} = {} & t^2_{\check{q}}\left(\frac{1}{6}\left(2k^2_{\check{q}}-3k_{\check{q}}+1\right)+\frac{k_{\check{q}}}{c_{\check{q}}}\left(\frac{k_{\check{q}}}{c_{\check{q}}}-k_{\check{q}}+1\right)\right)\\ & + \left(2d_{\hat{q}}\,(y-1)+d_{\check{q}}\right)\left(t_{\check{q}}\left(\frac{2k_{\check{q}}}{c_{\check{q}}}-k_{\check{q}}+1\right)+2d_{\hat{q}}\,(y-1)+d_{\check{q}}\right)\\ & - \frac{\overline{s^1_{2,\check{q}}}}{k_{\check{q}}}\left(2d_{\hat{q}}t_{\check{q}}\,(y-1)\left(\frac{2k_{\check{q}}}{c_{\check{q}}}+1\right)+d_{\check{q}}t_{\check{q}}\left(\frac{2k_{\check{q}}}{c_{\check{q}}}-1\right)\right.\\ & \left. + 4d_{\hat{q}}\,(y-1)\left(d_{\hat{q}}\,(y-1)+d_{\check{q}}\right)+t^2_{\check{q}}\left(\frac{k^2_{\check{q}}}{c^2_{\check{q}}}+\frac{k_{\check{q}}}{c_{\check{q}}}-\frac{5}{6}\right)\right)\\ & + \frac{\overline{s^2_{2,\check{q}}}t_{\check{q}}}{k_{\check{q}}}\left(d_{\check{q}}+2d_{\hat{q}}\,(y-1)+\left(\frac{k_{\check{q}}}{c_{\check{q}}}+\frac{1}{2}\right)t_{\check{q}}\right)-\frac{\overline{s^3_{2,\check{q}}}t^2_{\check{q}}}{3k_{\check{q}}}.\end{aligned} \tag{4.64}$$

The mean delay of packets first transmitted on faster paths is given by

$$\bar{\tau}_{Y_q>Z_q=1} = \frac{1}{k_q}\sum_{s=0}^{k_q-1}\left(\sum_{i=0}^{s-1}\left(\left(\frac{k_q}{c_q}-i\right)t_q + d_q + (d_{\check{q}}-d_q) - \left(\frac{k_{\check{q}}}{c_{\check{q}}}-k_{\check{q}}\right)t_{\check{q}}\right)\right.$$
$$+\sum_{i=0}^{k_q-s-1}\left((k_q-s-i+(k_q/c_q-k_q))\,t_q\right.$$
$$\left.\left.+\,d_q+2\,(y-1)\,d_{\hat{q}}\right)\right)p_{S_q|Y_q}(s|y) \tag{4.65}$$

$$= t_q\left(\frac{k_q}{c_q}-\frac{k_q}{2}+\frac{1}{2}\right)+\frac{\overline{s^1_{2,q}}}{k_q}\left(d_{\check{q}}-t_{\check{q}}\left(\frac{k_{\check{q}}}{c_{\check{q}}}-k_{\check{q}}\right)\right)$$
$$+\,(2d_{\hat{q}}\,(y-1)+d_q)\left(1-\frac{\overline{s^1_{2,q}}}{k_q}\right). \tag{4.66}$$

The second moment for the delay of these packets is determined by squaring the components shown in (4.65):

$$\bar{\tau}^2_{Y_q>Z_q=1} = d_q^2 + d_q t_q\left(\frac{2k_q}{c_q}-k_q+1\right)+\left(\frac{k_q^2}{c_q^2}-\frac{k_q}{c_q}(k_q-1)+\frac{1}{3}k_q^2-\frac{1}{2}k_q+\frac{1}{6}\right)t_q^2$$
$$+\frac{2d_{\hat{q}}}{k_q}(y-1)\left(k_q-\overline{s^1_{2,\check{q}}}\right)\left(2d_{\hat{q}}\,(y-1)+2d_q+t_q\left(\frac{2k_q}{c_q}+1\right)\right)$$
$$-\frac{1}{k_q}\left(-d_{\check{q}}+d_q+t_{\check{q}}\left(\frac{k_{\check{q}}}{c_{\check{q}}}-k_{\check{q}}\right)\right)\left(\overline{s^1_{2,\check{q}}}\left(d_{\check{q}}+d_q-t_{\check{q}}\left(\frac{k_{\check{q}}}{c_{\check{q}}}-k_{\check{q}}\right)\right.\right.$$
$$\left.\left.+\,t_q\left(\frac{2k_q}{c_q}+1\right)\right)-\overline{s^2_{2,\check{q}}}t_q\right)+\frac{2d_{\hat{q}}t_q}{k_q}(y-1)\left(\overline{s^2_{2,\check{q}}}-k_q^2\right). \tag{4.67}$$

It is important to note that the additional time to retransmit $dofs$ is not taken into account (see the assumptions made in Section 4.4).

4.6.3 Case 3: $Z_q > Y_q \geq 1$

In this case, there is no need to differentiate between packets first transmitted on the slowest path or faster paths. The generation containing packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+k_q}$ first transmitted on path q completes in a round prior to a generation containing any packet $\boldsymbol{p}_j$, $j < i$ (i.e., a previous generation is causing head-of-line blocking). As

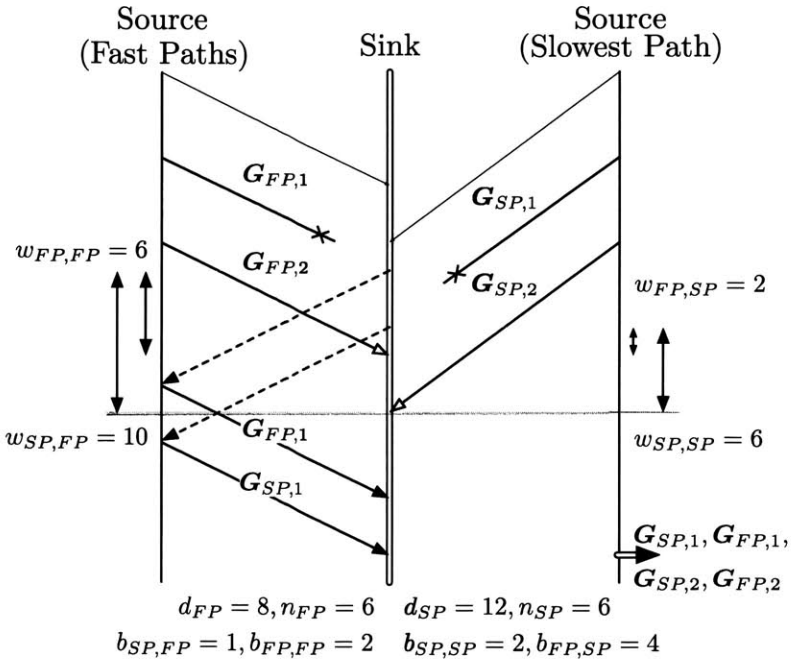


Figure 4-9: Case 3: $Z_q > Y_q \geq 1$. Packets in $\boldsymbol{G}_{FP,2}$ and $\boldsymbol{G}_{SP,2}$ cannot be delivered until both $\boldsymbol{G}_{SP,1}$ and $\boldsymbol{G}_{FP,1}$ are decoded and delivered where $n_q = {}^{k_q t_q}/_{c_q}$.

a result, packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+k_q}$ are buffered until all packets $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{i-1}$ have been delivered. Once there are no earlier generations preventing in-order delivery, packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+k_q}$ can be immediately delivered.

An example is provided in Figure 4-9. While packets in $\boldsymbol{G}_{SP,2}$ and $\boldsymbol{G}_{FP,2}$ are received in round $Z_q = 1$, they cannot be delivered until both $\boldsymbol{G}_{SP,1}$ and $\boldsymbol{G}_{FP,1}$ are delivered. In this example, generation $\boldsymbol{G}_{SP,1}$ is the last to be decoded (i.e., we select the path with the minimum $W_{q,q'}$ to determine the delay). Taking the expectation over all packets within the generation and using the above lemma, the expected delay is provided by

$$\bar{\tau}_{Z_q>Y_q\geq 1,Z_q>1} = \frac{1}{k_q}\sum_{i=1}^{k_q}\Bigg(\left(\left\lceil k_q/c_q \right\rceil - k_q + i\right)t_q + d_q + 2d_{\hat{q}}\left(z-1\right) - \mathbb{E}\left[W_{\min,q}|Z_q = z\right]\Bigg) \tag{4.68}$$

$$\geq t_q\left(\frac{k_q}{c_q} - \frac{k_q}{2} + \frac{1}{2}\right) + d_q + 2d_{\hat{q}}\left(z-1\right) - \max_{q'\in\mathcal{P}}\left(d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}\left(z\right)}+1\right)\right), \tag{4.69}$$

where $\overline{v^1_{q,q'}\left(z\right)}$ is given by (4.31) and the bound is a result of Lemma 4.8. The second moment for this case is given by

$$\bar{\tau}^2_{Z_q>Y_q\geq 1,Z_q>1} = \gamma - \left(4d_{\hat{q}}\left(z-1\right) + 2d_q + t_q\left(\frac{2k_q}{c_q} - k_q + 1\right)\right)\mathbb{E}\left[W_{\min,q}|Z_q = z\right] + \mathbb{E}\left[W^2_{\min,q}|Z_q = z\right] \tag{4.70}$$

$$\geq \gamma - \left(4d_{\hat{q}}\left(z-1\right) + 2d_q + t_q\left(\frac{2k_q}{c_q} - k_q + 1\right)\right)\mathbb{E}\left[W_{\min,q}|Z_q = z\right] \tag{4.71}$$

$$\geq \gamma - \left(4d_{\hat{q}}\left(z-1\right) + 2d_q + t_q\left(\frac{2k_q}{c_q} - k_q + 1\right)\right) \cdot \max_{q'\in\mathcal{P}}\left(d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}\left(z\right)}+1\right)\right), \tag{4.72}$$

where

$$\gamma = 2d_{\hat{q}}\left(z-1\right)\left(t_q\left(\frac{2k_q}{c_q} - k_q + 1\right) + 2d_q + 2d_{\hat{q}}\left(z-1\right)\right) + t_q^2\left(\frac{k_q}{c_q}\left(\frac{k_q}{c_q} - k_q + 1\right) + \frac{1}{3}k_q^2 - \frac{1}{2}k_q + \frac{1}{6}\right) + d_q\left(d_q + t_q\left(\frac{2k_q}{c_q} - k_q + 1\right)\right). \tag{4.73}$$

The first inequality in (4.71) is possible because $\mathbb{E}\left[W^2_{\min,q}|Z_q = z\right] \geq 0$, and the second

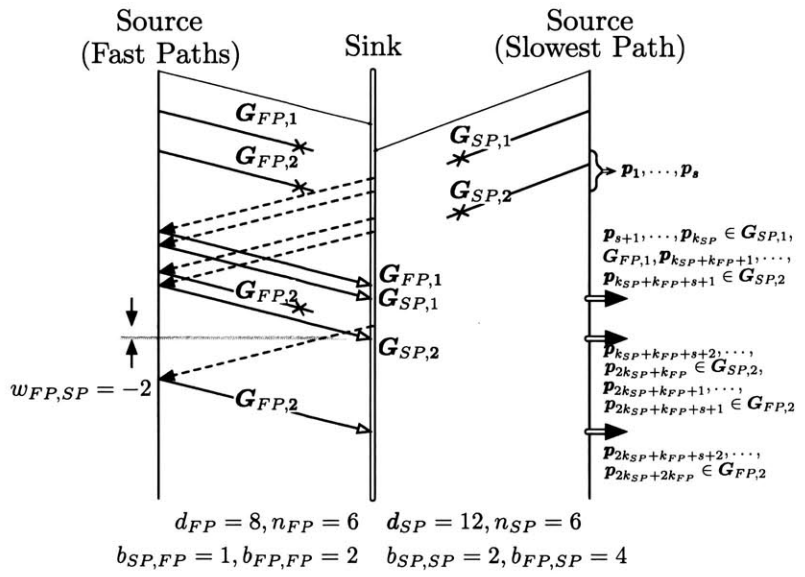


Figure 4-10: Case 4: $Y_q \geq Z_q > 1$. The first s packets in generation $\boldsymbol{G}_{FP,2}$ are delivered when $\boldsymbol{G}_{SP,2}$ is decoded in round $Z_{FP} = 2$. The remaining $k_{FP} - s$ packets are delivered when $\boldsymbol{G}_{FP,2}$ is decoded in round $Y_{FP} = 3$. Note $n_q = {}^{k_q t_q}/_{c_q}$.

inequality in (4.72) is possible because $z \geq 1$ and $0 < c_q \leq 1$. Unfortunately, finding a lower bound for $\mathbb{E}\left[W^2_{\min,q} | Z_q = z\right]$ is not straight forward. However if there is only a single path, the second moment can be found easily without having to use this bound.

4.6.4 Case 4: $Y_q \geq Z_q > 1$

This case is a mixture of the last two. The generation containing packets $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+k_{\tilde{q}}}$ requires the same or more rounds to decode than all of the previously transmitted generations containing packets $\boldsymbol{p}_j$, $j < i$. Packets received prior to the first loss in the generation (i.e., $\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+s}$) are buffered until all previous generations are delivered. Once there are no generations causing head-of-line blocking, these packets are delivered in-order. Packets received after the first loss (i.e., $\boldsymbol{p}_{i+s+2}, \ldots, \boldsymbol{p}_{i+k_q}$) are buffered until the generation is decoded in round Y_q. Figure 4-10 provides and example where packets $\boldsymbol{p}_{2k_{SP}+k_{FP}+1}, \ldots, \boldsymbol{p}_{2k_{SP}+k_{FP}+s+1} \in \boldsymbol{G}_{FP,2}$ are delivered in round $Z_{FP} = 2$, while packets $\boldsymbol{p}_{2k_{SP}+k_{FP}+s+2}, \ldots, \boldsymbol{p}_{2k_{SP}+2k_{FP}} \in \boldsymbol{G}_{FP,2}$ are delivered once $\boldsymbol{G}_{FP,2}$ is decoded in round $Y_{FP} = 3$.

In general, the delay for the case when $Y_q > Z_q$ may not be the same as the case when $Y_q = Z_q$. This is due to head-of-line blocking created by slower paths preventing packet delivery on the faster paths. The mean delay for $Y_q > Z_q$ is given by

$$\begin{aligned}
\bar{\tau}_{Y_q>Z_q>1} = \frac{1}{k_q}\sum_{s=0}^{k_q-1}\Bigg(\sum_{i=1}^{s}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q + d_q + 2d_{\hat{q}}\left(z-1\right) - \mathbb{E}\left[W_{\min,q}|Z_q\right]\right) \\
+ \sum_{i=s+1}^{k_q}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q + d_q + 2d_{\hat{q}}\left(y-1\right)\right)\Bigg)p_{S_q|Y_q}\left(s|y\right) \qquad (4.74) \\
\geq t_q\left(\frac{k_q}{c_q} - \frac{k_q}{2} + \frac{1}{2}\right) + d_q + \frac{2d_{\hat{q}}\overline{s^1_{2,q}}}{k_q}\left(z-y\right) + 2d_{\hat{q}}\left(y-1\right) \\
- \frac{\overline{s^1_{2,q}}}{k_q}\max_{q'\in\mathcal{P}}\left(d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}\left(z\right)}+1\right)\right), \qquad (4.75)
\end{aligned}$$

while the mean for the case when $Y_q = Z_q$ is

$$\begin{aligned}
\bar{\tau}_{Y_q=Z_q>1} = \frac{1}{k_q}\sum_{s=0}^{k_q-1}\Bigg(\sum_{i=1}^{s}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q + d_q + 2d_{\hat{q}}\left(z-1\right) - \mathbb{E}\left[W_{\min,q}|Z_q\right]\right) \\
+ \sum_{i=s+1}^{k_q}\Bigg(\left(\frac{k_q}{c_q}-i+1\right)t_q + d_q + 2d_{\hat{q}}\left(y-1\right) \\
- \min\left(\mathbb{E}\left[W_{\min,q}|Z_q\right],0\right)\Bigg)\Bigg)p_{S_q|Y_q}\left(s|y\right) \qquad (4.76) \\
\geq \frac{1}{k_q}\sum_{s=0}^{k_q-1}\Bigg(\sum_{i=1}^{s}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q + d_q + 2d_{\hat{q}}\left(z-1\right) - \mathbb{E}\left[W_{\min,q}|Z_q\right]\right) \\
+ \sum_{i=s+1}^{k_q}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q + d_q + 2d_{\hat{q}}\left(y-1\right)\right)\Bigg)p_{S_q|Y_q}\left(s|y\right) \qquad (4.77) \\
\geq t_q\left(\frac{k_q}{c_q} - \frac{k_q}{2} + \frac{1}{2}\right) + d_q + \frac{2d_{\hat{q}}\overline{s^1_{2,q}}}{k_q}\left(z-y\right) + 2d_{\hat{q}}\left(y-1\right) \\
- \frac{\overline{s^1_{2,q}}}{k_q}\max_{q'\in\mathcal{P}}\left(d_q - d_{q'} + \frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}\left(z\right)}+1\right)\right). \qquad (4.78)
\end{aligned}$$

