A CONTENT CACHING STRATEGY FOR NAMED DATA NETWORKING

SEYED MOSTAFA
SEYED REZAZAD DALALY

NATIONAL UNIVERSITY OF SINGAPORE

2014

A CONTENT CACHING STRATEGY FOR NAMED DATA NETWORKING

SEYED MOSTAFA SEYED REZAZAD DALALY
(M.ENG, SHARIF UNIVERSITY OF TECHNOLOGY, 2004)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2014

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Seyed Mostafa Seyed Rezazad Dalaly

20 December 2014

Acknowledgments

First and foremost, I have to thank my research supervisor, Professor Y.C. Tay. Without his supervision and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. I would like to thank you very much for your support and understanding over these past five years.

I would also like to show gratitude to my committee, including Dr. Chan Mun Choon, and Dr. Richard TB Ma. I discussed of the CCndnS Cache Policy with Dr. Chan Mun Choon during the weekly meeting and he raised many precious points in our discussion and I hope that I have managed to address several of them here. Dr. Richard TB Ma was my teacher for Advance Computer Networking course and his teaching style and enthusiasm for the topic made a strong impression on me and I have always carried positive memories of his classes with me.

My sincere thanks to Professor Mohan Kankanhalli. He was the one who believed in me and with his recommendation I could join SoC.

I would like to express my warm thanks to Professor Sarbazi Azad for not only supervising me during my Masters degree in Iran but also for being my life mentor. I know I always can trust his guidance and his friendship is extremely invaluable for me.

I thank School of computing and all staffs working there especially staffs in Deans office Ms. Loo Line Fong, Ms. Agnes Ang and Mr. Mark Christopher for helping me in several administrative matters.

Getting through my dissertation required more than academic support, and I have many, many people to thank for listening to and, at times, having to tolerate me over the past five years. I cannot begin to express my gratitude and appreciation for their friendship. I am extremely grateful to Mr. Saeid Montazeri, Dr. Padmanabha Venkatagiri, Dr. Xiangfa Guo, Dr. Shao Tao, Dr. Yuda Zhao, Mr. Nimantha Baranasuriya, Mr. Girisha Durrel De Silva, Mr. Kartik Sankaran, Mr. Mobashir Mohammad, Mr. Sajad Maghare. I have been unwavering in their personal and

professional support during the time I spent at the University. I must also thank all my friends from all over the world, Mr. Mohammad Olia, Dr. Ghasem and Sadegh Nobari, Dr. Hashem Hashemi Najaf-abadi, Dr. Hamed Kiani, Mr. Mohammad Reza Hosseini Farahabadi, Mr. Sasan Safaie, Mr. Hooman Shams Borhan, Mr. Amir Mortazavi, Dr. Abbas Eslami Kiasari. They always supported me in any circumstances. I would also like to thank all my flat mates during these five years. Dr. Mojtaba Ranjbar, Dr. Mohammadreza Keshtkaran, Mr. Hassan Amini, Mr. Mehdi Ranjbar, Dr. Hossein Eslami, Mr. Sai Sathyanarayan, for making our home warm and joyful and for your kind friendship and hospitality.

Most importantly, none of this could have happened without my family. To my parents and my adorable sisters it would be an understatement to say that, as a family, we have experienced some ups and downs in the past five years. This dissertation stands as a testament to your unconditional love and encouragement. My lovely fiance, who offered her encouragement through phone calls every day. With her own brand of humor and love, Mojgan Edalatnejad has been kind and supportive to me.

Contents

A CONTENT CACHING STRATEGY FOR NAMED DATA NETWORKING

by

Mostafa Rezazad

Submitted to the School of Computing
on 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The type of applications that Internet is being used for is completely different from what it was invented for. Whilst resource sharing was the first goal of networking, accessing huge data, such as multimedia files, is the main usage of the Internet now. The nature of multimedia content requires multicasting which is hard to provide in current point-to-point paradigm of TCP/IP protocol. In addition, concepts like mobility, security, efficiency, billing etc were not the first concern of designers of the Internet. That explains the recent movements toward designing a more efficient Internet which matches the current requirements.

Named Data Networking is one of the successful proposals that has received a lot of attention. By giving name to content, NDN enables in-network caching. However, efficiency of in-network caching has been questioned by experts. Therefore, in this thesis we propose a cache policy, CCndnS, which can increase the efficiency of in-network caching. The idea can be generalized to the domain of Content Networking but we analyzed our approach with NDN.

We realize that the source of inefficiency in a network of caches is the dependency between caches. To break the dependency, each cache regardless of its location in the network should receive independent set of requests. Without such characteristic, only misses of the downstream caches make their way to the upper caches. That filtering effect establishes a hidden dependency between neighboring caches. CCndnS breaks files into smaller segments and spreads them in the path between requesters and publishers. Requests for a segment skip searching intermediate caches to search only the cache with corresponding segment.

We present mathematical equations for cache and network hit rate when CCndnS is applied. We show that how CCndnS can simplify this task. The model can be used for further studies on cache performance or in a real application such as Service Level Agreement application.

Using CCndnS we suggest some techniques to improve the forwarding architecture of an NDN router for a better match with line speed throughput.

Performance of a cache can even be improved more with partitioning scheme. A dynamic partitioning scheme is presented in this thesis. The scheme can be used to enhance other features like fairness as well.

All ideas and proposed techniques are tested with an event-driven simulator that we implemented.

Thesis Supervisor: Y.C. TAY
Title: PROFESSOR

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Future Internet

The attraction of many useful Internet applications caused a huge growth of the Internet's user population and in the same way increasing the number of users make a great platform for new applications in different domains to emerge. Originally the Internet was designed for remote login and resource sharing. Nowadays the Internet is the main media for communication (VoIP, web conferencing), advertisement, business (eBay, PayPal, online banking), broadcasting (TV channels, streaming) and pleasure (on-line gaming, on-line gambling). Although, the user demand which defines the type of traffic has been changed completely, the architecture of the Internet still remains intact. The rigid point-to-point architecture of the Internet is not capable of supporting the requirements of current demands. Scalability, security, mobility, distribution (multicast, broadcast), etc are among the requirements that the current Internet poorly provides. The biggest debate among network professionals is whether the current architecture of the Internet will meet the aforementioned requirements subject to minor modification, or a completely new architecture should be devised.

The origin of the problems of the current transmission protocols (mainly TCP/IP) is at the design level. Many of the current requirements such as mobility, security,

billing, etc., were not an issue at the time when the protocols were designed. Shifting from one application domain (resource sharing at the beginning) to another application domain (multimedia recently), requires reconsidering the communication paradigm as well.

Considering the size of the Internet, a sudden replacement is impossible. Hence, middle-boxes have been largely used to keep the main architecture intact. Middle-boxes have been used for different demands:

- Enhance functionality (e.g., firewall).
- Overcome shortages (e.g., NAT for IP shortage).
- Filter redundant traffic for bandwidth efficiency (e.g., web proxies).

In addition to the middle-boxes, distributed systems started to come in the picture to enhance the performance of the Internet. Ideas such as, Distributed Web Proxy Caching [15] [26] [70], Peer-to-Peer (P2P) systems, Content-distribute servers like Google servers and Akamai [60] are good examples. Although these ingenious ideas are admirable for making the Internet more stable for the current usage, it seems that the Internet is reaching its limit and no application layer remedy would be able to scale the Internet for future demands. So a new movement has emerged to find out the best architecture for the Future Internet applications. Some proposals are trying to suggest a completely new architecture whilst others are aiming to push down the solutions to the lower layer of the current architecture.

Among the new proposals, we will focus specifically on NDN (Named Data Networking) since it promises for incremental deployment and facilitate all aforementioned requirements. The NDN proposal claims that it is not a clean-slate approach but it only pushes down all the available solutions from application layer into the lower layers.

The main common characteristic of NDN and all other content-oriented protocols are their new perspective toward the communication model which is content based

communication. However, they are different in interpretations and implementations. Based on content-oriented model, network will only focus on content not the location of the content. Location based communication is the current communication paradigm which is named Client/Server paradigm. In this model a client needs to find the location of the content first and then asks for the desired content. In other words there is a one-to-one map between content and the location of the content. This paradigm works well for resource sharing which was the main intention of the Internet at the beginning. By growing the size of content and the number of content producers, the current communication paradigm fails to handle the traffic. With aid of application layer solutions like Content Delivery Networks (CDN) ISPs try to localize the traffic. CDNs bring content closer to the clients by caching them in different regions. Akamai reported that it handles 15-30 % of web traffic everyday[1]. Google claims that it handles 25 % of all web traffic in North-America[2]. However, CDN is not for all applications, it is not free, it is not part of the network architecture and it is very complicated.

Statistics[3] show huge growth in caching market in the near future. In-network caches effectively bring content closer to the consumers and save considerable amount of time and bandwidth. In fact, future Internet will use memory and processing power to increase the good-put of the network unlike the traditional networking which was designed to increase the utilization of computational resources.

Using content based communication paradigm at the heart of NDN proposal makes it a highly competent network. In short, NDN creates new opportunities like:

- **Application friendly:** although applications are interested in data, the current communication model forces them to look first for the location of the data.

[1] $http://www.akamai.com/html/about/facts_figures.html$

[2] $http://www.datacenterdynamics.com/focus/archive/2013/07/google-dominates-internet-traffic-figures$

[3] $http://www.prnewswire.com/news-releases/global-transparent-caching-market-2014-2018-key-vendors-are-blue-coat-systems-juniper-networks-peerapp-and-qwilt-275100511.html$

To simplify the task, application-specific middleware has been used to map the location to data (like DNS). Since NDN names data, applications can ignore all the middleware complications by immediate addressing Data chunk; the smallest addressable unit of data is called *Data chunk*.

- **Security:** many attacks like phishing emerged from the location-based security pitfall. Securing only the communication channels and the devices in between two IP machines is not an adequate security policy for an end-to-end communication model. By providing security at the chunk level, NDN not only offers better security but also ends all the location-based security difficulties.

- **Broadcast friendly:** Data chunks in NDN network can be broadcasted easily without any concern about looping. Moreover, requests can be aggregated at the routers to increase bandwidth efficiency. This property of NDN makes it a preferable platform for streaming applications (especially live streaming).

- **Mobility friendly:** finding the location of mobile content or requester in an IP network is quite challenging (which is not a concern in NDN). In addition, caching contents inside all communication devices (including switches and routers) simplifies mobility algorithm for NDN. In case of packet lost (which happens a lot in wireless communication) or moving a requester from one base to another, the requested Data can be found from a closer device by the resented request.

However, this paradigm shifting from location to content networking involves rigorous research on many subjects. One of the major obstacles is memory access latency and performance of in-network caching.

Considering the diversity of the traffic passing through a router, to have an efficient caching system the size of the memory should be relatively large. However, fast memory technologies are still not inexpensive. Slower memory technology increases the packet processing latency at a router which increases the risk of incompatibility

of routers' speed with line speed. Therefore, solutions like better lookup algorithms, efficient memory architecture and caching policies could be a more realistic way of dealing with the memory problem.

Considering memory system as the main obstacle of an NDN router, in this thesis we mainly focused on a cache policy which can be used to enhance the efficiency of in-network caching and increase the throughput of router's memory architecture.

1.2 NDN Architecture

An NDN router contains three main tables, *Forwarding Information Base (FIB)*, *Pending Interest Table (PIT)* and *Content Store (CS)*. The abstract version of the forwarding plane is illustrated in Figure 1-1. The functionality of the **FIB** table is similar to the IP routing table. Information of which data can be accessed from which interfaces is provided from FIB. The **PIT** provides the reverse path information for the data to send back to the requester(s). When a data packet arrives to a node it is sent out to all interfaces obtained from the PIT. It also aggregates all the received requests for the same content and forwards only one request to the upstream node. This strategy prevents inefficient usage of bandwidth. The **CS** keeps a copy of the data chunks for a short period of time (subject to replacement policy and incoming traffic) to satisfy future requests for the same data chunk.

As communication in NDN is receiver driven, users need to request every Data chunk of an object individually. A request which is named *Interest packet* contains the Data chunk name (as the address). If an Interest packet is satisfied by a node (from CS of an intermediate node or from the data base of the publisher), a Data packet contains Data chunk, the name of the data and the signature of the data will be sent back to the requester(s).

The footprint of an Interest packet in a router is as follows:

- *Search on CS-index table*, the result of this search determines presence of a

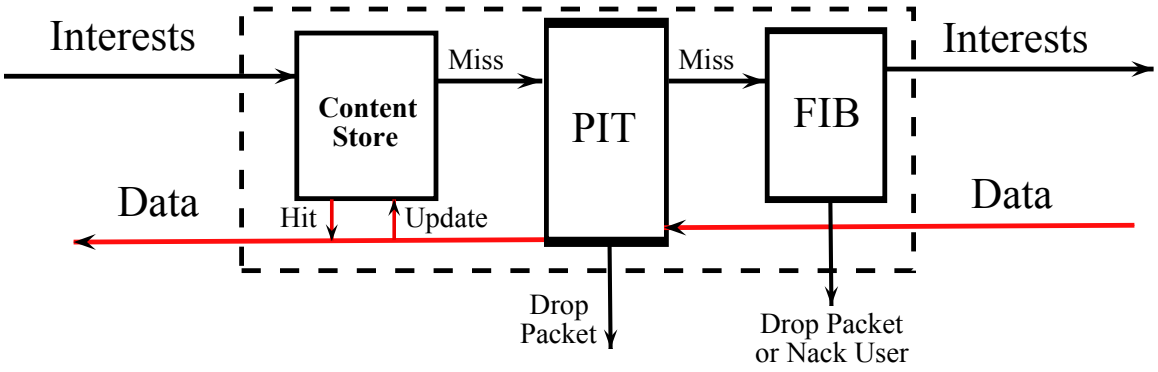


Figure 1-1: The forwarding plane of an NDN router which consists of CS, PIT and FIB.

Data chunk in the CS. In case of hit, the Interest packet will be dropped and retrieved Data chunk from CS will be sent back in a Data packet. Otherwise the Interest packet will be sent to the PIT.

- *Search PIT*, for the same entry (in terms of content address) to prevent sending duplicate request of the same content (which wastes bandwidth). In case of PIT hit, PIT entry will be updated with the new interface address and Interest packet will be dropped. In case of PIT miss, Interest packet will be forwarded to FIB table after the entrance interface of the packet is registered in PIT.

- *Find Longest Prefix Match (LPM)*, before the Interest packet can be searched on FIB, the LPM of the address should be found. This can be done by a few bloom filters. LPM will be searched against FIB entries later.

- *Find the proper Interface from FIB*, after all in the last step the Interest packet finds the Interface(s) from FIB or in case of FIB miss from the strategy plan, and leaves the router. Interest packet can be dropped or a Nacked the client by rules defined by the administrator of the network.

Interest packets can be dropped at different occasions:

- Behind a long queue to search CS.

- Timer expires when waiting for Data chunk at PIT.
- When there is not a free space in PIT.
- When there is the same entry in PIT for the same Data chunk.
- Because of congestion at the out links (determined by FIB).

The other important features of NDN architecture are as follows:

- *Hierarchical Naming*, two important features of the hierarchical naming system are: capability of applying LPM on the names and it is easier to comprehend content from its name. Routing table size is one problematic issue for a named content networking. LPM helps to rectify this problem by shrinking the needed address part for routing. In addition, address part of a packet can carry useful information for users and applications. Address can provide information like the location of the content, the version of the application or data, sequence of the packet, etc. None of these information and features can be easily applied using flat naming system. Notice that the names should be unique in their domain, requesting area. If the domain of a name is as small as a campus, it should be unique on that campus. Globally used names should be unique through entire globe.
- *Security*, which is built into data itself rather than being a function of *where* or *how* it is obtained [41]. Data packets carry the signature of the publisher for each piece of data. The security (signature) field (unlike in a TCP/IP packet) is not optional but compulsory. Since the location of the data is not important and it is not known, the signature provides sufficient information to determine data provenance.
- *Routing* can be done in a similar fashion to today's IP routing. To maintain FIB table any routing protocol like BGP or OSPF can be used. An NDN router

and NDN publisher employs announcement mechanism to update the routing tables. The accuracy of the FIB table depends on the scope of the routing algorithm. So it is possible that a router does not have any entries for a specific data name. Predefined strategies by the administrator would be applied to such cases. Depending on the strategy, an NDN router might broadcast the Interest to all interfaces or multicast it to a group of interfaces based on their performance. Then by receiving the feedback (by receiving the first data chunk) the router can automatically add a new entry to its FIB table by measuring the performance of different interfaces.

All aspects of NDN architecture are still considered as open problems and subject for active research.

1.3 Our Contribution

Our main contribution is to enhance the functionality of in-network caching, specifically for NDN, by introducing CCndnS [69] cache policy and developing a partitioning scheme for it. Contributions are listed below:

- CCndnS is a cache policy to increase the performance of in-network caching and reducing the load of the routers. The objective is devising a cache policy which requires less meta data or coordination to make it possible to be implemented in practice.

- Build a mathematical model to study CCndnS.

- We show how the equation for CCndnS can be used in a real application like Service Level Agreement (SLA).

- Introducing cache miss equation partitioning scheme for further improvement of a single cache in a network of caches.

- Study the performance of NDN forwarding plane and suggest to use parallel search on memories with CCndnS instead of pipelined design.

1.4 Thesis Organization

Literature review related to our proposal is provided in Chapter 2. Studies show that *cache policy* is the key element of in-network caching enhancement. We look at different proposed cache policies and pros and cons of some of the influential cache policies. In addition, we review some studies on how to match the memory access delay of IP routers with line speed. That is the motivation on pursuing research on architecture of NDN routers.

Our main contribution which is a cache policy, CCndnS, is given in Chapter 3. The algorithm of how CCndnS can break files into segments and spread them on the path along with the CS skipping scheme are given in this Chapter. We see how CCndnS can reduce the content dependency between nodes in the network of caches and that makes the process of modeling the cache hit, straightforward. The mathematical model is presented in the Chapter along with the simulation experiments to evaluate and compare the performance of our scheme with other similar approaches.

The given equation in Chapter 3 can be applied on various applications and studies. As a proof of our claim, the equation is used in a real SLA application in Chapter 4. We can offer QoS (Quality of Service) by making cache reservation for a particular customer and estimate the cost of such agreement using our cache hit equation.

To increase memory manageability, in Chapter 5, a dynamic partitioning scheme is provided using a cache miss equation. The cache miss equation has been successfully applied on various applications before and in this Chapter we show that the equation can be applied on in-network caches as well.

Our last contribution is some suggestions on NDN router architecture and is pre-

sented in Chapter 6. We claim that using CCndnS with a parallel search on PIT and FIB can significantly reduce the forwarding latency of NDN.

Conclude our thesis with some possible future work in Chapter 7.

Chapter 2

Related Work

Since CCN has been proposed by Van Jacobson [41], many related articles have been published covering different parts of the architecture. Here we cover some of the most related ones to our research and those which provide important insights into Information-Centric Networking (ICN) [97].

The idea of addressing data by name can be traced back to TRIAD [16], and has many other forms. These alternative proposals are generally classified under ICN. Giving name to content rather than location changes the networking paradigm from point-to-point communication to publish/subscribe communication. That enables caching at the network devices. These two major changes open many new topics to research in networking field. The open problems can be categorized into six main groups [105]:

- ***Naming***- Although there was a debate between hierarchical structured naming [41] and flat naming [29, 77], most of the proposed naming structures are based on hierarchical structure [58, 98]. The most important challenges for naming are [105]:

  1. Length of the names. That affects routing algorithm and table sizes.
  2. An efficient and deterministic algorithm to reach the same name at both

producer and consumer side using the available information about the content.

