MIT-CSAIL-TR-2016-009 June 28, 2016

Evaluating Caching Mechanisms In Future Internet Architectures

Yuxin Jing

Evaluating Caching Mechanisms in Future Internet Architectures

by Yuxin Jing

S.B., Massachusetts Institute of Technology (2015),
Computer Science and Engineering

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Computer Science and Engineering
at the
Massachusetts Institute of Technology
June 2016

Author: ____
Department of Electrical Engineering and Computer Science
May 20, 2016

Certified by: ____
Karen Sollins
Principal Investigator
MIT Computer Science and Artificial Intelligence Laboratory
May 20, 2016

Accepted by: ____
Dr. Christopher Terman, Chairman, Masters of Engineering Thesis Committee

Acknowledgements:

I’d like to thank everyone involved in helping me through this thesis: my supervisor Karen, my labmates, and my friends for supporting me through the last stretch of getting everything in. This work was funded in part under NSF Grant 1413973, “NeTS: Large: Collaborative Research: Location-Independent Networks: Evaluation Strategies and Studies”

Table of Contents

List of Figures

List of Tables

Abstract

This thesis seeks to test and evaluate the effects of in-network storage in novel proposed Internet architectures in terms of their performance. In a world where more and more people are mobile and connected to the Internet, we look at how the added variable of user mobility can affect how these architectures perform under different loads. Evaluating the effects of in-network storage and caching in these novel architectures will provide another facet to understanding how viable of an alternative they would be to the current TCP/IP paradigm of today's Internet. In Named Data Networking, where the storage is used to directly cache content, we see its use of storage impact the locality of where things are, while in MobilityFirst, where storage is used to cache chunks to provide robust delivery, we look at how its different layers work together in a mobility event.

Chapter 1

Introduction

More and more, people are connected to the Internet through the ever increasing number of smart devices. The shift in not only how we connect to the web but also what kinds of content catches our attention in the last couple decades has been drastic and in a direction that the original creators of the Internet could not have foreseen. Generally, we think of the Internet as having an hourglass shape, with the TCP/IP portion of the transport layer as the narrowest part due to the fact that there are really only those few protocols the network and transport layers. However, the basis for those protocols has remained mostly the same since their inception and most of any changes to those protocols have been reactionary. This has led to several projects looking to redesign the Internet from a clean-slate approach. Each of these projects focuses on some select scopes of issues with the current Internet, ranging from user mobility, the end-to-end structure of communication, and routing based the content, not the end host.

Traditionally, we think of caching as a client side action or as an overlay on top of the Internet infrastructure. Once you've downloaded a page, the next time you want to access that page, everything loads much faster from the local cache. However, the price for persistent memory has followed an exponential trend downwards, leading to more options to where caches could potentially be located. Some of these clean-slate approaches include the use of in-network caching of both metadata as well as the actual content. Instead of just a local cache, routers themselves have the infrastructure to cache content inside the network. Thus, caching in these systems has the opportunity to affect more than just the individual user but provide overall gains to the whole system.

1.1 Problem Statement

These clean-slate approaches are often in the very experimental phases. Looking at two of these approaches, namely NDN, named data networking, and MobilityFirst, we gain a clearer picture of the role of caching across different architectures and the issues that are inherent to these alternative architectures. Because persistent in-network storage on the router is built into the basic protocols of these two architectures -- content caching for NDN and delay-tolerant delivery in MobilityFirst, not only will we see the overall performance gains caching provides but also the trends in the system that caching introduces in the architectures.

1.2 Caching in Today's Internet

1.2.1 Three Levels to Caching

Internet browsers will cache information at the client end host. First, the browsers dictate the caching policy and the cache size set aside for each browser. Second, a common form of web caching is caching at HTTP proxy servers, which are intermediaries between a browser and web servers on the Internet. These proxies cache HTTP requests and the responses to those requests for all users accessing the Internet through that proxy. Third, generally, ISPs set up HTTP proxy servers that clients can query from. Users can either specify a proxy they wish to use, or an ISP may funnel all traffic through their transparent proxies. In reverse proxy caching, caches are deployed near the origin of the content. It is used to support web hosting farms, or virtual domains mapped to a single physical site.

1.2.2 Hierarchical vs. Distributed Caching

There are two main caching schemes commonly proposed, hierarchical and distributed[17]. For hierarchical caching, client caches exist at the bottom level of the hierarchy. When there is a cache miss, the request is redirected to the institutional cache – ie up a level. If the document is not found at any cache level, the national cache contacts directly the origin server. When the document is found, it travels down the hierarchy, leaving a copy at each of the intermediate cache. Hierarchical caching has lower connection times than distributed caching and, thus, placing redundant copies in intermediate cache levels reduces the connection time

In a totally distributed caching scheme, caches, which exist only at the bottom level of the network, cooperate. No higher level caches are set up, and institutional caches serve each other's misses. In order to decide from which institutional caches to retrieve a miss document, metadata information or location hints are used. By sharing metadata information, every institutional cache knows locally about the content of every other cooperating institutional cache. Distributed caching has lower transmission times than hierarchical caching since most of the traffic flows through the less congested low network levels[17].

1.3 Thesis Contribution

In this thesis, we have evaluated how the use of in-network storage and caching has affected the semantics of the systems we look at, NDN and MobilityFirst. In NDN, we see how caching changes the locality of requests and how caching in a network topology affects the data flow of requests. We then evaluate how these elements affect caching strategies and optimal cache resource placement. In MobilityFirst, we look at the relationship of the different layers of in-network storage and client buffering and how factors like chunk size and distance from the server change the performance of the system as a whole. Looking at these components, we get an understanding of how mobility events are handled. We see that because in-network storage is available for these architectures, they are able to approach and solve some of the issues that continue to plague today's Internet.

1.4 Thesis Organization

In the remainder of this thesis, we will provide the background to the two architectures in Chapter 2, introduce the experimental setup in Chapter 3, present results in Chapter 4, discuss opportunities for future work in Chapter 5 and conclude in Chapter 6.

Chapter 2

Background

This section will provide an overview on the mechanics of the architectures we will be looking at as well as the caching technology relating to each paradigm. We will look at their design goals, protocols, and prototype. Later, we will compare these architectures to current internet CDNs and overlay technologies.

2.1 Architectures

2.1.1 Named Data Networking

NDN decouples location from identity, security, and access, and instead uses names to retrieve content. This design choice is driven by the fact that in today’s world, people value what content it has, but communication protocols still focus on the where. Instead of having to trust the host of the connection, people can trust the content itself being secured. NDN achieves this by replacing the end-to-end TCP/IP paradigm by Interest and Data packets that follow a flat naming scheme.

Named Data Networking[8] is a network architecture building on the concept of named data. There are two types of packets in NDN, interest and data. A user will broadcast his interest in a named piece of data, which will propagate through network until it reaches a node that is authoritative for the data or has it data cached locally. At that point, that node will forward the data to the original user, reversing the path that the interest packet took and potentially leaving a trail of newly cached copies. NDN routers are composed of three main blocks: a content store (CS), a pending interest table (PIT) and a forwarding information base (FIB). For every incoming interest, the router first checks its CS and, if the requested chunk is present, it returns the chunk directly. If not, it will check to see if a pending PIT entry has been already created for that interest. If there is, it adds the new request to the PIT so when a data packet comes back, it will forward to both the previous interests as well as the new interest. Otherwise, it creates a new PIT entry and forwards the Interest to a next hop determined via the FIB and forwarding strategy. By these means, popular data packets will become widely cached and available for more efficient responses to successive interest packets for it.

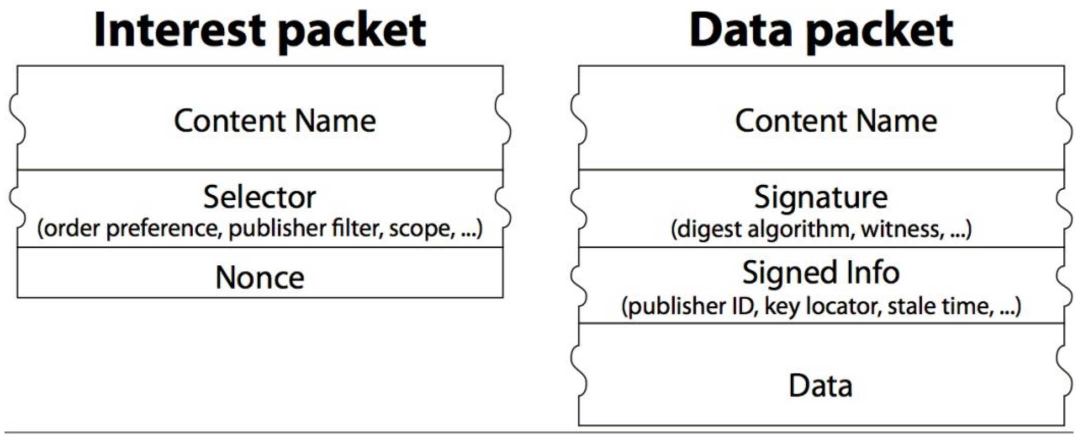

Interest packet
Content Name
Selector
(order preference, publisher filter, scope, ...)
Nonce
Data packet
Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

Figure 2.1 NDN Packet Headers and Payload

The basic block in NDN is at the packet level. A large file is broken up into individually named packets and a request for the entire file requires the user to send a request for each individual packet. Currently, each packet is treated independently in most caching schemes.