By removing the additional delay due to head-of-line blocking in (4.76), the delay can be bounded. This results in the same delay as the case when $Y_q > Z_q$. Similarly, the

second moments can be determined as

$$\begin{aligned}
\bar{\tau}^2_{Y_q>Z_q>1} &= \gamma - \frac{1}{k_q}\left(\overline{s^1_{2,q}}\left(4d_{\hat{q}}\left(z-1\right)+2d_q+t_q\left(\frac{2k_q}{c_q}-1\right)\right)\right. \\
&\quad \left.+\overline{s^2_{2,q}}t_q\right)\mathbb{E}\left[W_{\min,q}|Z_q\right]+\frac{\overline{s^1_{2,q}}}{k_q}\mathbb{E}\left[W^2_{\min,q}|Z_q\right] \qquad (4.79)\\
&\geq \gamma - \frac{1}{k_q}\left(\overline{s^1_{2,q}}\left(4d_{\hat{q}}\left(z-1\right)+2d_q+t_q\left(\frac{2k_q}{c_q}-1\right)\right)\right. \\
&\quad \left.+\overline{s^2_{2,q}}t_q\right)\mathbb{E}\left[W_{\min,q}|Z_q\right] \qquad (4.80)\\
&\geq \gamma - \frac{1}{k_q}\left(\overline{s^1_{2,q}}\left(4d_{\hat{q}}\left(z-1\right)+2d_q+t_q\left(\frac{2k_q}{c_q}-1\right)\right)\right. \\
&\quad \left.+\overline{s^2_{2,q}}t_q\right)\max_{q'\in\mathcal{P}}\left(d_q-d_{q'}+\frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}\left(z\right)}+1\right)\right), \qquad (4.81)
\end{aligned}$$

for $Y_q > Z_q$ and

$$\begin{aligned}
\bar{\tau}^2_{Y_q=Z_q>1} &= \frac{1}{k_q}\sum_{s=0}^{k_q-1}\left(\sum_{i=1}^{s}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q+d_q+2d_{\hat{q}}\left(z-1\right)-\mathbb{E}\left[W_{\min,q}|Z_q\right]\right)^2\right. \\
&\quad +\sum_{i=s+1}^{k_q}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q+d_q+2d_{\hat{q}}\left(y-1\right)\right. \\
&\quad \left.\left.-\min\left(\mathbb{E}\left[W_{\min,q}|Z_q\right],0\right)\right)^2\right)p_{S_q|Y_q}\left(s|y\right) \qquad (4.82)\\
&\geq \frac{1}{k_q}\sum_{s=0}^{k_q-1}\left(\sum_{i=1}^{s}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q+d_q+2d_{\hat{q}}\left(z-1\right)-\mathbb{E}\left[W_{\min,q}|Z_q\right]\right)^2\right. \\
&\quad \left.+\sum_{i=s+1}^{k_q}\left(\left(\frac{k_q}{c_q}-i+1\right)t_q+d_q+2d_{\hat{q}}\left(y-1\right)\right)^2\right)p_{S_q|Y_q}\left(s|y\right) \qquad (4.83)\\
&\geq \gamma - \frac{1}{k_q}\left(\overline{s^1_{2,q}}\left(4d_{\hat{q}}\left(z-1\right)+2d_q+t_q\left(\frac{2k_q}{c_q}-1\right)\right)\right. \\
&\quad \left.+\overline{s^2_{2,q}}t_q\right)\max_{q'\in\mathcal{P}}\left(d_q-d_{q'}+\frac{k_{q'}t_{q'}}{c_{q'}}\left(\overline{v^1_{q,q'}\left(z\right)}+1\right)\right), \qquad (4.84)
\end{aligned}$$

for $Y_q = Z_q$ where

$$\gamma = d_q^2 + d_q t_q \left(\frac{2k_q}{c_q} - k_q + 1\right) + t_q^2 \left(\frac{k_q}{c_q}\left(\frac{k_q}{c_q} - k_q + 1\right) + \frac{1}{6}\left(2k_q^2 - 3k_q + 1\right)\right)$$
$$+ 4d_{\hat{q}}^2 \left((y-1)^2 - \frac{1}{k_q}\overline{s_{2,q}^1}\,(y-z)\,(y+z-2)\right) + 4d_{\hat{q}} d_q \left(y - 1 + \frac{1}{k_q}\overline{s_{2,q}^1}\,(z-y)\right)$$
$$+ 2d_{\hat{q}} t_q \left(\left(\frac{2k_q}{c_q} - k_q + 1\right)(y-1) - \frac{1}{k_q}\left(\overline{s_{2,q}^1}\left(\frac{2k_q}{c_q} + 1\right) - \overline{s_{2,q}^2}\right)(y-z)\right). \quad (4.85)$$

Within each of the the above equations, the expectations over all S_q and all packets within a generation are taken, in addition to employing Lemma 4.8 to help bound each moment. The second moment is further bounded by removing $\mathbb{E}\left[W_{\min,q}^2 | Z_q\right]$. Since $W_{\min,q}$ is not a major contributor to the delay, especially when Z_q and Y_q are large, removing it will not greatly effect the overall result.

4.6.5 Mean and Variance of the In-Order Delivery Delay

Given the expected delay for each of the above cases, the expected in-order delay for packets first transmitted on path q can now be determined.

Theorem 4.9. *The first two moments of the in-order delivery delay for packets first transmitted on path q is lower bounded by*

$$\mathbb{E}\left[\mathrm{T}_q^i\right] \geq \sum_{z_q \geq 1} \sum_{y_q \geq 1} \bar{\tau}_{y_q, z_q}^i p_{Y_q}(y_q)\, p_{Z_q}(z_q), \quad (4.86)$$

where $i = \{1, 2\}$ and each $\bar{\tau}_{Y_q, Z_q}$ are given by (4.57), (4.60), (4.63), (4.66), (4.69), and (4.75). The distributions for $p_{Y_q}(y_q)$ and $p_{Z_q}(z_q)$ are given by (4.10) and (4.17) respectively.

The first two moments of the in-order delivery delay can then be determined by averaging over all of the paths:

$$\mathbb{E}\left[\mathrm{T}^i\right] = \frac{1}{\sum_{q' \in \mathcal{P}} \left(c_{q'}/t_{q'}\right)} \sum_{q \in \mathcal{P}} \left(\frac{c_q}{t_q}\right) \mathbb{E}\left[\mathrm{T}_q^i\right], \quad (4.87)$$

where $c_q/t_q \sum_{q' \in \mathcal{P}} c_{q'}/t_{q'}$ is the fraction of information packets transmitted on path q. The variance of the in-order delivery delay, or the jitter, is simply $\sigma_T^2 = \mathbb{E}[T^2] - \mathbb{E}[T]^2$.

4.7 Determining the Cost of Reducing the Delay

The above results show that adding redundancy into a packet stream can decrease the in-order delivery delay. However, this improvement comes with a cost. To characterize this cost, this section will determine the efficiency η of the coding algorithm. Before determining η, several several random variables need to be defined. Let R_q, $q \in \mathcal{P}$, be the number of $dofs$ received at the sink as a result of transmitting a generation of size k_q. Given that $Y_q = 1$, the number of $dofs$ received at the sink has the following distribution:

$$p_{R_q|Y_q}(r_q|1) = \frac{B\left(\frac{k_q}{c_q}, r_q, \epsilon_q\right)}{\sum_{i=k_q}^{k_q/c_q} B\left(\frac{k_q}{c_q}, i, \epsilon_q\right)} \tag{4.88}$$

$$= \frac{1}{p_{X_0,q}(0)} B\left(\frac{k_q}{c_q}, r_q, \epsilon_q\right), \tag{4.89}$$

for $r_q \in [k_q, {}^{k_q}/_{c_q}]$. This leads to the following expectation:

$$\mathbb{E}[R_q|Y_q = 1] = \sum_{r=k_q}^{k_q/c_q} r p_{R_q|Y_q}(r|1). \tag{4.90}$$

Assuming that $Y_q > 1$, the Markov chain defined in Section 4.5 can be used to help determine R_q. Define N_i to be the total number of $dofs$ received by the sink for any set of transitions starting in state i and ending in state 0. Furthermore, define the random variable N_{ij} as the number of $dofs$ received by the sink as a result of a single transition from state i to state j (i.e., $i \to j$) in the same Markov chain. N_{ij} is deterministic (e.g., $n_{ij} = i - j$) when $i, j \geq 1$ and $i \geq j$. For any transition $i \to 0$,

$i \geq 1$, $n_{i0} \in [i, {}^{i}/_{c_{\hat{q}}}]$ has probability:

$$p_{n_{i0}}(n_{i0}) = \frac{B\left(\frac{i}{c_{\hat{q}}}, n_{i0}, \epsilon_{\hat{q}}\right)}{\sum_{j=i}^{{}^{i}/_{c_{\hat{q}}}} B\left(\frac{i}{c_{\hat{q}}}, j, \epsilon_{\hat{q}}\right)}. \tag{4.91}$$

Therefore, the expected number of $dofs$ received by the sink in a single transition is:

$$\mathbb{E}[N_{ij}] = \begin{cases} i - j & \text{for } i, j \geq 1, i \geq j, \\ \sum_{n=i}^{{}^{i}/_{c_{\hat{q}}}} n \cdot p_{N_{i0}}(n) & \text{for } i \geq 1, j = 0. \end{cases} \tag{4.92}$$

Given $\mathbb{E}[N_{ij}]$ $\forall i, j$, the total number of $dofs$ received by the sink when trying to transfer i $dofs$ is:

$$\mathbb{E}[N_i] = \frac{1}{1 - a_{ii}} \left(\sum_{j=0}^{i-1} (\mathbb{E}[N_{ij}] + \mathbb{E}[N_j]) a_{ij} \right), \tag{4.93}$$

where $\mathbb{E}[N_0] = 0$ and

$$a_{ij} = \begin{cases} B({}^{i}/_{c_{\hat{q}}}, i - j, \epsilon_{\hat{q}}) & \text{for } j \in (0, i], \\ \sum_{x=i}^{{}^{i}/_{c_{\hat{q}}}} B({}^{i}/_{c_{\hat{q}}}, x, \epsilon_{\hat{q}}) & \text{for } j = 0. \end{cases} \tag{4.94}$$

This framework can now be used to determine the expectation of R_q when $Y_q > 1$:

$$\mathbb{E}[R_q | Y_q > 1] = \sum_{r=1}^{k_q} (k_q - r + \mathbb{E}[N_r]) \frac{p_{X_{0,q}}(r)}{1 - p_{X_{0,q}}(0)}. \tag{4.95}$$

Combining (4.90) and (4.95), the following theorem is obtained.

Theorem 4.10. *The efficiency η, defined as the ratio between the total number of information packets transmitted and the number of $dofs$ received by the sink, is*

$$\eta = \frac{1}{\sum_{q' \in \mathcal{P}} {}^{c_{q'}}/_{t_{q'}}} \sum_{q \in \mathcal{P}} \frac{k_q c_q}{t_q \mathbb{E}[R_q]}, \tag{4.96}$$

where

$$\mathbb{E}[R_q] = p_{X_{0,q}}(0)\,\mathbb{E}[R_q|Y_q = 1] + (1 - p_{X_{0,q}}(0))\,\mathbb{E}[R_q|Y_q > 1], \tag{4.97}$$

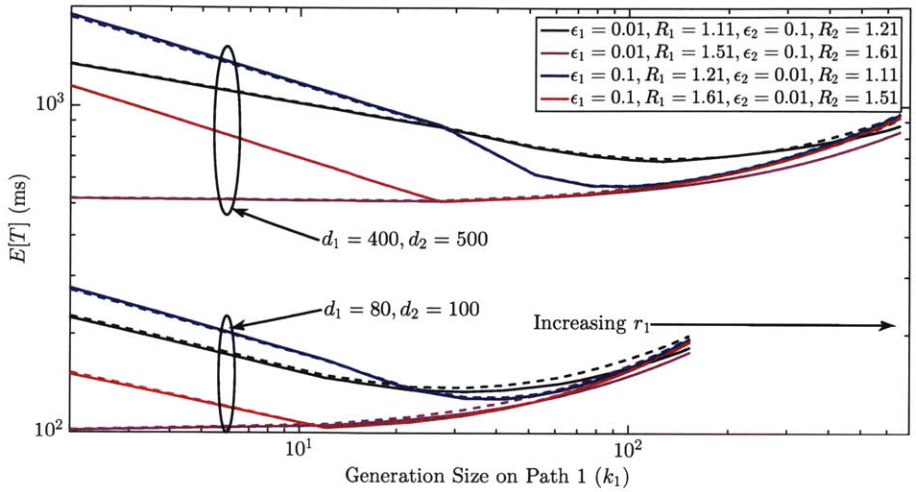
$\mathbb{E}[R_q|Y_q = 1]$ *is given in (4.90), and* $\mathbb{E}[R_q|Y_q > 1]$ *is given in (4.95).*

4.8 Numerical Results

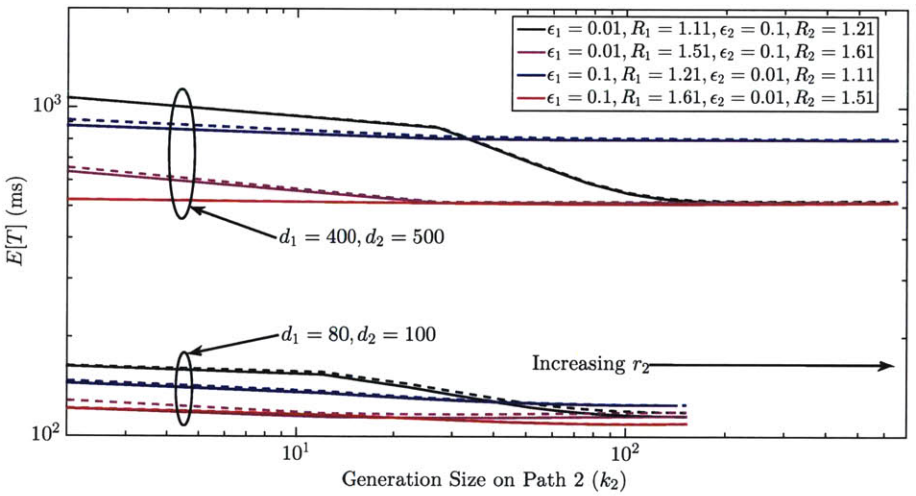
Unfortunately, a closed form expression for the in-order delivery delay cannot be found due to the complexity of the process. However, numerical results help show that (4.87) is fairly tight for a large number of regimes. As a result, the above analysis can be treated as a fairly accurate estimate for the true mean in-order delivery delay.

Before proceeding, several items need to be noted. First, the terms in (4.86) where $p_{Y_q}(y)\,p_{Z_q}(z) < 10^{-6}$ when calculating the moments are not considered since they have little effect on the overall calculation. Second, the analytical curves are sampled at local maxima. As the generation sizes increase, the number of in-flight generations, b_q, on each path decreases incrementally. Upon each decrease in b_q, a discontinuity occurs that causes an artificial decrease in $\mathbb{E}[\mathrm{T}]$ that becomes less noticeable as k_q increases towards the next decrease in b_q. This transient behavior in the analysis is more prominent when $c_q \approx 1 - \epsilon_q$ and less so when $c_q \ll 1 - \epsilon_q$. This behavior is removed from the results presented within this section and the next if it is large enough to notice. Third, it should be noted that k_q/c_q may not be an integer. To overcome this issue, $\lceil k_q/c_q \rceil - k_q$ and $\lfloor k_q/c_q \rfloor - k_q$ coded packets are generated and transmitted with probability $k_q/c_q - \lfloor k_q/c_q \rfloor$ and $\lceil k_q/c_q \rceil - k_q/c_q$ respectively. This time-sharing approach allows the analysis and simulations presented here to acheive an average transmitted generation size of k_q/c_q.

The analytical in-order delivery delay given by (4.87) is compared with simulated results in Figure 4-11. The delay is shown for the case where two disjoint network paths are available. Per Algorithms 1 and 2, information packets are partitioned into generations of size k_1 and k_2 depending on the path which they are initially trans-



(a) $k_1 =$ Variable, $r_1 =$ Variable, $k_2 = 30, r_2 = 10$ Mbps



(b) $k_1 = 30, r_1 = 10$ Mbps, $k_2 =$ Variable, $r_2 =$ Variable

Figure 4-11: The in-order delivery delay over two paths for various combinations of erasure rates ($\epsilon_i = \{0.01, 0.1\}$, $i = \{1, 2\}$) and propagation delays ($d_1 = \{80, 400\}$ and $d_2 = \{100, 500\}$) as a function of the generation size k_i on path $i = \{1, 2\}$. The analytical and simulated results are represented using solid and dotted lines respectively. Note the log scale of both the x-axis and y-axis, in addition to the variable rates on the secondary path due to assumption (4.6).

mitted. Redundancy is added on a path-by-path basis, and each of these generations are then transmitted to the client. If feedback, received on the fastest path, indicates additional $dofs$ are needed to decode a generation, retransmissions are made using the fastest path. Figure 4-11a shows the mean in-order delivery delay when k_1 is varied over the size of the bandwidth-delay product of the fastest network, while k_2 is fixed at 30 packets. Figure 4-11b is similar except k_2 is varied while k_1 is fixed. The simulated results are represented by the dashed lines and the analytical results are represented by the solid lines. It should be noted that the rate of the network where k is variable is also variable. This is owing to the assumption made by (4.6) that states ${}^{1}/_{t_i} = {}^{k_i c_j}/_{k_j t_j c_i}$.

Both subfigures indicate that the analysis is a fairly good estimate of the in-order delivery delay that might be expected on parallel networks using the coding scheme presented within this chapter. While this is the case, interpreting these figures is a challenge since multiple variables are changing at the same time. However, Figure 4-11a does show the trade-off between head-of-line blocking created by previous generations and the coding delay. The next section will expand on this in greater detail using only a single network path to help properly interpret the results.