3. How to make sure that names are unique.
4. How to find a name for a particular content.

- ***Security***- NDN secures each and all Data packets by cryptographically sign them. Although it is claimed that this is the most secure system, network performance wise signing all data packets is not very efficient.

- ***Application***- This has been the major playground of the inventors of NDN. By developing many different kinds of applications, they have tried to find out the major open problems in the area. Applications such as Video streaming [45], real-time conferencing [103, 107], building automation system (BMS) [74], vehicular networking [31], etc., have been developed and tested so far. They are still encouraging developers to work on different type of applications.

- ***Routing***- Hierarchical naming structure enables NDN to use the current routing algorithm. OSPFN is an extension of OSPF to advertise the names [46]. The current NDN routing protocol, NLSR, can name even physical entities like routers and links [35]. NLSR can use any underlying communication channel to exchange routing information.

- ***Forwarding***- NDN poses a completely different forwarding scheme than an IP router. That makes forwarding a brand new research area for NDN. Scalability and supporting wire-speed operations on memory lookups for variable names are among the most challenging issues. There have been some efforts to cope with these problems especially for FIB table processing. FIB is considered as the largest table that holds a huge amount of routing information (i.e., names and paths) [78,87,101,102]. There are many papers focusing on this issue which they mainly try to introduce better compressing algorithm to reduce the size

of information in FIB or using architectural solutions like using fast memory for popular contents and slow memory for others to maintain both performance and cost [19, 37, 58, 66].

- ***Caching***- Recently Ager et al., [2], by analyzing the Internet traffic, observed that a large portion of traffic is cacheable. The cacheable traffic mostly consists of software downloads and traffic coming from CDNs. In a similar study Anand in [4] stated that they could save the campus bandwidth up to 75-90% by using a middleware redundant traffic eliminator device. These studies prove the importance of caching elements to reduce the inefficient usage of bandwidth and the load on the servers, and increase the quality that users perceive. Although NDN community considers caching as an optional feature of NDN (because of its technical issues when it is used at router level and need to match with line speed), there are still many on-going researches to improve the performance of the in-network caching.

Apart from those aforementioned research topics, there are many other papers written about other aspects of NDN such as:

- window-based flow control [9],
- QoS-aware path selection [44],
- impact of traffic mix [25],
- mobile ad hoc networks [54], etc.

In this thesis, we mainly focus on two aspects of the NDN which raised some skepticism over whether ICN in general, and NDN in particular, are fundamentally feasible. We particularly look at usefulness of in-network caching and the forwarding architecture of NDN.

2.1 Caching

Although, cache is optional in the newest version of NDN proposal[1], still considerable amount of studies in this domain targeted in-network caching issues. Among them, there are some which challenge the efficiency of in-network caching. For Ghodsi et al., ICN's in-network caching is futile because most hits will occur only at the edge, where the popular files are cached [30]. Using LRU replacement policy with Leave Copy Everywhere (LCE) cache policy, they found that having a very big cache (equal to the total sum of all caches in the network) at the edge of the network is sufficient to receive almost all benefits from in-network caching. We will show that their finding is actually valid for their assumptions. Based on our findings, in-network caching is truly futile without a proper cache policy.

D. Rossi in [72] provides a comprehensive simulation study considering wide range of parameters like different replacement policies, range of content popularities, different catalog sizes, cache policies, topologies, etc. They emphasized on the importance of cache polices for the performance of a network of caches. Cache policy is addressing the question of which content should be cached where. There are different opinions on whether more content should be cached at edge routers or at core routers. For Rossi in [73] caching at core (giving more memory space for caches at the core) results in a better performance. Opposite results are reported in [24, 63] where a larger memory size suggested to be assigned at the edge routers.

Perino and Varvello's calculations show that memory technology (e.g. size and speed) cannot match with line speed at reasonable cost for NDN [62]. They argue that memory lookup is not affordable at core level (assuming core routers have higher traffic density) when lookups most likely result in cache misses. Although their analysis looks sound, their assumptions are too rigid. Firstly, they did not consider any traffic elimination from lower level caches (no cache hit at the lower level caches). Secondly, they did not make use of any proper cache policy in their evaluation.

[1] $http://named-data.net/project/annual-progress-summaries/2013-2014/$

Using default caching policy LCE (Leave Copy Everywhere), caches at the edge of the network will get most of the hits and act as a filter. This effect is most obvious for hierarchical systems; it is known for web caching [14], and recently observed for CCN [42, 64]: Caches at the edge are dominated by popular files, while those in the core are occupied by less popular or redundant copies of the popular content that get few hits. This is largely why Fayazbakhsh et al. believe that most of the benefit of in-network caching for ICN can be obtained with TCP/IP by putting all the caches at the edge [24]

On the other hand, Tyson et al.'s simulations with BitTorrent traces show that, if cache sizes are sufficiently large, then there is significant caching in Tier-2 Autonomous Systems using CCN instead of TCP/IP [86].

We group cache policies into two categories: cooperative caching and algorithmic coordination.

2.1.1 Cooperative caching

In-network caching resembles cooperative web caching. Several papers have proposed protocols for such cooperation [20, 23, 55, 65, 94], but Wolman et al.'s analysis shows that they have marginal benefits for large populations [92].

However, that analysis uses traffic data from some 16 years ago, when the Web was quite different from today. In the case of on-demand TV, for example, Li and Simon recently show that their cooperative caching strategy can significantly reduce cross-domain traffic [50].

Nevertheless, all cooperative caching policies require extra traffic to the network for the updates between the peers. For instance in [94] a Content Router (CR) broadcast the content of its Bloomfilter, contains the name of the cached content, to its neighbors. The aim of cooperation among in-network caches is to increase the efficiency of caches by reducing redundancy of cached content in several nodes. Cooperative caching usually enables off-path routing too. Off-path routing refers

to a routing algorithm in which Interest packets can be forwarded toward a closer cache with content rather than further source of the content. Cooperation adds extra complexity into the routers to measure or collect some information (such as counters to measure content popularity) and to aggregate the received information from other routers.

A cooperative caching scheme is called coordinated caching if a router redirects a Data packet to a specific direction to be cached there. Coordination can be done at the publisher of the content as well by adding extra headers to the packet. Notice that coordination happens to Data packets while off-path routing operates on Interest packets.

Some of the newly introduced cooperative or collaborative cache policies designed for NDN are [21, 49, 93, 95, 96]. Li et al. in [49] proposed an inter-ISP coordinated caching scheme where content selectively get cached in routers. In order to make a better decision each router maintains a list of counters which calculates the popularity of some selected content. In each interval, routers share their measured popularity with others. Although the algorithm might increase the efficiency of the cache, it imposes extra traffic on the network to exchange the meta information (the volume of the traffic depends on the frequency of the intervals). Their algorithm also requires extra memory space to store counters.

The idea in [49] considers only one gateway in an ISP. Later they improved their mechanism to have multiple gateways and multi-path routing in an ISP [95].

Tecc [96] is a way to decouple traffic engineering and collaborative caching. That gives freedom to the administrator to manage different network resources.

Dong et al. [21] presented a localized cache policy where the decision making on what to cache takes place in a router and it is independent from other nodes. Nevertheless they still need to broadcast their index of caches by each update on the caches. They formulated the replacement policy as an optimization problem to minimize the overhead of the network.

Walter et al. in [93] are using a kind of skipping scheme. In their probabilistic scheme, passing traffic through a router can be cached at the router based on a probability. When a router decides not to cache some content, it keeps track of the interface that the content is forwarded to. When an Interest packet arrives for the same content, it will be forwarded to outgoing interfaces that the corresponding Data chunk had left before.

2.1.2 Algorithmic cache policies

In an algorithmic cache policy, nodes make decision without using or having information from other nodes. Thus they do not need to exchange their cache indices. Index dissemination is a costly process especially for in-network caching where the life time of content in a cache is very short.

Algorithmic cache policy which works well for en-route caching, is mainly aiming to increase the efficiency of network of caches by eliminating or reducing content redundancy between caches. One reason en-route caching strategy leaves the core caches cold lies in the lack of cache diversity, i.e. the core contains copies of the content at the edge. This redundancy can be reduced by the two major algorithmic cache policies LCD and MCD considered by Laoutaris et al. for hierarchical web caches [47, 48]. In both algorithms a router will cache content if content is from the memory of the upstream cache. In other words, if a cache gets hit for a content, the content will be cached at the downstream router. The difference between the two algorithm is that LCD just gives a copy to the downstream router whilst MCD moves the content completely to the downstream router by removing it from the upstream cache. Although both LCD and MCD are very effective for edge routers (only popular files reach the edge), other intermediate routers do not get much benefit from these cache policies.

Another possibility is for a router to probabilistically decide whether to cache a chunk that is passing through, i.e. a randomized way of reducing redundancy [63].

WAVE is another algorithmic cache policy to reduce the cache pollution by caching only popular content at the edge router [18]. The behavior and the result of WAVE is very similar to MCD. Unlike MCD which has a window size equal to one Data chunk, WAVE has a variable window size and it increases exponentially. That means for each hit at a router, the number of chunks that will be cached at the downstream routers will be doubled for WAVE.

As another effort to increase the life span of content in caches, in [90] a three level caching scheme is proposed in which contents first cached in the lowest transient cache. They may promoted to higher level caches by their popularity (pre-defined number of cache hit) or demoted back to the lower level after a certain period of time without any hit.

Chai in [13] argued that maybe caching less data would be better. By caching less data they want to increase the resident time of data chunks in a cache. In their algorithm, a Data chunk caches at a router with probability p. When the Data chunk is cached in a node the probability p will be set to a very small number. That reduces the chance of the Data chunk to be cached at the next immediate node (connected neighbor). Thus it is unlikely that a Data chunk will be cached at two connected routers. However, the value of p increases hop-by-hop until it gets cached again somewhere further from the first caching node.

CCndnS [69] is an algorithmic cache policy which does not use any information from other caches in the network but still reduces redundancy. We show in Chapter 3 that CCndnS is indeed effective to improve the efficiency of both edge and core routers. CCndnS can be considered as a coordinated cache policy (location of cache for each Data chunk is determined by the source) but it definitely is not a cooperative policy. Like the cache less idea of Chai, CCndnS prolongs the life time of cached content by selectively caching content.

2.2 Cache Management

Storage management in the context of content routers can be translated to cache management. Storage or cache management might have positive effect on the caching system. Carofiglio studied this issue by comparing the static storage partitioning with shared storage for NDN routers. They found that storage partitioning can reduce cache misses whilst it increases bandwidth usage [12]. They argued that their proposed dynamic partitioning can improve the performance of cache by increasing its utilization with slightly more bandwidth usage than shared storage (but better than static partitioning).

Storage partitioning is not a new concept. Lu presented a dynamic partitioning for web caches in [51]. However, monitoring based algorithm is too costly for a chunk level caching. We propose dynamic cache partitioning based on an equation in Chapter 5. In addition, the equations in Chapter 4 are used to introduce a mechanism for estimating the effect of memory reservation for a specific content on the performance of the whole cache or on the other content in the shared part of the cache.

2.3 Cache hit equation

The mechanism we use to evaluate the effect of Service Level Agreement (SLA) on the cache performance is based on a cache hit equation. Modeling a cache in a network of caches has far more complexity than modeling a single stand alone cache. Carofiglio presented a model to study the behavior of a single cache hit ratio using LRU replacement policy for a chunk level caching system where content retrieval is receiver-driven [10]. The filtering effect that makes en-route caching ineffective also makes the analytical modeling of a caching network "extremely difficult" [71] and "far beyond the borders of tractability" [47]. This is why many models consider only hierarchical topologies [42, 56, 64].

A mathematical model can facilitate studying different aspects of a new concept.

For instance, in [91] the equation is used to study the optimum cache placement in the network. They mapped their placement problem to the knapsack problem to consider all possible ways of cache placement. This can be considered as another proof of difficulty of modeling the cache hit rate as they preferred to choose a very time consuming algorithm rather than model the cache hit rate.

We realized that one of the sources of the complexity is the correlation among the routers in caching content. By caching content at a granularity smaller than file size and larger than Data chunk size, CCndnS facilitates skipping, diminishes the filtering effect, and thus decouples the behavior among caches. We can therefore construct a very simple model (Sec. 3.3) that can be used to predict and analyze individual and aggregated router performance.

2.4 Router architecture

Apart from the efficiency problem of in-network caching, another question is whether NDN router architecture is good enough to handle line speed traffic rate. Part of the problem is related to the architecture of the cache and the cache policy. Memory technology is not ready to facilitate in-line caching. Updating and retrieving data from memory requires multiple access to memory (depending on the lookup algorithm) which might induce a queue of waiting packets to look up the cache.

Arianfar et al. see probabilistic caching as a load-sharing strategy, having pointed out the difficulty for memory latency to match line speed [5]. Chai et al. also note that the need for an NDN router to work at line speed rules out any collaborative caching algorithms that require significant information exchange among the routers [13]. In another similar research, Perino and Varvello check the feasibility of implementing an NDN router considering the current available software and hardware technologies [62]. They concluded that today's technology is not ready to support an Internet scale NDN deployment, whereas a CDN or ISP scale can be easily afforded. Narayanan

and Oran, for example, calculated that an NDN routing table would have to be orders of magnitude larger than current tables [57]. Both issues, and more, were raised by Ghodsi et al. [30]. Our proposed CCndnS not only can increase the efficiency of the cache, but also rectify the architectural problem by not caching everything at a router and skipping cache lookup for some of the Interests.

Apart from cache lookup, NDN routers require at least another two memory units in its forwarding architecture. Yuan et al. in [102] studied the key components of a scalable forwarding plane using CCNx[2] as their platform. They identified that the delay of CCNx lookups cannot cope with the incoming rate into the routers considering the current link throughput. Many researches have been conducted to overcome the problem. Yi proposed a GPU based approach for name lookups to make FIB lookup faster and by some extra techniques make FIB smaller [99]. Considering PIT as the key component to reach wire-speed, Varvello studied different placement of PIT table at egress or ingress interfaces and finally proposed their third party solution to speed up PIT look up [53]. Hwang proposed a multi level memory architecture mainly for FIB to maintain the acceptable lookup performance whilst keeping the memory cost reasonable [38].

However, cache is not the only difficulty of the router architecture to cope with the line speed processing. The current IP routers requires only one memory lookup on routing table and still many ideas have been proposed to match the forwarding latency with the line speed. Aweya in [6] provides a thorough survey on routing lookup difficulties and algorithm. Although it might be argued that memory technology is advancing and better lookup algorithm might come to the picture, the networking equipment and link bandwidth are advancing rapidly as well.

Similar studies show the memory lookup as the main bottleneck of IP routers and Many of them suggested software or hardware solutions to ease the problem [7, 22, 32, 59, 61, 79, 88, 104].

[2] $http://www.ccnx.org/$

The goal in [61] is to process 32 million packets per second, meaning a router can only spend 30 nanoseconds on one packet. Current router can achieve higher bandwidth (Cisco CRS-3-1 supports 100Gbps interfaces); however, the memory technology has not improved much. Still the network processor needs to access SRAM or DRAM. Definitely DRAM with an average 50 nanosecond delay cannot be a proper choice. The paper assumes that tables can be fit in the SRAM cache memory inside the processor. However, in case of NDN, the size of tables might prevent us of using expensive SRAM memory.

The other approaches work on the memory architecture to tailor it for a particular traffic and router design [28, 34]. For instance, as an effort to match the throughput of DRAM memory with Network Processor Hasan, et all., [34] tried to increase the locality of data in DRAM to reduce row miss penalty. According to the paper, multithreading and multiple engines to process multiple packets in parallel can hide memory latency and increase system throughput. However, packets need to be written and read from buffers. DRAM implementation of buffers (more cost effective technology) needs to provide a high bandwidth communication to compensate its latency and feed in enough Data to the processors. Their solution is to get as much data as possible by reading a row of a DRAM. Accessing data blocks in a row of DARM is much faster than random access for the same data blocks.

Buffers not only used to store routing table (index table) but also to buffer incoming packets from links. There is a tendency to increase the size of memory buffers to reduce the risk of packet drop. However, huge SRAM memory is not affordable. To match the buffer latency with high speed links (even 10Gbps) a hybrid memory architecture consisting of SRAM and DRAM is proposed in [40]. The paper and the extended versions of it [52], [27] try to overlap the delay of reading and writing from and to DRAM by overlapping the delays of different memory banks. One small SRAM memory is receiving packets from the link and write them to different bank of DRAM memories.

Throughput matching appears an ongoing problem as line card and processor throughput increase much faster than memory throughput. The latest memory technology used in the recent routers (e.g., Cisco CRS-3-1) is DDR3 which is a SDRAM memory using a special interface which can reduce the access latency to 10 nanosecond. This memory technology and probably very complicated and specialized interface for it to match the 100Gbps of the router's interfaces makes the cost of the router more than a million dollars [3].

The update procedure of the routing table might be troublesome as well. However, updates are less challenging for IP routers compared to NDN routers with three tables. In an IP router not only there is only one index table (forwarding table), but also the update of the table can be decoupled from searching the table. The two mechanisms of updating FIB table in an IP router which might be used for NDN are:

- Using two or more memory banks to perform the lookup algorithm on one and update on another [61] [32]. The idea needs to maintain two copies of the FIB table.

- The second idea is using a special hardware in help the router's processor to update the current FIB table without interfering with FIB lookups [32].

However, both schemes might be troublesome for NDN routers as memory space and processor cycle both are very scarce resources on NDN. Considering the size complexity of FIB table (because of the name length in NDN and number of possible content), adding an extra FIB might not be affordable. Also processing three tables and security check for all packets (we do not go into the details of the security in this study) put a huge load on the processors in an NDN router and makes the second approach infeasible as well.

CS-index table, same as FIB table, would not be updated by Interest packets. This index table only receives updates on arrival of new Data chunks (but not for a

[3] $http://www.globalpricelists.com/globalpricelistcisco.php?groopa = 536$

duplicate Data chunk received from another interface at the same time). This means CS-index should be searched for the Data before it can be updated. Another concern is that based on the cache policy (e.g., CCndnS, WAVE, Probabilistic caching, etc) CS-index might not get updated by all Data packets.

PIT table needs to be updated with both Interest and Data packets. For both packet types, PIT needs to be searched first. For Data packets that is founded entry, needs to be deleted from the table but for Interest packet either a completely new entry needs to be added to the table or the entry needs to be updated by adding the new interface address. Unlike CS-index, PIT will get updated for every Interest and Data packet.

Thus, each table introduces different number of memory access and delay in dealing with updates; making the process of router design even more challenging to handle the throughput problem.

Pipeline design not only is hard to implement for each individual table, because of unknown name length and memory lookup algorithms, but also it is difficult to maintain it among the three memory units as their nondeterministic behavior in dealing with different packets.

- CS-index and PIT need exact much for indexing and FIB needs LPM. This means there should be at least two different types of data structures which have different lookup time complexity. Not only different implementation or data structures might be the source of the variable delay, variable name length results in variable delay as well. It is shown in [102] that forwarding delay of CCNx (the software prototype of CCN architecture [4]) strongly depends on the length of the names.

- Packets can be dropped at any stage. An Interest packet can find data from cache and be replaced with the corresponding Data packet or the Interest packet can be aggregated with another packet in PIT and get dropped.

[4] www.ccnx.org

- As discussed earlier another source of nondeterminism is how the router treats Interest and Data packets differently (which is not the case for IP router).

Therefore, memory access latency of the current NDN design is very much non-deterministic. We conclude that making NDN router pipelined is hardly possible.

The above arguments are stating that the proposed forwarding pipeline in the last update of NDN project [3] might not that effective. NDN forwarding pipeline consists of many decision points in between pipes which might reduce the throughput of the bottlenecked component.

Hence, we introduce our parallel architecture to resolve the problem [68]. In our parallel design all memory units always have something to process even if the output of the unit will not be used. We are trying to always keep the bottlenecked unit busy.