There are many examples in which this mode of operation might not be optimal. For example, when we stream a video from Netflix, caching the most recent segments of the stream would be of little use for other users around us unless they were at around the same place in the video. A more effective caching scheme might instead cache the first few packets of the video so that any user starting to watch can have a faster starting time. Thomas and al.[23], for example, have proposed a Object-oriented caching scheme.

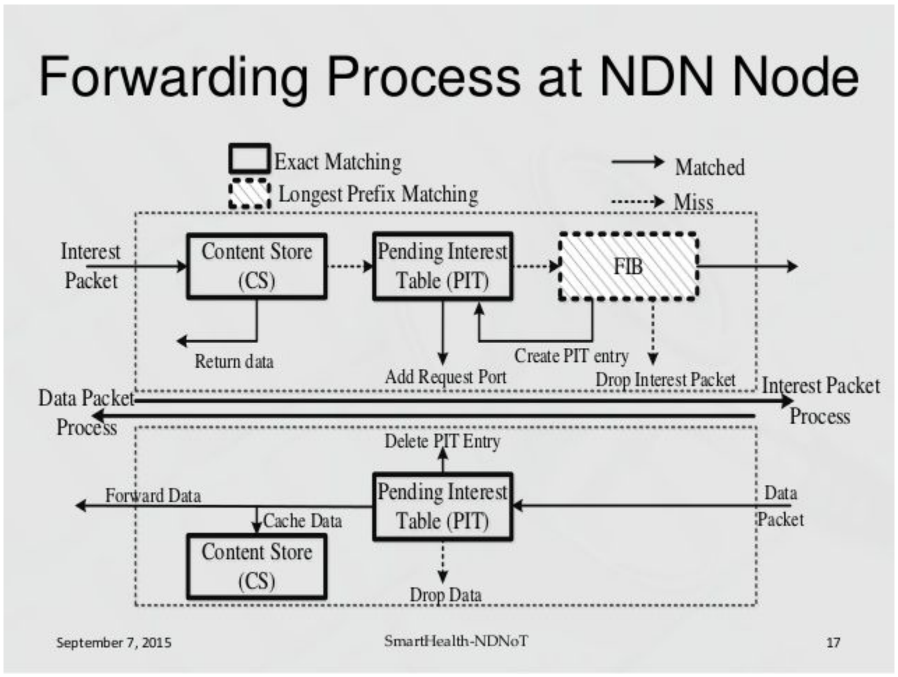


Figure 2-2 NDN Forwarding Pipeline [39]

While the NDN name space is unbounded, NDN naming conventions[20] encourage the use of human-readable clear-text strings as name components, which resembles the file system naming scheme. Octet values 0xC0, 0xC1, and 0xF5 to 0xFF, normally prohibited in UTF-8, are used in NDN naming to indicate special components that are used for roles such as versioning, segmentation, time stamping, and sequencing. These conventions are highly encouraged, but not strictly enforced. Not all octet values are in use, so there is an opportunity for expansion in NDN naming conventions. Currently, name conventions are specific to applications but *opaque t*o the network, i.e., routers do not know the meaning of a name. The network understands the boundary between name components but it is up to a producer and a consumer to agree on name conventions indicating versioning and segmentation. In addition, not all the names need to be globally unique.

2.1.2 MobilityFirst

MobilityFirst seeks to provide an architecture to support mobility as the norm with dynamic host and network mobility at scale[14][1]. With the rise of mobile devices, MobilityFirst looks to replace the fixed-host/server model. MobilityFirst also hopes to provide robustness in a mobile environment by generalized delay-tolerant routing and hop-by-hop transport protocols that use in-network storage.

MobilityFirst introduces the concept of a global unique identifier (GUID)[21], which is long lasting and can be abstracted to cover multiple contexts beyond just a physical device or a piece of data. Instead of an endpoint IP address, GUIDs can freely move in the network and attach themselves to one or more network addresses (NAs), which can be either GUIDs or, at the lowest level, IP addresses. This creates a hierarchical routing structure where MobilirtyFirst first routes a chunk of data to the NA, which will then do internal routing to get the chunk to the destination GUID. Because a GUID can map to more than one NA, MobilityFirst offers extra built in delivery services, including multihoming, multicasting, and anycasting.

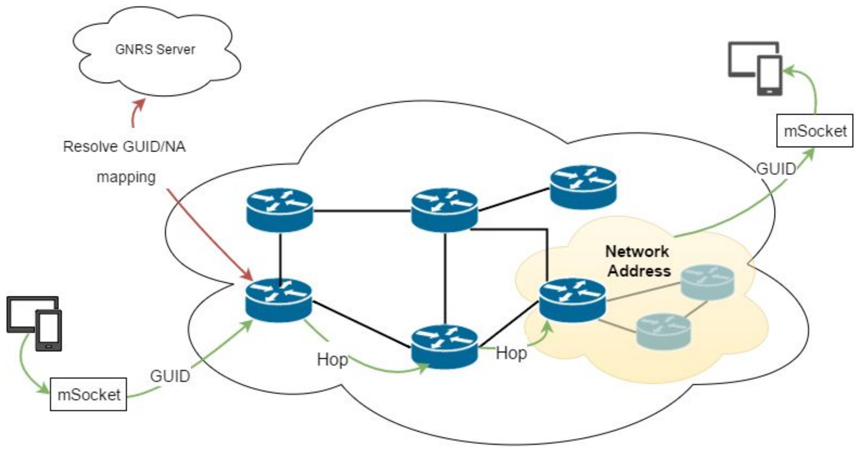


Figure 2-3 MobilityFirst Network

To quickly resolve the GUID to NA mapping, MobilityFirst introduces a Global Name Resolution Service (GNRS) which can dynamically resolve the GUID to NA(s) mapping. Currently, the MobilityFirst project has two GNRS prototypes, Auspice[24] and DMap[25]. Auspice is a two-layered Paxos distributed system while Dmap is logically centralized but physically distributed hash table. While both are independent services, with a scalable name-based API like a DNS server, Auspice is an overlay service, while DMap is located on the routers but accessed through a socket protocol.

The Generalized Storage Aware Routing (GSTAR)[22] protocol runs on the routing level. Each router periodically floods the network with link state advertisements of their immediate neighbors. There is both a short term and long Expected Transmission Time in these advertisements, which allows the routers to have an up-to-date view of the current link qualities as well as historic link qualities within their current partition in the network. Packets are then routed on this information to the destination based on the shortest path when a path is available. Otherwise, when there is either no current link quality information, or no current NA which the GUID is attached to, the router caches the data and periodically retries sending the data.

As opposed to the traditional TCP packet transport, MobilityFirst uses a block transport protocol, Hop, to transport large chunks of contiguous data at a time in a per-hop-reliable manner[12]. At each hop, a sender first sends a control message downstream to the next hop and then sends the contiguous data chunk, and when the receiver has received all the packets of the chunk, it sends back to the sender an acknowledgment of all the correctly received packets of that chuck. At each hop, the routers keep track of successful chunks sent downstream, and if it sees that the acknowledgment from downstream is under a certain threshold compared to how many chunks it has successfully received from upstream, it will stop sending acknowledgments upstream. This feedback, at no extra cost, eventually reaches the original sender who can then try to send again.

On top of these transport layer protocols, MobilityFirst also has a session layer application called msocket[26]. When a server starts, it creates a msocket without any connections and waits for clients to connect to it. Once a client connects to the server, a connection information is created for the server/client pairing. Each connection consists of one or more flows, determined by the number of interfaces that each side can send on. The connection keeps track of the overall sending and receiving of all data across its flows. Each flow also keeps track of the connection between the specific flow. The server follows a policy that determines which flows to send on, and keeps a buffer of the outgoing data. The client keeps a buffer of received data that allows for out of order transmissions. Once the client successfully receives a packet, it sends an acknowledgment back to the server. Both sides keeps timers for connectivity, and when there is a timeout, whichever side figured out about the timeout initiates a migration protocol.

2.2 Caching

2.2.1 NDN

Each router in the NDN network has the ability to cache individual packets. Every node comes with a content store that has an attached policy that determines how items in the content store get replaced. Common policies include: LFU(Least Frequently Used), LRU (Least Recently Used), FIFO (First in First Out), and RANDOM. In current NDN research, LRU is the most commonly adopted caching policy given the observation that that about half of the caching benefits at packet level happen in the first 10 seconds.

Replication strategies for cache networks determine how an object gets back to the requester and how to cache the object along its path. LCE, leave-copy-everywhere, is cached along all nodes in the path. LCP, leave-copy-probabilistically[16], is cached with probability q along the nodes. LCD, leave-copy-down, is only cached at the preceding node of where the data is found, ie the first hop from the source or a cached version. Other strategies propose an overlay architecture for caching[10].