4.9 A Special Case: In-Order Delivery Delay Over a Single Path

Assume that only a single path connects the source and sink. In this situation, the analysis provided in the previous sections can be greatly simplified and is largely the same as the delay of the slowest path $\tilde{q}$ in the previous analysis. To help simplify this section, the subscript q will be removed from each variable.

The first two moments of the delay when $Z = 1$ remain the same as those determined in (4.57) and (4.60). However, the cases where $Z > 1$ can be simplified since it is no longer necessary to find the path that is preventing packets from being delivered. The number of previously transmitted generations that can cause head-of-line

blocking is reduced to $b = \lceil \frac{2dc}{kt} \rceil - 1$, and the auxiliary variable $W_{q,q'}$ becomes

$$W = \frac{kt}{c}(V+1). \tag{4.98}$$

This makes the first two moments of W equal to

$$\mathbb{E}[W|Z=z] = \frac{kt}{c}\left(\overline{v^1(z)}+1\right), \tag{4.99}$$

and

$$\mathbb{E}[W^2|Z=z] = \frac{k^2t^2}{c^2}\left(\overline{v^2(z)}+2\overline{v^1(z)}+1\right), \tag{4.100}$$

respectively where $\overline{v^1(z)}$ and $\overline{v^2(z)}$ are given in Lemma 4.7. Inserting the first moment into (4.68) and (4.74), the resulting delay is

$$\bar{\tau}_{Z>Y\geq 1,Z>1} = d(2z-1) - t\left(\frac{k}{c}\overline{v^1(z)} + \frac{k-1}{2}\right), \tag{4.101}$$

and

$$\bar{\tau}_{Y\geq Z>1} = d\left(\frac{2}{k}(z-y)\overline{s_2^1} + 2y - 1\right) - t\left(\frac{1}{c}\left(\overline{v^1(z)}+1\right)\overline{s_2^1} - \frac{k}{c} + \frac{k-1}{2}\right). \tag{4.102}$$

Similarly, the two moments above can be used to simplify both (4.70) and (4.79). This results in the following:

$$\begin{aligned}\bar{\tau}^2_{Z_q>Y_q\geq 1,Z_q>1} = {} & t^2\left(\frac{k^2}{c^2}\overline{v^2(z)} + (k-1)\left(\frac{k}{c}\overline{v^1(z)} + \frac{1}{6}(2k-1)\right)\right) \\ & + d^2(2z-1)^2 - td(2z-1)\left(\frac{2k}{c}\overline{v^1(z)} + k - 1\right),\end{aligned} \tag{4.103}$$

and

$$\bar{\tau}^2_{Y_q \geq Z_q > 1} = \frac{1}{k} s_2 \left(2dt\,(y-z) - \frac{k}{c} t^2 (v_1 + 1) \right) + \frac{1}{k} s_1 \left(\frac{k}{c} t^2 \left(\frac{k}{c} (v_2 - 1) + v_1 + 1 \right) \right.$$
$$\left. - 4d^2 (y-z)(y+z-1) - 2dt \left(\frac{k}{c} (v_1 (2z-1) + 2y - 1) + y - z \right) \right)$$
$$+ (2y-1) \left(dt \left(2\frac{k}{c} - k + 1 \right) + d^2 (2y-1) \right)$$
$$+ t^2 \left(\frac{k}{c} \left(\frac{k}{c} - k + 1 \right) + \frac{1}{6} \left(2k^2 - 3k + 1 \right) \right). \tag{4.104}$$

Using the above moments, the moments of the in-order delivery delay for a single path becomes

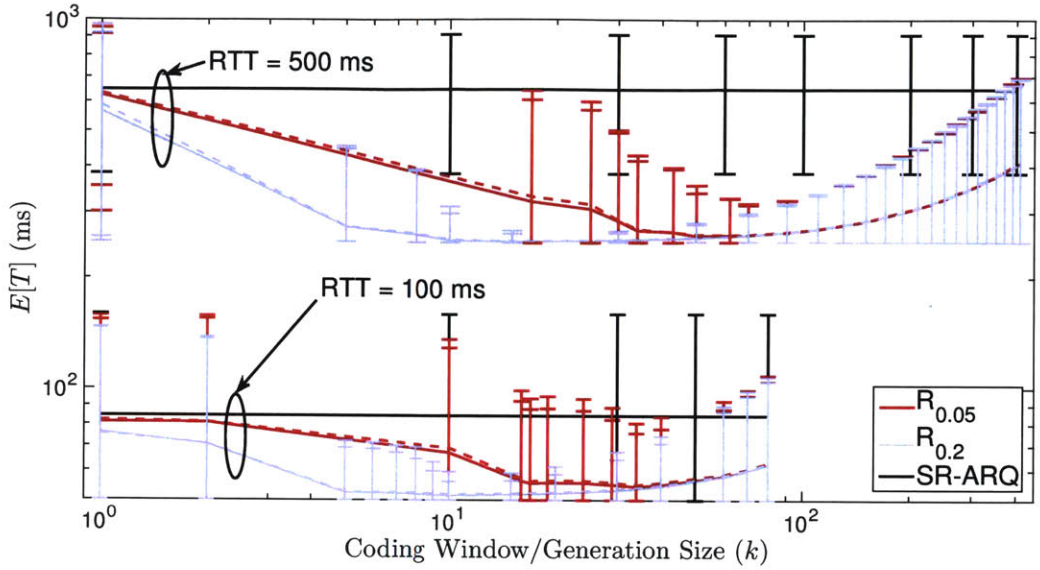
$$\mathbb{E}\left[\mathrm{T}^i\right] = \sum_{y \geq 1} \sum_{z \geq 1} \bar{\tau}^i_{y,z} p_Y(y)\, p_Z(z)$$

for $i = \{1, 2\}$. This results in a mean delay of $\mathbb{E}[\mathrm{T}]$ and variance $\sigma^2_{\mathrm{T}} = \mathbb{E}[\mathrm{T}^2] - \mathbb{E}[\mathrm{T}]^2$.

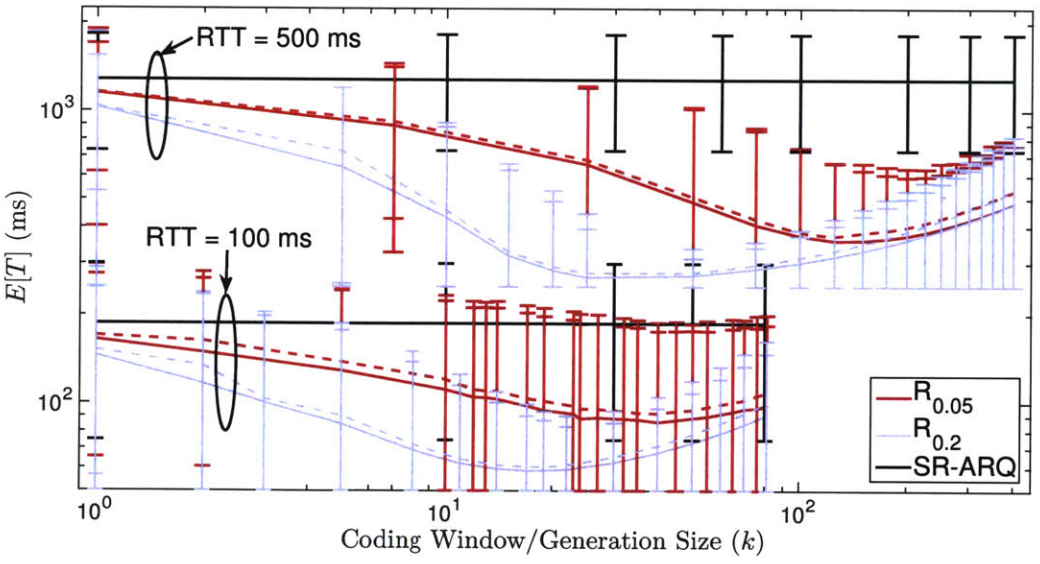
4.9.1 Coding Window Size and Redundancy Selection

The results above help inform decisions on the selection of the coding window size and required redundancy injected into the transmitted packet stream. Figure 4-12 shows the in-order delivery delay and its standard deviation for four different networks/links. The simulated results were developed in Matlab using a model similar to that presented in Section 4.3, although several of the assumptions are relaxed. The time it takes to retransmit coded packets after round one is taken into account. Furthermore, the number of generations preventing delivery is not limited to a single bandwidth-delay product (BDP) of packets. This increases the probability of head-of-line blocking; and in general, these relaxations effectively increase the delay experienced by a packet. Finally, the figure shows the delay of an idealized version of SR-ARQ where it is assumed both the source and sink have infinite buffer sizes.

Figure 4-12 reiterates that adding redundancy and/or choosing the correct coding window/generation size can have major implications on the in-order delay. Not only



(a) $\epsilon = 0.01$



(b) $\epsilon = 0.1$

Figure 4-12: The in-order delivery delay over a single path with erasure rates $\epsilon = 0.01$ and $\epsilon = 0.1$ as a function of the generation size k. The error bars show $2\sigma_T$ above and below the mean and $R_z = {}^{(1+x)}/_{(1-\epsilon)}$. The analytical and simulated results are represented using solid and dotted lines respectively. Note the log scale of both the x-axis and y-axis.

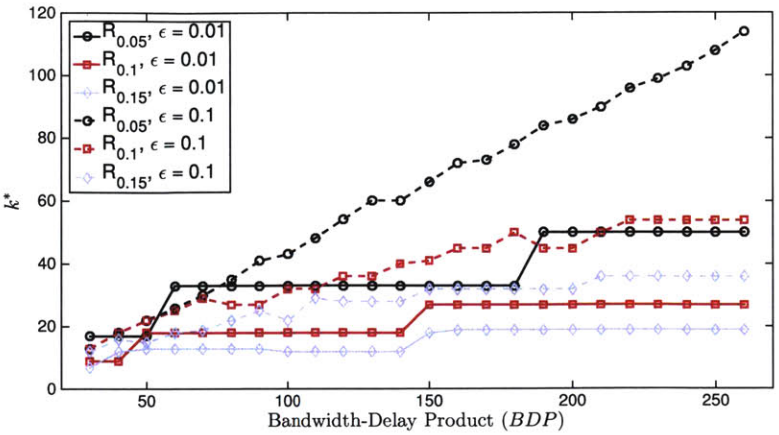


Figure 4-13: k^* as a function of the BDP.

does choosing correctly reduce the delay, but doing so can also reduce the jitter. However, it is apparent that the proper selection of k for a given code rate $c = 1/R$ is critical for minimizing both $\mathbb{E}[\mathrm{T}]$ and $\mathbb{E}[\mathrm{T}^2]$. In fact, Figure 4-12 indicates that adding redundancy and choosing a moderately sized generation is needed in most cases to ensure both are minimized.

The shape of the curves in the figure also indicate that there are two major contributors to the in-order delay that need to be balanced. Let k^* be the generation size where $\mathbb{E}[\mathrm{T}]$ is minimized for a given ϵ and $c = 1/R$, i.e.,

$$k^* = \arg\min_k \mathbb{E}[\mathrm{T}]. \tag{4.105}$$

To the left of k^*, the delay is dominated by head-of-line blocking and re-sequencing delay created by previous generations. To the right of k^*, the delay is dominated by the time it takes to receive enough $dofs$ to decode the generation. While there are gains in efficiency for $k > k^*$, the benefits are negligible for most time-sensitive applications.

Figure 4-13 shows k^* for a given ϵ and $c = 1/R$ as a function of the BDP. The figure indicates that the coding window size k^* increases with ϵ, which is opposite of what we would expect from a typical erasure code [62]. In the case of small ϵ, it is better to try and quickly correct only some of the packet losses occurring within

a generation using the initially transmitted coded packets while relying heavily on feedback to overcome any decoding errors. In the case of large ϵ, a large generation size is better where the majority of packet losses occurring within a generation are corrected using the initially transmitted coded packets and feedback is relied upon to help overcome the rare decoding error. Furthermore, decreasing c also decreases k^*. This due to the receiver's increased ability to decode a generation without having to wait for retransmissions. Finally, k^* is not very sensitive to the BDP (in most cases) enabling increased flexibility during system design and implementation.

Before proceeding, it is important to note that a certain level of redundancy is needed to see benefits. Each curve in Figures 4-12 and 4-13 show results for $c < 1-\epsilon$. For $c \geq 1-\epsilon$, it is possible to see in-order delays and jitter worse than the idealized ARQ scheme. Consider an example where a packet loss is observed near the beginning of a generation that cannot be decoded after the first transmission attempt. Since feedback is not sent/acted upon until the end of the generation, the extra time waiting for feedback can induce larger delays than what would have occurred under a simple ARQ scheme. This time can be reduced by reacting to feedback before the end of a generation; but it is still extremely important to ensure that the choice of k and c will decrease the probability of a decoding failure and provide improved delay performance.

4.9.2 Rate-Delay Trade-Off

While transport layer coding can help meet strict delay constraints, the decreased delay comes at the cost of throughput, or efficiency. Let $\mathbb{E}[\mathrm{T}^*]$, σ_{T}^*, and η^* be the expected in-order delay, the standard deviation, and the expected efficiency respectively that corresponds to k^* defined in eq. (4.105). The rate-delay trade-off is shown by plotting $\mathbb{E}[\mathrm{T}^*]$ as a function of η^* in Figure 4-14. The expected SR-ARQ delay (i.e., the data point for $\eta = 1$) is also plotted for each packet erasure rate as a reference.

The figure shows that an initial decrease in c (or η) has the biggest effect on $\mathbb{E}[\mathrm{T}]$. In fact, the majority of the decrease is observed at the cost of just a few percent (2-5%) of the available network capacity when ϵ is small. As c is decreased further,

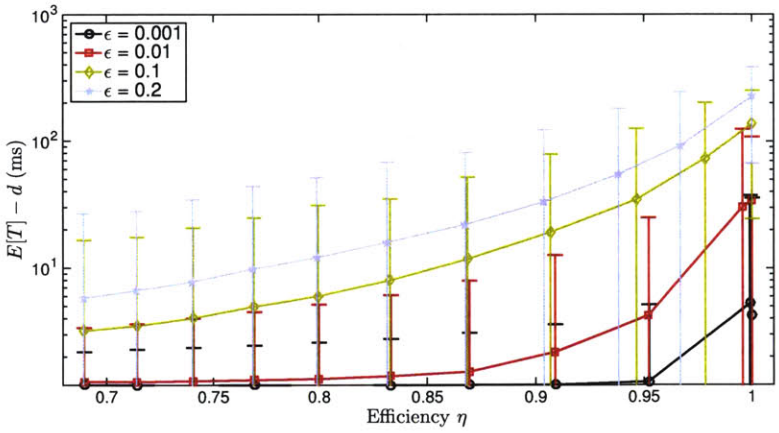


Figure 4-14: Rate-delay trade-off for a 10 Mbps link with a RTT of 100 ms. The error bars represent $2\sigma_{\mathrm{T}}$ above and below the mean, and the delay for ARQ is shown for $\eta = 1$. Note the log scale of the y-axis which skews the appearance of the results for large ϵ.

the primary benefit presents itself as a reduction in the jitter (or σ_{T}). Furthermore, the figure shows that even for high packet erasure rates (e.g., 20%), strict delay constraints can be met as long as the user is willing to sacrifice throughput.

4.9.3 Real-World Comparison

Finally, the analysis is compared with experimentally obtained results in Figure 4-15. The figure shows that the analysis above provides a reasonable approximation to real-world protocols. The experiments were conducted using Coded TCP (CTCP) over an emulated network similar to the one used in [3]. The only difference between the setup used to generate the figure here and the setup in [3] is the congestion control used in both. In Figure 4-15, CTCP's congestion control window size (*cwnd*) is fixed so that it is equal to the BDP of the network. This was done in order to eliminate the affects of fluctuating *cwnd* sizes on the in-order delivery delay.

There are several contributing factors for the differences between the experimental and analytical results shown in the figure. First, the analytical model approximates the algorithm used in CTCP. Where the analysis assumes feedback is only acted upon at the end of a generation, CTCP proactively acts upon feedback and does not wait

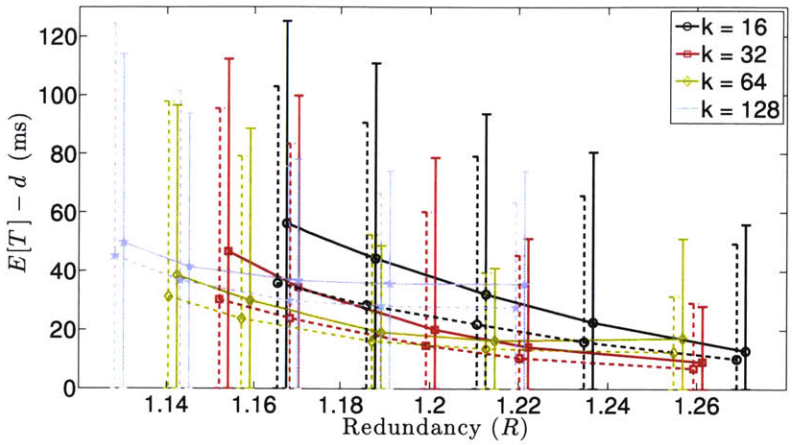


Figure 4-15: Experimental (solid lines) and analytical (dotted lines) results for various k over a 25 Mbps link with $RTT = 60$ ms and $\epsilon = 0.1$. Note, $c = 1/R$.

until the end of a generation to determine if retransmissions are required. Second, the experiments include additional processing time needed to accomplish tasks such as coding and decoding, while the analysis does not. Finally, the assumptions made in Sections 4.4 effectively lower bounds $\mathbb{E}[T]$ and $\mathbb{E}[T^2]$. The bounds are fairly tight for large k and small $c = 1/R$, but they can be very loose for either small k or large c. An example of this is evident in Figure 4-15 for $k = 16$ and $R = 1.165$ where there is a significant difference between the experimental and analytical results. However, the simulation results suggest neither small k nor small R result in k^*. Therefore, making these bounds tight in these regimes becomes less important. For all other choices of k and c, the analysis can provide a fairly good estimate of the in-order delay and can be used to help inform system decisions.