Using CCndnS, there would be no meta traffic for caching purpose between routers. The Interests and Data chunks at most carry a few extra bytes to locate the router to cache (for Data chunk) or to search (for Interest packet). Nonetheless, it spreads content so the load for checking CS is spread out. This reduces queuing delays at the CS and significantly postpones router saturation [68]. The redundancy reduction ensures that caching capacity in the core is not wasted. CCndnS thus partly addresses the line speed and edge/core issues raised by Perino and Varvello, and Ghodsi et al.

2.5 Discussion

In this thesis, we make no effort to examine whether NDN is feasible. Rather, the question we pose is: How can we improve caching for NDN?

We add granularity of segment in the system and propose a cache policy that caches content not at chunk or file level but at segment level. Therefore, routers can cache different parts of a file or files can be distributed among different nodes. Doing that, we eliminate the effect of edge router filtration of popular content. The cache

policy (CCndnS) improves the performance of the network of caches. Additionally, CCndnS helps to reduce the forwarding delay of the routers to match the processing delay of the router with the line speed. Moreover, CCndnS makes modeling the cache hit rate very straightforward. The model can be used in an application such as SLA or to study other aspects of NDN.

Chapter 3

CCndnS

As mentioned earlier, content naming enables caching at router level. However, based on the newly revised version of NDN prototype, caching at routers is optional (i.e., a router might or might not cache content). The main reason for such modification is the complex forwarding algorithm of NDN architecture. There is some skepticisms that a NDN router can cope with the current line throughput [5,30,57,62]. To reduce the processing burden, it is decided to make caching optional.

To maximize the performance of a network of cache, it is crucial to have an efficient cache policy. Although the aim of an efficient cache policy for a network of cache is to improve the network cache hit rate, from routers' perspective an efficient cache policy is the one that is simple enough for implementation and fast enough to not add extra processing delay to the router. In fact, it appeared that the overhead of the cache policy is the more important parameter and the inventors of NDN decided to choose the simplest possible cache policy which is cache everything everywhere.

Although it seems it is the easiest and fastest way to process an Interest, performance wise caching everything everywhere is not efficient. there are several issues with this idea:

(I1) A chunk may be cached at multiple routers, and routers may cache unpopular chunks that are rarely hit. Both are a waste of caching capacity.

(I2) The number of content files in the Internet is huge, and many are large (e.g. movies). To hold a significant fraction of these files, the caches will also have to be large, and therefore use cheaper and slower technology (e.g. DRAM instead of SRAM) [62]. The need to access this memory (to check if Data for an Interest is there, or to insert a chunk into the cache) thus slows down the router. This delay can increase exponentially if link rates are high and queues build up.

(I3) A router that caches some popular Data will filter out Interests for this Data, and forward Interests for unpopular Data to upstream routers. If content popularity has a long tail, then chances are slim that the filtered Interests will find their Data at some intermediate router before the source [30]. Routers in the **core** of the network thus cache less popular content (I1). The resulting imbalance in workload degrades performance.

(I4) Filtering (I3) causes a cache to affect its upstream caches. This coupling makes it difficult to construct a mathematical model for analyzing the performance of the network.

Hence, we propose **CCndnS** [69], a content caching strategy for NDN that is a modification of **CCndn** [68]. CCndnS not only improves the performance of the network of caches but also it requires very small processing at routers and the most of the process happens at the requester and publisher sides. CCndnS divides a file into **segments**, where a segment is larger than a chunk and smaller than a file. A data chunk is the smallest unit of data which receives a name. Therefore, a segment is a smallest unit of content which consist of group of data chunks that are cached together in the same router, but different segments may be cached at different routers. A file is thus spread across multiple routers. CCndnS addresses the above issues in the following ways:

(I′1) The caching strategy reduces the number of redundant copies of a chunk. By

spreading popular files among routers, CCndnS also reduces the cache space taken by unpopular files.

(I′2) It allows an Interest to **skip** a cache, i.e. pass through a router without checking if its Data is there. This reduces cache misses and shortens memory queues, thus reducing router latency.

(I′3) It increases hit rates for caches in the core, so they are better utilized, and balances the workload between caches in the core and at the edge.

(I′4) Skipping decouples the dependency among caches. This makes it possible to model aggregate behavior of the network with some simple equations.

We claim that the overhead of CCndnS is negligible. Although CCndnS requires some extra fields on the packet headers which is relatively small, the extra information compensates processing overhead. Later it is shown that the most of the processes will be done by the clients and/or by the publisher of the content. A router merely needs an extra simple comparator to compare two small integers. That can be simply implemented in a hardware.

This Chapter begins by describing how CCndn spreads a file. Section 3.2 then describes how Interests can skip router checks under CCndnS. The model for CCndnS is derived in Section 3.3. Along the way, we present simulation results for a realistic topology to examine the caching strategy and validate the model.

3.1 CCndn: Spreading Content

In most cache policies like LCE or LCD, this is only the edge router which receives most of the requests for the popular files. That is because the most popular file are cached at the edge of the network entirely.

So intuition behind segmentation of CCndn is to give an equal chance to the routers other than edge routers to receive requests for some part of the popular files.

At first, this idea might not look effective as the popular file will be cached in further routers and downloading the entire file might take longer time. However, there are two main reasons to support such decision:

- First, caching a portion of a popular file gives a chance to other popular files to get closer to the clients. In other words, if there are many popular files in an area, a subset (few segments) of each of them can be found in an edge router.
- Second, mainly a large and popular file which worth to cache is a multimedia file. For a multimedia file, the time to consume the first segment of the file which is closer to the consumer can be overlapped with the download time of the rest of the file which is cached inside the network.

To fully exploit the idea, we first describe CCndn in Section 3.1.1, then present results from experiments in Section 3.1.2.

3.1.1 CCndn: Description

In NDN, clients request content by sending an Interest that explicitly addresses a Data chunk of a content file The content may be an email, a voice call, etc., but, for convenience, we refer to them as **files**.

Each NDN router has a **content store** (CS) for caching Data chunks. (we assume all routers participate in caching). CCndn is a content caching policy that specifies which router should cache a chunk and how chunks should be replaced.

By default, NDN caches a chunk at every router that the chunk visits [41]. This redundancy wastes caching capacity, especially when CS size is already small compared to the number of files. Moreover, hits at edge routers filter out their Interests, so copies in the core get few hits [30], again wasting capacity.

A possible improvement is to selectively cache popular content at the edge, and unpopular ones in the core. Doing so with file popularity is impractical, since that requires ranking a large number of files on the fly, as new files are added and popularity

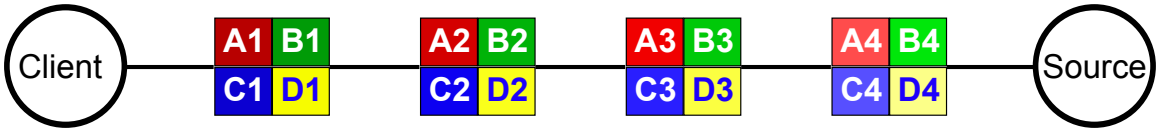


Figure 3-1: Each router in the path caches one segment of each file. File A is the most popular file and file D is the least popular in this example.

shifts. Moreover, the popular files can be large, and so squeeze out other files at the edge. Instead, CCndn spreads a file over multiple routers along the discovered path from client to source, with the head nearer the client and the tail nearer the source (Figure 3-1). We have three reasons for doing so :

(i) Previous traffic measurements have shown that users tend to abort downloads [85, 100]. So chunks at the head of a file are more popular than those at the tail. CCndn is thus consistent with the well-known observation that popular content eventually saturates the edge in a network of caches [24, 30, 42].

(ii) Cache space that is close to clients is precious. By caching only the head at the edge, this valuable resource can be shared by more files.

(iii) For content like video, user-perceived latency depends on how soon the application can get enough of the file to get started. Delays in retrieving the rest of the file can be hidden by the time it takes for the application to consume the head; there is a similar idea in file layout for solid state disks [37].

Although the issue of user tendency to abort is one of our main motivation to spread files along the path, we do not consider it in our experiments as we could not find any reasonable distribution to express the behavior of users to abort downloads. However, later in Section 3.1.2 we will show through some experiments that the first few routers closer to the clients are more precious than further routers. Download termination stops the tail of the stream to be cached on the routers closer to the source which are precious to the clients around the source. So from this point of

S	CCndn's parameter for number of file $\mathcal{F}$'s segments
H	hop count parameter that limits spread of file $\mathcal{F}$
h	hop count from client to Data source
$m_{\mathcal{F}}$	number of segments cached at a router for file $\mathcal{F}$
$N_{\mathcal{F}}^{\text{chunk}}$	number of chunks in file $\mathcal{F}$
$N_{\text{seg}}^{\text{chunk}}$	number of chunks in segment

Table 3.1: Notation for the content caching strategy.

view, the performance of the CCndn in real world even might be better than what we present in our experiments.

To spread a file, CCndn divides a file into segments. The chunks for a segment are cached at the same router. In the following, we use the notation listed in Table 3.1.

How should a file be segmented? We could use a hop count h to divide a file into $h-1$ segments, and cache segment i at router i along the path from client to source. However, if two clients with different hop counts request the same file, they would thus use different segment sizes and cannot share segments. Instead, CCndn fixes the number of segments S, so each file $\mathcal{F}$ has S segments, with size:

$$N_{\text{seg}}^{\text{chunk}} = \lceil N_{\mathcal{F}}^{\text{chunk}}/S \rceil. \tag{3.1}$$

CCndn uses another parameter H that does not exceed the smallest hop count between client and source. Suppose an Interest $\mathcal{I}$ from client $\mathcal{C}$ takes a path through routers $\mathcal{R}_1, \ldots, \mathcal{R}_{H-1}, \ldots$ to reach the source, where $\mathcal{R}_i$ is i hops from $\mathcal{C}$. Let $m_{\mathcal{F}} = \lceil S/(H-1) \rceil$. Then $\mathcal{R}_1$ caches segments $1, 2, \ldots, m_{\mathcal{F}}$; $\mathcal{R}_2$ caches segments $m_{\mathcal{F}}+1, \ldots, 2m_{\mathcal{F}}$, etc.

For example, suppose $S = 22$ and $h = 7$ for a pair of requester-source. If $\mathcal{F}$ has 1024 chunks and $H = 7$ ($H = h$ means all routers in the path will involve in caching the segments of $\mathcal{F}$), then there are $\lceil 1024/22 \rceil = 47$ chunks per segment (except for the last segment); $m_{\mathcal{F}} = \lceil 22/6 \rceil = 4$, so $\mathcal{R}_1$ caches segments 1,2,3 and 4, $\mathcal{R}_2$ caches segments 5,6,7 and 8, ..., $\mathcal{R}_6$ caches segments 21 and 22.

If $S \leq H-1$, then only the first S routers cache 1 segment each. Although H can be as large as the hop count from client to source, a smaller H will keep $\mathcal{F}$'s tail away from the edge at the source, so that premium space can be used by clients there.

How does a router know which segments it should cache? There is no addressing scheme for locating a router in NDN. However, since the path taken by a chunk exactly reverses the path taken by its Interest, we can use hop count to identify a router, as follows:

An Interest $\mathcal{I}$ from a client $\mathcal{C}$ searches the CS in every router along its path to the source. $\mathcal{I}$ keeps track of the hop count from $\mathcal{C}$ in its search. $\mathcal{I}$ may hit its Data along the way and thus not reach the source $\mathcal{Z}$. If $\mathcal{I}$ reaches $\mathcal{Z}$, $\mathcal{Z}$ knows the hop count h from $\mathcal{C}$. It then chooses an appropriate H value and calculates $m_{\mathcal{F}}$. $\mathcal{Z}$ then puts h and i in each chunk's header. As the chunk passes through a router on the reverse path to $\mathcal{C}$, h is decremented. This value becomes i at router $\mathcal{R}_i$, so $\mathcal{R}_i$ knows it has to cache that chunk; other routers do not cache that chunk as it passes through (this is also true if Data is retrieved from an intermediate router).

When a cache is full, CCndn uses LRU (least recently used) replacement. This policy is close to optimal in practice [76], but we will later suggest a modification.

3.1.2 CCndn: Experiments

We now present experiments for tuning CCndn and comparing it to other caching policies.

Performance Metrics

One obvious performance metric for a content caching strategy is the probability $P^{\text{hit}}_{\text{router}}$ of hitting a router cache, i.e. an Interest finds its Data in that CS. We also use the following metrics to measure a strategy's performance over the entire network of caches:

$P^{\text{hit}}_{\text{router}}$	probability of a CS hit at a router
$P^{\text{hit}}_{\text{net}}$	probability that Interest finds Data before reaching source
N_{hops}	average number of hops before Interest finds its Data
N_{copies}	average number of copies (among routers) per Data chunk
α	parameter for Zipf distribution

Table 3.2: Notation for the experiments.

network hit probability $P^{\text{hit}}_{\text{net}}$:

the probability that an Interest finds its Data at some intermediate router, instead of the source; a strategy with a higher $P^{\text{hit}}_{\text{net}}$ is more effective in using the caches to relieve load on the source.

redundancy N_{copies}:

the average number of copies per Data chunk among the routers; a strategy with a smaller N_{copies} reduces memory wastage and can cache a bigger variety of chunks in the network.

hop count N_{hops}:

the average number of hops before an Interest finds its Data; a strategy with a smaller N_{hops} has a smaller latency for retrieving Data, and reduces bandwidth and power consumption.

The best caching strategy is the one with higher *network hit probability* and, at the same time, reduces both *redundancy* and *hop count*. All three metrics indicate the efficacy of the caches but the hop count is also an indicator of bandwidth usage and power consumption. Table 3.2 lists the notation used for the experiments.

Simulation Set-up

Many experiments in the literature on network of caches use a tree topology [25, 42, 48, 56, 64]. Such networks do not test a caching strategy's effectiveness for multipaths and cross traffic. Others use large random graphs [13, 71], but these make it difficult

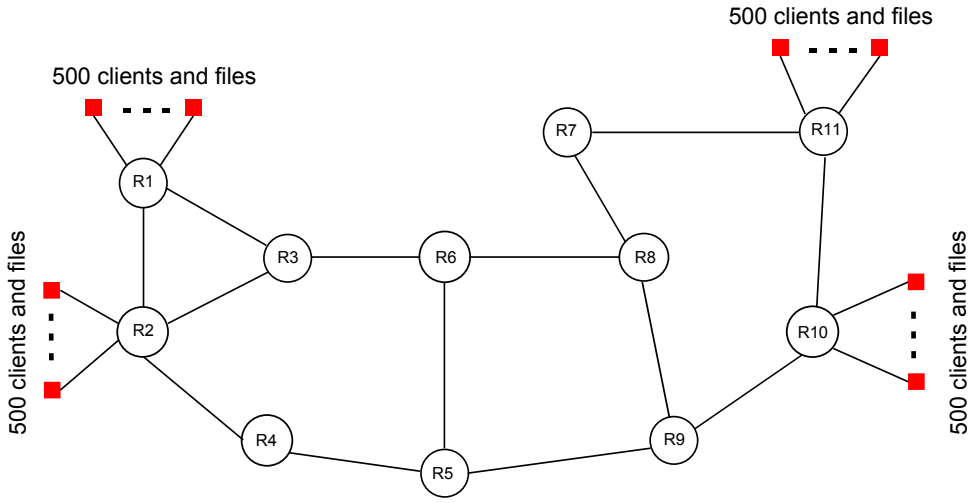


Figure 3-2: Abilene topology and its clients set-up.

to control the experiments and analyze their results. Instead, we choose a realistic, Abilene-like topology[1], as shown in Figure 3-2.

We attach clients only at routers $R1, R2, R10$ and $R11$, so these are edge routers; the other seven are core routers. Each edge router has 500 clients, and each client generates a file that has a geometrically-distributed size of average 500 chunks. The catalog of all files thus has about $4 \times 500 \times 500 = 1$ million chunks.

Unless otherwise stated, routers have the same CS size, which we vary from 1K to 25K chunks. This is 0.1% to 2.5% of the catalog size, comparable to the 5% used by Fayazbakhsh et al. [24]. We also follow these authors in using a request-level simulator that does not model details like router queues and TCP congestion control.

A client sends Interests for a file at a constant rate. The time between the first Interests of two files is exponentially distributed. This means that, sometimes, a client may be downloading multiple files concurrently. The clients at $R1$ and $R2$ request files from clients at $R10$ and $R11$, and vice versa, so there are $2 \times 2 \times 2 = 8$ flows. There is cross traffic in all routers because of Interests flow to and fro.

Interests are routed via the shortest path to the source. If there are multiple paths from a router R to the source, R multicasts the first Interest of the file to all of them;

[1] http://nic.onenet.net/technical/category5/core_topology.htm

R then chooses the one that brings back the Data chunk first, and uses that for the rest of the simulation.

As is common practice, we use a Zipf distribution with parameter α to model the popularity of files [8]:

$$p_N(i) = \frac{\omega}{i^\alpha}$$

where

$$\omega = (\sum_{i=1}^{N} \frac{1}{i^\alpha})^{-1}$$

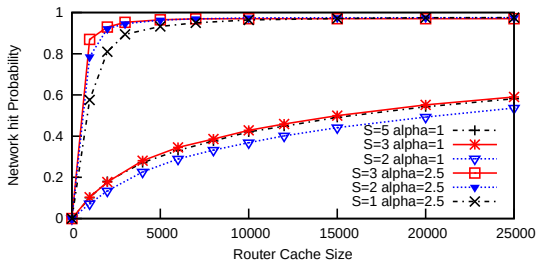
In the plots below, each data point is the average of 5 runs but, to avoid clutter, we omit error bars. Simulation run time is different for different scenarios but in general we wait until each client downloads at least 50 files. Although warm-up is not important, we do not consider the first 10% of messages in our measurements as warm-up period.

Tuning S and H

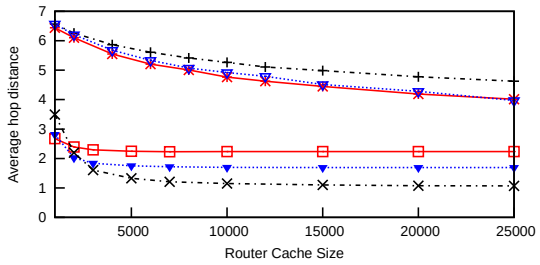
CCndn has two parameters, S (number of segments) and H (for hop count), to be tuned to suit the workload and topology. What should these values be for our experimental set-up?

Figure 3-3 plots network hit probability $P_{\text{net}}^{\text{hit}}$ and hop distance N_{hops} for $H = 7$. Since $S \leq H - 1$, the number of routers involved in caching Data chunks for a path is equal to S. The other way of doing this experiment is to set S larger than 7 (since $h = 7$) and vary H from 1 to 5. These two sets of experiment are equivalent for CCndn but not for CCndnS.

For Zipf $\alpha = 2.5$, Figure 3-3(a) shows that $P_{\text{net}}^{\text{hit}}$ quickly exceeds 0.9 when CS size increases. This is because the file popularity is heavily skewed, so the edge caches are large enough to contain the small number of popular files. In fact, Figure 3-3(b) shows that, for $\alpha = 2.5$, N_{hops} is minimum when $S = 1$, i.e. the entire file can be cached in one router at the edge.



(a) Network hit probability $P_{\text{net}}^{\text{hit}}$



(b) Average distance to content N_{hops}

Figure 3-3: CCndn performance for $H = 7$, Zipf $\alpha = 1$ and $\alpha = 2.5$. All routers have the same CS size.

A caching network is of marginal interest if the popularity is so skewed that the caches at the edge suffice to achieve a high $P_{\text{net}}^{\text{hit}}$. Besides, it is well-known that file sizes have a long tail distribution. In particular, a recent study of traces from a content delivery network shows that files (text, images, multimedia, binaries, etc.) have sizes that fit a Zipf distribution with $\alpha \approx 1$ (specifically: 0.99, 0.92 and 1.04) [24].