Individual cache policies combine with a replication strategy for an overall network strategy. An example of this is Betw+LRU[3], where an object is cached at one location, the most central, along a path, using LRU replacement at each node.

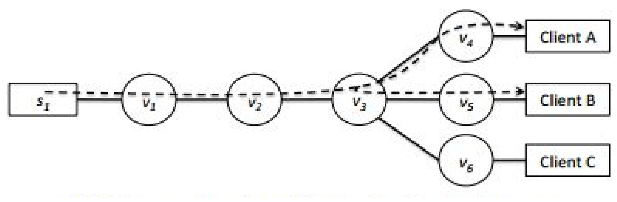


Figure 2-4 Optimal Caching

2.2.2 MobilityFirst

In MobilityFirst, caching is implemented in two layers. The client sends a large chunk, ~1MB of sequenced packets, all at once to the network. In the first layer, routers cache chunks of packets sent on each hop-by-hop segment. This setup also allows for virtual retransmission, where a Hop sender, upon a timeout, only needs to send a token of the chunk if it is already cached in the next downstream Hop router. It only physically retransmits chunks if the chunk is not cached downstream. If a router either cannot calculate the next hop, or the NA the GUID was bound to had changed during transmission, it has the ability to store the chunks in a buffer as it tries to figure out the next hop destination. Periodically, the router will try to resend chunks saved in its buffer if a forwarding path opens up.

The second layer of caching involves the msocket application in the user space. Msockets on each end of the sender-client relationship keep a buffer for incoming and outgoing data. This buffer is designed so that out of order transmission are cached in the buffer. For each successful completed chunk received, the receiver send an ack to the sender so it knows not to retransmit that chunk. Each msocket application allows for multiple flows, ie multiple paths between the two endpoints. The msocket application keeps tracks of these flows in its connection with the server, and sends and receives on each of these paths based on a policy, such as a round robin policy. If one of the flows times out without receiving an ack, the msocket will initiate a migration protocol, where it will try to reconnect to the server.

2.3 Related Work

2.3.1 Content Delivery Networks

Content delivery networks focus specifically on the issue of replication of data to provide for reliability and performance. Constructed from an overlay network of strategically placed and geographically distributed servers, CDNs generally use a hierarchical topology to form their networks. CDNs typically incorporate dynamic information about network conditions and load on the servers, to redirect requests and balance loads among their servers[19].

Over the last two decades, CDNs have evolved to serve over 34% of all Internet traffic in 2013[18]. This has garnered a lot of research in the effectiveness of CDNs in many different situation. Because of topological similarities between CDNs and NDN as well as CDN's focus on caching, insight gathered from CDNs can help us in determining the effectiveness of our caching strategies in NDN as well as a larger comparison between NDN and traditional IP/TCP networks. Akamai, delivers hundreds of billions of Internet interactions daily and is made up of more than 60,000 servers in over 70 countries[15].

2.3.2 BitTorrent

BitTorrent, one of the most popular peer-to-peer implementations, tries to redistribute the cost of upload to downloaders [4]. In P2P, unlike in a CDN, all users are both server and client, and act without the need for a centralized server. A file is broken up in fix-sized pieces and each user keeps track of which pieces it owns. When a user wants to download a file in BitTorrent, the user looks at all the available seeds(other users) and decides on from which seeds it can start downloading. This allows the user to download parts of the file from different locations in order to maximize its download speed. Here, we see one major difference between BitTorrent and NDN at the level of who makes the decisions in where a user gets the packets. This difference changes the overall path a packet takes in the network and might provide some insight on how this affects the overall traffic in BitTorrent versus NDN.

BitTorrent presents a system in which a file is broken down into many different pieces and in which a distributed system allows users to coordinate and effectively download all the different pieces from the different endpoints that host them. The level in which BitTorrent operates is congruent to how NDN uses packets in its system. In a way, the temporary connectivity of peers[4] also exhibit similarities of caching in the larger system and allows the user to dynamically choose where to send their queries.

2.4 Conclusion

Current caching mechanisms operate at different layers of the transport stack, either as an overly on top of the TCP/IP protocols or at the application layer. The new proposed architectures integrate caching on a much lower layer. Thus, while current caching schemes greatly impact the overall performance of the system, the caching in the new proposed architectures will do the same as well as having the opportunity to impact the actual semantics of the network.

Chapter 3

Evaluation Factors to Caching

The primary goal of this thesis is to evaluate the impact of caching on future architectures. To do so, we will look at both the factors in the network that would affect how caching is employed as well as how to relate these factors to the actual quantitative impact of caching. Elements such as network topology, user request distribution, caching policies, and user mobility all contribute to the overall flow of the network. We will also look at how to define the metrics to quantify the effects of caching.

Because MobilityFirst and NDN are both clean slate approaches to a new Internet architecture, the way each utilizes caching differs considerably. In NDN, caching lives in a single layer of the architecture, while MobilityFirst actually has multiple parts of its system where caching comes into play. In the case of MobilityFirst, we must also look at how the layers of caching interplay in the overarching system. While NDN and MobilityFirst are very different architectures, we have tried to use similar network topologies and user requests to gain a better perspective between the two systems.

In this section, we will discuss what factors we will look at to quantify the performance and systematic differences that caching yields.

3.1 Network Factors to Overall Performance

3.1.1 Network Topology

Studies into the topology of the internet have shown that it has a hierarchical domain structure, with transit domains, stub domains, and end user nodes attached to stub nodes[27]. Each domain consists of interconnected nodes that share routing information. Connections between nodes in a domain stay within the set of nodes in the domain while transport domains interconnect stub domains effectively, such that sub domains do not need to route to each other directly. Sub domains connect to transit domains usually through a gateway node.

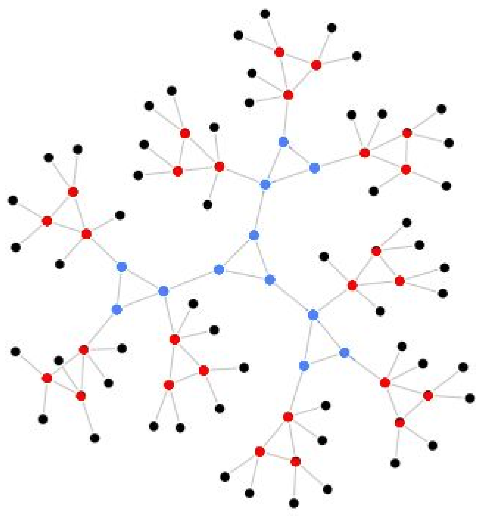


Figure 3-1, 3 Ring, 3 Layer, 2 Edge Nodes

We used these topology properties to generate a simpler model consisting of a star ring configuration, as seen above. A ring of nodes represent a domain. Each ring is connected to each other in a hierarchical order. Inner rings represent transit domains, while rings at the edge are stub domains. The tree structure of the inner rings represent the hierarchical core network. Each node in the stub domain rings has edge leaves, representing the end user nodes. While this topology generation does not allow for the factor of randomness, we believe that it is a good enough approximation of a model to a realistic network topology. Along with this topology, we will also use a simple mesh network to provide a comparative topology.

3.1.2 Request Distribution

Most attempts at modeling user internet requests follow the assumption that user requests are both independent and zipf-like in distribution[7] following studies into user internet traces which were conducted starting from the 90s[5]. Breslau and al. argue that the individual popularity of an item is proportional to a $1/i^{\alpha}$ frequency, where $0 < \alpha <= 1$. This suggests that the top few popular items make up the majority of the total requests, which concurs with results from previous traces that show the top 1% of the documents accounting for about 20% to 35% of all requests.

Breslau and al. acknowledge that the assumption that requests are independent is obviously an over-simplification. This simplification ignores the impact of spatial and temporal locality[9]. For example, if we look at the rise in a viral video, aspects of these localities come into play. Through social media, people influence, in a short amount of time, those around them, who are usually spatially close. As more and more people become interconnected and influenced by each other, these localities should be given more weight. JJ Garcia-Luna-Aceves et al[2] have presented a model of stochastically generating user requests to populate a simulated trace, with the added factor of locality by using a poisson process .

The distribution of user requests directly affects the effectiveness of caching. Some of the most popular caching replacement strategies today, LRU and LFU, manage to do so well because they are well suited to the expected distribution of requests. We will use both the zipf-like distribution as well as the augmented version in our tests.

3.1.3 User Mobility

With over 25% of internet traffic coming from smartphones and tablets in 2014, and an estimated 90% of internet traffic coming from mobile devices by 2020 [30], we see that user mobility will become a major factor in how well the internet will be able to deliver data. People are often on the move -- the daily commute most people have from home to work is one example of this. With mobile devices becoming the major producer of traffic in areas such as music streaming[31], users are clearly not only moving, but downloading information at the same time. Since MobilityFirst is built on the premise that the Internet is approaching an historic inflection point, with mobile platforms and applications poised to replace the fixed-host/server model that has dominated the Internet since its inception[14], a mobility model must be involved when testing the system.