4.10 Conclusions

This chapter illustrated how network coding can help reduce the in-order delivery delay of a packet stream transmitted over multiple parallel networks. This is particularly useful for applications like Internet video, Voice on Demand, etc. where any delays in packet delivery can cause significant application layer disruptions that reduces the quality of service. The coding algorithm used throughout the chapter

is similar to the one used in CTCP, which was described in Chapter 2. Packets are partitioned into generations and redundancy is added on a generation-by-generation basis to help recover from packet erasures quickly. The primary problem addressed in this chapter was how to properly select the generation size for a given amount of added redundancy. The analysis helped show that there is a trade-off between head-of-line blocking caused by previous generations and the coding delay. Furthermore, decreasing the delay incurs a cost where this cost is quantified in terms of efficiency. It was then shown that the delay can be reduced significantly with just a minor loss of efficiency.

While the coding scheme used throughout this chapter is easy to implement, the partitioning of packets into generations places artificial constraints on the usefulness of transmitted coded packets. For example, a coded packet transmitted in generation $\boldsymbol{G}_i$ cannot help correct a packet loss that occurred in generation $\boldsymbol{G}_j$, $i \neq j$. Removing this constraint can potentially help reduce the in-order delivery delay further. This will be the topic of the next chapter where a streaming code construction is presented that employs a sliding window approach to determine the information packets used in the generation of each coded packet.

Chapter 5

In-Order Delivery Delay for Multi-Path Streaming Codes

5.1 Introduction

The analysis of the generation-based, or block, coding scheme illustrate that coding can be used to decrease the in-order delivery delay. Unfortunately, it also showed that minimizing the delay is largely dependent on selecting the proper generation size k for a given set of links. Furthermore, this minimization is not straightforward. Selecting a generation size that is too small increases both the head-of-line blocking probability and delay due to packet re-sequencing. Selecting a generation size that is too large increases the coding delay. This problem is a direct result of partitioning packets into generations and coding on a generation-by-generation basis. The streaming code discussed within this chapter removes this artificial constraint and shows that the in-order delivery delay can be decreased further by increasing the linear space spanned by each redundant coded packet.

A simple random linear network code (RLNC) [8] is used to insert redundant coded packets into a packet stream traversing multiple parallel networks or links. Unlike previous chapters, the span of information packets, or the coding window, used to generate these coded packets is not dependent on a predetermined partition of information packets. Rather, the coding window is adjusted based on the state-

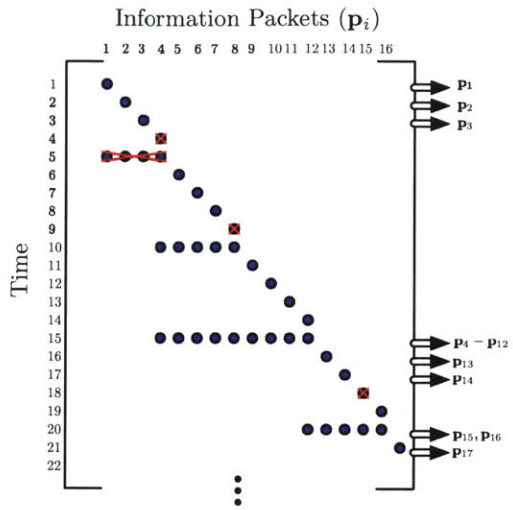


Figure 5-1: An example of the streaming code considered within this chapter.

space of the client with the intention of ensuring frequent and timely opportunities to recover from packet erasures.

An example of the streaming code used within this chapter is provided in Figure 5-1. The columns of the matrix in the figure represent the information packet $\boldsymbol{p}_i$ that must be reliably delivered to a client. The rows represent the composition of the actual transmitted packet in a given time-slot. For example, packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_4$ are transmitted in time-slots 1 through 4. A coded packet $\boldsymbol{c}_1$ consisting of a linear combination of packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_4$, depicted by multiple dots in the a single row, is transmitted in time-slot 5. The double arrow to the right of the matrix indicates the time where an information packet is delivered in-order to a client's application layer. In addition, it is assumed that delayed feedback is used to communicate the last successfully delivered information packet. In the case of the example, feedback is delayed by 6 time-slots.

In the figure, the first three information packets can be delivered immediately upon reception. However, the information packet $\boldsymbol{p}_4$ transmitted in time-slot 4 is erased and prevents delivery of any subsequent packets until the erasure is corrected. This doesn't happen until the coded packet $\boldsymbol{c}_3$ is received in time-slot 15. Unlike the generation based code described in Chapter 4, the span of information packets used to

generate coded packets is not confined to a small subset of packets. Rather, the coding window can grow, shrink, and slide based on the pattern of packet erasures, network characteristics, etc. The example, and this chapter, assumes that an information packet is added to the coding window immediately after it is first transmitted; and it is removed from the coding window once feedback has been obtained by the server indicating that it has been delivered to the client's application layer. Therefore, the coding window for the first transmitted coded packet in the figure contains information packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_4$. By the time that the second coded packet is transmitted, four additional packets have been added to the window and the first three have been removed. This is because feedback was received from the client informing the server that the first three packets were delivered.

The remainder of this chapter will describe an algorithm to deploy this code over multiple paths, develop an analysis of the expected in-order delivery delay, and provide comparisons with the generation based coding approach outlined earlier in the thesis. Simulation results will also be provided showing how the streaming code performs when the packet erasures are correlated. These results will ultimately help show the value of feedback when trying to minimize delay.

5.2 In-Order Delivery Delay Over a Single Path

The results presented in this chapter build upon the single path, low delay coding scheme originally proposed by [63, 64]. In their proposal, a single coded packet is inserted into the packet stream every l packets resulting in the code rate $c = {}^{(l-1)}/_{l}$. Each of these coded packets is assumed to be a linear combination of all transmitted information packets. For example if packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_n$ have been transmitted, then the next coded packet is $\boldsymbol{c}_n = \sum_{i=1}^{n} \alpha_{n,i}\boldsymbol{p}_i$. Let the packet erasures be i.i.d. with probability ϵ, then the in-order delivery delay for $l\epsilon < 1$ is given by the following equation:

$$\mathbb{E}[\mathrm{T}] = \frac{1}{2}\epsilon(l-1)^2\left((1-\epsilon)^{-1}+\epsilon l(1-l\epsilon)^{-2}\right). \tag{5.1}$$

This equation is obtained using a mixture of techniques from both coding theory and queuing theory. In particular, the delivery of packets is modeled as a renewal process and distributions describing the busy time of a G/D/1 queue are necessary to produce (5.1). Furthermore, the analysis also assumes that feedback is not available.

Karzand et al. [63, 64] also explore the decode failure probability. They show through numerical and simulated results that the decode failure probability is approximately zero when the coding coefficients $\alpha_{n,i}$ are chosen at random from a large enough field size (e.g., $\alpha_{n,i} \in \mathbb{F}_{2^q}$ where $q > 4$). This is a powerful result that essentially says that feedback is not necessary to provide reliability and it will be assumed that the multi-path streaming code developed here has the same decode failure probability because of the similarities in construction. However, simulated results presented later in this chapter will show that feedback is still necessary in some situations.

5.3 Multi-Path Streaming Code Algorithm

This section describes a coding scheme that allows a server to communicate over parallel paths or networks while helping to reduce the overall in-order delivery delay. Like the last chapter, a systematic code based on RLNC [8] will be used. However, the artificial constraints placed on the code by partitioning packets into generations is removed.

Information packets $\boldsymbol{p}_i$ are injected into each network uncoded. Note that the server has limited knowledge of the packets that will be sent in the future (i.e., it does not have access to the entire file). If an opportunity arises that allows the server to transmit a new packet, it does so without attempting to ensure specific packets arrive at the client in a predetermined order. After a specific number of information packets have been transmitted on any given path, the server will generate and transmit

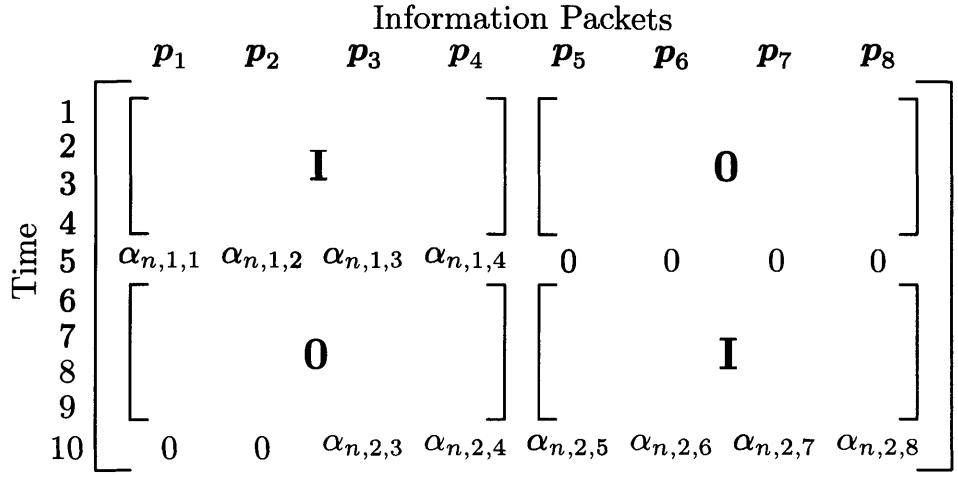


Figure 5-2: Example generator matrix used to produce the streaming code. The elements of the matrix contain the coefficients used to produce each transmitted packet.

a coded packet $\boldsymbol{c}_i$ on a path of its choosing to help overcome any packet losses that may have occurred.

Define l_i to be the duration between transmitted coded packets on path $i \in \mathcal{P}$ (i.e., $l_i - 1$ information packets are transmitted followed by a single coded packet), resulting in a code rate of $c_i = {}^{l_i-1}/_{l_i}$. If a path is idle, the server will transmit either an information packet or coded packet depending on the previously transmitted packets on that specific path. When a coded packet is generated, the information packets used to produce the linear combination are drawn from a dynamically changing coding window defined by the 2-tuple $w = (w_L, w_U)$. This results in the following packet:

$$\boldsymbol{c}_{n,k} = \sum_{i=w_L}^{w_U} \alpha_{n,k,i} \boldsymbol{p}_i. \tag{5.2}$$

The coefficients $\alpha_{n,k,i} \in \mathbb{F}_q$ are chosen at random and each information packet $\boldsymbol{p}_i$ is treated as a vector in $\mathbb{F}_q$. All of this is summarized in Algorithm 3 where $\mathbb{1}_i$ is a vector of size i consisting of all ones. In addition, an example of the generator matrix used to generate the streaming code is provided in Figure 5-2. In the example, information packets $\boldsymbol{p}_1$ through $\boldsymbol{p}_4$ and $\boldsymbol{p}_5$ through $\boldsymbol{p}_8$ are transmitted systematically

in time-slots 1 through 4 and 6 through 9 respectively. In time-slots 5 and 10, coded packets $\boldsymbol{c}_{n,1} = \sum_{i=1}^{4} \alpha_{n,1,i}\boldsymbol{p}_i$ and $\boldsymbol{c}_{n,2} = \sum_{i=3}^{8} \alpha_{n,2,i}\boldsymbol{p}_i$ are transmitted respectively. It is assumed in this example that the server has obtained feedback from the client by time 10 indicating that it successfully received and decoded packets $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. This allows the server to adjust the lower edge of the coding window to exclude the packets during the generation of coded packet $\boldsymbol{c}_{n,2}$.

Algorithm 3: Streaming Multi-Path Code Generation

```
Initialize k = 1 and u = 1_{|P|}
while k ≤ M do
    n ← First idle path found
    if u_n ≤ l_n then
        Transmit packet p_k
        u_n ← u_n + 1
        k ← k + 1
    else
        Transmit coded packet c_{n,k} = Σ_{i=w_L}^{w_U} α_{n,k,i} p_i
        u_n ← 1
```

Before proceeding, it must be noted that Algorithm 3 does not explicitly take advantage of feedback in determining when to inject coded packets into the packet stream. Rather, feedback is only used to estimate the packet erasure probability ϵ_i on each path $i \in \mathcal{P}$. This is done in order to simplify the analysis that will be presented later in the chapter. However, using feedback can only improve the algorithm's performance; and implemented versions should use feedback intelligently when determining when to inject redundancy to help reduce the delay further.

While Algorithm 3 is fairly simple, two topics jump out that require special consideration. First, the selection of code rates c_i on every path $i \in \mathcal{P}$ must be done properly to ensure that the client can decode within a reasonable time period regardless of the observed packet losses. Second, management of the coding window w must be performed carefully to ensure coded packets add to the knowledge space of the client. These two topics will be addressed by defining the following:

Definition 5.1. A coding policy π determines the code structure and rates used on each path between a server and client.

In other words, the coding policy defines the code rates used to generate coded packets on each path, as well as the algorithm for managing w. There are, in fact, an infinite number of coding policies. However, policies that allow the client to decode with high probability within a reasonable time frame are the only ones of interest. This leads to the next definition:

Definition 5.2. A coding policy that ensures the client will decode with probability equal to 1 is said to be admissible.

Let $\prod = \{\pi_1, \pi_2, \pi_3, \ldots\}$ be the set of all admissible coding policies. The code rates and code window management rules for policy π_i will be referred to as the $|\mathcal{P}|$-tuple $c(\pi_i) = \left(c_1(\pi_i), \ldots, c_{|\mathcal{P}|}(\pi_i)\right)$ and $\mathcal{W}(\pi_i)$ respectively. The following sub-sections will help define both $c(\pi_i)$ and $\mathcal{W}(\pi_i)$ for each policy $\pi \in \Pi$.

5.3.1 Code Rate Selection

An admissible coding policy must ensure the client's capability to decode. One of the most important parts is choosing the appropriate code rate $c(\pi_i)$. The following theorem helps determine this rate.

Theorem 5.3. *An admissible coding policy π must satisfy the following constraints:*

$$\sum_{i \in \mathcal{P}} (1 - \epsilon_i)\left((1 - \epsilon_i) - c_i(\pi)\right) r_i > 0, \tag{5.3}$$

and

$$c_i(\pi) \in [0, 1]. \tag{5.4}$$

Proof. A path with code rate $c_i(\pi)$ and transmission rate r_i packets/seconds results in a coded packet being generated every $(1 - c_i(\pi)) r_i$ seconds. Therefore, the expected rate that coded packets arrive at the client on path i is $(1 - \epsilon_i)(1 - c_i(\pi)) r_i$ resulting in $\sum_{i \in \mathcal{P}} (1 - \epsilon_i)(1 - c_i(\pi)) r_i$ total coded packets/second. Now consider the case when each path is treated as a separate session. The code rate on path i must satisfy $c_i < 1 - \epsilon_i$ in order to ensure the client's ability to decode (i.e., the probability of a

decoding error $Pr\{\mathcal{E}\} \rightarrow 0$ as the file size increases for all $c_i < 1 - \epsilon_i$). Allowing the code rate to be $c_i = 1 - \epsilon_i$, the expected rate at which coded packets arrive at the client on path i is then equal to $(1 - \epsilon_i)\,\epsilon_i r_i$ resulting in the sum rate $\sum_{i \in \mathcal{P}} (1 - \epsilon_i)\,\epsilon_i r_i$. This produces the following bound:

$$\sum_{i \in \mathcal{P}} (1 - \epsilon_i)\,(1 - c_i\,(\pi))\,r_i > \sum_{i \in \mathcal{P}} (1 - \epsilon_i)\,\epsilon_i r_i. \tag{5.5}$$

Rearranging, we arrive at (5.3). With regard to (5.4), it is obvious that $c_i\,(\pi)$ must satisfy $c_i\,(\pi) \in [0, 1]$. □

Transmitting coded packets over multiple networks maybe the appropriate strategy in some cases; but in others, it maybe better to only send coded packets over a single network. This leads to the following corollary.

Corollary 5.4. *For the case when coded packets are only transmitted over a single path and there exists a π such that $c_i\,(\pi) \in [0, 1]$ and $c_j\,(\pi) = 1$ for $i, j \in [1, |\mathcal{P}|]$, $i \neq j$, the admissible coding policy π must satisfy the following:*

$$c_i\,(\pi) \;\; < \;\; 1 - \frac{\sum_{k \in \mathcal{P}} (1 - \epsilon_k)\,\epsilon_k r_k}{(1 - \epsilon_i)\,r_i}. \tag{5.6}$$

5.3.2 Code Window Management

For admissible coding policies, the code window used to generate coded packets must provide the potential for the client's knowledge space to increase in the presence of packet losses. There are many ways of accomplishing this goal ranging from schemes that code on a generation-by-generation bases to schemes that code over the entire packet stream. While there is no guarantee that the scheme proposed here is optimal, it does lead to an admissible coding policy.

As a reminder, it is assumed that coded packets are used solely for redundancy. If the path or network is error-free, coded packets will not contribute to the knowledge space of the client. In addition, it is assumed that coding occurs over a packet stream

where the server has limited to no knowledge of packets that will be sent in the future. Therefore, any decisions regarding the code window management must be made using information packets that have already been sent and information from feedback that is at least RTT seconds old. Before an algorithm is proposed, the concept of a *seen* packet from [19] must be established.