Henceforth, we focus on $\alpha = 1$.

For $\alpha = 1$, Figure 3-3(a) shows that $S = 3$ and $S = 5$ have $P_{\text{net}}^{\text{hit}}$ higher than $S = 2$, whereas $S = 2$ and $S = 3$ have smaller N_{hops} than $S = 5$. These suggest that $S = 3$ is the best choice for this topology and workload.

Why is $S = 3$ best although $H = 7$ for this experiment? Note that $H = 7$ is about the number of hops between requesters and sources in Figure 3-2. With $S = 3$, each file is spread over 3 routers, so the tails avoid contending for cache space with the heads. In fact, $H = 4$ will work just as well.

Our experimental set-up is relatively symmetrical (same number of clients per

edge router, same popularity distribution, etc.), so $S = 3$ (or $H = 4$) is intuitively right. In general, where there is more asymmetry, the best value for H (or the choice of number of routers to cache content in a path) will depend on the traffic and topological imbalance.

Since each router caches $m_{\mathcal{F}} = \lceil S/(H-1) \rceil$ segments, why not just set $S = H-1$, so $m_{\mathcal{F}} = 1$? Recall that different clients may have different hop counts to the source, so they have different best values for H. On the other hand, segment size S must be fixed, so clients with different hop counts can share segments. Fixed size of S is the only constraint that CCndn requires. However, the value of S affects CCndnS where Interests can skip searching CS. The right choice of S depends on the application and size of $\mathcal{F}$.

S and H should therefore be set independently. We will revisit S and H tuning after introducing CCndnS.

Comparison With Other Strategies

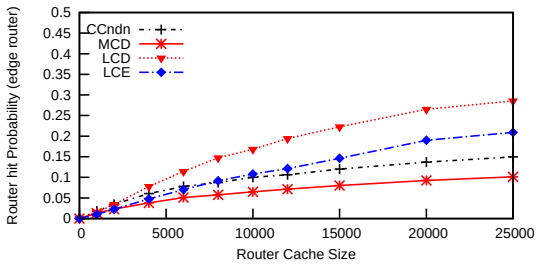
There are several proposals for how a network of caches can cooperate to improve performance [23, 55, 92] that one can use to compare with CCndn. However, cooperative caching requires information exchange that would significantly slow down NDN packet traffic [13]. We therefore consider three simpler strategies (the names follow terminology for hierarchical caches [47]):

LCE (Leave Copy Everywhere):

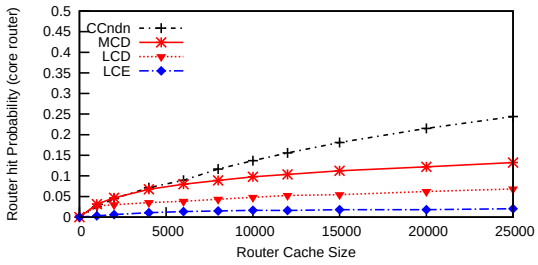
A copy of the Data is cached at every router along the path traced by its Interest; this is the default strategy for NDN [41]. Since it does not apply any rules for Data placement, the effect of the replacement policy (e.g. LRU) is more dominant when using LCE.

LCD (Leave Copy Down):

If an Interest finds its Data after i hops, a copy of the Data is cached at hop



(a) CS hit probability for an edge router (R1)



(b) CS hit probability for a core router (R6)

Figure 3-4: Comparing the strategies' $P^{\text{hit}}_{\text{router}}$ for edge and core routers ($S = 3$).

(1)

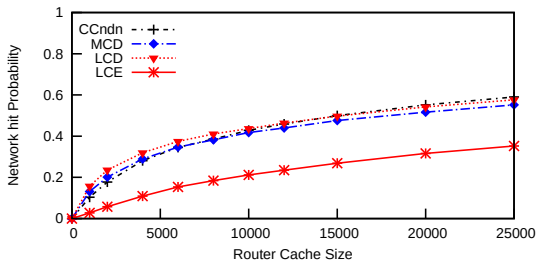
$i - 1$.

MCD (Move Copy Down):

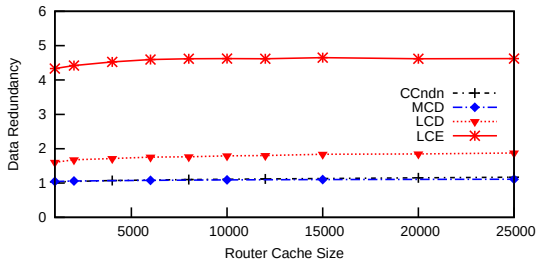
If an Interest finds its Data after i hops, a copy of the Data is cached at hop $i - 1$, and the chunk is deleted from hop i, unless that is the source.

Figure 3-4 compares the $P^{\text{hit}}_{\text{router}}$ of CCndn, LCE, LCD and MCD for $S = 3$. Figure 3-4(a) shows that, for an edge router ($R1$), $P^{\text{hit}}_{\text{router}}$ is higher for LCE and LCD than for CCndn. This is because they cache the most popular files in their entirety at the router, whereas CCndn only caches the heads.

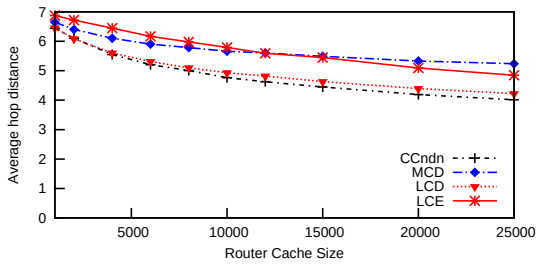
For the core router $R6$, however, Figure 3-4(b) shows that CCndn has significantly higher $P^{\text{hit}}_{\text{router}}$ than all three alternatives. In fact, a comparison of Figure 3-4(a) and Figure 3-4(b) shows that $P^{\text{hit}}_{\text{router}}$ for CCndn at the core $R6$ is even higher than at the edge $R1$. This is because $R6$ gets hits from clients at both $R1$ and $R2$ for the same files, whereas $R1$ only gets hits from its own clients.



(a) Network hit probability $P_{\text{net}}^{\text{hit}}$



(b) Data redundancy N_{copies}



(c) Average distance to content N_{hops}

Figure 3-5: Comparing CCndn (S=3) with alternatives.

Figure 3-5 shows how router hits translate into aggregated network performance. Specifically, CCndn is similar to LCD in outperforming LCE and MCD in terms of network hits $P_{\text{net}}^{\text{hit}}$ and hop distance N_{hops}, and similar to MCD in minimizing redundancy measure N_{copies}.

Chai et al. noted that hop reduction does not imply higher net hit probability [13]; indeed, Figure 3-5 shows that, although CCndn has a significantly smaller average hop distance than LCE and MCD, its $P_{\text{net}}^{\text{hit}}$ value is much higher than that for LCE but similar to that for MCD.

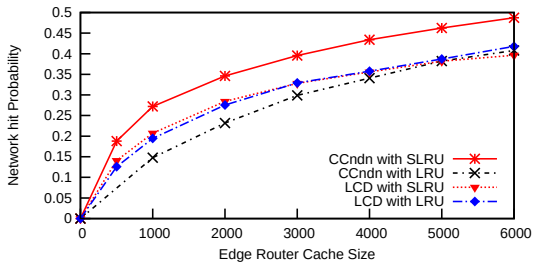
Intuitively, one might expect to see a smaller hop count for LCE because it caches a popular file entirely at the edge router. However, one important observation from Figure 3-5 is that Data chunks are actually closer to the clients under CCndn. Although CCndn caches only part of each file (only popular files are important) at the edge, that increases the diversity of files at the edge. If the network has a large number of popular files, then more of them will have chunks at the edge. If the popular files are mainly multimedia files (e.g. Youtube videos), the delay in downloading the farther segments can be overlapped by the time it takes for requesters to consume the segments near the edge. In this way, having the head of popular files closer to the clients is more effective than having fewer popular files that are cached entirely at the edge.

Moreover, CCndn increases the cache efficiency in the core, so the network effectively offers a larger caching capacity under CCndn. The increase in cache efficiency increases the network hit probability and reduces average hop distance.

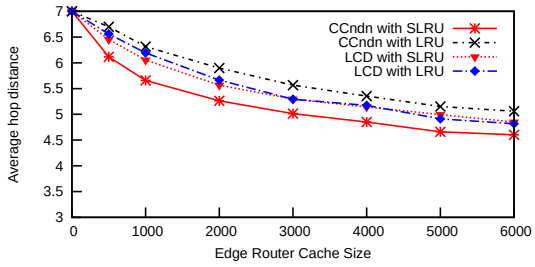
For the experiment in Figure 3-5, LCD has an advantage over CCndn in that the edge caches are occupied by popular content for LCD, but are polluted with unpopular content for CCndn. One simple improvement for CCndn is to adopt the **SLRU** policy previously proposed for disk systems [43]. (SLRU stands for *segmented* LRU, but "segment" in SLRU has a different meaning.)

In our SLRU implementation, a newly arriving chunk c that is to be inserted into the cache does not occupy the head of the LRU list. Instead, if this list has ℓ positions, then c is inserted at position $0.9 \times \ell$, near the tail. Later insertions then push c down this LRU list. If c is not hit, it gets pushed out of the list (i.e. replaced from the cache) sooner than other chunks; if c is hit, then it goes to the top of the list, like vanilla LRU. In this way, unpopular chunks stay in the cache for a shorter time than popular chunks.

Figure 3-6 shows that LCD has similar performance if it uses SLRU instead of LRU. Since LCD performance for a network is dominated by performance at the edge,



(a) Network hit probability $P_{\text{net}}^{\text{hit}}$



(b) Average distance to content N_{hops}

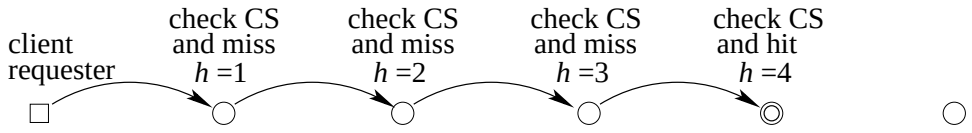
Figure 3-6: Comparing CCndn and LCD, using LRU and SLRU ($S = 3$). $R3, R4, R7, R8$ and $R9$ have 50% more cache space, while $R5$ and $R6$ have 100% more, than the edge routers $R1, R2, R10$ and $R11$. LCE and MCD are omitted since LCD has better performance.

and edge routers are not polluted by unpopular content under LCD, SLRU does not help LCD. For CCndn, however, using SLRU makes a significant difference, resulting in a clear performance improvement over LCD.

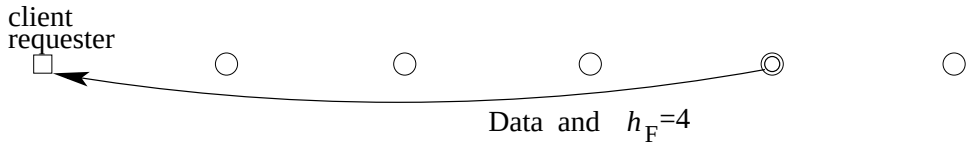
3.2 CCndnS: Decoupling Caches

CCndn uses a segment as the granularity (bigger than a chunk but smaller than a file) for caching content. It follows that if an Interest finds chunk k of a file in router R_x, then the Interest for chunk $k+1$ should skip other routers and go directly to R_x. This is the key idea in CCndnS which is pictured in Figure 3-7.

As before, Section 3.2.1 below describes CCndnS and Section 3.2.2 presents some experiments for analyzing its behavior.



(a) Request for chunk 1 of file $\mathcal{F}$ checks all CS along the way.



(b) Requester receives Data and $h_{\mathcal{F}} = 4$.

(c) Next Interests for file $\mathcal{F}$ only checks CS when $h = h_{\mathcal{F}}$.

Figure 3-7: Data for chunk k of $\mathcal{F}$ passes hop count $h_{\mathcal{F}}$ to request for chunk $k+1$. The latter checks a CS only when its hop count matches $h_{\mathcal{F}}$.

3.2.1 CCndnS: Description

In CCndn, an Interest searches every CS in its path for its Data. With CCndnS, an Interest can use hop count to skip this search for some routers, as follows:

(S1) The first Interest of a file checks every CS along its path to the Data source (Figure 3-7(a)).

(S2) An Interest keeps track of hop count in its search. If it finds its Data at some *intermediate* router R_x, the Data copies the hop count from the Interest and carries it back to the client (Figure 3-7(b)). The Interest for the next chunk of the file uses this hop count to skip CS checks, until it reaches R_x (Figure 3-7(c)).

(S3) Suppose an Interest does not find its Data at intermediate routers and retrieves the chunk from the source $\mathcal{Z}$. $\mathcal{Z}$ puts in the header of this chunk the file size $N_{\mathcal{F}}^{\text{chunk}}$, the segment size $N_{\text{seg}}^{\text{chunk}}$, and the value i for the router R_i that should cache this Data. When the client receives this chunk, it learns hop count i and puts that in the Interest it sends out for the next chunk so, like in (S2), CS

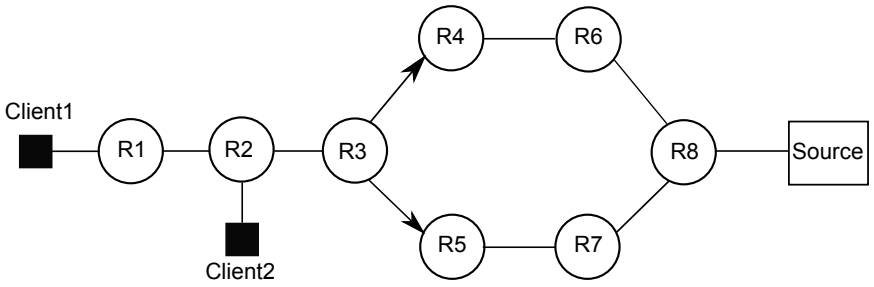


Figure 3-8: An example of a multipath routing problem.

checking can be skipped for $R_1, \ldots, R_{i-1}$. If the Interest does not find its Data at R_i, it skips subsequent routers and retrieves from $\mathcal{Z}$.

(S4) There is an exception to (S3): When a client learns the segment size, it can determine if the next interest $\mathcal{I}$ is for the first chunk in a segment; if so, $\mathcal{I}$ returns to (S1) and checks every CS. This is because the next segment of a file may be cached at an *earlier* router, possibly because some other client has retrieved it. So all Interests requesting the first Data chunk of any segment, search every CS but the other Interests of a segment only check the designated router R_i for that segment.

To illustrate (S4), consider the topology in Figure 3-8. Assume Client1 downloaded file $\mathcal{F}$, which has 6 segments $seg1, seg2, \ldots, seg6$ cached in the following way:

seg1, seg2 at R1; seg3, seg4 at R2; seg5, seg6 at R3; ...

Suppose Client2 also wants $\mathcal{F}$. Since its hop count to the source is different, the caching scheme for Client2 may be

seg1, seg2, seg3 at R2; seg4, seg5, seg6 at R3; ...

When Client2 sends the first Interest for $seg4$, it may expect to find it at $R3$ when, in fact, $seg4$ is cached at a closer router $R2$. This is why CCndnS requires the first Interest of a segment to search all CS in its path to the source.

Next, we use Figure 3-8 to illustrate a separate issue in (S2) and (S3), namely **multipaths**. Assume a segment $segA$ has 4 chunks. When $R3$ forwards Client1's Interests towards the source, it may send the first two Interests to $R4$, then change its routing decision and send the next two Interests to $R5$. The first two chunks of $segA$ may thus be cached at $R6$, say, and the next two chunks are cached at $R7$.

Suppose Client2 now sends Interests for $segA$, and $R3$ forwards them via $R5$. It is possible that the Interests for the chunks at $R7$ skip that router. This situation happens because caching for $segA$ is disrupted by a change in routing decision at $R3$. The only reason for a router to change its decision and select different path for Interest i of $segA$ from Interest $i-1$ of the same segment is that the path for $i-1$ become congested. We assume such incidents are rare because usually routers keep their strategy to forward packets intact during a short period of time. The assumption is that during that short period of time there is not much fluctuation in network traffic to affect routers decision.

Note that forwarding different Interests of a file to different paths are the only case that is considered as problem. As for the chunks of $segA$ cached at $R6$, Client2's Interests may miss them because $R3$ forwards them via $R5$. Such a miss can happen even if there is no skipping.

Finally, it is very important to highlight that CCndnS is not forcing another obstacle to the CCN forwarding architecture. As discussed earlier, the complexity of CCN forwarding architecture forced the designers to make caching at routers optional. Looking through the entire cache for every Interest packet might not feasible for the current memory speed. However, in CCndnS only fraction of Interests will search CS. The time complexity of deciding to whether search a CS is just one comparison of two integer values (i.e., hop distances). This latency is not only much smaller than searching a slow memory, it reduces the length of the potential queue at the CS.

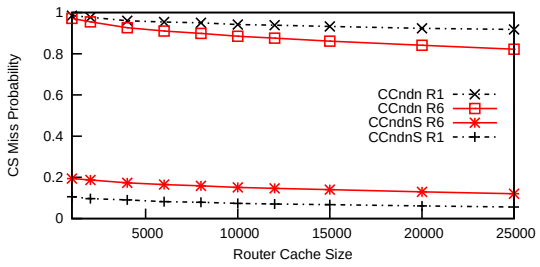


Figure 3-9: Skipping drastically reduces miss probabilities at both edge and core routers. ($S = 5, H = 7$)

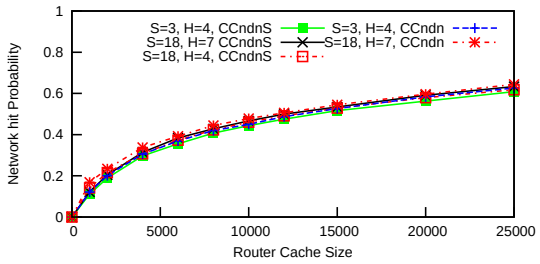


Figure 3-10: Skipping does not affect $P_{\text{net}}^{\text{hit}}$.

3.2.2 CCndnS: Experiments

By targeting a specific router that is likely to cache a desired chunk, CCndnS should reduce the probability of a miss. Indeed, Figure 3-9 shows a big reduction in misses for both edge router $R1$ and core router $R6$.

Earlier in the beginning of this chapter we mentioned of skepticisms that whether a NDN router can cope with line throughput. Considering the origin of this doubt that is accessing slow memories for all packets, introducing skipping scheme can help to rectify the problem largely. As it is shown in Figure 3-11, around 80% of the traffic would not search cache. A complete solution of the problem depends on the architecture of the cache but we believe searching cache only for 20% of incoming traffic, even might reduce the complexity of such memory design.