One difficulty to modeling user mobility involves integrating the physical movements of a user to the movement of the device in the network. Walking or driving around results in a change in the cell/LTE tower as the user moves from the signal strength of one tower to another.

This small geographical change results in similarly small changes of location in the network. However, the switch between networks, such as Wifi and LTE, while the user is not necessarily physically moving, also results in a change in user location in the eyes of the network. Even more importantly, this switch between networks results in greater change in user locality in the network.

Yang and al.[27] defines a transition matrix between one network and another as on the scale of one transition every 15 minutes. However, their study used traces of user access to IMAP servers, a small subset of possible user requests. Kim and al.[11] suggest modeling a user as either mobile or stationary using a mobile to stationary ratio. A mobile user's next destination is then chosen based on the probabilities in the region transition matrix, with the number of points visited on the way based on another matrix. They argue that there are natural hot spots that users congregate to.

Using these observations about user requests and user mobility, we incorporated both aspects into a simulated trace of overall network usage.

3.1.4 Caching Resources

While memory and disk space have exponentially decreased in price and increased in availability, there are only so much resources that can be set aside for caching. JJ Garcia-Luna-Aceves et al[2] suggests that spending the majority of cache resources on the edge of the network actually approaches optimal caching. We will look at how the cache budget, the cache size at each router relative to the total number of possible objects, will affect the overall performance gains.

3.2 Evaluating Caching Performance Metrics

Below, we define the metrics we will look at to quantify the benefits caching provides. Individually, these metrics measure the performance gains of caching, but by comparing them to the network factors, we attain a greater overall understanding of the trends of the architectures. Some of these metrics, such as cache hit ratio and hop count, map more easily to NDN since NDN allows for caching of data that is independent of the requester.

- Cache Hit/Miss probability directly measures the effectiveness of a particular cache replacement strategy.
- Hop count is defined as the number of hops between the user and the closest replica of the content. In NDN, a hop count tag is added to each packet, which gets incremented every time the packet is forwarded and reset any time the packet is pulled from a content store (cache).
- Total Transmission Time gives us an idea of the end user experience. In a congested environment, a request might get dropped and need to get retransmitted before successfully reaching the end user.
- Throughput measurements gives us an overview on how the entire network is functioning.

3.3 Architecture Specific Responses

3.3.1 User Mobility

In the event that a user moves before the request has had time to finish, NDN assumes that the request will time out, and the user will simply request again at the new location. Because there is caching along the path of the request, NDN argues that a change in locality can still take advantage of a cached copy nearby. However, current implementations of NDN forwarding strategies do not take into account nearby caching states when making forwarding decisions. Thus, the locality of nearby cached copies can be entirely ignored if the path the forwarding strategy chooses bypasses those locations.

MobilityFirst, on the other hand, actively tries to guarantee delivery of data, even when the user is in transit during the transmission. They do this using their GSTAR service, in which they will cache data until the user finishes changing locations in the network and reroute at that time. In this sense, MobilityFirst is acting as more of a long term buffer to individual data.

3.3.2 MobilityFirst NA to GUID Binding

MobilityFirst provides the ability for a device to multihome on multiple interfaces connected to different NAs. There are two mechanisms to resolve the binding of the GUID to a

NA in the case where there are multiple NA destinations possible for a GUID. One is through the use of the msocket, while the other is done inside the network by the routers. In msocket, the server side determines which NA to specifically send the chunk, resulting in an early binding of NA to GUID. The other method allows for the chunk to pass through the network, with the set of NAs it can bind to, until it reaches a point in the network where a decision can be made on which path and resulting NA that chunk needs to be forwarded.

Both offer benefits and costs. Late binding, letting the routers decide which NA to send to, allows for the network to respond in real time to network changes. Routers, through GSTAR, keep track of link state. The best path at the time is chosen, and less chunks need to be dropped overall. However, msocket, at a greater cost, provides a higher level view on the overall network. Because msocket keeps track of acknowledgements from successfully sent chunks, it can automatically retry, on a specifically different path, after a timeout. While a mobility event or a bad connection results in a complete retransmission of a chunk, msocket operates with a greater set of information of the overall network from one GUID to another.

Chapter 4

Evaluation

In this chapter, we will be describing the exact experiments we conducted to test the effects of caching in NDN and MobilityFirst.We will discuss the basic testbed setup used before moving on to the specific tests for each architecture. Because NDN and MobilityFirst are separate architectures, many of our tests share similar aspects but contain various differences.

4.1 Testbed

4.1.1 NS3

There is currently an implementation of NDN called ndnSIM[13] using ns-3, a discrete-event network simulator. One benefit to this simulation mode is that we can easy create any scenario to test out the system, adjusting for topology, link distance, and throughput. Many of the basic node level caching strategies such as FIFO, LRU, etc, have already been implemented in ndnSIM. All future work will modify this implementation of NDN to simulate scenarios and to compare our results with the various caching strategies.

4.1.2 Orbit Lab

The experimental setup for MobilityFirst exists in a testbed called Orbit-lab located in the MobilityFirst lab in Rutgers. They have a 20 x 20 grid setup of interconnected radio nodes with both a self-contained ethernet subnet as well as wifi capabilities.

4.2 NDN

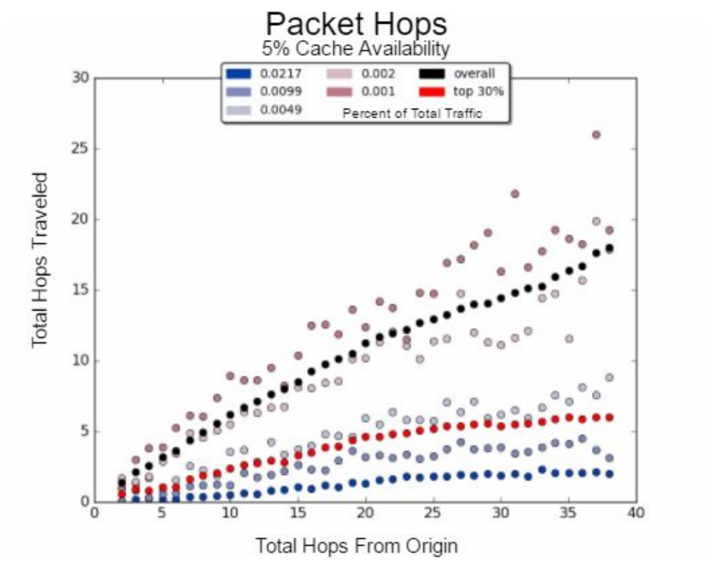
4.2.1 Baseline

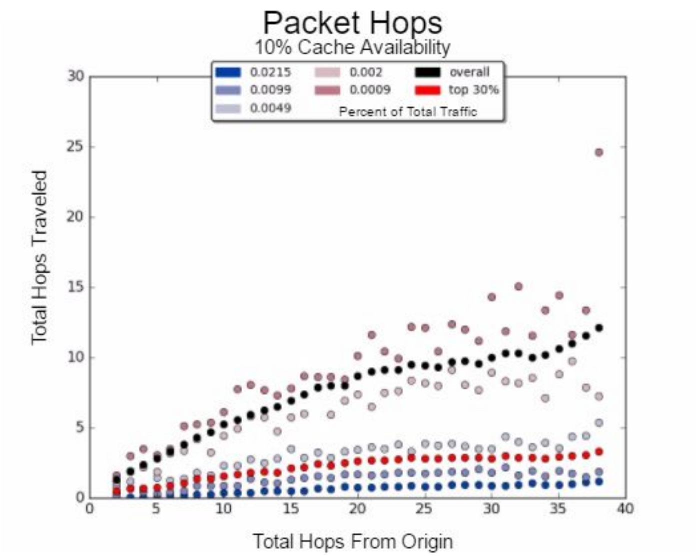
We started off with a 20x20 mesh topology, with each node using LFU with a cache size of 100. 40 consumers were randomly distributed across the network, and a single origin was located at the 0,0 corner of the mesh. Each consumer used a zipf distribution, with an alpha of 0.7. We varied the total number of data objects in accordance to the cache budget -- a cache budget of 5% would mean that we used 2000 objects.

When a timeout occurs, where the consumer had to retransmit the Interest packet, there are multiple scenarios that could have happened. In some cases, the consumer might just be too far from a cached copy anywhere and times out before the interest packet finds a cached/origin copy, and in other times the data packet might had started its way back to the requester before it got dropped in the network. However, we have no way of measuring how far along the path the data packet had made its way before the consumer issued a new request along that path. The packet might now be cached closer to consumer, and the consumer has no idea how much extra work had been done.

Because in our simulation we know the maximum distance from any consumer to the origin, we looked at various ways of including this factor when adjusting for the actual hop count when a consumer receives a packet that had previously timed out. We looked at multiple approaches, including either a multiplicative or an additive factor to the received hop count, but we found that neither had much overall effect on the performance of the network as a whole. Thus we concluded that we could exclude timeout events from our data without significant impact to the overall performance.