Definition 5.5. The client is said to have *seen* a packet $\boldsymbol{p}_i$ if it has enough information to compute a linear combination of the form $(\boldsymbol{p}_i + \boldsymbol{q})$ where $\boldsymbol{q} = \sum_{k>i} \alpha_k \boldsymbol{p}_k$ with $\alpha_k \in \mathbb{F}_q$ for all $k > i$. Therefore, $\boldsymbol{q}$ is a linear combination involving information packets with indices larger than i.

Define **seen** to be the index of the last seen information packet at the client that is composed of the set of all consecutive *seen* information packets. It is assumed that the client informs the server of the value of **seen** through feedback. Once the feedback has been received by the server, **seen** will be used to set the lower edge of the code window. The upper edge of the code window will be managed based on the index of the last transmitted uncoded information packet. This is summarized in Algorithm 4, which is executed by the server and is agnostic to the path on which any one packet is transmitted.

Algorithm 4: Code Window Management

Initialize $(w_L, w_U) = (0, 0)$ and $j = 0$
if $\boldsymbol{p}_i$ *transmitted uncoded* **and** $i > w_U$ **then**
    $w_U \leftarrow i$
if *Feedback received* **and** **seen** $> w_L$ **then**
    $w_L \leftarrow$ **seen**

Since **seen** is required to be the last *seen* packet out of the set of consecutive packets, the client will eventually be able to decode given the transfer of enough degrees of freedom. Furthermore, using *seen* packets to manage the code window helps to decrease the size of the coding/decoding buffers on the server/client respectively.

5.4 System Model

A time-slotted model is assumed where a single server-client pair are communicating with each other over multiple parallel networks. Denote this set of disjoint networks as $\mathcal{P}$. Data is first placed into information packets $\boldsymbol{p}_1, \boldsymbol{p}_2 \ldots$. These information packets are then used to generate coded packets $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \ldots$. Depending on the coding policy, the server chooses to transmit either an information packet or coded packet over one of the network paths. The time it takes to transmit either type of packet is $t_i = 1/r_i$ seconds where r_i is the transmission rate in packets/second of network $i \in \mathcal{P}$. Furthermore, it takes each packet d_i seconds to propagate through network i (e.g., $RTT_i = t_i + 2d_i$ on network i assuming that the size of the feedback packet is sufficiently small).

Delayed feedback is available to the server allowing it to estimate each paths' i.i.d. packet erasure rate ϵ_i and round-trip time RTT_i (in seconds). However, the server is unable to determine the cause of the packet erasures (e.g., poor network conditions or congestion). Furthermore, the server has knowledge of each network's transmission rate r_i, which can either be determined from feedback obtained from the client or from the size of the server's congestion window on any specific path (e.g., $r_i = {}^{cwnd_i}/_{RTT_i}$). This feedback can also be used to communicate to the server the number of $dofs$ received by the client. While the analysis later in the chapter assumes feedback does not contain this information, numerical and simulated results will use feedback to dynamically adjust the code rate depending on the client's dof deficit.

5.4.1 Analysis of the In-Order Delivery Delay

Before proceeding, several assumptions are required to simplify the analysis. First, it is assumed that coded packets are only sent over a single network and the code rates conform to Corollary 5.4. The rate and packet erasure probability of the network used to send coded packets will be referred to as r_c and ϵ_c respectively. Second, packets transmitted over faster networks are delayed so that they arrive in-order with packets transmitted over slower networks. For example, assume that packets

are transmitted over two disjoint networks with propagation delays d_1 and d_2 where $d_1 < d_2$. Packets transmitted over network 1 will be delayed an additional $d_2 - d_1$ seconds. This assumption affects the analysis by over-estimating the delay since there is a possibility that packets transmitted over the faster networks can be delivered in-order without waiting for a packet from the slower network. However, the number of packets transmitted over the faster networks that can be delivered without packets from the slower networks is relatively small. Third, the coding window used for each coded packet contains all transmitted information packets. This assumption is not necessary if the code window management follows Algorithm 4. However, it does remove any ambiguity regarding the usefulness of a received coded packet.

The in-order delivery delay for the code provided in Algorithm 3 can be determined using a renewal-reward process based off of the number of transmitted coded packets on path $p_c \in \mathcal{P}$. More accurately, a renewal occurs whenever a received coded packet results in a decoding event. This occurs whenever the number of received coded packets is greater than or equal to the number of lost information packets. Per Algorithm 3, a coded packet is transmitted every

$$l_c = \frac{1}{1 - c_c(\pi)} \tag{5.7}$$

packets on path p_c. This results in the transmission of

$$\alpha_i = (l_c - 1)\frac{r_i}{r_c} \tag{5.8}$$

information packets on each network $i \in \mathcal{P}$ for every transmitted coded packet. Consider a time-slotted model where each time slot has duration l_c/r_c. Now define the sequence $X_1, X_2, \ldots$, where $X_n = 0, 1, 2, \ldots$ slots, to be the independent and identically distributed (i.i.d.) inter-arrival times between decode events with first and second moments $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$ respectively. The arrival process is then a sequence of increasing random variables, or arrival epochs, $0 \leq S_1 \leq S_2 \leq \cdots$ where the nth epoch $S_n = \sum_{i=1}^{n} X_n$.

In order to determine the distribution and moments of $X_1, X_2, \ldots$, several addi-

tional random variables need to be defined. Let the random variable $Y_{n,i}$, $i = 1, 2, \ldots$, be the number of lost packets (both information and coded) between $S_{n-1} + (i-1)$ and $S_{n-1} + i$ in the nth arrival epoch. The exact distribution is the convolution of $|\mathcal{P}|$ binomial distributions with parameters α_i and ϵ_i for each $i \in \mathcal{P}$. In order to simplify the analysis, this distribution is approximated by the following Poisson distribution:

$$p_{Y_{n,i}}(y_{n,i}) = \frac{\lambda^{y_{n,i}}}{y_{n,i}!} e^{-\lambda}, \quad y_{n,i} = 0, 1, \ldots, \tag{5.9}$$

where

$$\lambda = \epsilon_c + \sum_{i \in \mathcal{P}} \alpha_i \epsilon_i. \tag{5.10}$$

If $Y_{n,1} = 0$, the number of received packets between S_{n-1} and $S_{n-1}+1$ is $1+\sum_{i\in\mathcal{P}} \alpha_i$ while only $\sum_{i\in\mathcal{P}} \alpha_i$ packets were necessary to decode (i.e., the coded packet is of no benefit and is dropped). Therefore, $X_n = 0$. However if $Y_{n,1} > 0$, at least one packet was lost which may prevent delivery. Therefore, the extra dof obtained from coded packets will help correct these erasures and eventually lead to a decode event. As an example, consider the case when $Y_{n,1} = 1$. A single packet was lost, but enough $dofs$ were received to decode all of the packets transmitted between S_{n-1} and $S_{n-1} + 1$. Therefore, the inter-arrival time is $X_n = 1$. Now consider the case when $Y_{n,1} = 2$ and $Y_{n,2} = 0$. It is impossible for a renewal to occur between at S_{n-1} or $S_{n-1} + 1$; however a renewal does occur at $S_{n-1}+2$. This results in an inter-arrival time $X_n = 2$. Continuing on in this way, it becomes clear that a renewal occurs the first time Z that $\sum_{i=1}^{Z} Y_{n,i} \leq Z$.

In fact, Z is a random variable and can be modeled as a M/D/1 queue with a constant service time of 1 packet per slot and an arrival rate of λ packets per slot. Define $Y = \sum_{i=1}^{Z} Y_{n,i}$, then Z conditioned on Y has the following Borel-Tanner distribution [65]:

$$p_{Z|Y}(z|y) = \frac{y z^{z-y-1} \lambda^{z-y} e^{-z\lambda}}{(z-y)!}, \quad \text{for } y = 1, 2, \ldots \text{ and } z = y, y+1, \ldots. \tag{5.11}$$

Equations (5.9) and (5.11) can now be used to determine the distribution of X_n and

its first two moments.

Theorem 5.6. *Let the number of packets lost in a single time-slot be independent and identically distributed according to equation (5.9). The distribution of the time between decode events for all ϵ_i and r_i, $i \in \mathcal{P}$, that satisfy Corollary 5.4 is*

$$p_{X_n}(x_n) = \begin{cases} e^{-\lambda} & \text{for } x_n = 0 \\ \lambda e^{-\lambda} & \text{for } x_n = 1 \\ \frac{(x_n-1)^{x_n-2}}{x_n(x_n-2)!}\lambda^{x_n}e^{-x_n\lambda} & \text{for } x_n \geq 2 \\ 0 & \text{otherwise,} \end{cases} \tag{5.12}$$

with first and second moments

$$\mathbb{E}[X] = \frac{\lambda}{1-\lambda}e^{-\lambda} \tag{5.13}$$

and

$$\mathbb{E}\left[X^2\right] = \mathbb{E}[X] + \frac{\lambda^2}{(1-\lambda)^3}e^{-\lambda}. \tag{5.14}$$

Proof. The inter-arrival time X_n takes the values of $x_n = 0$ and $x_n = 1$ if and only if $y_{n,1} = 0$ with probability $e^{-\lambda}$ and $y_{n,1} = 1$ with probability $\lambda e^{-\lambda}$ respectively. For all $X_n \geq 2$, we must have $y_{n,1} \geq 2$. This results in a decoding error in the first time-slot. Conditioning on $Y_{n,1}$, we can use equation (5.11) to find the probability for $X_n \geq 2$ by setting $Z = X_n - 1$ and $Y = Y_{n,1} - 1$:

$$p_{X_n}(x_n) = \sum_{y_{n,1}=2}^{x_n} p_{Y_{n,1}}(y_{n,1})\, p_{Z|Y}(x_n - 1 | y_{n,1} - 1) \tag{5.15}$$

$$= \sum_{y_{n,1}=2}^{x_n} \frac{\lambda^{y_{n,1}}e^{-\lambda}}{y_{n,1}!} \cdot \frac{(y_{n,1}-1)(x_n-1)^{x_n-y_{n,1}-1}\lambda^{x_n-y_{n,1}}e^{-(x_n-1)\lambda}}{(x_n-y_{n,1})!} \tag{5.16}$$

$$= \sum_{y_{n,1}=2}^{x_n} \frac{(y_{n,1}-1)(x_n-1)^{x_n-y_{n,1}-1}}{x_1!(z-x_1)!} \tag{5.17}$$

$$= \frac{(x_n-1)^{x_n-2}}{x_n(x_n-2)!}\lambda^{x_n}e^{-x_n\lambda}. \tag{5.18}$$

To determine the moments of X_n, first note that

$$\sum_{x_n=2}^{\infty} \frac{(x_n-1)^{x_n-2}}{(x_n-2)!}\lambda^{x_n}e^{-x_n\lambda} = \mathbb{E}[X] - \lambda e^{-\lambda} \tag{5.19}$$

and

$$\sum_{x_n=2}^{\infty} \frac{x_n(x_n-1)^{x_n-2}}{(x_n-2)!}\lambda^{x_n}e^{-x_n\lambda} = \mathbb{E}\left[X^2\right] - \lambda e^{-\lambda}. \tag{5.20}$$

We can then take the first and second derivatives of

$$\sum_{x_n=0}^{\infty} p_{X_n}(x_n) = 1, \tag{5.21}$$

i.e.,

$$\frac{\partial^i}{\partial\lambda^i}\left(e^{-\lambda} + \lambda e^{-\lambda} + \sum_{x_n=2}^{\infty} \frac{(x_n-1)^{x_n-2}}{x_n(x_n-2)!}\lambda^{x_n}e^{-x_n\lambda}\right) = 0 \tag{5.22}$$

for $i = \{1, 2\}$, to find $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$ respectively. For both $\mathbb{E}[X] < \infty$ and $\mathbb{E}[X^2] < \infty$, the rate of packet loss across all paths must be $\lambda < 1$. This corresponds with Corollary 5.4 after substituting in equations (5.7) and (5.8) when $l_c\epsilon_i \approx 1\ \forall i \in \mathcal{P}$. □

The first two moments of X_n can now be used to determine the the renewal-reward process that describes the in-order delivery delay. Before this is done, the following lemma from [66] is needed.

Lemma 5.7. *Let $\{R(t); t > 0\}$ be a non-negative renewal-reward function for a renewal process with expected inter-renewal time $\mathbb{E}[X] < \infty$. If each R_n is a random variable with $\mathbb{E}[R_n] < \infty$, then with probability 1,*

$$\lim_{t\to\infty} \frac{1}{t}\int_{t=0}^{t} R(\tau)\, d\tau = \frac{\mathbb{E}[R_n]}{\mathbb{E}[X]}. \tag{5.23}$$

Rather than defining the renewal reward function $R(t)$ and the renewal-reward process using the inter-arrival times X_n, an estimate is considered where the inter-arrival times of this new process are $W_n = \max(X_n, 1)$ (i.e., $W_n = 1, 2, \ldots$). The distribution on W_n and its first moment are defined in the following.

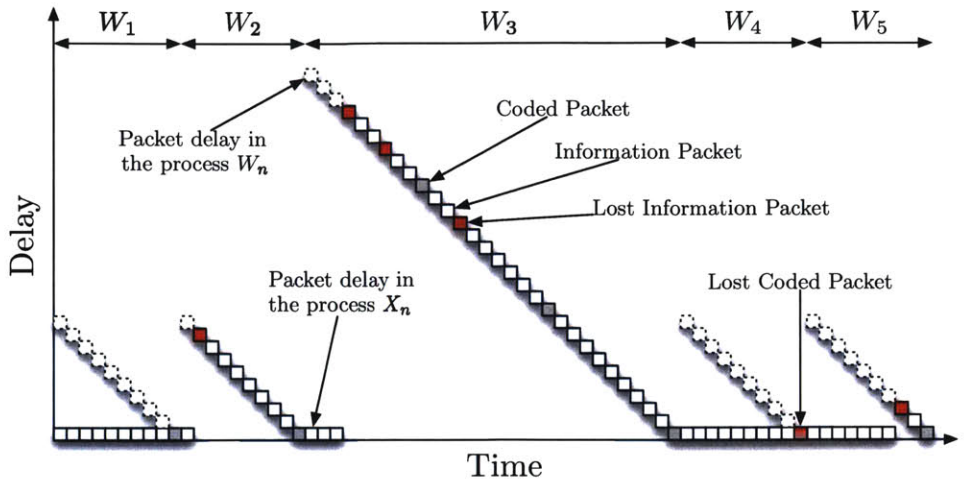


Figure 5-3: An example of the processes X_n and W_n and the reward function $R(t)$ for a single path.

Corollary 5.8. *Let the number of packets lost in a single time-slot be independent and identically distributed according to equation (5.9) and define* $W_n = \max(X_n, 1)$. *The the distribution of inter-arrival times that satisfy Corollary 5.4 is*

$$p_{W_n}(w_n) = \begin{cases} (\lambda+1)\, e^{-\lambda} & \text{for } w_n = 1 \\ \frac{(w_n-1)^{w_n-2}}{w_n(w_n-2)!}\lambda^{w_n} e^{-w_n\lambda} & \text{for } w_n \geq 2 \\ 0 & \text{otherwise,} \end{cases} \tag{5.24}$$

where

$$\mathbb{E}[W] = \frac{1}{1-\lambda} e^{-\lambda}. \tag{5.25}$$

Figure 5-3 provides a sample function of both renewal processes X_n and W_n. The reward function that describes the in-order delay precisely is the area under the curve shown for process X_n (i.e., the blocks with solid borders). However, the reward function $R(t)$ that describes the delay for process W_n is the one that is used. The packet delay in this process is shown by the blocks with dashed borders in the figure. The in-order delivery delay can now be determined by combining Lemma 5.7, Theorem 5.6, and Corollary 5.8.