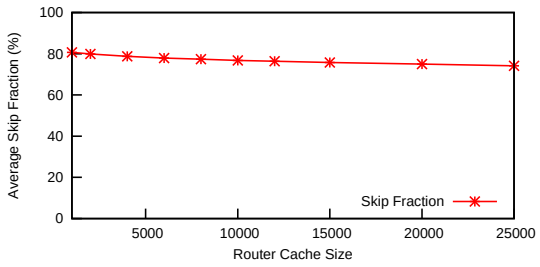


Figure 3-11: Average skip fraction in the network

Hits

However, Figure 3-10 shows that the network hit probability $P_{\text{net}}^{\text{hit}}$ is unaffected. This may seem unintuitive: How is it possible to drastically reduce router miss probability without affecting network hit probability? With a little derivation, this becomes clear:

$$\text{Prob(check CS)} + \text{Prob(don}'\text{t check CS)} = 1,$$

$$\text{i.e.Prob(check CS)} + \text{Prob(skip)} = 1,$$

$$\longrightarrow \text{Prob(check\&hit)} + \text{Prob(check\&miss)} + \text{Prob(skip)} = 1.$$

Using conditional probability:

$$\text{Prob(hit|check)Prob(check)} + \text{Prob(miss|check)Prob(check)} + \text{Prob(skip)} = 1$$

therefore

$$\text{Prob(hit|check)} = \ (1 - \text{Prob(skip)})/\text{Prob(check)} - \text{Prob(miss|check)}. \qquad (3.2)$$

Without skipping, we have Prob(skip)= 0 and Prob(check)= 1, and Equation (3.2) has the usual form Prob(hit) = 1 − Prob(miss), so Prob(hit) is affected by any change in Prob(miss). With skipping, however, the increase in Prob(skip) is balanced by the decrease in Prob(miss|check) in Equation (3.2), so Prob(hit|check) is unaffected.

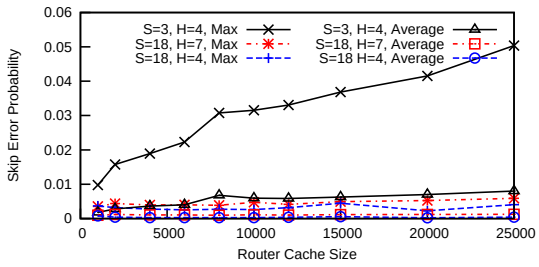


Figure 3-12: Tuning S can drastically reduce maximum skip error.

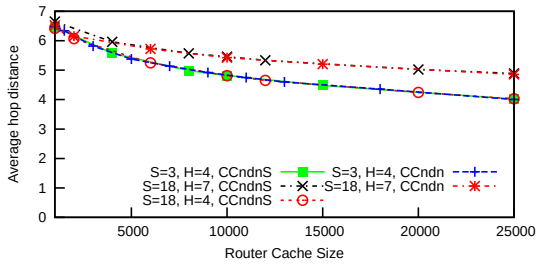


Figure 3-13: Average distance to content N_{hops}.

Skip Errors

It is possible that an Interest does not check a CS when its Data is in fact there, thus committing a **skip error**. Figure 3-12 shows that this error is less than 0.01 when averaged over all routers in our experiment.

However, for $S = 3$ and $H = 4$, the maximum (over all routers) is much larger than the average, and increases with CS size.

Tuning S and H

Interests from a client skip a CS for all non-first chunks in a segment, so any skip error is amplified by the size of a segment. One can thus decrease skip error by reducing segment size, i.e. increasing the number of segments S. Figure 3-12 shows that, when S is increased from 3 to 18 for $H = 4$, the maximum skip error is drastically reduced.

However, there is a price to pay: increasing S spreads a file over more routers, thus increasing hop count N_{hops}; we can see this from Figure 3-13, where S is increased from 3 ($H = 4$) to 18 ($H = 7$).

To reduce maximum skip error by increasing S but without increasing N_{hops}, we can use H to limit the spread of the files. Figure 3-13 shows that, when $H = 4$, increasing S from 3 to 18 does not increase N_{hops}; Figure 3-10 also shows no change in $P_{\text{net}}^{\text{hit}}$.

As a general guideline, increasing S will reduce skip error as there is more Interests which searches the entire path for Data (the first Interest of each segments does). However, that increases the overhead of the routers. If $S = filesize$ (segment size is one Data chunk) then **CCndnS** behaves like **CCndn**.

H value affects the hop distance. Larger H means more routers at core will engage in caching the file. An administrator of the network can set this parameter based on the memory size of the router and the traffic volume passing through the routers. If cache hit rate of the routers is not satisfactory and the traffic is loaded then maybe increasing H helps to reduce the amount of the content need to be cached in each router and increase the life time of content in the cache and therefore increase the cache hit rate. On the other hand, if the hop distance is too high and download time for users exceeds their desire, then reducing H might brings Data closer to the clients.

Eliminating the Filter Effect

One fundamental issue with the default caching strategy for NDN is the filter effect, where only Interests that miss their Data at the edge enter the core. This can lead to traffic imbalance among the routers. Such an imbalance can seriously degrade performance, since traffic load has a nonlinear effect on queuing delay.

We see this imbalance in Figure 3-14 where, under LCE, there is a wide range in the number of Interests received by different routers in the same simulation run. In contrast, except for $R2$, the routers receive a similar number of Interests under CCndnS. $R2$ is different because it is both an edge router and a core router, there are more flows running through it, it is assigned more chunks to cache, and so it receives more Interest traffic.

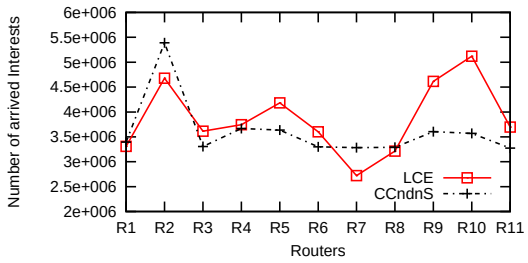


Figure 3-14: CCndnS balances workload among edge and core routers, except for $R2$ (it is both core and edge).

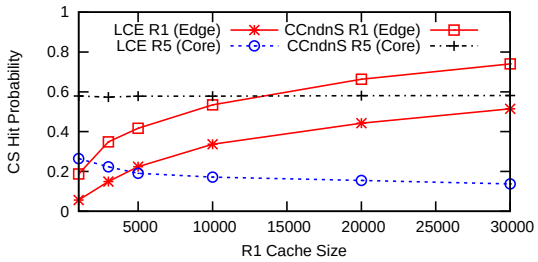


Figure 3-15: Under CCndnS, changing CS size of edge router $R1$ does not affect $P^{\text{hit}}_{\text{router}}$ in the core.

More importantly, filtering reduces the effectiveness of caches in the core. We ran an experiment where the core $R5$ and $R6$ each has a 20K cache, and all other routers have a 1K cache. Figure 3-15 shows that, when the edge router $R1$ has its CS size increased to 30K, $P^{\text{hit}}_{\text{router}}$ for $R5$ steadily decreases for LCE. In contrast, for CCndnS, $P^{\text{hit}}_{\text{router}}$ for $R5$ is unaffected by the change in CS size at the edge.

CCndnS thus decouples the cache performance of edge and core routers.

3.3 CCndnS: Analytical Model

In general, the input to a cache in a network depends on the miss rates in upstream caches. This makes it very difficult to construct a mathematical model to analyze their performance. However, the skipping in CCndnS considerably weakens this dependency, as illustrated in Section 3.2.2. This decoupling makes it possible to model cache behavior with some simple equations.

M_r	CS size at router r
C_r	size of the catalog subset eligible for caching at router r
λ_r	arrival rate of Interests at router r
T_r	average time a Data chunk stays in router r
p_k	probability that an arriving Interest requests file k
f_k	size ratio of file k's chunks to C_r
P_r^{hit}	probability of a hit at router r

Table 3.3: Notation for the analytical model

We first derive equations for $P_{\text{router}}^{\text{hit}}$, then use it to compute the aggregate measures $P_{\text{net}}^{\text{hit}}$ and N_{hops}. In each case, we validate the model with the simulator. Table 3.3 lists the variables used in the model.

3.3.1 Router Hit Probability $P_{\text{router}}^{\text{hit}}$

Consider router r. Let M_r = size of content store CS at router r, λ_r = arrival rate of Interests at router r and T_r = average time a Data chunk stays in router r.
By Little's Law [82],

$$M_r = \lambda_r T_r. \tag{3.3}$$

Under CCndnS, only a subset of the catalog is eligible for caching at router r; let C_r be the size of this subset.

Suppose the part of file k to be cached at router r is a fraction f_k of C_r. Let p_k be the probability that an arriving Interest requests file k. Let M_{rk} be the size of router r's CS that is occupied by chunks from file k. Then the probability of a cache hit at router r is

$$P_{rk}^{\text{hit}} = \frac{M_{rk}}{C_r f_k} \tag{3.4}$$

The Interest arrival rate for file k is $p_k\lambda_r$, so Little's Law gives, in steady state,

$$M_{rk} = \begin{cases} (p_k\lambda_r)T_r & \text{if } p_k\lambda_r T_r < C_r f_k \\ C_r f_k & \text{if } p_k\lambda_r T_r \geq C_r f_k \end{cases}$$

where the second case applies if M_r is so large that router r can accommodate all chunks of file k that is to be cached there. By Equation (3.3), we get

$$M_{rk} = \begin{cases} M_r p_k & \text{if } M_r p_k < C_r f_k \\ C_r f_k & \text{if } M_r p_k \geq C_r f_k \end{cases} \tag{3.5}$$

The probability of a hit at router r is, by Equation (3.4),

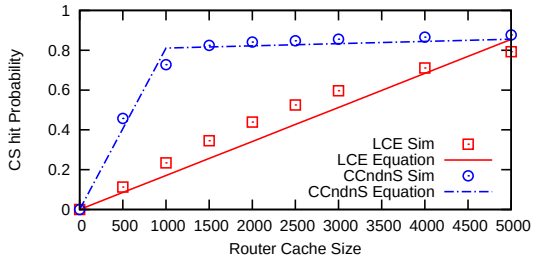
$$P_r^{\text{hit}} = \sum_k P_{rk}^{\text{hit}} p_k = \sum_k \frac{M_{rk}}{C_r f_k} p_k. \tag{3.6}$$

To validate Equation (3.6), we conduct a simple experiment with a chain of 5 routers. We use just two popularity classes: one with 20 files and $p_1 = 0.9$, the other with 180 files and $p_2 = 0.1$; as before, file length is geometrically distributed with a mean of 500 chunks.

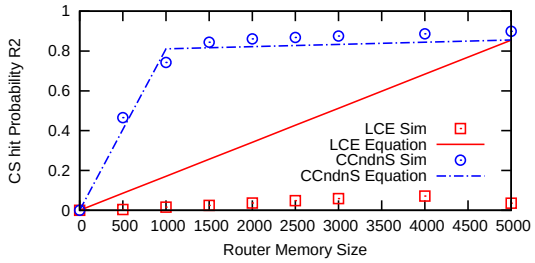
Figure 3-16 shows that, according to Equation (3.6), $P_{\text{router}}^{\text{hit}}$ for CCndnS first increases steeply, as the popular class takes advantage of the increase in cache size; after the 20 popular files are all cached in the routers, $P_{\text{router}}^{\text{hit}}$ increases very slowly for the unpopular class. We see that, for CCndnS, there is excellent agreement between the equation and the measured $P_{\text{router}}^{\text{hit}}$, at both edge and core routers. In contrast, for LCE, the equation gives a reasonable fit only for an edge router.

To validate Equation (3.6), we further test the model with the Abilene topology. Clients at $R1$ and $R10$ request each other's files, and similarly for $R2$ and $R11$, so cross traffic is heaviest at $R2$. The 500 files at each edge router have popularity distributed as Zipf with $\alpha = 1$, so $f_k = 1/500$, and $p_k = p_0/k$, where $p_0 = 1/\sum_{i=1}^{500} \frac{1}{i}$. Figure 3-17 shows that , with 500 popularity classes, the equation gives an excellent fit for the simulated $P_{\text{router}}^{\text{hit}}$ at both edge and core routers.

The two main parameters in the proposed equation 3.6 are catalog size of traffic passing through a router and popularity of the cached content in the router. We have claimed that the popularity of a file remains the same for different routers in



(a) $P^{\text{hit}}_{\text{router}}$ for an edge router



(b) $P^{\text{hit}}_{\text{router}}$ for a core router

Figure 3-16: Equation (3.6) works for CCndnS (but not LCE) at both edge and core routers for a chain topology.

the different network levels. Although that is true for a single file, the rank of the file might not be the same in different routers. We consider finding the global popularity of files or finding the popularity at the publisher (which is the only popularity we need in our equation) remains for our future work.

3.3.2 Network Hit Probability $P^{\text{hit}}_{\text{net}}$

To calculate an aggregated metric for the network, we need to determine Prob(check), i.e. the probability that an Interest checks a CS (see Section 3.2.2).

Consider a file $\mathcal{F}$ with $N^{\text{chunk}}_{\mathcal{F}}$ chunks, and a router r. How many of the $N^{\text{chunk}}_{\mathcal{F}}$ Interests will check the CS at router r?

If $S \leq H-1$, then router r caches at most 1 segment, which has $N^{\text{chunk}}_{\mathcal{F}}/S$ chunks; if $S > H-1$, then the router caches, on average, $N^{\text{chunk}}_{\mathcal{F}}/(H-1)$ chunks. Router r thus gets approximately $I_1 = N^{\text{chunk}}_{\mathcal{F}}/\min(S, H-1)$ Interests for the chunks that it caches.

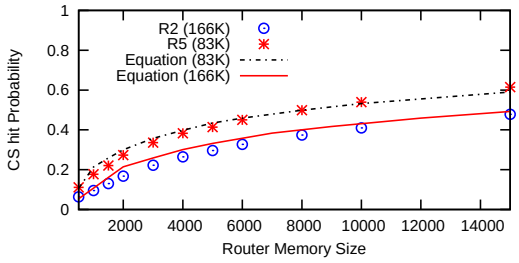


Figure 3-17: Validating Equation (3.6) for $P^{\text{hit}}_{\text{router}}$ at edge router R_2 ($C_r = 166$K) and core router R_5 ($C_r = 83$K) for the Abilene topology.

In addition, CCndnS requires the Interest for the first chunk of a segment to probe every router; at worst, router r gets $I_2 = S$ Interests for this probing. We therefore estimate Prob(check) as $(I_1 + I_2)/N^{\text{chunk}}_{\mathcal{F}}$, i.e.

$$\text{Prob(check)} \approx \frac{1}{\min(S, H-1)} + \frac{S}{N^{\text{chunk}}_{\mathcal{F}}}. \tag{3.7}$$

This approximation does not model how Prob(check) depends on r (e.g. Interest for the first chunk of a segment checks every CS in its path only until it gets a hit); we assume the effect is minor.

Now suppose the route taken by Interests passes through routers $1, 2, \ldots, k$ before reaching the source. The network hit probability would be:

$$P^{\text{hit}}_{\text{net}} = \sum_{r=1}^{k} \text{Prob(check)} P^{\text{hit}}_{r}. \tag{3.8}$$

Figure 3-18 shows that this equation fits two very different sets of $P^{\text{hit}}_{\text{net}}$ measurements in our experiment.

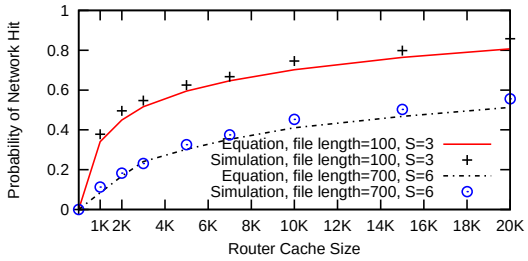


Figure 3-18: Validating Equation (3.8) for $P_{\text{net}}^{\text{hit}}$ ($H = 7$).

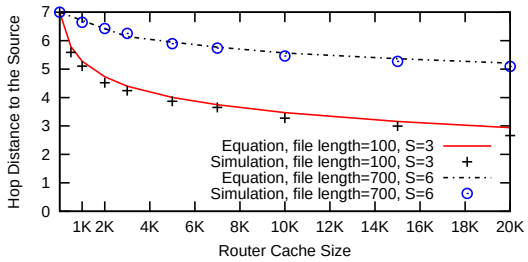


Figure 3-19: Validating Equation (3.9) for N_{hops} ($H = 7$).

3.3.3 Average Hop Count N_{hops}

Similarly, the average hop count is

$$N_{\text{hops}} = \sum_{r=1}^{k} r\text{Prob}(\text{check})P_r^{\text{hit}} + (k+1)(1 - P_{\text{net}}^{\text{hit}}). \tag{3.9}$$

The equation assumes the hop distance between a client and a source is $k + 1$. The first part of Equation 3.9 is for when an Interest finds its Data from network and the second part is for hitting the source.

Figure 3-19 shows that this equation also gives a good N_{hops} fit for the two sets of measurements in our experiment.

In Appendix B we present some other results with a traced base topology.

Chapter 4

SLA with CCndnS

A Service Level Agreement (SLA) is a part of a service contract between the producer of content and a distributer of content. Having cache in NDN router gives the opportunity to ISPs to behave like a small content distributor. ISPs can dedicate some part of their routers' memory to a particular content based on an agreement with the content producer. Here the question is: what is the effect of such memory partitioning on the other cached content in the router? If such agreement has huge negative impact on the cache performance, then the ISP might need to increase its memory size to balance the performance of the cache.

Such effect could be used as a metric to evaluate the cost of the agreement as well.

One of the most important features of CCndnS policy is reducing the Data correlation among caches. Since each segment of a file is cached at a single router, the content popularity can be used by all routers. In other words, the popularity does not need to be measured at each router. For instance the number of hits of a Youtube clip can be used as its popularity and it can be used at all routers. This might be considered as the easiest way of measuring content popularity.

Therefore, knowing the range of the popularities, an ISP can use Equations 3.6 and 3.5 to calculate the cost of an agreement for a file when it is in different level of popularities. The customer (content provider) can use the information to make the

decision on the level of the agreement (or the amount of memory) they need.

Through previous sections we have seen the effect of network of caches on two important networking metrics, hop distance to content and router cache hit rate. The two main parameters in our equation are:

1. the ratio of memory size to the catalog size (i.e., the number of distinct Data chunk passing through a router)

2. the range of popularities.

If an ISP accepts SLA agreements then the memory can be divided into two sections: shared area and dedicated area. Taking some memory away from the shared area to be used as dedicated partition would affect the performance of the files in the shared area. Here in this Chapter we study the effect of such decisions.

We show this effect is negligible if the agreement is for a very popular content. In fact, dedicating memory to the most popular content can improve the overall performance of the cache.

When the agreement is for an unpopular content, the equations can be used to estimate the extra memory space should be provided for a cache to perform at the same level of performance when there was not any SLA agreement.

4.1 Simulation Parameters

A chain network of 11 routers is used in our experiments (Figure 4-1). There are 50 files on one side of the chain (attached to router $R11$) and 50 clients requesting the files on the other end of the network (connected to the router R1). Notice that we validated the equation in Section 3.3. Thus the purpose of experiments here is not to validate the equation but to show how the model can be used in an application. We intentionally chose a simple topology to be able to calculate all required parameters in the equation without any help from the simulation. Calculating the catalog size at

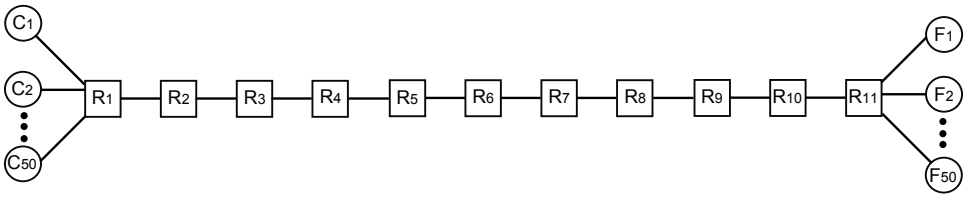


Figure 4-1: Chain of 11 routers. 50 clients are attached to $R1$ requesting 50 files attached to $R11$.

a router for a more sophisticated topology is not a straightforward task. All equation inputs in this Section are calculated and with that we show the accuracy of the model. However, we present the similar results with a larger tree-based network at Appendix C. Since all files are located in a node connected to the root, still we could calculate the catalog size of the traffic passing through the routers.

Another important concern is content popularity. If clients from separate ISPs generate requests with different popularity distribution (i.e., content A is very popular for ISP1 but not very popular for ISP2), then calculating popularity at a core router is not easy.