In the figure below, we measure the number of router hops the data packet took to get back to the requester vs the total number of hops to the requester from the origin of the packet. Here, the black line represents the overall average hop count, while the red line represents the average hop count of the data objects that constitute the top 30% of the total traffic. We see that even with a cache budget of 5%, the benefit to the entire network is noticeable. By 20% of cache budget, the gains are significant.





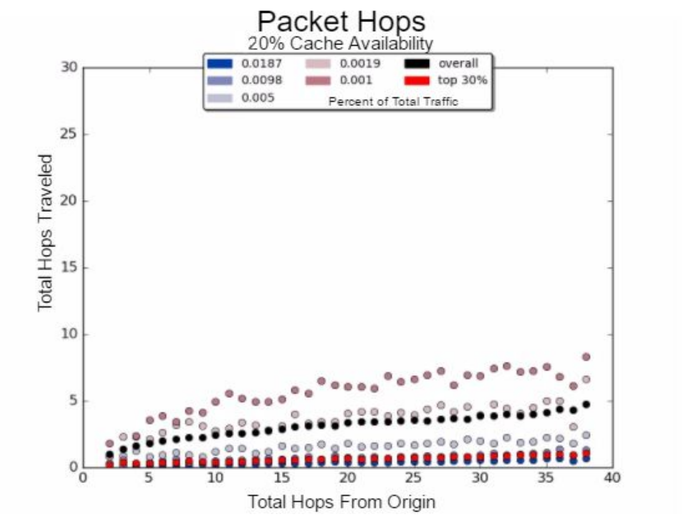


Figure 4-1 Average Hop Counts with Content Caching

One notable trend we see is the leveling out of hop counts, especially among the top 30% of the traffic. Instead of a linear relationship between the total distance from the source and the average distance from a cached copy, we see that the distance eventually comes to a constant factor that is far lower than the total distance the farther away it is. This indicates a level of locality of objects that results from the use of caching in the system. This kind of locality could be exploited, especially when looking at higher level caching strategies. When we think about the internet of today, the median hop count from anywhere in the continental USA is around 14 [6], with each packet passing multiple Autonomous systems. A low average hop count means that the packet doesn't have to pass through very many ASs, thus avoiding many of the problems that come with inter-domain protocols such as BGP.

4.2.2 Cache Replacement Strategies

In this experiment, we used the same setup as previously, changing only the cache replacement strategies installed on each node. We tested for LRU, LFU, and Random Replacement. The cache hit ratios are per object basis. The first CDF is over the total number of data objects, while the second is normalized over the total number of requests per data object.

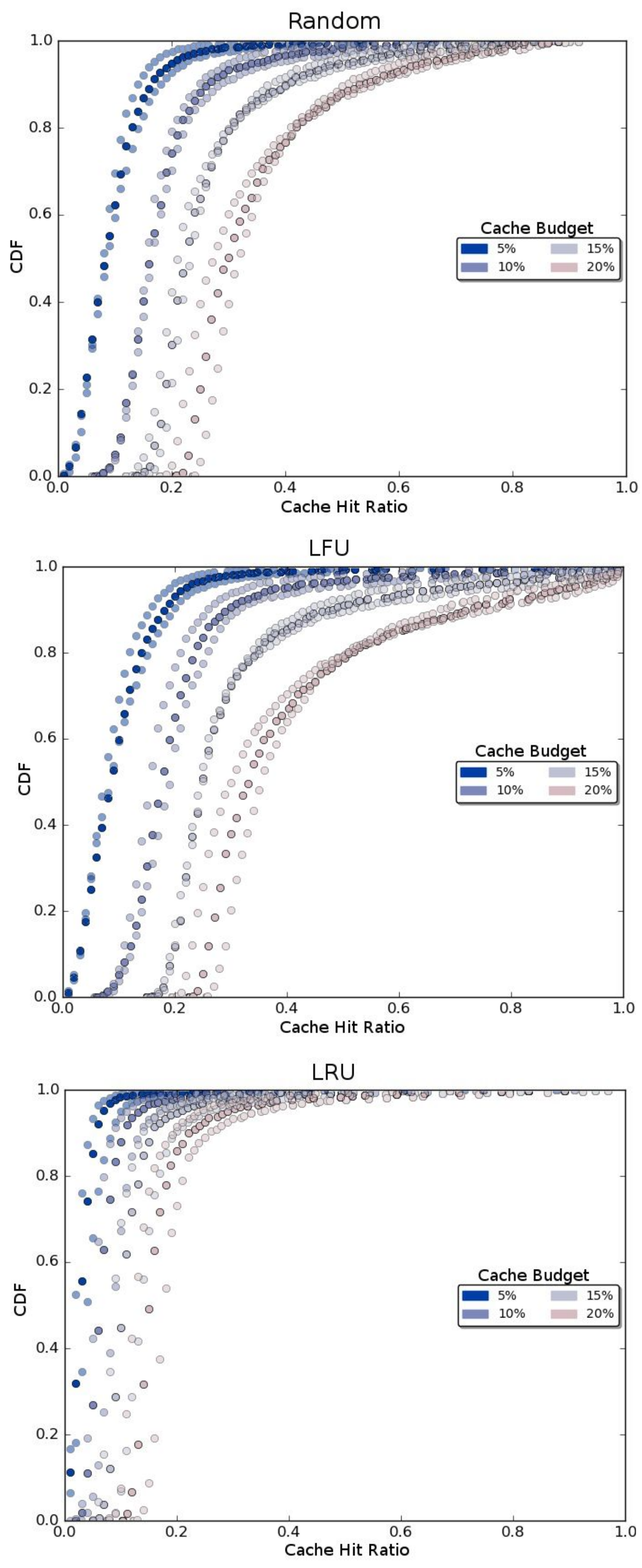


Figure 4-2 CDF of Cache Hit Ratios of Popular Cache Replacement Strategies

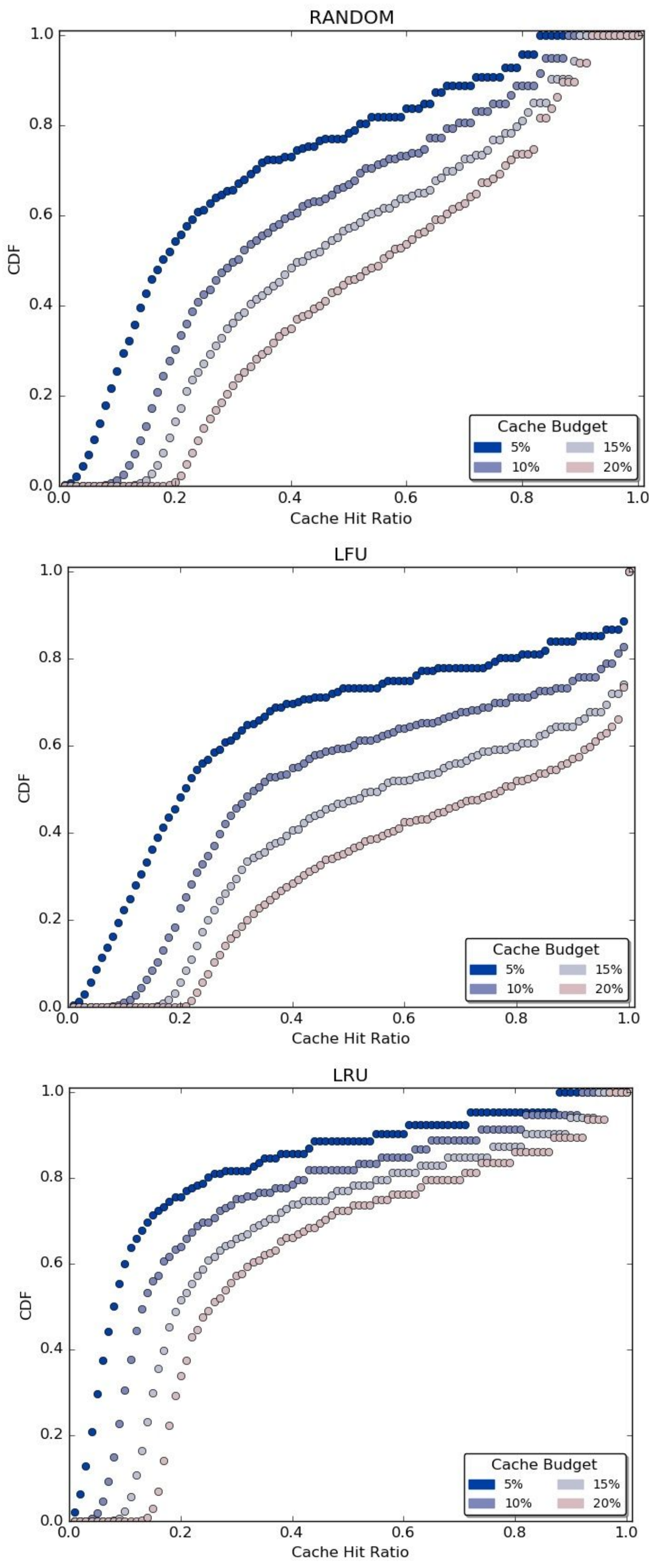


Figure 4-3 Normalized CDF of Cache Hit Ratios to Total Requests, $\alpha = 0.85$

Above, we have a CDF of the total number of data objects over the average cache hit ratio of an object across three different cache replacement strategies, LRU, LFU, and Random Replacement. The main center line uses a zipf distribution alpha of 0.7, while the two other lines represent alphas of 0.55 and 0.85. This experiment used the same setup as the first experiment.