Theorem 5.9. *Consider the coding scheme described by Algorithm 3 where redundant packets are only transmitted on path $i_c \in \mathcal{P}$. With probability one, the in-order delivery delay $\mathbb{E}[\mathrm{T}]$ is given by*

$$\mathbb{E}[\mathrm{T}] = \frac{\left(r_c^3(l_c-1) + l_c \sum_{j\in\mathcal{P}\setminus i_c} r_j^3\right)\mathbb{E}[X^2]}{2r_c^2\mathbb{E}[W]\sum_{i\in\mathcal{P}} r_i} - \frac{\mathbb{E}[X]\sum_{j\in\mathcal{P}\setminus i_c} r_j^2}{2r_c\mathbb{E}[W]\sum_{i\in\mathcal{P}} r_i}. \tag{5.26}$$

Proof. The renewal-reward function to determine the delay experienced by an information packet transmitted on path $P \in \mathcal{P}$ is similar to the residual life of the process with some modifications. Define R_n given W_n to be the sum delay of all information packets on path P:

$$R_n = \begin{cases} \sum_{k=1}^{W_n l_c} k - l_c \sum_{k=1}^{W_n} k & \text{for } P = i_c \\ \frac{r_c}{r_i} \sum_{k=0}^{\frac{W_n l_c r_i}{r_c}-1} k & \text{for } P \neq i_c \end{cases} \tag{5.27}$$

Taking the expectation of R_n, we obtain the following

$$\mathbb{E}[R_n] = \sum_{i\in\mathcal{P}} \sum_{w_n=1}^{\infty} \mathbb{E}[R_n | P=i, W_n=w_n]\, p_P(i)\, p_{W_n}(w_n) \tag{5.28}$$

$$= \frac{1}{\sum_{j\in\mathcal{P}} r_j} \sum_{i\in\mathcal{P}} \sum_{w_n=1}^{\infty} r_i \mathbb{E}[R_n | P=i, W_n=w_n]\, p_{W_n}(w_n) \tag{5.29}$$

$$= \frac{1}{\sum_{j\in\mathcal{P}} r_j} \sum_{w_n=1}^{\infty} \Bigg(r_c \mathbb{E}[R_n | P=i_c, W_n=w_n] + \sum_{i\in\mathcal{P}\setminus i_c} r_i \mathbb{E}[R_n | P=i, W_n=w_n] \Bigg) p_{W_n}(w_n) \tag{5.30}$$

$$= \frac{1}{\sum_{j\in\mathcal{P}} r_j} \sum_{w_n=1}^{\infty} \Bigg(r_c \mathbb{E}\left[\sum_{k=1}^{w_n l_c} k - l_c \sum_{k=1}^{w_n} k\right] + \sum_{i\in\mathcal{P}\setminus i_c} r_i \mathbb{E}\left[\frac{r_c}{r_i} \sum_{k=0}^{\frac{w_n l_c r_i}{r_c}-1} k\right] \Bigg) p_{W_n}(w_n). \tag{5.31}$$

Substituting x_n for w_n from Corollary 5.8,

$$\mathbb{E}[R_n] = \frac{1}{\sum_{j\in\mathcal{P}} r_j} \sum_{x_n=0}^{\infty} \left(r_c \mathbb{E}\left[\sum_{k=1}^{\max(x_n,1)l_c} k - l_c \sum_{k=1}^{\max(x_n,1)} k \right] + \sum_{i\in\mathcal{P}\setminus i_c} r_i \mathbb{E}\left[\frac{r_c}{r_i} \sum_{k=0}^{\frac{\max(x_n,1)l_c r_i}{r_c}-1} k \right] \right) p_{X_n}(x_n) \tag{5.32}$$

$$= \frac{1}{\sum_{j\in\mathcal{P}} r_j} \sum_{x_n=0}^{\infty} \left(\frac{l_c r_c}{2}(l_c-1)x_n^2 + \sum_{i\in\mathcal{P}\setminus i_c} \frac{l_c r_i^2}{2r_c^2}\left(l_c r_i x_n^2 - r_c x_n\right) \right) p_{X_n}(x_n) \tag{5.33}$$

$$= \frac{1}{\sum_{j\in\mathcal{P}} r_j} \left(\frac{l_c r_c}{2}(l_c-1)\mathbb{E}[X^2] + \sum_{i\in\mathcal{P}\setminus i_c} \frac{l_c r_i^2}{2r_c^2}\left(l_c r_i \mathbb{E}[X^2] - r_c \mathbb{E}[X]\right) \right). \tag{5.34}$$

Since both $\mathbb{E}[X] < \infty$ and $\mathbb{E}[X^2] < \infty$, the expectation $\mathbb{E}[R_n] < \infty$ and Lemma 5.7 can be applied. Keeping in mind that every time-slot in the process defined by W_n is divided into l_c smaller time-slots, the expected in-order delivery delay is

$$\mathbb{E}[\mathrm{T}] = \lim_{t\to\infty} \frac{1}{l_c t} \int_0^t R(\tau)\, d\tau \tag{5.35}$$

$$= \frac{\mathbb{E}[R_n]}{l_c \mathbb{E}[W]} \tag{5.36}$$

$$= \frac{1}{\sum_{j\in\mathcal{P}} r_j} \left(\frac{r_c}{2}(l_c-1)\mathbb{E}[X^2] + \sum_{i\in\mathcal{P}\setminus i_c} \frac{r_i^2}{2r_c^2}\left(l_c r_i \mathbb{E}[X^2] - r_c \mathbb{E}[X]\right) \right). \tag{5.37}$$

□

5.5 Numerical and Simulation Results

The in-order delivery delay $\mathbb{E}[\mathrm{T}]$ derived in the last section provides a useful approximation that can be used to determine the performance of streaming codes operating over multiple parallel network paths. Unlike the analysis in [64], the multi-path anal-

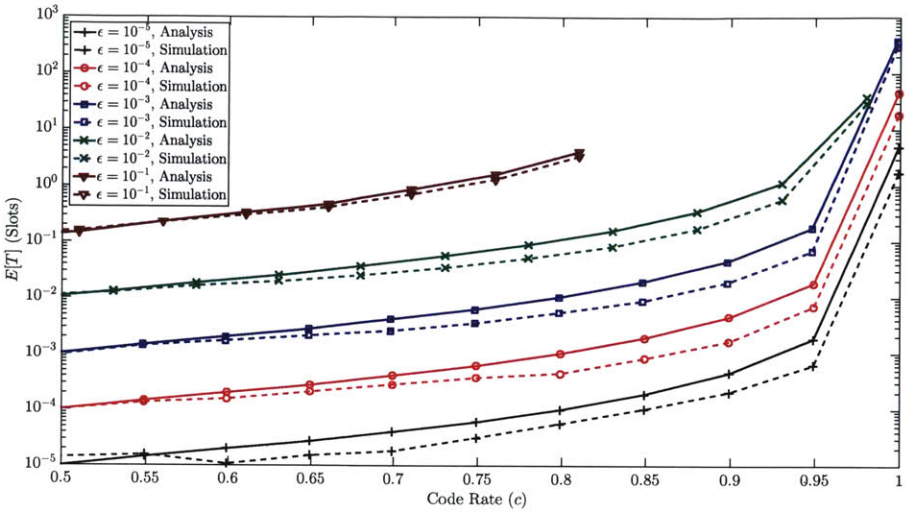


Figure 5-4: Simulated and analytical in-order delivery delay for a streaming code over a single path.

ysis is not technically an upper bound since approximations were made with regards to λ and the α_i's (see (5.10) and (5.8) respectively). Regardless, the results presented within this section show that the approximation is fairly good over a range of network conditions.

Over a single path, the delay given by (5.26) is reduced to

$$\mathbb{E}\left[\mathrm{T}\right] = \frac{(l-1)\,\mathbb{E}\left[X^2\right]}{2\mathbb{E}\left[W\right]}, \tag{5.38}$$

which is similar to the in-order delivery delay calculated in [64]. A comparison of this delay with simulated results is provided in Figure 5-4. The figure demonstrates that the approximation above is a fairly good measure of the true in-order delivery delay over a range of code rates and packet erasure probabilities.

Figure 5-5 also provides a comparison between the analytical and simulated delay for two paths. A single path with transmission rate r_c and packet erasure rate ϵ_c is used to transmit all of the coded packets in addition to information packets. The second path, which is only used to transmit information packets, has transmission rate r_n and packet erasure rate ϵ_n. The analytical delay given by (5.26) in this case

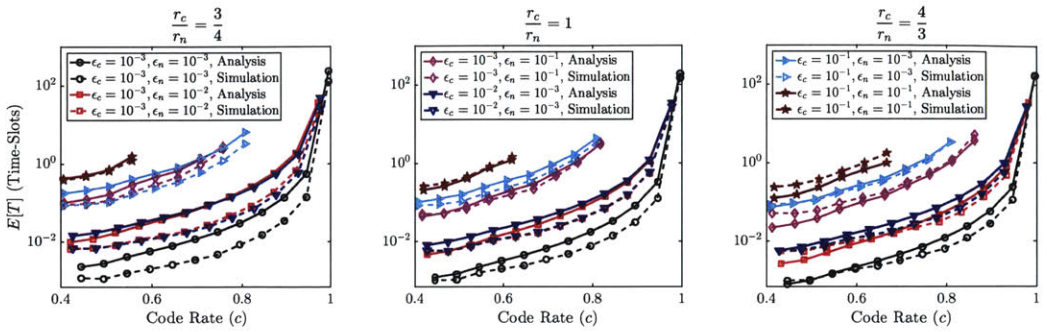


Figure 5-5: Simulated and analytical in-order delivery delay for a streaming code over two disjoint paths. The rate of the coding path is r_c and the rate of the non-coding path is r_n.

is reduced to

$$\mathbb{E}\left[\mathrm{T}\right] = \frac{\left(r_c^3\left(l_c - 1\right) + l_c r_n^3\right)\mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right] r_n^2 r_c}{2 r_c^2 \mathbb{E}\left[W\right]\left(r_c + r_n\right)}, \tag{5.39}$$

and is plotted for various combinations of packet erasure rates and transmission rates. The first subfigure, labeled $r_c/r_n = 3/4$, shows the delay of the multi-path streaming code scheme over two paths where the coding path has a slower transmission rate than the non-coding path; the second subfigure, labeled $r_c/r_n = 1$, shows the case where the two paths with the same transmission rate; and the third subfigure, labeled $r_c/r_n = 4/3$, shows the case where the coding path has a faster transmission rate. When $r_c < r_n$, the analysis tends to overestimate the delay while the analysis tends to underestimate the delay when $r_c > r_n$. Regardless, the simple analysis provided above gives a fairly good approximation of the in-order delivery delay that can be expected when communicating over multiple paths.

5.6 A Comparison Between Generation Based Codes and Streaming Codes

Both the generation based code discussed in Chapter 4 and the streaming code discussed within this chapter have their benefits and drawbacks. It is easy from a cod-

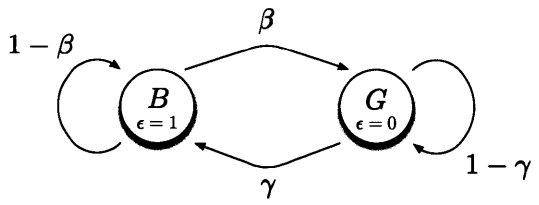


Figure 5-6: Gilbert channel used to produce correlated losses.

ing perspective to implement the generation based coding scheme, and these schemes achieve capacity when the generation size $k \to \infty$. However, partitioning packets into generations adds artificial restrictions on the code's capability to recover from losses; and doing so may not be as efficient as streaming code schemes. Furthermore, generation based schemes can increase the complexity of the feedback process, especially for reliable data transfers. Streaming code schemes, on the other hand, can outperform generation based schemes in terms of efficiency and delay. Unfortunately, code window management can be difficult and these schemes typically cannot guarantee that a decoding event occurs before the termination of a session. In addition, the size of the code window maybe much larger than the generation based schemes leading to increased decoding complexity and communication overhead.

This section will provide a comparison between generation based codes and streaming codes. While only single path transmission is covered; comparisons of these codes in both reliable and unreliable settings are provided. The only difference in code construction between the reliable, or closed-loop, approaches presented earlier in the thesis and the unreliable, or open-loop, approaches included here is how the server responds to feedback. In the closed-loop case, the server uses feedback to insert additional $dofs$ into the network as necessary. In the open-loop case, the server only uses the feedback to estimate the packet loss probability of the network. The net effect of this difference is that the application layer observes a non-zero packet erasure rate or, in some cases, increased delay.

In addition, a correlated packet loss model is also considered. This model uses the simple Gilbert channel shown in Figure 5-6. The probability of transitioning from the "good" state G, which has a packet erasure rate equal to zero, to the "bad" state

B, which has a packet erasure rate equal to one, is γ. Similarly, the probability of transitioning from B to G is β. This results in the following the transition probability matrix:

$$M = \begin{bmatrix} 1-\gamma & \gamma \\ \beta & 1-\beta \end{bmatrix}. \tag{5.40}$$

The steady-state distribution of B,

$$\pi_B = \frac{\gamma}{\gamma + \beta}, \tag{5.41}$$

and the expected number of packet erasures in a row, or burst duration

$$\mathbb{E}[L] = \frac{1}{\beta}, \tag{5.42}$$

will be used as the primary parameters for determining the transition probabilities of the channel model. Either the erasure rate ϵ or the 2-tuple $(\pi_B, E(L) = 1)$ (note that π_B equals the erasure rate) will be used to denote the i.i.d. packet loss model. The 2-tuple $(\pi_B, E(L) > 1)$ will be used when the correlated packet loss model is used. It should also be noted that the correlated packet loss model does not necessarily capture the affects of fading, which can have a duration equal to hundreds of milliseconds to hours. Instead, the use of the Gilbert channel is intended to help model the cases where the signal-to-noise ratio (SNR) is such that the performance of the underlying physical layer code is degraded; but the situation does not warrant the need to change to a more robust physical layer modulation/coding scheme.

5.6.1 Closed-Loop Performance

In the closed-loop setting, feedback is used, regardless of the coding scheme, to improve both delay and throughput performance. Algorithm 2 determines how the server responds to feedback when using the generation based code. For the streaming code, the server adjusts the code rate based on the expected number of $dofs$ in-flight

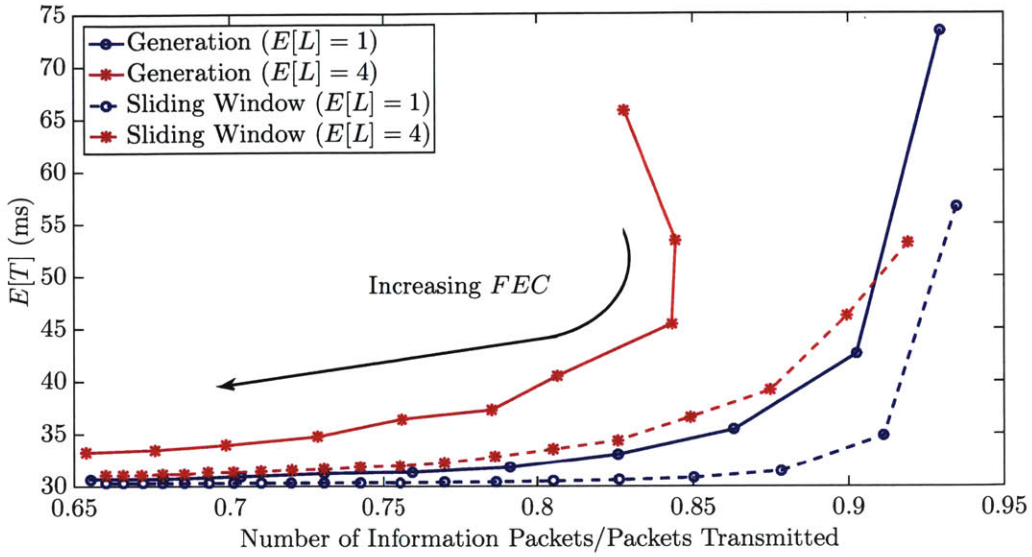
and the number of *dofs* required to decode that is reported by the client.

Figure 5-7 compares the in-order delivery delay of both codes as a function of the observed code rate. The figure also shows results for two average packet loss rates, π_B, and two expected packet loss burst durations, $\mathbb{E}[L]$. The observed code rate is defined as the number of information packets divided by the number of transmitted packets and is dependent on the number of observed packet losses. For the generation based code, an initial code rate is chosen that is used to generate coded packets for forward error correction (FEC). If a generation cannot be decoded and retransmissions are required, the additional transmitted coded packets further reduce the code rate. The streaming code's observed code rate is similar. An initial code rate is used to determine when to insert coded packets, but feedback is used to adjust the code rate if the client is unable to decode in a timely manner.

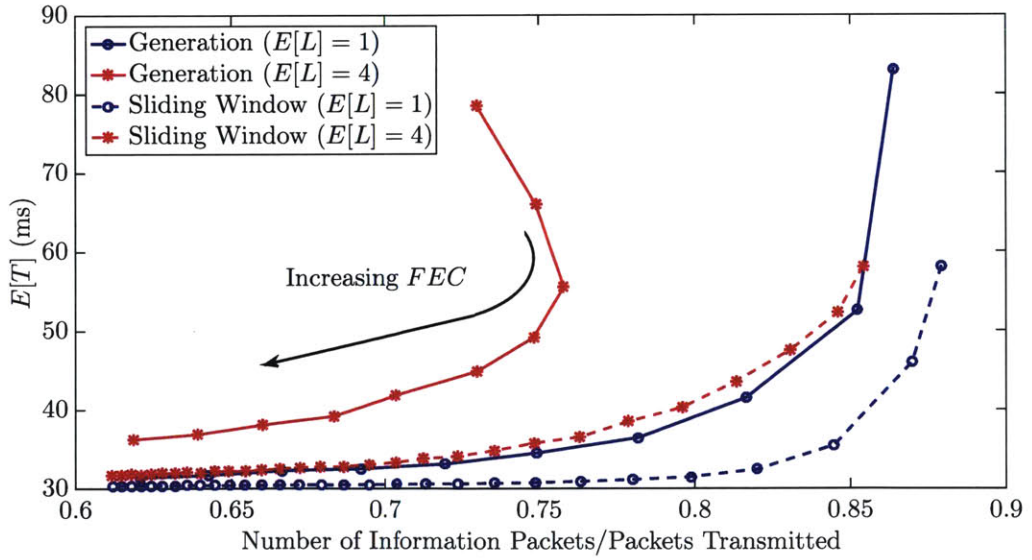
It is clear from the figure that the streaming code provides a lower in-order delivery delay than the generation based code regardless of π_B or $\mathbb{E}[L]$. While the difference in delay for i.i.d. packet losses (i.e., $\mathbb{E}[L] = 1$) is not that large, the difference when $\mathbb{E}[L] = 4$ is significant. Both subfigures show that the generation based code is unable to obtain a high code rate when the losses are correlated. This is illustrated by the non-uniqueness in the abscissa. As previously noted, an initial code rate is used to determine how many coded packets are used for forward error correction. When this code rate is high, the probability of a decode error is fairly large resulting in the server retransmitting additional *dofs*. This effectively decreases the code rate while exhibiting a very large in-order delivery delay. Because coded packets are able to correct for packet erasures over a larger span of packets, a higher code rate can be used to achieve the same in-order delivery delay.