We assume that the contract is between one ISP and one content provider. We also assume there is no internal contract between ISPs. In that case and in a steady state (no fault, no congestion), there would be only one path between clients and the publisher of content. That explains our choice of using a chain of routers.

The popularity distribution is Zipf with $\alpha = 1$. The file name 0 is the most popular file and consequently the file name 49 is the least popular file.

The inter-file request time of a client is exponentially distributed. The time of requesting each chunk of a file is constant (i.e., CBR).

The length of files are fixed to 330 Data chunks for all files except for the SLA files which are 990 length. Since there are 11 routers in the path, the segment size for SLA files is 90 Data chunks and for other files is 30 chunks. Since we are studying the effect of SLA files on the other files, larger SLA files highlights such effect more. That is the reason of having two different file lengths for the two different file types.

Routers are using LRU (Least Recently Used) replacement policy when the cache is full.

We plot the result of the experiments for memory size of 200 Data chunks up to 700 Data chunks. In our simulation setup, there are 48 files with size 330 and 2 files (SLA files) with size 990 Data chunks. Therefore the catalog size of each router is: $48 * 330/11 + 2 * 990/11 = 1620$ Data chunks.

As mentioned earlier, the ratio of cache size to catalog size is an important metric. Here such ratio changes from 12% to 43% when CS size changes from 200 Data chunks to 700 Data chunks. Since the size of each SLA segment is 90 Data chunks and we assumed there are 2 SLA files, that defined our minimum CS size.

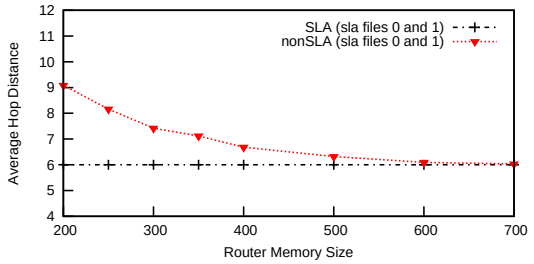
Although 12% ratio of memory to catalog size might not look realistic, still we show that even with that large memory size, SLA has huge impact on the performance of the system and its effect must be known in advance. Smaller ratio of memory to catalog size increases the impact of SLA.

4.2 Full Path SLA Agreement

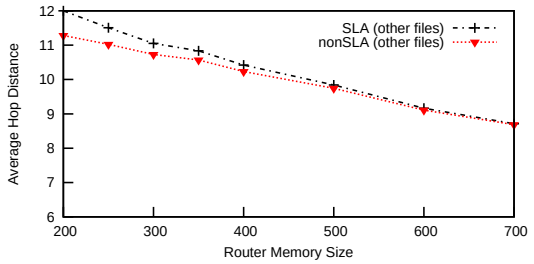
The SLA contract can engage any number of routers in the path. A full path contract can cache all segments of the file. However, a half path, for instance, can only reserve memory in half of the routers in the path. Having full path caching definitely reduces the hop distance to content to the minimum possible value. However, there is always tradeoff between cost and performance. Therefore, we first study the effect of having a full path agreement and later in Section 4.3 we study the effect of having agreement for half of the space of SLA files.

4.2.1 SLA for Very Popular Content

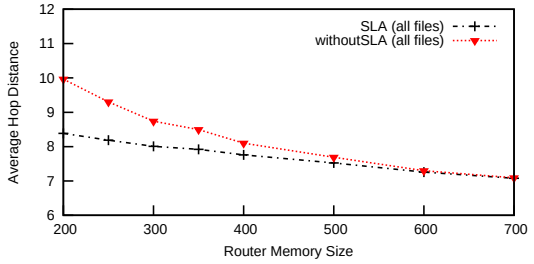
In this Section we evaluate the effect of choosing the first two most popular files as SLA files. When there is enough space in memory for the whole segment of a file, the



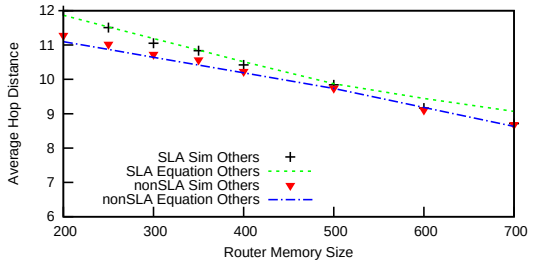
(a) The effect on popular files 0 and 1.



(b) The effect on other files except the SLA files.



(c) The effect on all files including the SLA files.



(d) The results of simulation match with the equation output.

Figure 4-2: The effect of full path memory reservation for the two most popular files. In the legend, nonSLA means there is no memory reservation for file 0 and 1.

hop distance to the file would be equal to the half of the distance to the source which is the minimum distance with CCndnS. Since Data chunks spread along the path with CCndnS, if all Data chunks can be found from caches the hop distance would be equal to the half of the hop distance from requester to the source. Figure 4-2 represents the results for only SLA files in (a) and for other files in (b) and for all files in (c) when there is an agreement for full path partitioning for the two most popular files.

The Figure shows that with SLA the network performs much better specially when $\frac{memorysize}{catalogsize}$ is small. Reducing the hop distance for the most popular files reduces the total hop distances for all files. The penalty for the non-SLA files is negligible. Using this result, an ISP might decide to always keep the most popular files in the cache even if there is not any SLA agreement. That improves its cache performance. Of course the most popular file can be changed during the time when there is no SLA agreement.

Figure 4-2(d) shows a perfect match between simulation and Equation 3.9 results. Thus, instead of doing simulations, ISPs can use the equation to estimate the effect of the partitioning on other files.

We can use our hop distance Equation 3.9 to predict the hop distance for other files except SLA files. Figure 4-2(d) compares the results of the equation against the experiments.

4.2.2 SLA for Less Popular Files

SLA contract has more meaning for a newly generated unpopular content. That might give clients an incentive to download the unpopular file.

Figure 4-3(a) shows the fact that with a SLA contract the unpopular files (i.e., file numbers 40 and 49) get much closer to the requesters. Here again as the contract is for the full path, the hop distance reaches to the minimum possible value with SLA agreement. Maximum hop distance is 12 and the first segment can be found at hop

Without SLA			With SLA		
Hop Distance		Memory size	Hop Distance		Memory size
Equation	Simulation		Equation	Simulation	
9.61	9.78	200	9.61	9.74	380
9.11	8.95	300	9.11	8.90	480
8.74	8.49	400	8.74	8.44	580
8.48	8.13	500	8.48	8.06	680

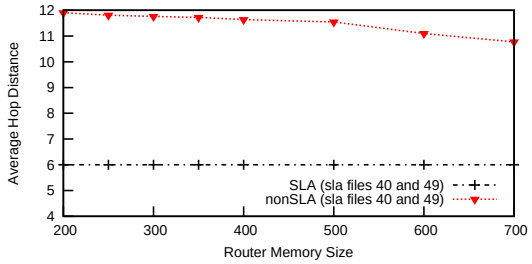
Table 4.1: The equation can be used to find the extra memory size needed to keep the hop distance the same as with SLA. The results obtained for SLA files 40 and 49.

1, segment 2 at hop 2,... so the average hop distance would be half of the maximum hop distance which is 6.

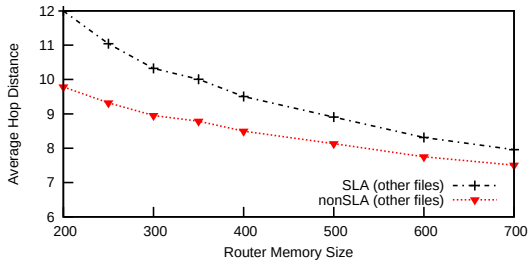
The impact of such improvement on other files (including the popular ones) are shown in Figure 4-3(b). As an example, consider a case where the routers have memory size for 200 Data chunk. Without SLA the average hop distance for all files (in Figure 4-3(c)) and for other files except selected files (Figure 4-3(b)) is around 10 hops away from the requesters. The same Figures show that with SLA the average hop distance increases to 12 hops in both cases.

The performance of a cache mainly depends on how they can serve the most popular files. Therefore, it may be more appropriate if we look at the effect of such contract on the hop distance of the most popular file in Figure 4-4. The Figure shows drastic increase on hop distance for the most popular file when there is not enough space in the memory of the routers for that file.

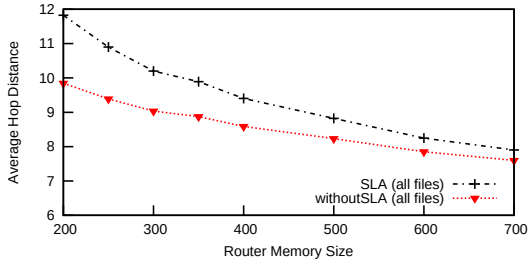
To show the power of the equation we present table 4.1. The column for *Without SLA* shows the hop distance for various memory sizes. For instance, we know that our router memory size is 200. Using the equation we find the average hop distance knowing other information like catalog size of our router and popularity of files. The hop distance found from equation matches with simulation result as well. Now we are interested to find out the memory size needed to keep the hop distance as equal as before (9.6 hops) when we have SLA for unpopular files 40 and 49. Based on equation, the memory size we need is 380. Therefore, the router needs extra 180 memory size



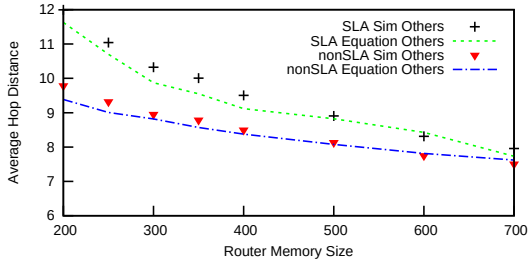
(a) The effect on unpopular files 40 and 49.



(b) The effect on other files except the SLA files.



(c) The effect on all files including the SLA files.



(d) The results of simulation matches with the equation output.

Figure 4-3: The effect of full path memory reservation for two unpopular files 40 and 49.

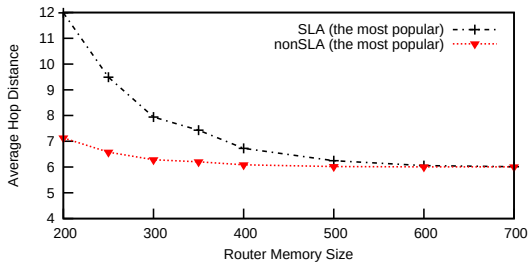


Figure 4-4: The effect of SLA for unpopular files on the most popular file 0.

to keep the hop distance the same when there are SLA for two unpopular files.

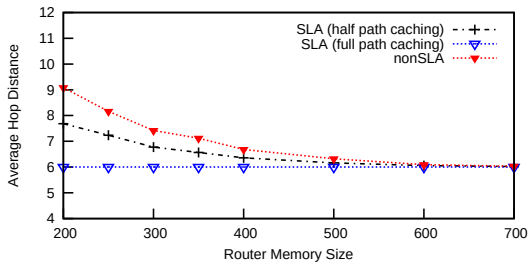
4.3 Half Path Caching

To reduce cost, a customer might request for less memory space for its file. In this Section we evaluate the effect of such decision on the other files and compare it with the full path SLA agreement.

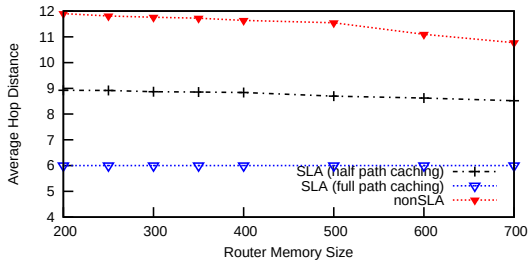
In the new setup of the network, only half of the routers (i.e., the first 5 of them closer to the clients) reserve memory for SLA files. Meaning only 5 segments out of 11 segments of SLA files are cached in the path.

4.3.1 The Effect on SLA Files

Since only 5 routers out of 11 are reserving memory for SLA files, the average hop distance in this case is more than previous case (where all routers engaged in SLA agreement). However, the closer routers to the clients are more effective on reducing the hop distance. That is obvious as for example the last router (the router in hop distance 11th from clients) brings data only one hop closer to the clients whilst this reduction is 11 for the edge router. The results for both popular and unpopular SLA files is presented in Figure 4-5.



(a) When the SLA is for popular files 0 and 1.



(b) When the SLA is for unpopular files 40 and 49.

Figure 4-5: Compare full and half path memory reservation for both popular and unpopular files.

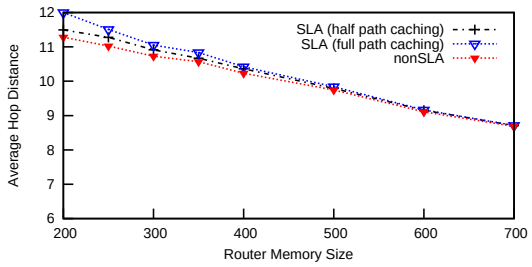
4.3.2 The Effect on Other Files

Figure 4-6 shows that reducing the memory reservation for SLA file to half of the routers does not reduce the hop distance for the other files in the network. In our simulation setup the maximum saving is only one hop.

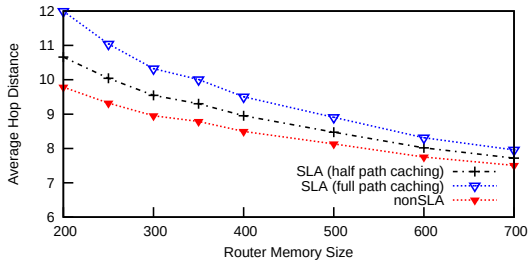
Performance of a cache in a network highly depends on how the cache is serving the most popular files. Figure 4-7 shows that when the SLA file is unpopular how dedicating memory for that affects the most popular file. Even reducing the reservation to half of the routers does not help the most popular file much.

4.3.3 The Effect on All Files

Again as the most popular files dictate the performance of the cache, the result of partitioning on all files depends on whether the SLA file is the most popular file or not. If so the SLA (full or half) improves the performance.



(a) When the SLA is for popular files 0 and 1.



(b) When the SLA is for unpopular files 40 and 49.

Figure 4-6: Results for other files. The negative impact of memory reservation on other files, is largely reduced by cutting the path to half.

The results for all files are presented in Figure 4-8.

4.3.4 Validating the Equation for Half Path Caching

Figure 4-9 proves that the equation can be used when only a subset of routers participate in a SLA contract. Half path analysis is important for two reasons:

- To reduce the cost of the contract.
- The cache miss equation parameters H and S might be set in a way that only half of the routers of the ISP falls in the path between client to publisher.

4.4 Single Router Analysis

So far we assumed multiple routers are engaged in the caching process. However, it is possible that an ISP owns only one router or as another scenario, a customer wants

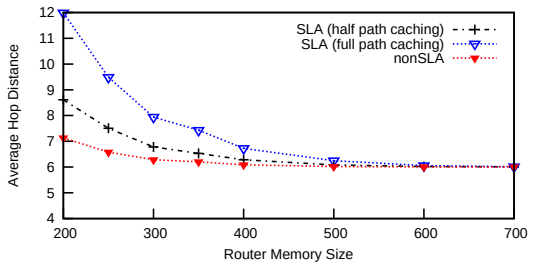
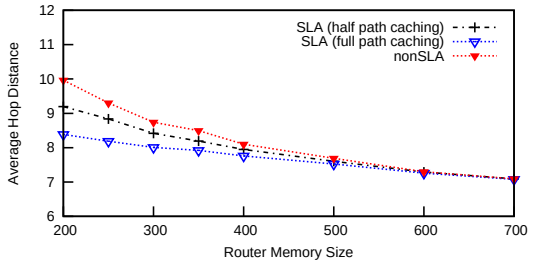
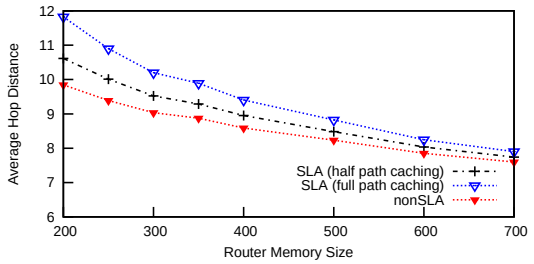


Figure 4-7: The effect of SLA for unpopular files on the most popular file 0.



(a) When SLA is for popular files 0 and 1.

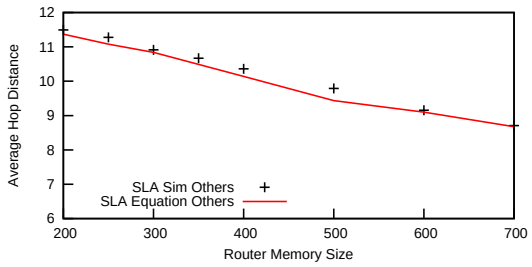


(b) When SLA is for unpopular files 40 and 49.

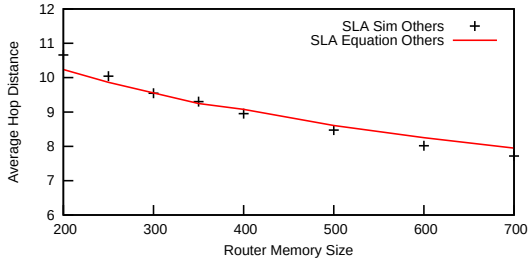
Figure 4-8: The impact of full and half path caching on all files.

to have a dedicated memory space only at the edge router of a specific clients domain. The incentive could be that caching only the head of a file at the edge router would be sufficient for the file to become popular. When the file becomes popular, the rest of the file most likely will be cached in the rest of the routers with the aid of LRU replacement policy.

The issue here is that, hop distance might not be a good metric when there is only one router in the contract. In fact Figure 4-10 shows very small differences between SLA and without SLA scenarios when only the edge router is to cache the SLA file.



(a) Validating the equation when SLA is for popular files 0 and 1.



(b) Validating the equation when SLA is for unpopular files 40 and 49.

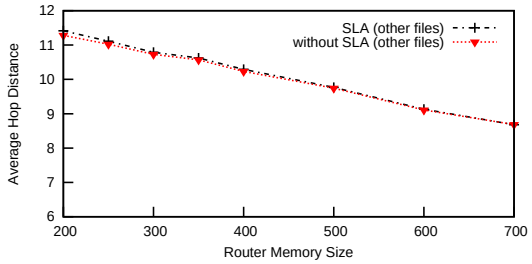
Figure 4-9: The equation is validated for half-path memory reservation.

Cache hit rate in Figures 4-11 and 4-12 could be a better metric for a single router. Figure 4-13 shows that the equation fits well with the simulation results and can be used to estimate cache hit rate for a single cache analysis.

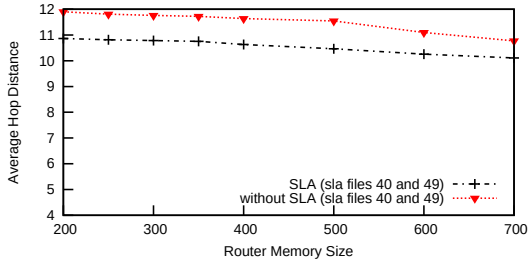
The same methodology can be used here for CS hit rate to project the effect of the SLA contract on the CS hit probability of the other files and estimate how much extra memory is required to keep the level of CS hit probability the same as before when there was not any SLA contract.

In table 4.2 the CS hit probability for different memory size when there is no SLA contract is reported in the first half of the table. Then keeping the CS hit probability the same, in the second half of the table, using the equation we find the memory size require to reach the same probability.

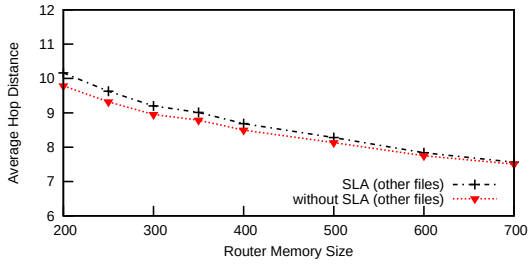
(a) The effect on popular files 0 and 1.