First, we can see the difference the caching strategy makes in the overall effectiveness of caching. As expected, LFU is the most effective strategy, given that we are using a zipf distribution. Interestingly, Random Replacement also works reasonably well, and is comparable to LFU in the low to mid range cache hit ratios. We see here that varying the alpha values of the zipf distribution does make some difference. In Random Replacement and LFU, a higher alpha value decreases the effectiveness in lower cache hit ratios, and crosses over to improve the effectiveness in higher hit ratios. When Breslau and al. explained that they modeled requests with a zipf-like distribution, they noted that the alpha values can vary from 0 to 1. For LFU and Random Replacement, we see that the most impact of varying alpha values occurs in the worst performing hit rates. These hit rates correspond with the least requested data objects, whose impact is minimal to the overall effectiveness of the system. However, for LFU, the difference is significant. Different alpha points significantly shift the performance rate of the highest requested objects, which then have a significant impact on the overall performance of the network. A higher alpha, which correspond to a steeper zipf distribution, benefits LRU significantly higher than lower alpha, which makes sense, as a higher alpha means that there is a greater probability that there would be another request on popular objects in the span of time the cache size.

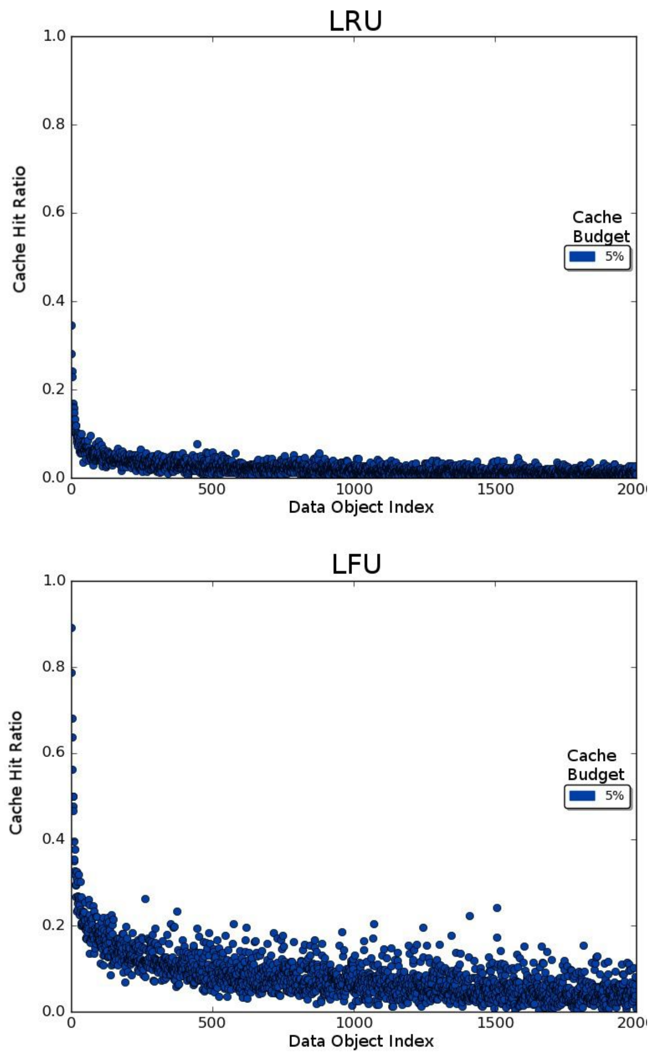


Figure 4-4 Cache Hit Ratio vs Popularity, α = 0.55

In the above graph, when the alpha was set at .55, we see that LRU responds poorly to a flatter zipf distribution. Especially when we consider that the cache size is relatively small compared to the total number of possible requests, we see that LRU kicks out cached data before a new request can come. Because LRU has essentially a snapshot of the current state of the network, LRU functions best when there is short term temporal locality in requests. On the other hand, we see that Random Replacement and LFU are not as affected by alpha values since LFU keeps track of the history over the entire span of time and Random Replacement keeps no history.

4.2.3 Network Topology

In our next experiment, we used a star ring topology described in section 3.1.1, using a ring size of 6 and a 3 layer setup, with 1 leaf node at each bottom layer ring. Due to this topology, the nearest neighbor is 3 hops away. A cached size of 100 was used for 2000 total data objects.

Object Popularity Rank	**1**	**2-4**	**5-10**	**11-30**	**31-100**	**101-500**	**501-2000**
Ring Layer 1	**0.44**	**0.48**	**0.52**	**0.54**	**0.5**	**0.38**	**0.17**
Ring Layer 2	**0.36**	**0.35**	**0.37**	**0.39**	**0.32**	**0.19**	**0.06**
Ring Layer 3	**0.28**	**0.22**	**0.2**	**0.19**	**0.13**	**0.06**	**0.02**

Table 4-1 Average Cache Hit Ratios Across Data Objects(LFU)

In the table above, we averaged the cache hit rates over data objects of different popularities. Column 1 has the average hit rate for the most popular content, column 2 has the average for the 2nd-4th most popular, and etc. Data object 1 has the highest popularity, while by object 500, the popularity is relatively small. When we look at caching across the different layers of the topology, we see that the core network is consistently better at serving cached versions of the data object than surrounding outer layers because the amount of traffic that goes through the core gives routers at that level the best overview of the network as a whole. However, the most popular content is more likely to be cached in the lower layers, since they are requested so frequently. We see that by the mid range objects of 10-100, considering the fact that our test used a cached size of 100, those objects have filtered up to the higher and higher levels. This correlates with the earlier findings of the leveling off of hops by suggesting that it really doesn't matter how far away a consumer is from the origin of the data object, but how far the consumer is from the core of the network. Because NDN decouples data from origin location, the use of caching in the network allows for an independence from where the data originated from.

When we think about where caching would be most effective, we see that the cache hit ratio in ring layer 3 follows a straight zipf-like curve. If we consider putting more cache resources at the edges, the hit ratios for popular content would effectively go up in the lower layers. Layer 2 would see less of the most popular content because they would have already gotten a cache hit earlier, and thus would see most of the traffic the mid range data objects. Layer 1 would then see the same filtering process and would be best at caching for even less popular content. Overall, that benefits the network, as more content would be served from the edges of the network.

4.2.4 Spatial Temporal Locality

Using the algorithm described by JJ Garcia-Luna-Aceves et al[2], we reran our experiments using generated traces. There is an initial seeding of requests across all time steps, and at each future timestep, a poisson process determines if and how many children requests are propagated. These children requests are randomly distributed across the neighbors of the original request. We compare the results when we run the same trace generation on the star ring topology. There seems to be very little effect to our system as a whole.

Locality	**1**	**2-4**	**5-10**	**11-30**	**31-100**	**101-500**	**501-2000**
0	0.28	0.22	0.2	0.19	0.13	0.06	0.02
0.3	0.3	0.24	0.21	0.21	0.15	0.07	0.03
0.6	0.29	0.25	0.21	0.21	0.16	0.07	0.03

Table 4-2 Cache Hit Ratios in Layer 3

4.2.5 NDN Conclusion

In this section, we have seen that there is a locality in where user requests are served that is independent of the origin of the data. While LFU is resilient to those changes in the distribution of requests, other cache replacement strategies such as LRU is significantly affected because of the short term temporal history it keeps track of. Finally, we see that the topology of the network greatly affects the distribution of requests to nodes in different locations in the network.

4.3 MobilityFirst

4.3.1 MobilityFirst Stack

MobilityFirst targets both the network and the transport layers as well as offering a session layer application, msocket. However, the msocket and transport/network layer were prototyped separately and ignorant of each other's architecture. In order to fully evaluate the overall architecture, we made modifications to both prototypes to make them compatible to each other by layering msocket on top of the routing layer. On the routing layer, we modified the code so that the user can directly choose which NA they wanted to attach to the GUID. On the msocket side, each flow, or connection, was originally tied to an IP address and the data was transferred through a java socketChannel. Instead, we made each flow to be tied to a NA, and all data transfers used the underlying MobilityFirst routing prototype.

4.3.2 Simulating Mobility and Multihoming

One of the major aspects of MobilityFirst is built on the fact that the users are physically mobile. However, with the Orbit main testbed consisting of static 20x20 radio nodes, we needed a way to simulate mobility. To do this, we wrote a script that would save the status of the msocket application, scp it to another node, and restart the application as if it had physically moved locations. The msocket would try to reconnect to the server, and it would seem as if a msocket migration had occurred. Msockets initialized on separate machines simulate multihoming across separate NAs. They use the same connection info, and to the server, it seems as if there is one machine that is multihomed over multiple interfaces.