5.6.2 Open-Loop Performance

Data streams such as real-time voice and video do not necessarily require 100% reliability. However, decreasing the underlying packet erasure rates may still drastically improve upper layer quality of service. Recent work in this area has shown that network coding is one tool that can help improve performance [67], [68]. This section will



(a) $\pi_B = 0.05$



(b) $\pi_B = 0.1$

Figure 5-7: Closed-loop in-order delivery delay as a function of the code rate on a 25 Mbps link with a RTT of 60 ms.

compare both the generation based and streaming code schemes with respect to the upper-layer packet erasure probabilities and the expected in-order delivery delays.

The generation based coding scheme shown in Algorithm 1, without using Algorithm 2, is ideally suited to the case where there is a delay constraint and packet delivery is not guaranteed. Packets within each generation are delivered in-order until the first packet loss is encountered. Once the entire generation has been received, the client attempts to decode it. If the generation cannot be decoded, only the successfully received information packets are delivered. If the generation can be decoded, every information packet contained in the generation is delivered in-order.

Modifying the streaming code scheme shown in Algorithm 3 for unreliable data streams is somewhat difficult. The code is constructed in such a way that the probability of a decode error is nearly zero [64]. In addition, the code window cannot be arbitrary changed to accommodate delay constraints if they exist. For example, assume that a lost information packet $\mathbf{p}_i$ is no longer necessary due to its delivery time exceeding some specified value. One approach would be to move the left side of the code window to the right so that $\mathbf{p}_i$ is no longer used in the generation of future coded packets (i.e., $\mathbf{c}_j = \sum_{k=i+1}^{j} \alpha_{j,k}\mathbf{p}_k$). In order for these new coded packets to be useful, the decoder must discard any coded packet containing $\mathbf{p}_i$ that it has already received. Not only does this decrease the efficiency of the coding scheme, but it also potentially increases the delay for subsequent packets $\mathbf{p}_j, i < j$. As a result, it is assumed that Algorithm 3 is left unchanged in this scenario.

Figure 5-8 shows the in-order delivery delay $\mathbb{E}[\mathrm{T}]$ for both the generation based code and streaming code on a link with i.i.d. packet erasures (Figure 5-8a) and correlated losses (Figure 5-8b). Each curve represents $\mathbb{E}[\mathrm{T}]$ for a given application layer packet erasure rate (PER). Furthermore, the curves shown for the generation based code are created by adjusting the generation size k so that a given PER is obtained. The larger the generation size in the generation based scheme, the better the error performance; but the cost is increased latency. In the figure, a small generation size k results in large PER but small $\mathbb{E}[\mathrm{T}]$, while small values of PER require large generation sizes leading to higher $\mathbb{E}[\mathrm{T}]$. The curves shown for the streaming code

show a PER that is approximately zero.

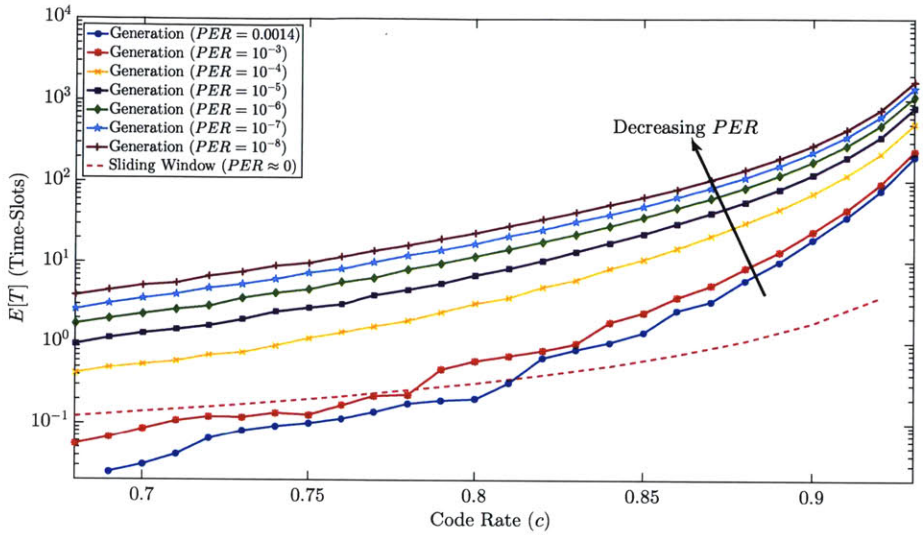
The figure shows that the streaming code typically achieves a smaller in-order delivery delay than the generation based code. In fact, the streaming code achieves an in-order delivery delay that is several orders of magnitude smaller than a generation based code that has similar PER (i.e., the generation based code's $PER < 10^{-8}$ while the streaming code's $PER \approx 0$). The only situation that the delay of the streaming code is not smaller than the generation based code is when the latter's PER is very large (i.e., $PER > 10^{-3}$ for code rates $c \leq 0.8$). To obtain this small delay, the generation size of the generation based code is only 1 to 3 packets making it similar to a repetition code. As the generation size is increased, quantization due to these small block sizes results in the large fluctuations shown in the delay-rate curves for small PER.

Finally, correlated losses can have a significant impact on the performance of the generation based code. This is a result of partitioning information packets into generations, which places artificial constraints the ability of the code to recover from packet losses. The streaming code, on the other hand, is much less sensitive to correlated losses due to the ability of a coded packet to recover from erasures over a larger span of information packets.

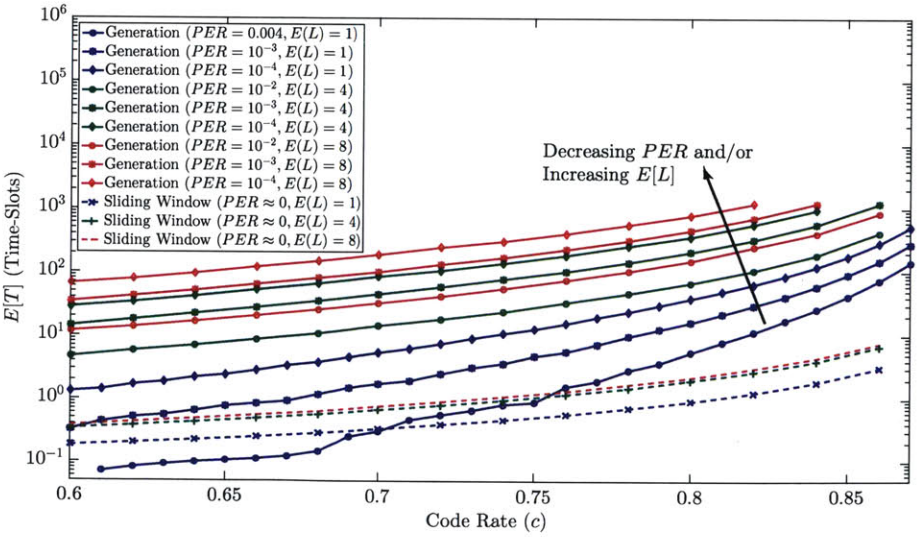
5.6.3 An Unfair Comparison: Closed-Loop Generation Based Codes versus Open-Loop Streaming Codes

The previous subsections showed that the streaming code can achieve a smaller $\mathbb{E}[T]$ than the generation based code regardless of the use of feedback. In the closed-loop comparisons, both the generation based and streaming codes were able to achieve 100% reliability. However, only the streaming code was able to come close to achieving this goal in the open-loop case with a $PER \approx 0$. This subsection compares the performance of the closed-loop generation based code with the open-loop streaming code to help illustrate the trade-off between using feedback and delay.

This comparison is shown in Figure 5-9. Both Figure 5-9a and Figure 5-9b show

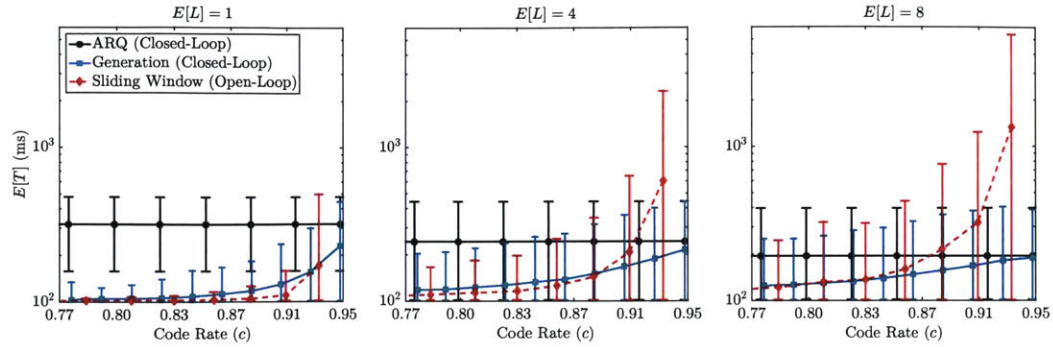


(a) I.I.D. Packet Losses with $\pi_B = 0.05$ and $\mathbb{E}[L] = 1$.

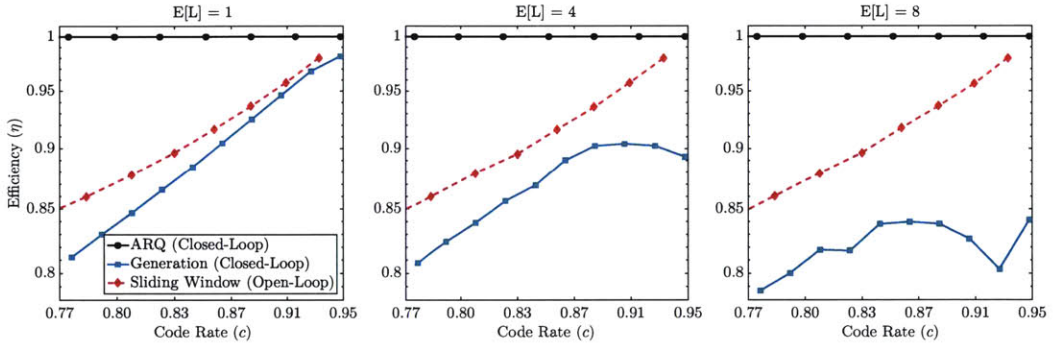


(b) Correlated Packet Losses with $\pi_B = 0.1$.

Figure 5-8: Open-loop in-order delivery delay as a function of the code rate. The packet erasure rate (PER) at the application layer after coding has attempted to correct all erasures is listed for each curve.



(a) In-order delivery delay.



(b) Efficiency η (Number of Information Packets/Total Number of Packets Received).

Figure 5-9: A comparison of an open-loop generation based code and a closed-loop streaming code on a 10 Mbps link with $RTT = 200$ ms and mean packet erasure rate $\pi_B = 0.05$.

$\mathbb{E}[T]$ and the efficiency η respectively as a function of the code rate c. In the case of the generation based code, the code rate refers to the FEC code rate and not the observed code rate that was used in Section 5.6.1. The error bars represent two standard deviations above and below the mean, and an idealized version of ARQ is shown for reference.

When packet erasures are i.i.d., the open-loop version of the streaming code has lower in-order delivery delay than the closed-loop generation based code while also achieving a higher efficiency. This illustrates that code construction has a major impact on efficiency. Since coding occurs over more information packets in the streaming code scheme, coded packets can help recover from packet erasures that occur over a larger span of time (i.e., multiple generations if we compare it with the generation based scheme). However, the figures showing the performance when losses are corre-

lated (i.e., $\mathbb{E}[L] = \{4, 8\}$) indicate that feedback is critical for ensuring that the delay does not grow unbounded. The decrease in the generation based scheme's efficiency as $\mathbb{E}[L]$ increases, as well as its non-increasing behavior for a given $\mathbb{E}[L] > 1$, is an indication that retransmissions are necessary to provide reliability. In fact, the generation based scheme almost always requires retransmissions to be made when $\mathbb{E}[L] = 8$. Regardless, the streaming code is still able to achieve a the same or lower delay and higher efficiency for a small enough code rate.

5.7 Conclusions

A multi-path streaming code is presented within this chapter that removes the artificial constraints of partitioning packets into generations. Rather, coded packets are generated using a code window that is managed based on the state-space of the client. An analysis of the in-order delivery delay experienced by information packets transmitted under this coding scheme is developed using a renewal-reward process; and a simple expression is found that closely approximates the true delay. Finally, a comparison of this streaming code with the generation based code provided in Chapter 4 is presented. This comparison shows that the streaming code construction achieves a lower in-order delivery delay than the generation based code while also obtaining a higher efficiency.

Chapter 6

Conclusions

6.1 Summary

The shift in the way end users' access the Internet and the type of traffic they generate require new approaches and techniques to meet increasingly strict quality of service requirements in networks that are becoming more unreliable. Furthermore, the adoption of new technologies, especially in the mobile environment, is providing new opportunities that can be leveraged to improve quality of service further. This thesis focused on applications of network coding for reliable transport as a method to overcome the challenges mentioned above and utilize all available network resources to fullest extent possible. This included a study of various methods to implement network coding in a multi-path transport layer and different approaches for generating network codes. It was shown throughout the thesis that transport layer coding can increase throughput, as well as decrease in-order delivery delay. This combination of gains results in significant improvements to application layer performance.

The primary motivation for the thesis was provided in Chapter 2 where an overview of a single-path network coded transport protocol called Coded TCP (CTCP) was presented. CTCP used a combination of network coding and congestion control modifications to increase transport layer performance. Experimental results showed that CTCP achieves considerable throughput gains over standard TCP in networks with high packet erasure rates. In addition, application layer measurements also hinted at

gains in delay. Indications of these gains were highlighted through the measurements of HTTP request completion times and the number of buffer under-runs experienced during the playback of a streamed video. While the throughput gains mentioned earlier were a major contributor to these application layer gains, they did not fully explain them.

Before exploring the reason for the gains shown in application layer performance, Chapter 3 first addressed extensions of network coded single-path transport layers to the multi-path environment. Empirical measurements helped map the multi-path, mobile environment by simultaneously collecting path statistics over three parallel heterogeneous networks. These statistics helped show the network availability one might expect and the conditions that may be present during communication. A multi-path transport protocol called Multi-Path TCP with Network Coding (MPTCP/NC) was then presented. This protocol used two layers of coding to both increase connection resiliency and provide erasure protection. A mean-field analysis of the protocol's throughput was developed; and a fusion of this analysis with the empirical measurements described above helped show the possible throughput gains over non-coded multi-path transport protocols such as Multi-Path TCP (MPTCP).

Chapter 4 helped to address the reason behind the non-throughput related application layer gains shown in Chapter 2. An analysis of the systematic generation based network code used in CTCP, although extended to the multi-path environment, was developed. This analysis helped show that there is an inherent trade-off between the delay caused by head-of-line blocking and coding. Unfortunately, a small generation size increases the probability of decode error. This results in an increase in the probability of head-of-line blocking. As the code's generation size increases, the probability of a decode error is reduced resulting in fewer retransmissions and less head-of-line blocking. This should reduce the overall in-order delivery delay; however, the coding delay increases as the generation size grows resulting in increased in-order delivery delay. Therefore, the analysis helped determine the generation size that balanced the delay caused by head-of-line blocking and coding. The analysis also showed that this minimum can be made as close to the network propagation delay as needed.

The only drawback is that reducing the delay requires a reduction in efficiency, or goodput. This rate-delay trade-off was shown using numerical results.

Finally, Chapter 5 looked at an alternate code construction that removed the artificial constraints placed on the generation based code as a result of partitioning packets into fixed length generations. This multi-path streaming code uses a sliding window approach to coding where the composition of coded packets is based on the state-space of the client's decoder. A renewal-reward process was used to determine the expected in-order delivery delay; and both numerical and simulated results showed that this streaming code can achieve lower in-order delivery delay with higher efficiency than the generation based code discussed in Chapter 4. Finally a comparison of the closed-loop version of the generation based code and an open-loop version of the streaming code was made. This helped show the importance of using feedback, especially in networks where packet losses are correlated.

6.2 Implementation Considerations

This thesis has presented multiple methods to implement network coding within the transport layer. Chapters 2 and 3 provided two different transport layer designs, while Chapters 4 and 5 explored two different coding approaches. Each of these approaches have their benefits and drawbacks. While not everything can be addressed, a discussion of some of the most important implementation considerations are provided here.

The first major consideration is where to perform the coding and decoding operations. Ideally, redundancy should be added at any point in the network where packet losses occur. This includes locations such as queues or links where the physical layer cannot provide 100% reliability. Furthermore, the amount of added redundancy should only be enough to help recover from losses that occur between network nodes that can code. This can be motivated by the simple example shown in Figure 6-1 where a source S wants to transmit N packets to the destination D. However, these packets must travel over a tandem network where each link $i \in \{1, 2, 3\}$ has an i.i.d.

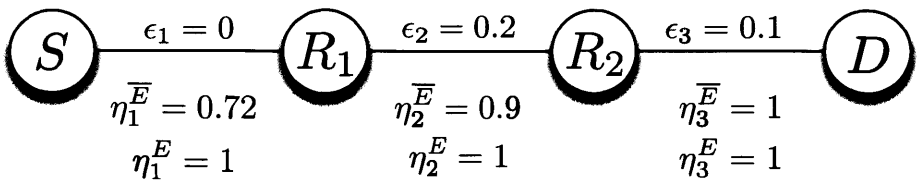


Figure 6-1: A simple example showing that coding within the network is more efficient than end-to-end coding. η_i^j is the efficiency on link $i \in 1, 2, 3$ when coding is performed end-to-end ($j = \bar{E}$) or at each intermediate network node ($j = E$).

packet erasure probability ϵ_i. If end-to-end coding is used, $N\left(\prod_i (1-\epsilon_i)^{-1} - 1\right)$ coded packets must be generated at S and transmitted through the network. This results in an inefficient use of links closer to the source than would be necessary if redundancy is included into the packet stream at each node $R_i, i \in 1, 2$.