(b) The effect on other files except 0 and 1.

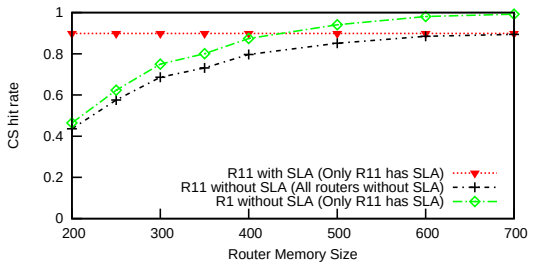


(c) The effect on unpopular files 40 and 49.

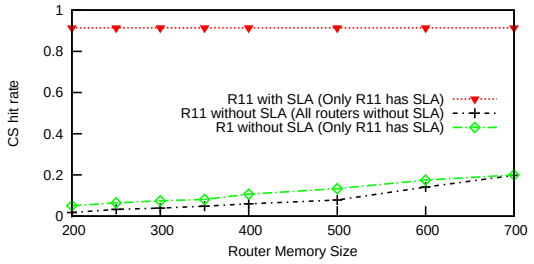


(d) The effect on other files except files 40 and 49.

Figure 4-10: Hop distance is not a good metric when there is only one edge router in the contract.

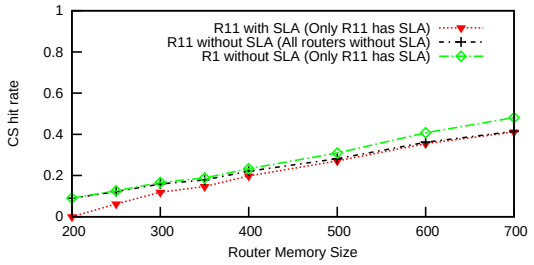


(a) The effect on popular files 0 and 1.

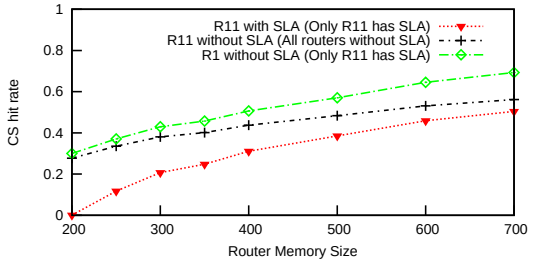


(b) The effect on unpopular files 40 and 49.

Figure 4-11: The effect of partitioning on cache hit rate for the SLA files on the edge router attached to the clients.



(a) The effect on popular files 0 and 1.



(b) The effect on unpopular files 40 and 49.

Figure 4-12: The effect of partitioning on cache hit rate for the other files except the SLA files on the edge router attached to the clients.

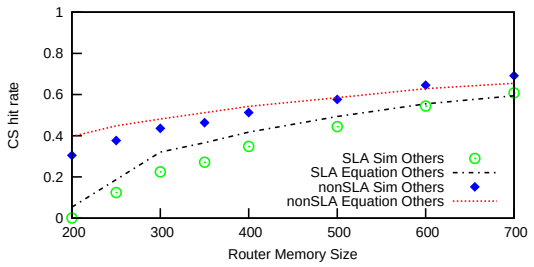


Figure 4-13: The equation is validated for CS hit rate of a single router and for unpopular files 40 and 49 as SLA files.

Without SLA			With SLA		
CS hit probability		Memory size	CS hit probability		Memory size
Equation	Simulation		Equation	Simulation	
0.30	0.27	200	0.30	0.28	380
0.39	0.38	300	0.39	0.38	480
0.45	0.43	400	0.45	0.44	580
0.49	0.48	500	0.49	0.48	680

Table 4.2: The equation can be used to find the extra memory size needed to keep the CS hit probability for the edge router. Results are for having SLA for unpopular files 40 and 49 only at the edge router R11.

Chapter 5

CS Partitioning Based on Cache Miss Equation

Through the previous Chapters we observed that the performance of the cache in a router with limited amount of memory space, very much depends on how the cache deals with the popular files. That in general means managing the cache inside the router is an important task.

Memory partitioning is a powerful tool for per-application memory management. Per-application memory management allows to set some predefined constraints on how to assign resources (here in particular memory) to different applications to reach a particular goal. For instance if a specific application needs a strict level of requirements or ISPs needs to maintain fairness among different applications, they all can be done by proper memory partitioning scheme.

In [11] the effectiveness of using memory partitioning in an NDN router instead of using shared storage (without any partition) on user experience and bandwidth utilization is studied. The major part of the study is focused on comparing the static partitioning against shared storage. Static partitioning means the partition size is evaluated and assigned once and it remains till the next evaluation (i.e, the partition sizes would not change by traffic pattern alteration). They also introduced

two dynamic partitioning algorithms which are priority storage management (PSM) and Weighted fair storage management (WFSM). These two algorithms are based on priority where different classes compete on CS. When the partition of a class is full the algorithm replaces one of the lower priority class data chunk with the newly arrived data chunk (i.e., reduces the partition space of the lower priority class and give it to the higher priority class). The problem of this type of algorithm is that if one of the flows becomes bursty for a period of time it is going to occupy larger space of the CS. In case of PSM if this flow has highest priority this space never goes back to other flows. In case of WFSM it takes some time (which might not be short) to refine the partition sizes.

They mentioned that dynamic partitioning based on monitoring is not affordable in a chunk level caching system. Previous work on cache partitioning typically uses a linear approximation of the miss rate curve, then adjusts the partition iteratively over multiple epochs (e.g., gradient descent [80] or ARMA controller [75]). Oscillations and convergence can be an issue for such techniques.

In contrast, we use our (nonlinear) cache miss equation to calculate the desired partition that minimizes miss probability or enforces fairness, and enforce the partition in one epoch (without iteration) [67]. The algorithm assigns fixed size of partition to each flow and each flow can only replace new data with old data in its own partition (unlike priority based algorithms). The system is stable since it does not need to continuously change the partition sizes to experimentally find the best fit. Using the cache miss equation the system can find the best partition sizes very quickly and keep it till the next considerable changes in traffic pattern. Thus the system needs to monitor the input but does not keep changing the partition sizes. So this scheme might be considered as a semi-monitoring system.

The results are evaluated with our simulator.

Note that packet classification (e.g., [39] and [33]) is not a concern of this Chapter and we assume that the router can distinguish different traffics from each other.

We need to highlight that Equation 3.6 cannot be used for this purpose. The Equation 5.1 is a more general form of a cache miss equation than our cache hit Equation 3.6. Equation 3.6 works only with CCndnS cache policy. In addition, this equation requires some knowledge from network. Content popularity and catalog size of the passing traffic needed in advance. That makes the equation inefficient for a dynamic system. The cache miss Equation 5.1 does not require such information as it adapts with passing traffic by sampling the traffic.

Therefore, these two equations can be used for different purposes. Equation 3.6 is effective in estimating or studying the performance of the system when a static or permanent change imposes to the system (e.g., increase the cache size). On the other hand, Equation 5.1 is perfect for dynamic changes. Moreover, Equation 5.1 can only be used for a single cache. As one of our future works we plan to extend the equation to be used for estimating the cache miss of the whole network.

5.1 Cache Miss Equation

The essence of the memory partitioning algorithm which is going to be introduced in Section 5.3 is based on the following buffer miss equation:

$$P^{\mathbf{miss}} = \frac{1}{2}(H + \sqrt{H^2 - 4})(P^* + P_c) - P_c \tag{5.1}$$

where

$$H = 1 + \frac{M^* + M_{\mathbf{b}}}{M + M_{\mathbf{b}}}, for M \leq M^*.$$

The source of this equation is from the page fault equation introduced by Tay and Zou [83]. The equation is used in [84] to dynamically tune database buffers. The concept of partitioning buffer size is the same for any buffering system and so we will investigate here to find out if the equation works well for partitioning the cache inside of an NDN router.

The power of the equation is that, having few sample points of memory miss probability for given memory size, the equation can fill in the memory miss probability for other memory sizes in between the sample points.

The equation would stay intact by changing network parameters or caching configurations like replacement policy (i.e., the process of chunk selection in a memory to be replaced with a newly arrived chunk) or cache policy (i.e., the decision process of which data chunk should be cached in a router). Changing environmental parameters only affects the four main parameters of the equation.

The four parameters are listed in table 5.1.

- $\mathbf{P^*}$ - Cold miss is the minimum cache miss probability that might have been obtained by increasing the cache size of a router. Then the minimum cache size with miss probability of P^* gives the M^*. For instance in Figure 5-2(a) P^* is about 0.4 and M^* is around 6000.

- $\mathbf{M^*}$ - The smallest memory size for which $P^{\mathbf{miss}} = P^*$. After this point the curve would be flat which means increasing the memory size more than M^* has no or very little effect on $P^{\mathbf{miss}}$.

- $\mathbf{M_b}$ - The memory inside a router might be used for other purposes like indexing, hashing, etc.. The memory used for such purposes (anything other than Data chunks) is represented by $M_{\mathbf{b}}$.

- $\mathbf{P_c}$ - Many parameters like replacement policy or cache policy has effect on cache miss probability ($P^{\mathbf{miss}}$). These are modeled by $P_{\mathbf{c}}$. In effect, $P_{\mathbf{c}}$ controls the convexity of the $P^{\mathbf{miss}}$ curve. Tran, et. al., in [84] provides more derivations of the equation.

To demonstrate the power of the equation that accurately can project memory miss probability for all memory sizes and with any replacement policies, in Figure 5-2 the output of the equation and the results of simulating a network with different re-

Table 5.1: Main parameters

M	cache size.
M_b	size of non-cache memory.
M^*	smallest cache size for which $P^{miss} = P^*$.
P^*	probability of a cold miss.
P_c	parameter that controls convexity of P^{miss} curve.

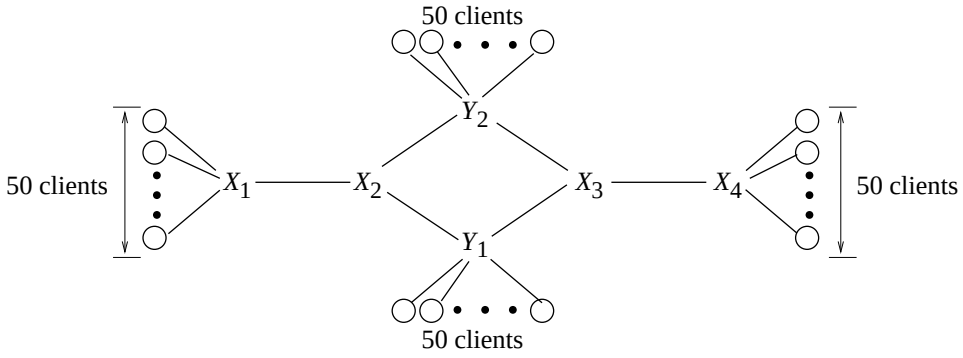


Figure 5-1: Topology for the experiment. X_1, X_2, X_3, X_4 (along the "x-axis") and Y_1, Y_2 (along the "y-axis") are routers. This design models cross traffic and multi-path routing.

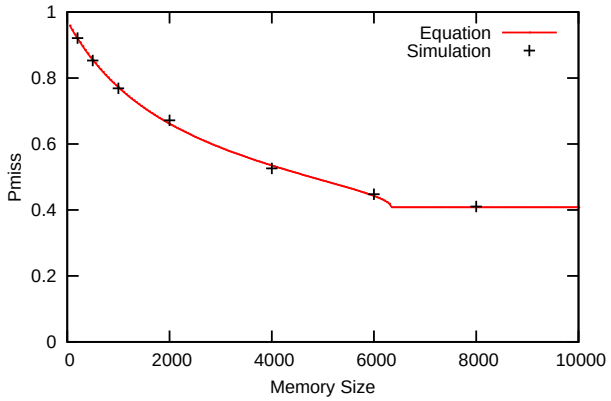
placement policies and other network parameters are shown. The details of simulation setup and the network topology is presented in Section 5.2.

5.2 Simulation Setup

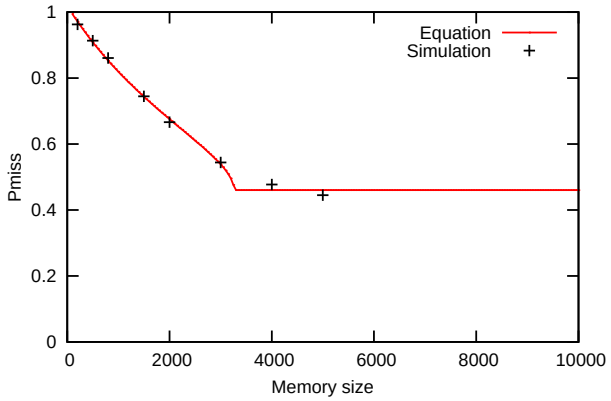
We use the same simulator explained in Section 6.4 and the same topology as Figure 5-1.

Here again each client hosts one file and all clients can request the files on other clients except the clients in their local network (attached to the same edge router). The inter-file request time is exponentially distributed with rate λ_{file}. We consider both CBR and nonCBR traffics.

File popularity is Zipf with different parameters for different traffic classes. Files on each local network (i.e., files on clients attached to the same router) have their own popularity set. In this way the traffic of the network would be more even on each



(a) Random replacement policy, traffic nonCBR, file length 120 packets, Zipf $\alpha = 1.6$.



(b) LRU replacement policy, traffic CBR, file length 60 packets, Zipf $\alpha = 2.3$.

Figure 5-2: Validate the equation with different replacement policies. Results are for router Y1.

side. There are three different set of file popularities representing three distinguish applications. Table 5.2 summarizes the details.

An Interest packet size is 500Bytes, and a Data chunk is 50KBytes. Again each flow in our experiments has different file size (table 5.2). CS size is set to 600 packets which is 4.7% of the total catalog size of the network. Different choices of replacement policy, traffic type (CBR/nonCBR), popularity (α), etc., produce different shapes for the simulation data. Figure 5-2 shows that the equation can fit these different shapes. We examined both random and LRU replacement policies and we reached the same

conclusion using either one of them.

This study is a comparison study which compares shared CS with partitioned CS and we tried to provide equal environment to make the comparison fare.

5.3 Static Allocation

We first show how the equation can be used for partitioning a memory size M into k (M_1,M_2,...M_k) partitions. Here we assume that the ISP or in general the network administrator determines the number of traffic classes or number of partitions (This can be done based on agreement with content producers). Now the question is, what is the best partition set size? For this Section, our goal is to minimize the CS miss probability. CS miss probability has direct effect on link bandwidth utilization (more hit means less bandwidth usage) and packet delivery time which affects user performance.

So here the target is to provide a mechanism to divide memory size M to several partitions whilst keeping the total CS miss rate minimum. Suppose we want to partition M into three subsets (M_1,M_2,M_3) for three workloads with three sets of miss probabilities. To do that first we need some sample points of (M,$\mathrm{P}^{\mathbf{miss}}$) for each flow. There are some suggestions in [84] of how to collect the sample points for the equation but in general they can be obtained from a simple simulation or by letting the network operate for a short time and collect the miss rate results for different memory sizes.

Now suppose we have a set of (M,$\mathrm{P}^{\mathbf{miss}}$) data points. In case that P^{*} is known we use regression in [83] to find values for M^{*}, $\mathrm{M}_{\mathbf{b}}$ and $\mathrm{P}_{\mathbf{c}}$ to give a best fit between the points and the curve defined by the buffer miss equation. If P^{*} is not known we search iteratively for a P^{*} value that gives a best fit.

We need to do the above procedure for each and all flows determined by the administrator to assign a partition size for them. In our example there are three

Table 5.2: Characteristics of the three traffic classes

	Class 1	Class 2	Class 3
file size	10	60	120
Zipf α	1	2.3	1.6
Traffic	CBR	nonCBR	nonCBR

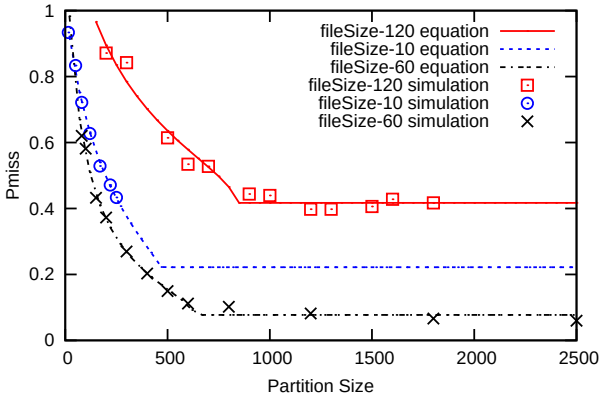


Figure 5-3: P^{miss} prediction for the three distinct traffics.

flows and so we need to have three curves. Figure 5-3 shows the three curves of our example for router X1 of our network. The exact procedure is that we run the simulator for few selected memory sizes for each flow individually. Then using the method explained above the graph is filled in.

Notice again from Figure 5-3 that the equation fit the data for three traffic classes that have very different miss behavior (P^*, curvature, etc.).

In this step we are only trying to evaluate the equation and study if it works well for an NDN router. To do that we are applying the idea on the router X1 in Figure 5-1. One problem we have to deal with is that having cross traffic or two way traffics (i.e., from X1 to X4 and from X4 to X1) brings some requests which already passed through some routers and the CS of the other routers filter out some part of the traffic. This means the incoming rate to a router is under effect of CS hit rate of other routers which that is under direct influence of the routers partition sizes.

To eliminate any effect of cross traffic on our study, we turned off the cross traffic by splitting the nodes to two sets of clients and content generators. In the new setup

for the X network all clients are connected to the router X1 and all nodes with a file are attached to the router X4. The same setup goes for Y network and only clients on router Y1 are requesting files on router Y2.

Later, for dynamic partitioning evaluation, we relax this constraint and let the network work per normal with cross and two-way traffic.

The Equation 5.1 helps us to get a miss probability function $f_i(M_i)$ for any partition sizes M_i. The aggregate $\mathbf{P^{miss}}$ for partition set size (M_1, M_2, M_3) then can be predicted by

$$P^{\mathbf{miss}}(M_1, M_2, M_3) = w_1 f_1(M_1) + w_2 f_2(M_2) + w_3 f_3(M_3) \tag{5.2}$$

where w_i is the probability that a data reference belongs to partition set i. To evaluate the idea we run three sets of simulations with three distinct traffic classes. Table 5.2 summarizes the characteristics of the three traffics. For the three traffic flows, we tried to find three distinguished traffic classes which return different P^{miss} by increasing the partition sizes. So the classes need to have different file sizes and popularity properties. The three classes in table 5.2 can be representative of three applications, HTTP web [1], web media [17] and user generated contents respectively.

Running the simulator for different CS sizes to find their CS miss probability, provides the sample points for the equation to fit in the curves (Figure 5-3). We also measure the number of access to the CS L_i at router X1 for each traffic classes, and estimate w_i by $w_i = L_i/(L_1 + L_2 + L_3)$. Now that we can find the P^{miss} for different memory sizes of each traffic class and we know the incoming rate to the CS for each traffic, using the Equation 5.2 the partition sizes which return the minimum aggregate P^{miss} can be found.

Table 5.3 represents the results obtained from equation and simulation using different partition sizes. The same results are presented for Random replacement policy in table 5.4. The minimum P^{miss} from equation belongs to partition set size: $(1, 208, 391)$ for LRU policy and $(1, 205, 395)$ for Random policy.

Table 5.3: Finding the partition size with minimum P^{miss} with LRU replacement policy.