4.3.3 Hop Architecture

One of the variables in the Hop Architecture is the size of a chunk. A hop segment will wait until all packets in a chunk are received before sending the chunk downstream. A larger chunk size means fewer overall chunks sent over the network, even though there is the same amount of data transferred. In the network, routers have a backpressure mechanism where a router will stop sending downstream after a certain threshold between the total number of chunks sent downstream minus the CSY_ACK for that chunk from the downstream router is met. By default, there is also a buffer of 10 chunks in the socket layer on the client end. The client is forced to wait until a space opens up in the buffer, which happens when a CYS_ACK for a sent chunk returns.

In this experiment, we used a simple four router link line topology, with a client attached to an access point at either ends. We sent a 20MB file at chunks of size 256KB, 512KB, and 1024 KB while varying the size of the buffer at the client end.

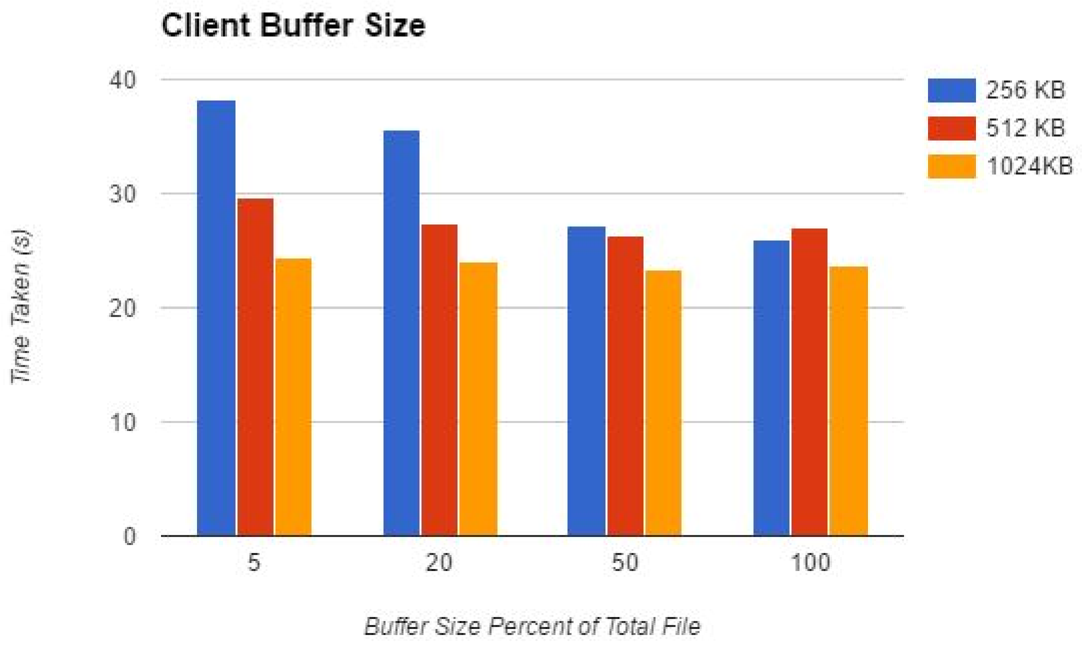


Figure 4-5 Download Times for 20MB File

We see from above that the fact that the buffer is based on number of chunks versus a set cache size favors chunk sizes that are larger, as expected. Using smaller chunks creates for more overhead because clients need to wait for proportionally more round trip times between each hop segment before a router can initializing further network sends downstream. At a certain point of chunk size compared to the overall size of the file, we see that the benefits of increasing the chunk size decreases.

4.3.4 Mobility on One Interface

In this experiment, a user utilizes one interface while moving around in the network. The topology is a 2 layer, 3 ring, 2 edge node topology as described in section 3.1.1, and the server attempts to send a 10MB file. When a mobility event happens, the msocket disconnects and reconnects to the server at some later time. After each migration, the server checks to see what data chunks need to be resent. When the client reconnects to the network, the client sends the server the sequence number of the highest received in-order chunk and if the server sees that it has tried to send more to the client, it will try to resend all chunks from the highest in-order chunk to the current chunk. On the router layer, once the msocket reconnects to a NA after a mobility event, the GSTAR service kicks into place to try to deliver chunks originally meant for a previous NA to wherever the user reattaches to a new NA.

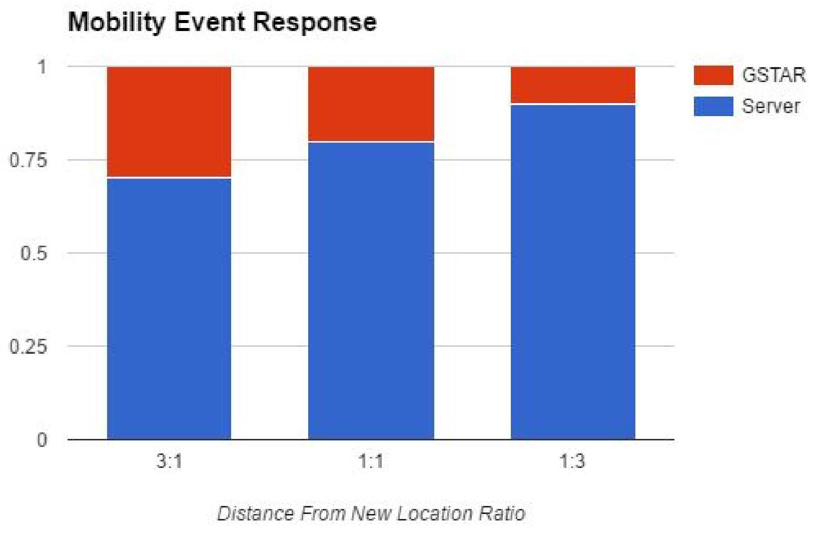


Figure 4-6 Chunks Received from Resend vs GSTAR Service

Above, we show the ratio of the data received from either the original server or the GSTAR service over the first 10MB. The ratio is the distance to the new location from the server to the GSTAR previous location. We see that overall, it is a lot faster for the server to resend the packets than for the GSTAR to deliver them. One reason for this is that the GSTAR service, when dealing with a chunk with no NA binding, tries to resend chunks by making a GNRS request to lookup the GUID to NA binding in the order of every couple seconds, by which time the server has already started sending to the new location. This delay is exaggerated in our test scenario, since all of our nodes in the testbed are physically close together and are unable to change the link speeds.

Sequence of arrival	1	3	0	2	4	5	6	7	8	9
Displacement	-1	-2	+2	+1	0	0	0	0	0	0

Table 4-3 Example Displacement Seen From Client End

We see that even when, over one interface, the end router sends chunk in order, the msocket still might receive chunks out of order. The msocket layer allows for out of order delivery by storing out of order chunks in a buffer. However, it only sends an acknowledgement of successfully received chunk based on the highest in-order chunk that the end user has retrieved from the buffer. Thus, on a mobility event, the server will try to resend all chunks past the last acknowledged chunk, potentially resending a lot of redundant chunks. One of the benefits of using GSTAR, in this instance, is that the router where the chunks were stored knows which chunks were successfully delivered, and it will try to forward the remaining chunks to the msocket layer. The first chunks that the moved msocket will receive from GSTAR will be a chunk that isn't already in its buffer.

4.3.5 Multihoming

As a session layer application, msocket manages the connection between a client and a server. One of the benefits of msocket outside of its buffer is its management of multiple flows. A user might have the capability to use multiple interfaces, and each interface is a separate flow connected to the server. In this section, we look at the benefits of multihoming over the MobilityFirst network.

We look at two scenarios, one where there are two stationary multihomed interfaces, and one where one is stationary and the other is mobile. An example of this is when a person uses both LTE and Wifi hot spots. In our experiment, both interfaces have the same link speed.

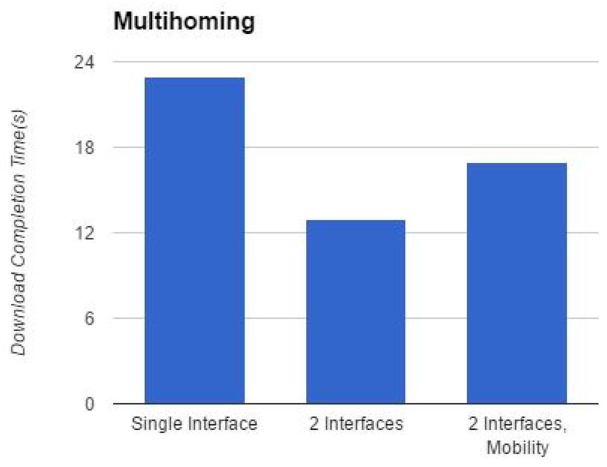


Figure 4-7 Multihoming

Above, we see that there is an additive benefit to multihoming when both interfaces are stationary. The flow paths to each interface, because they are on two separate networks, allow for the server to send on two paths that don't add congestion to each other. As seen in section 4.3.3, when a mobility event occurs, it is most likely that the server has to resend the chunk meant for the moving interface as opposed to the GSTAR service delivering it first. Overall, using two interfaces, even with one moving, is still better than one when considering the total download time. One issue would be if the user needed to access data as it was coming into the buffer, as it would happen during a video stream. In that case, the mobility of one interface could mean a long delay a much needed in-order chunk.