This simple fact can have major implications in networks with limited or expensive bandwidth. While coding should be performed as often as possible, network codes do not need to be decoded at each hop. In other words, coded packets can be generated at multiple points within the network while only needing to decode once at the client. In the example provided in Figure 6-1, coding can take place at S, R_1, and R_2; however, only D needs to decode.

The second consideration that needs to be taken into account is how to communicate the coding coefficients α_i used to the decoder. For generation-based coding schemes where k is typically small, one can simply insert each coding coefficient into the packet header of a coded packet, which would require approximately qk bits assuming each $\alpha_i \in \mathbb{F}_{2^q}$. Coding within the network only needs to modify the existing coefficients and does not increase the size of the coding coefficient vector. Of course, other approaches that require less than qk bits such as [69] or [70] can be used to decrease overhead.

Communicating the coefficients efficiently for streaming code schemes like the one presented in Chapter 5 is more challenging since the coding windows can be quite large. Existing methods typically use a pseudo-random number generator and communicate only the seed. This seed is then used by the decoder to generate the

coefficients used to create each coded packet. Unfortunately, this does not scale well when coding occurs at intermediate network nodes. As an example, assume that an intermediate node's coding window contains multiple coded packets that were generated by previous nodes. When the node generates a new coded packet, it must communicate the seed used to generate the packet; in addition to all of the seeds for each of the coded packets contained within its coding window. If the coding window and the number of coded packets contained within the window are large, the amount of overhead required to reproduce the coefficients can far exceed the payload size.

Finally, congestion control and file size can potentially dictate the coding approach used. Regardless of the type of data stream, some form of congestion control is typically needed. Common congestion control algorithms can cause bursts of packets, or packet trains, while they are ramping up to fully utilize the network. This behavior is even more pronounced when considering TCP flows over large latency networks. In these situations, it maybe preferable to use a coding scheme that provides a high probability of delivering every packet within a burst without needing retransmissions or waiting for the next packet burst to arrive. For example, a generation-based coding scheme can be used for small congestion window sizes and a streaming code can be used for large ones. The benefit of using RLNC is that switching between the two is relatively easy.

In a similar fashion, the coding strategy can also significantly impact the overall throughput for some file sizes. For example, consider a small file that can be transmitted using less than a single bandwidth-delay product worth of packets. A generation-based coding scheme, or a mixture of the generation-based and streaming code schemes, should be used so that the the probability of decoding the file after the first transmission attempt is made very large. While this may impact the efficiency of the network, it can have major benefits for the user's quality of service.

6.3 Possible Directions for Future Research

A number of different avenues for future research and extensions to this thesis are possible. These range from continued development of multi-path transport layer protocols to exploiting the synergetic affects of network coding. While not everything can be discussed here, a number of possible directions are outlined.

First, continued development of a coded multi-path transport layer is needed. A number of protocol designs and coding approaches were discussed throughout this thesis; however, the implementation of these approaches is necessary to verify the gains shown through analysis. Furthermore, extensive system testing is needed to confirm the ultimate goals and objectives of the transport layer are satisfied.

Second, methods to enable efficient coding within the network using streaming, or sliding window codes, is required. Chapter 5 showed that the multi-path streaming code that uses a sliding window to manage the generation of redundancy outperforms generation based approaches. Unfortunately, the number of information packets used to generate each coded packet can be fairly large making the use of coding vectors undesirable. As mentioned in the last section, a random number generator or a deterministic scheme can be used to define the coding coefficients and reduce overhead; but these approaches do not provide the necessary flexibility to code within the network itself unless access to lower network layers is available.

Finally, the approaches presented within this thesis used network coding as the primary workhorse to improve transport layer performance. While significant performance gains were observed, network coding was restricted to just the transport layer. Combining the techniques used throughout the thesis with those developed for different network layers can potentially have synergistic effects. For example, network coding has been shown to improve storage system performance [71], data link layer performance [72], physical layer performance [68], etc. However, each of these proposals, including the those discussed within this thesis, has had limited scope. Research into the benefits of using network coding at each of the layers concurrently and how these network coded layers supplement each other is another promising avenue of

research.

Bibliography

[1] "The Zettabyte Era: Trends and Analysis," White Paper, Cisco, May 2015.

[2] "Maximizing Audience Engagement: How Online Video Performance Impacts Viewer Behavior," White Paper, Akamai, January 2015.

[3] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard, "Congestion Control for Coded Transport Layers," in *IEEE International Conference on Communications (ICC)*, June 2014, pp. 1228–1234.

[4] G. Maral and M. Bousquet, *Satellite Communications Systems: Systems, Techniques and Technology.* John Wiley & Sons, August 2011.

[5] O. Bonaventure, M. Handley, and C. Raiciu, "An overview of Multipath TCP," *USENIX ;login:*, vol. 37, no. 5, October 2012.

[6] J. Cox, "Apple iOS 7 Suprises as First with New Multipath TCP Connections," September 2013. [Online]. Available: http://www.networkworld.com/news/2013/091913-ios7-multipath-273995.html

[7] J. Cloud, F. du Pin Calmon, W. Zeng, G. Pau, L. M. Zeger, and M. Médard, "Multi-Path TCP with Network Coding for Mobile Devices in Heterogeneous Networks," in *IEEE 78th Vehicular Technology Conference (VTC Fall)*, September 2013, pp. 1–5.

[8] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, October 2006.

[9] C. Caini, R. Firrincieli, M. Marchese, T. d. Cola, M. Luglio, C. Roseti, N. Celandroni, and F. Potorti, "Transport Layer Protocols and Architectures for Satellite Networks," *International Journal of Satellite Communications and Networking*, vol. 25, no. 1, pp. 1–26, 2007.

[10] A. Pirovano and F. Garcia, "A New Survey on Improving TCP Performances Over Geostationary Satellite Link," *Network and Communication Technologies*, vol. 2, no. 1, June 2013.

[11] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, December 1997.

[12] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, July 2008.

[13] C. Caini and R. Firrincieli, "TCP Hybla: A TCP Enhancement for Heterogeneous Networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, 2004.

[14] S. Trivedi, S. Jaiswal, R. Kumar, and S. Rao, "Comparative Performance Evaluation of TCP Hybla and TCP Cubic for Satellite Communication Under Low Error Conditions," in *IEEE 4th International Conference on Internet Multimedia Services Architecture and Application (IMSAA)*, December 2010, pp. 1–5.

[15] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle, "Loss-Tolerant TCP (LT-TCP): Implementation and Experimental Evaluation," in *Military Communications Conference (MILCOM)*, October 2012, pp. 1–6.

[16] V. Sharma, S. Kalyanaraman, K. Kar, K. K. Ramakrishnan, and V. Subramanian, "MPLOT: A Transport Protocol Exploiting Multipath Diversity Using Erasure Codes," in *27th IEEE Conference on Computer Communications (INFOCOM)*, April 2008, pp. 121–125.

[17] O. Tickoo, V. Subraman, S. Kalyanaraman, and K. K. Ramakrishnan, *Quality of Service - IWQoS 2005: 13th International Workshop, IWQoS 2005, Passau, Germany, June 21-23, 2005. Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ch. LT-TCP: End-to-End Framework to Improve TCP Performance Over Networks with Lossy Channels, pp. 81–93.

[18] S. Bauer, R. Beverly, and A. Berger, "Measuring the State of ECN Readiness in Servers, Clients,and Routers," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, 2011, pp. 171–180.

[19] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP: Theory and Implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, March 2011.

[20] G. Xylomenos, G. C. Polyzos, P. Mahonen, and M. Saaranen, "TCP Performance Issues Over Wireless Links," *IEEE Communications Magazine*, vol. 39, no. 4, pp. 52–58, 2001.

[21] S. Gheorghiu, A. L. Toledo, and P. Rodriguez, "Multipath TCP with Network Coding for Wireless Mesh Networks," in *IEEE International Conference on Communications (ICC)*, May 2010, pp. 1–5.

[22] X. Zhuoqun, C. Zhigang, Y. Hui, and Z. Ming, "An Improved MPTCP in Coded Wireless Mesh Networks," in *IEEE International Conference on Broadband Network & Multimedia Technology (IC-BNMT)*, October 2009, pp. 795–799.

[23] Z. Xia, Z. Chen, Z. Ming, and J. Liu, "A Multipath TCP Based on Network Coding in Wireless Mesh Networks," in *IEEE International Conference on Information Science and Engineering (ICISE)*, December 2009, pp. 3946–3950.

[24] A. ParandehGheibi, M. Médard, A. Ozdaglar, and S. Shakkottai, "Access-Network Association Policies for Media Streaming in Heterogeneous Environments," in *49th IEEE Conference on Decision and Control (CDC)*, December 2010, pp. 960–965.

[25] A. Kulkarni, M. Heindlmaier, D. Traskov, M.-J. Montpetit, and M. Médard, *NETWORKING 2011 Workshops: International IFIP TC 6 Workshops, PE-CRN, NC-Pro, WCNS, and SUNSET 2011, Revised Selected Papers.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. An Implementation of Network Coding with Association Policies in Heterogeneous Networks, pp. 110–118.

[26] E. Brosh, S. A. Baset, D. Rubenstein, and H. Schulzrinne, "The delay-friendliness of TCP," in *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2008, pp. 49–60.

[27] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," in *19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, March, pp. 1742–1751.

[28] A. Heidarzadeh, "Design and Analysis of Random Linear Network Coding Schemes: Dense Codes, Chunked Codes and Overlapped Chunked Codes," Ph.D. Thesis, Carleton University, Ottawa, Canada, December 2012.

[29] D. Lucani, M. Médard, and M. Stojanovic, "Broadcasting in Time-Division Duplexing: A Random Linear Network Coding Approach," in *Workshop on Network Coding, Theory, and Applications (NetCod)*, June 2009, pp. 62–67.

[30] ——, "Online Network Coding for Time-Division Duplexing," in *IEEE Global Telecommunications Conference (GLOBECOM)*, December 2010, pp. 1–6.

[31] D. Lucani, M. Stojanovic, and M. Médard, "Random Linear Network Coding For Time Division Duplexing: When To Stop Talking And Start Listening," in *IEEE International Conference on Computer Communications (INFOCOM)*, April 2009, pp. 1800–1808.

[32] T. Dikaliotis, A. Dimakis, T. Ho, and M. Effros, "On the Delay of Network Coding Over Line Networks," in *IEEE International Symposium on Information Theory (ISIT)*, June 2009, pp. 1408–1412.

[33] M. Nistor, R. Costa, T. Vinhoza, and J. Barros, "Non-Asymptotic Analysis of Network Coding Delay," in *IEEE International Symposium on Network Coding (NetCod)*, June 2010, pp. 1–6.

[34] E. Drinea, C. Fragouli, and L. Keller, "Delay with Network Coding and Feedback," in *IEEE International Symposium on Information Theory (ISIT)*, June 2009, pp. 844–848.

[35] A. Eryilmaz, A. Ozdaglar, and M. Médard, "On Delay Performance Gains From Network Coding," in *40th Annual Conference on Information Sciences and Systems*, March 2006, pp. 864–870.

[36] B. Swapna, A. Eryilmaz, and N. Shroff, "Throughput-Delay Analysis of Random Linear Network Coding for Wireless Broadcasting," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6328–6341, October 2013.

[37] Y. Xia and D. Tse, "Analysis on Packet Resequencing for Reliable Network Protocols," in *22nd Annual Joint Conference of the IEEE Computer and Communications (INFOCOM)*, vol. 2, March 2003, pp. 990–1000.

[38] H. Yao, Y. Kochman, and G. W. Wornell, "A Multi-Burst Transmission Strategy for Streaming Over Blockage Channels with Long Feedback Delay," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 10, pp. 2033–2043, December 2011.

[39] M. Nistor, J. Barros, F. Vieira, T. Vinhoza, and J. Widmer, "Network Coding Delay: A Brute-Force Analysis," in *Information Theory and Applications Workshop (ITA)*, January 2010, pp. 1–5.

[40] J. Sundararajan, P. Sadeghi, and M. Médard, "A Feedback-Based Adaptive Broadcast Coding Scheme for Reducing In-Order Delivery Delay," in *Workshop on Network Coding, Theory, and Applications (NetCod)*, June 2009, pp. 1–6.

[41] A. Fu, P. Sadeghi, and M. Médard, "Delivery delay analysis of network coded wireless broadcast schemes," in *IEEE ireless Communications and Networking Conference (WCNC)*, April 2012, pp. 2236–2241.

[42] W. Zeng, C. Ng, and M. Médard, "Joint Coding and Scheduling Optimization in Wireless Systems with Varying Delay Sensitivities," in *9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, June 2012, pp. 416–424.

[43] G. Joshi, Y. Kochman, and G. W. Wornell, "On Playback Delay in Streaming Communication," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, July 2012, pp. 2856–2860.

[44] G. Joshi, "On Playback Delay in Streaming Communication," Master of Science in Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, May 2012.

[45] G. Joshi, Y. Kochman, and G. Wornell, "The Effect of Block-Wise Feedback on the Throughput-Delay Trade-Off in Streaming," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2014, pp. 227–232.

[46] A. ParandehGheibi, M. Médard, A. Ozdaglar, and S. Shakkottai, "Avoiding Interruptions; A QoE Reliability Function for Streaming Media Applications," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 5, pp. 1064–1074, May 2011.

[47] A. ParandehGheibi, "Metrics, fundamental trade-offs and control policies for delay-sensitive applications in volatile environments," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2012.

[48] M. Tömösközi, F. H. Fitzek, F. H. Fitzek, D. E. Lucani, M. V. Pedersen, and P. Seeling, "On the Delay Characteristics for Point-to-Point Links using Random

Linear Network Coding with On-the-Fly Coding Capabilities," in *20th European Wireless Conference; Proceedings of European Wireless*, May 2014, pp. 1–6.

[49] R. Prior and A. Rodrigues, "Systematic Network Coding for Packet Loss Concealment in Broadcast Distribution," in *International Conference on Information Networking (ICOIN)*, January 2011, pp. 245–250.

[50] D. Lucani, M. Médard, and M. Stojanovic, "Systematic Network Coding for Time-Division Duplexing," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, June 2010, pp. 2403–2407.

[51] P. Sadeghi and M. Yu, "Instantly Decodable versus Random Linear Network Coding: A Comparative Framework for Throughput and Decoding Delay Performance," *CoRR*, vol. abs/1208.2387, 2012.

[52] J. Cloud, D. Leith, and M. Médard, "Network Coded TCP (CTCP) Performance Over Satellite Networks," in *International Conference on Advances in Satellite and Space Communications (SPACOMM)*, February 2014, pp. 53–56.

[53] ——, "A Coded Generalization of Selective Repeat ARQ," in *IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 2155–2163.

[54] "Wireless Technologies for Network Service Providers 2012-2013," White Paper, Technicolor, 2013.

[55] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, April 2000.

[56] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, October 1995.

[57] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *25th IEEE International Conference on Computer Communications (INFOCOM)*, April 2006, pp. 1–12.

[58] R. N. Shorten and D. J. Leith, "On Queue Provisioning, Network Efficiency and the Transmission Control Protocol," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 866–877, August 2007.

[59] Iridium Everywhere. (2016) Iridium 9522A satellite transceiver. Online. [Online]. Available: https://www.iridium.com/products/details/iridium9522satellitetransceiver

[60] "MPTCP IETF Working Group," https://datatracker.ietf.org/wg/mptcp/.

[61] M. Kim, M. Médard, and J. a. Barros, "Modeling Network Coded TCP Throughput: A Simple Model and its Validation," in *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, 2011, pp. 131–140.

[62] R. Koetter and F. Kschischang, "Coding for Errors and Erasures in Random Network Coding," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3579–3591, August 2008.

[63] M. Karzand and D. J. Leith, "Low Delay Random Linear Coding Over a Stream," in *52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, September 2014, pp. 521–528.

[64] M. Karzand, D. Leith, J. Cloud, and M. Médard, "Low Delay Random Linear Coding Over a Stream," *CoRR*, vol. abs/1509.00167, 2015.

[65] J. C. Tanner, "A derivation of the borel distribution," *Biometrika*, vol. 48, no. 1/2, pp. pp. 222–224, 1961.

[66] R. G. Gallager, *Stochastic Processes: Theory for Applications.* New York, NY: Cambridge University Press, 2013.

[67] S. Teerapittayanon, K. Fouli, M. Médard, M.-J. Montpetit, X. Shi, I. Seskar, and A. Gosain, *Multiple Access Communications: 5th International Workshop, MACOM 2012, Maynooth, Ireland, November 19-20, 2012. Proceedings.* Berlin,

Heidelberg: Springer Berlin Heidelberg, 2012, ch. Network Coding as a WiMAX Link Reliability Mechanism: An Experimental Demonstration, pp. 75–78.

[68] D. Adams, J. Du, M. Médard, and C. Yu, "Delay Constrained Throughput-Reliability Tradeoff in Network-Coded Wireless Systems," in *IEEE Global Communications Conference (GLOBECOM)*, December 2014, pp. 1590–1595.

[69] D. E. Lucani, M. V. Pedersen, J. Heide, and F. H. P. Fitzek, "Fulcrum Network Codes: A Code for Fluid Allocation of Complexity," *CoRR*, vol. abs/1404.6620, 2014.

[70] N. Thomos and P. Frossard, "Toward One Symbol Network Coding Vectors," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1860–1863, November 2012.

[71] U. J. Ferner, "Toward Sustainable Networking: Coded Storage and High-Traffic Networks," Doctor of Philosophy in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June 2014.

[72] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 497–510, June 2008.