4.3.6 MobilityFirst Conclusion

In this section, we have looked at the Hop protocol, the GSTAR service, and the msocket session application layer. We have discussed the performance overhead of using large chunks of packets versus smaller chunks of packets as well as the relationship between the GSTAR service and how msocket natively helps the user in a mobility event. We have found that the multihoming capabilities provided by msocket greatly improves the overall performance of the session. By utilizing in-network storage and a two-fold approach to deal with mobility events, MobilityFirst provides reliability and improved user experience in cases of user mobility. However, the overhead that comes with their use of caching in the network, in terms of CSY_ACKS and GNRS calls, can pose scalability issues.

Chapter 5

Future Work

This thesis conducted a general overview of the caching mechanisms in NDN and MobilityFirst. We looked at the default behaviors of the systems as described in their project overview. However, there have been continuing work to add functionality and complexity to the protocols that drive each architecture. Future work will look at these additions to the story and evaluate how they affect the data flow in novel ways.

5.1 NDN

- *Forwarding Plane*: Caching Replacement strategies present the lowest layer of the forwarding plane. On top of the caching mechanisms, there is a forwarding strategy, where the routers need to decide on which path to forward new requests. Currently, there are various strategies, ranging from best performance and random path to more complicated decision making strategies. Caching is currently on path of the interest request, so looking at how that path is decided on can gives us a better understanding on how requests move through the system and how they are cached.

- *Prefetching*: Currently, the default behavior of NDN is to cache on requests. Other higher level caching mechanisms[5] create overlay networks that have a better understanding of the network behavior as a whole, and can initiate prefetching of popular content so seed the caches. Looking at these strategies, we can compare their benefits and overhead and how these overlay networks affect the locality of requests.

5.2 MobilityFirst

- *Content Caching*: In an effort to provide a better user experience for the individual, in-network storage in MobilityFirst has been focused on providing for reliability for an individual flow as opposed to caching of content for the overall system. F. Zhang and al. [29] have proposed the caching of specific chunk numbers as a way for content caching at the edge of the network. Future work can evaluate the effects of this use of in-network storage in MobilityFirst
- *GUID to NA binding* : Currently, both in-network NA to GUID binding decision making and client side NA binding are supported. Looking at both methods, we can get a better understanding of the benefits of making such decision at different locations in the network and how that affects the traffic flow and use of caching in the network.
- *Larger Testbed*: One of the issues with the ORBIT testbed is the inability to emulate changing link rates in the 20x20 grid. This makes it difficult to simulate real life transfers of data across significant distances. MobilityFirst is also deployed on another testbed, GENI which is made up of real computers spanning multiple institutions across the US.

Chapter 6

Conclusion

In this thesis, we evaluated the caching mechanisms of two Future Internet projects, NDN and MobilityFirst. These clean-slate approaches to alternative Internet architectures capitalized on the advances in technology in recent years that have made in-network caching on the router level feasible. In their designs, these caching capabilities are integrated into the routers themselves.

The pervasive caching scheme in NDN allows for interesting trends to pop up. Because content is no longer tied to an end host and the content itself is visible to the network layer, the origin point plays a lot less of a factor to the overall expected distance to any consumer, especially the more popular the content. We see that the overall distribution of requests impacts some caching strategies more than others, with LFU reasonably resilient to those changes. Finally, from a topological standpoint, we see how caching affects the perceived distribution of requests to nodes in different locations in the network.

In MobilityFirst, we have evaluated how the different in-network storage and caching layers interact with each other. We looked at how the Hop protocol responds to different chunk sizes and its impact on the overhead. We see under which scenarios the GSTAR service would be most beneficial to the end user, and we looked at the additive benefits of multihoming through the use of msocket.

References

[1] V. Arun, J. Kurose, D. Raychaudhuri, S. Banerjee, and M. Mao. "MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture." Sigcomm (2014)

[2] A. Mirzazad-Barijough J. J. Garcia-Luna-Aceves. “Understanding Optimal Caching and Opportunistic Caching at ‘The Edge’ of Information-Centric Networks”, Proc. ACM ICN ‘14, Sept. 2014

[3] W. Chai, D. He, I. Psaras, and G. Pavlou. Cache “Less for More” in Information-Centric Networks (Extended Version). Comput. Commun., 36(7):758–770, 2013

[4] B. Cohen. “Incentives build robustness in BitTorrent.” In Proc. of IPTPS, 2003.

[5] C. Cunha, Azer Bestavros, and Mark Crovella. “Characteristics of WWW client-based traces” Technical Report TR-95-010, April 1995.

[6] A. Fei, G. Pei, R. Liu, and L. Zhang. Measurements on delay and hop-count of the Internet. Proc. IEEE Global Internet’98.

[7] S. Glassman. A caching relay for the world wide web. In First International Conference on the WorldWide Web, CERN, Geneva, Switzerland, May 1994.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard “Networking Named Content” CoNEXT 2009, Rome, December, 2009.

[9] S. Jin and A. Bestavros. Sources and characteristics of Web temporal locality. In IEEE/ACM Mascots, 2000.

[10] K. Katsaros, G. Xylomenos, and G. C. Polyzos, “Multicache: An overlay architecture for information-centric networking,” Computer Networks, to appear, 2011

[11] M Kim, D Kotz, S Kim Extracting a mobility model from real user traces Proc. IEEE Infocom, Jan 2006.

[12] M. Li, D. Agarwal, and V. A., “Block-switched networks: A new paradigm for wireless transport.”

[13] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, “ndnSIM 2.0: A new version of the NDN simulator for NS-3,” NDN, Technical Report NDN-0028, 2015

[14] S. C. Nelson, G. Bhanage, and D. Raychaudhuri, “GSTAR: Generalized storageaware routing for MobilityFirst in the future mobile Internet,” in Proceedings of MobiArch’11, 2011, pp. 19–24.

[15] E. Nygren , R. Sitaraman and J. Sun, "The Akamai network: A platform for high-performance Internet applications", *Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2-19, 2010

[16] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic In-Network Caching for Information-Centric Networks. In Proc. ACM SIGCOMM Workshop on ICN, pages 55–60, 2012.

[17] P. Rodriguez, C. Spanner, and E. Biersack. Analysis of Web Caching Architectures: Hierarchical and Distributed Caching. IEEE/ACM Trans. Netw., 9(4):404–418, 2001.

[18] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology. 2015. Web. May 10, 2016

[19] A. K. Pathan, and R. Buyya, “A Taxonomy and Survey of Content Delivery Networks,” Tech Report, Univ. of Melbourne, 2007.

[20] NDN Project Team, NDN, Technical Report NDN-0022, 2014. Web. May 11, 2016

[21] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri, “MobilityFirst Future Internet Architecture Project,” in MobilityFirst Project, Proc. ACM AINTec 2011.

[22] N. Somani, A. Chanda, S. C. Nelson, and D. Raychaudhuri, “Storage-Aware Routing for Robust and Efficient Services in the Future Mobile Internet,” in Proceedings of ICC FutureNet V, 2012

[23] Y Thomas, C Tsilopoulos, G. Xylomenos, G. Polyzos “Object-oriented Packet Caching for ICN” In Proceedings of the 2nd International Conference on Information-Centric Networking (ICN’15), San Francisco, CA, USA, Oct. 2015.

[24] A. Venkataramani, X. Tie, A. Sharma, D. Westbrook, H. Uppal, J. Kurose, and D. Raychaudhuri. Design Guidelines for a Global Name Service for a Mobility-Centric, Trustworthy Internetwork. COMSNETS, 2013.

[25] T. Vu, A. Baid, Y. Zhang, T. Nguyen, J. Fukuyama, R.P. Martin, D. Raychaudhuri Dmap: A shared hosting scheme for dynamic identier to locator mappings in the global internet. In Proceedings of IEEE ICDCS (June 2012).

[26] A. Yadav, A. Sharma, A. Venkataramani, E. Cecchet, msocket: System Support for Developing Seamlessly Mobile,Multipath, and Middlebox-Agnostic Applications(Extended technical report), UMASS CS Tech report http://people.cs.umass.edu/~ayadav/msocket.pdf

[27] S. Yang, J. Kurose, S. Heimlicher, A. Venkataramani Measurement and Modeling of User Transitioning Among Networks Infocom 2015

[28] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," Proc. Infocom, Apr. 1996.

[29] F. Zhang, C. Xu, Y. Zhang, K. Ramakrishnan, S. Mukherjee, R. Yates, and T. Nguyen, "EdgeBuffer: Caching and prefetching content at the edge in the MobilityFirst future internet architecture," in WoWMoM. IEEE, 2015, pp. 1–9

[30] Walker Sands Digital. Mobile Traffic Report Q3 2013. Web. May 12, 2016

[31] L. Johnson, Mobile Commerce Daily. Pandora exec: 70pc of traffic comes from mobile. 2012. Web. May 14, 2016

[32] D. Saxena, V Raychoudhury, N. SriMahathi. SmartHealth-NDNoT. 2015. Web. May 13, 2016.