Partition size	$P^{miss}_{Simulation}$	$P^{miss}_{Equation}$
(1,100,499)	0.606	0.632
(1,150,449)	0.565	0.61
(1,208,391)	0.545	**0.588**
(1,250,349)	**0.541**	0.608
(1,300,299)	0.549	0.619
(1,400,199)	0.572	0.663
(10,200,390)	0.547	0.606
(10,245,345)	0.55	0.609
(50,150,400)	0.552	0.619
(50,200,350)	0.572	0.615
(50,250,300)	0.56	0.622
(100,200,300)	0.584	0.634
(200,200,200)	0.622	0.689

Results of the simulation confirms that this partition size returns one of the least P^{miss} values for both policies. However, the minimum P^{miss} from the simulation belongs to partition size $(1, 250, 349)$. The difference between the two P^{miss} is very small and they fall into the same partition configuration (i.e., the partition size for the first class is almost 0 and the last class gains the largest CS size). This small difference could be caused by some randomness parameters in the simulation as well. The incoming rate into the routers is influenced by CS hit rate of other routers. Although we tried to lessen this effect still there is small variation. The small variation of incoming rate to the routers might enforce some error into the equation. However, still the equation could find a good guess for the best partition sizes.

Note that the optimal partition essentially does not cache chunks for class 1 traffic. Thus, although NDN is designed with objective of in-network caching, it is not optimal to do so for all traffic when there is multi-class traffic.

The results show that using the equation we could find a set of partition sizes which results in almost minimum P^{miss}.

Table 5.4: Finding the partition size with minimum P^{miss} with Random replacement policy.

Partition size	$P^{miss}_{Simulation}$	$P^{miss}_{Equation}$
(1,104,495)	0.574	0.608
(1,150,449)	0.524	0.579
(1,205,394)	0.515	**0.569**
(1,250,349)	**0.508**	0.574
(1,300,299)	0.540	0.590
(1,400,199)	0.571	0.651
(10,200,390)	0.519	0.571
(10,245,345)	0.567	0.611
(50,150,400)	0.532	0.589
(50,200,350)	0.538	0.583
(50,250,300)	0.523	0.592
(100,200,300)	0.559	0.605
(200,200,200)	0.606	0.675

5.4 Dynamic Partitioning

In Section 5.3 we showed that the Equation 5.1 can be used to accurately find the partition size for three different classes of applications which results in minimum P^{miss}. Now in this Section we use the equation to set the partition size automatically whilst the router is working.

This method, unlike other dynamic partitioning algorithm (e.g., [80]), is not based on constant partition size refinement. Constantly refine partition sizes for a chunk level router is almost impossible. In our method routers only modify the partition sizes during the learning stage to collect some sample points for each flow. Then the equation can be used to project the best partition size for specified flows which meets the requirements. This requirement can be any performance metrics for CS. In an ordinary dynamic partitioning scheme (e.g., [81]) system gradually finds the partition sizes for different classes.

In our partitioning scheme we examine the network for different partition sizes for a short period of time and then the desired partition sizes can be found very quickly using the cache miss equation. The router monitors the system but since the

router already has its partitioning this monitoring can be done in a longer intervals (depending on the router location or configuration this interval can be set to few minutes to few hours). Only when the router observes big changes in CS miss rate (not satisfactory miss rate), it can redo the partitioning to find the best partition size for the current traffic pattern.

Here we show how the equation can be used to find the partition sizes which first results in minimum aggregate CS miss rate and in the next Section we show how to maintain fairness among the flows.

The partition setting can be recalculated when a specific performance metric is violated (e.g., unsatisfactory CS miss rate). When a router realizes a major change in traffic pattern, it starts the learning stage again to find out the proper partition size for the current traffic pattern. Administrator can put the router in learning stage at any time as well. If there are any changes in number of flows or their type, the router needs to learn the new settings by initiating the procedure of repartitioning the memory again.

In learning stage, time is divided into rounds called epoch. In each epoch router sets different partition size for each application class and collects miss probability of each partition. The only constraints we apply is that $M_1 + M_2 + M_3 = M$ where M is the CS size and M_1, M_2, M_3 are partition sizes. Table 5.6 shows the partition size for each class and their miss probabilities for router X_1 when replacement policy is random. The results for LRU policy is reported in table 5.7 for router Y_1. In our example the simulator runs for 5000 file requests in each epoch.

In each step we try to find the proper partition size for only one of the routers in our network whilst the other routers are already partitioned or still using shared CS without any partitions. For instance, when the router X_1 is in learning state, all other routers are using shared CS without any partitioning. When router X_1 found its partition size its going to use it and router X_2 goes to learning state and all other routers keep running shared CS without partitioning.

Table 5.5: The four parameters obtained for the router X_1 and Y_1.

X_1 (Random)	M^*	M_b	P^*	P_c
Partition 1	179.973	202.398	0.562	-0.265
Partition 2	280.633	146.823	0.472	-0.234
Partition 3	457.343	1011.84	0.621	-0.21
Y_1 (LRU)	M^*	M_b	P^*	P_c
Partition 1	187.636	184.114	0.431	-0.093
Partition 2	283.224	277.582	0.423	-0.092
Partition 3	469.226	136.125	0.729	-0.658

At the end of the learning state the i pair of (M_i, P_i^{miss}) are going to be applied to the memory miss equation to find the partitions which minimize the aggregate P^{miss} in Equation 5.2.

The obtained four parameters from the regression [84] is presented in table 5.5.

Using the parameters in a nested iterative loop by changing the size of each partition, the partition sizes which returns the minimum P^{miss} is $(1, 281, 318)$ with $P^{miss} = 0.687$ for Random policy on X_1 and $(85, 284, 231)$ with $P^{miss} = 0.684$ for LRU on Y_1. .

The last three epochs (epoch 13, 14 and 15) show the results of running the simulator with the obtained partition sizes.

To save space we skip presenting information for other routers and only show the final results when all routers find their best partition sizes (table 5.8 and 5.9).

Since setting partition on CS of the routers might change the traffic pattern compare to shared CS, after all routers set their partition size, we put the router X1 in initial stage again to see how much refinement it needs. The new partition size and their probabilities are reported in table 5.10 for Random policy and table 5.11 for LRU policy. As it can be seen the new partition size and its CS miss probabilities are quite consistent with the first results for router $X1$.

Comparing the dynamic partitioning results in table 5.8 with results obtained for shared CS in table 5.12 reveals that P^{miss} is always smaller for dynamic partitioning although the difference for core routers like $X2$ and $X3$ is not much. The results

Table 5.6: Results of dynamic partitioning for router $X1$. The first 12 epochs are for calibrating the equation. In epoch 13, the equation is used to determine the partition that minimizes aggregate P^{miss} (**Random** replacement).

Epoch	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
1	(27,374,199)	0.909	0.404	0.918	0.722
2	(1,322,277)	0.998	0.447	0.883	0.728
3	(334,53,213)	0.403	0.884	0.894	0.848
4	(218,61,321)	0.559	0.877	0.835	0.827
5	(54,76,470)	0.813	0.875	0.736	0.786
6	(72,144,384)	0.791	0.724	0.83	0.79
7	(102,234,264)	0.719	0.559	0.866	0.745
8	(132,258,210)	0.659	0.566	0.909	0.77
9	(168,300,132)	0.623	0.539	0.945	0.774
10	(180,360,60)	0.584	0.479	0.981	0.746
11	(30,36,534)	0.908	0.946	0.729	0.808
12	(93,185,322)	0.749	0.683	0.821	0.771
13	(1,281,318)	0.998	0.581	0.756	0.717
14	(1,281,318)	0.997	0.573	0.755	0.715
15	(1,281,318)	0.999	0.523	0.784	0.717

are consistence with LRU policy as well (tables 5.9 and 5.13). By the way, the performance of such routers is always less than edge routers because of filter effect introduced in Chapter 3(i.e., edge routers filter the most popular files and for not edge routers the difference between file popularities are not much). Having less popular files in cache of a router reduces its cache efficiency.

One interesting result from comparing table 5.12 with table 5.8 (or tables 5.9 and 5.13) is that dynamic partitioning reduces P_2^{miss} about 40% for edge routers (i.e., for $X1$, probability of CS miss for class 2 traffic with shared CS is 0.728 whilst its 0.457 with partitioning). Recall that the Zipf α parameter is the highest for the second class of traffics. Thus this class has the most popular files and to improve cache performance it is better to keep the most popular files in the cache. So partitioning the CS could save data chunks of the traffic class 2 (the most popular files) from interference of other traffic flows. At the same time using our dynamic partitioning scheme we could keep aggregate P^{miss} minimum.

Table 5.7: Results of dynamic partitioning for router $Y1$. The first 12 epochs are for calibrating the equation. In epoch 13, the equation is used to determine the partition that minimizes aggregate P^{miss} (**LRU** replacement).

Epoch	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
1	(226,307,67)	0.437	0.450	0.912	0.735
2	(18,422,160)	0.891	0.405	0.847	0.716
3	(5,270,325)	0.964	0.489	0.767	0.702
4	(15,330,255)	0.908	0.434	0.803	0.701
5	(64,296,240)	0.778	0.434	0.813	0.694
6	(20,276,304)	0.906	0.526	0.795	0.728
7	(93,185,322)	0.698	0.631	0.786	0.733
8	(218,61,321)	0.428	0.825	0.755	0.744
9	(54,76,470)	0.821	0.769	0.721	0.745
10	(72,144,384)	0.737	0.687	0.781	0.748
11	(102,234,264)	0.656	0.497	0.789	0.684
12	(180,360,60)	0.492	0.359	0.936	0.726
13	(85,284,231)	0.731	0.4546	0.788	0.683
14	(85,284,231)	0.693	0.474	0.765	0.670
15	(85,284,231)	0.705	0.464	0.799	0.687

Table 5.8: Results of dynamic partitioning for all routers with Random policy

Router	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$X1$	(1,281,318)	0.999	0.457	0.866	0.732
$X2$	(15,301,284)	0.981	0.689	0.833	0.826
$X3$	(64,296,240)	0.848	0.705	0.879	0.835
$X4$	(1,360,239)	0.997	0.413	0.865	0.718
$Y1$	(5,270,325)	0.97	0.463	0.785	0.704
$Y2$	(33,264,303)	0.878	0.517	0.751	0.697

5.5 Fair Partitioning

The Equation 5.1 can be used in different ways to set the partition sizes. In previous Section the equation was used to find the partition sizes which results in minimum P^{miss} and in this Section we show how it can be used for fair partitioning.

Fairness can be defined in several ways and here we use the definition introduced in [83]. Their fairness is defined as equal number of reads per page of a process in hard disk (i.e., r value which is defined as $\frac{\left(P^{miss}+P_c\right)}{(P^*+P_c)}$). If there are n process then memory is divided into n partitions and to have a fair partition size they try to equalize the

Table 5.9: Results of dynamic partitioning for all routers with LRU policy

Router	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$X1$	(1,308,291)	0.997	0.598	0.875	0.789
$X2$	(264,84,252)	0.404	0.937	0.886	0.811
$X3$	(101,66,483)	0.820	0.942	0.767	0.814
$X4$	(1,191,408)	0.997	0.693	0.824	0.795
$Y1$	(85,284,231)	0.815	0.450	0.794	0.694
$Y2$	(1,243,356)	0.995	0.509	0.736	0.688

Table 5.10: Partition size and CS miss probabilities of router X1 after initiating the partition sizes for the second time. Replacement policy is Random

Router	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$X1$	(1,300,299)	0.997	0.440	0.860	0.733

number of times a page is going to be brought to the memory. Here we can translate each process to one of the traffic classes.

Assume the memory is already partitioned to n classes with size $M_1...M_n$, and to reach the fairness we need to trim $\triangle_i$ from M_i and $\triangle = \triangle_1 + ... + \triangle_n$. To find the proper $\triangle_i$, [83] found the formula:

$$\triangle_i = M_i - \frac{M_i'}{\sum_{j=1}^{n} M_j'} \left(\sum_{j=1}^{n} M_j - \triangle \right) \tag{5.3}$$

$$M_i' = M_i^* - M_{ib}$$

where M_i the memory size that process i has and M_i' is what the process i needs.

Since in CS we want to use the whole space (no change in total cache size), $\triangle = 0$. Equation 5.3 specifies the amount of memory ($\triangle_i$) to remove from partition i to make the partition fair. If the partition is already fair then $\triangle_i = 0$ and the formula becomes:

$$M_i = \left(\frac{M_i'}{\sum_{j=1}^{n} M_j'} \right) \sum_{j=1}^{n} M_j$$

M_j on the right hand side includes M_i, so this looks like that M_i on the right

Table 5.11: Partition size and CS miss probabilities of router X1 after initiating the partition sizes for the second time. Replacement policy is LRU.

Router	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$Y1$	(1,259,340)	0.995	0.599	0.843	0.768

Table 5.12: Results for shared CS (no partitioning) with Random policy

Router	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$X1$	0.935	0.728	0.758	0.765
$X2$	0.953	0.804	0.815	0.835
$X3$	0.941	0.816	0.824	0.842
$X4$	0.933	0.734	0.815	0.798
$Y1$	0.902	0.68	0.71	0.719
$Y2$	0.906	0.716	0.707	0.731

hand side should be known in order to calculate M_i on the left hand side. However, since $\triangle = 0$:

$$\sum_{j=1}^{n} M_j = M_1 + ... + M_n = M$$

i.e.

$$M_i = \left(\frac{M_i'}{\sum_{j=1}^{n} M_j'} \right) * M \tag{5.4}$$

Now the right hand side does not have M_i and the M_j' is already known from the curve fitting. So it is straightforward to calculate $M_1, ..., M_n$, to get the same r value.

In other words, in order to have fair partitioning regards to r value, all partitions (in our example the three partitions) should have the same $\frac{\left(P^{miss}+P_c\right)}{(P^*+P_c)}$ (in our example, $\frac{\left(P_1^{miss}+P_1^c\right)}{\left(P_1^*+P_1^c\right)} = \frac{\left(P_2^{miss}+P_2^c\right)}{\left(P_2^*+P_2^c\right)} = \frac{\left(P_3^{miss}+P_3^c\right)}{\left(P_3^*+P_3^c\right)}$).

In order to make a better comparison some of the simulation parameters have been modified. To see if the partitioning is fair we needed one of the traffic classes to bully the others. The CS size lessened to 550 packets to increase the race for memory space. Class 3 is set to generate the bullying traffic. The inter-request file time is half for this class and the rest of the configurations are presented in table 5.14.

Figure 5-4 presents the results of the experiments for shared CS and partitioned CS using the Equation 5.4 and the measured fairness parameter $\frac{\left(P^{miss}+P_c\right)}{(P^*+P_c)}$ for shared

Table 5.13: Results for shared CS (no partitioning) for with LRU policy

Router	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$X1$	0.968	0.825	0.789	0.817
$X2$	0.974	0.830	0.830	0.854
$X3$	0.976	0.786	0.830	0.843
$X4$	0.954	0.740	0.834	0.814
$Y1$	0.907	0.687	0.705	0.720
$Y2$	0.913	0.670	0.720	0.734

Table 5.14: Characteristics of the three traffic classes

	Class 1	Class 2	Class 3
file size	10	60	120
Zipf α	2.5	1.6	1
Traffic	nonCBR	nonCBR	CBR

CS varies from 1 to 6.4 for the 3 traffic classes whilst this range for fair partitioning is from 1 to 2. As mentioned earlier to have a fair partitioning the parametric values of the partitions should be equal (or closer together). The result is for router $X1$ and the partition sizes from equation are $(52, 77, 421)$ for partition 1, partition 2 and partition 3 respectively.

5.5.1 Extension

In this Chapter we have shown that the cache miss equation is effective in finding the proper partition sizes of a CS to reach different goals (e.g., minimum CS miss probability and fairness).

Still the equation can be used to study the effect of partition on other aspects of the network. These parameters can be different classes of traffics (e.g., text, video, image, social network, etc.), or different measurements like latency or bandwidth.

One of the most important issues is the effect of partitions of one router on other routers. So far, we used the equation to find the minimum CS miss probability for only one router. However, the minimum CS miss rate can be calculated for the whole network. In this scenario a partition size might not result in minimum CS miss rate

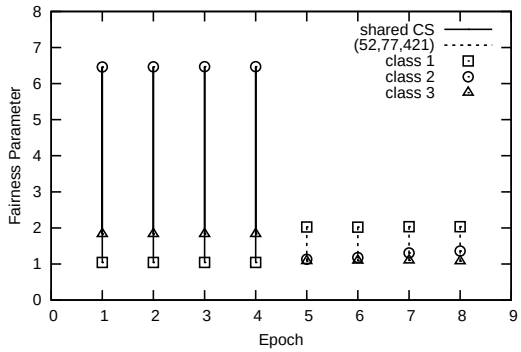


Figure 5-4: The first four epochs are for shared CS and the second four epochs are after partitioning the CS. the parametric values for the three partitions are much closer together when we have fair partitioning. Results is for router $X1$.

Table 5.15: The effect of partitioning of the edge router X1 on the core router X2. Router X1 does not cache traffic class 1. That allows router X2 to improve its cache hit rate by setting a larger partition size for this traffic class.

Router	Partition size	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
X1	(1,308,291)	0.997	0.598	0.875	0.789
X2	(264,84,252)	0.404	0.937	0.886	0.811

for a particular router but overall we might reach to the minimum miss rate for the whole network.

We have already seen the filter effect of edge routers on core routers (see Sec. 6.6.5). Because of this effect, the Interest packets arrive at the core routers do not have popularity property as the popular requests are already filtered by the edge routers. Without popularity the effect of caching system reduces significantly. One way of dealing with this issue could be using partitioning. An edge router does not respond to all popular contents and the partition size for a particular class can be very small (or zero) to let the next router handle this class of traffic. In this way by increasing the performance of CS of the core routers, we might be able to increase the performance of the whole network.

Through a simple experiment we observed the effect of partitioning on the performance of the two edge and core routers.

As it is shown in table 5.15, when router X1 does not cache traffic class 1, router

Table 5.16: Results for shared CS (no partitioning) for with LRU policy

Router	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
$X1$	0.968	0.825	0.789	0.817
$X2$	0.974	0.830	0.830	0.854

X2 has a better chance to respond to this type of traffic by setting a bigger partition size for this traffic class. The hit rate for the two routers can be compared with table 5.16 which is the results for the same routers but without memory partitioning scheme.

Chapter 6

A New Router Design

Throughout the previous chapters we studied the performance of network of caches mainly focused on dependencies and correlation among the nodes. Elimination (or reduction) of the correlation between nodes has two positive impact:

- Improves the performance of the entire network by increasing the performance of the core routers,

- Performance of the whole network of caches can be modeled by a simple mathematical equation.

Another major problem of NDN design is its router architecture. There is some skepticism that NDN architecture can match link speed. Router memory must be fast enough to match line speed and sufficiently large to effectively cache data chunks and host the routing information for them. Having a fast and large memory is very expensive. So by introducing a new router design, we try to reduce the packet forwarding delay using the slower memory technology.

In Section 6.2, we propose a new memory arrangement in an NDN router including its proper replacement policy and forwarding algorithm to reduce Interest packet processing time in the router.

From the experiments we have observed that even if we could manage to reduce the Interest packet forwarding delay, still the forwarding scheme suffers from huge queuing delay. Long queue length is mainly caused by caching all the Data chunks coming into the router in response to the Interest packets.

To rectify the queuing delay problem we use CCndnS, the cache policy proposed in Chapter 3, to reduce the memory queue length by selectively cache contents.

Before delving into our proposed architecture, we look at some difficulties of the pipeline design of forwarding architecture of NDN through the next Section.

6.1 Pipeline Design

Since a real NDN hardware design has not been disclosed so far, we make our discussion based on the proposals in the literatures and on NFD platform, the Name Forwarder Daemon published recently [3].

The main architecture of an NDN router consists of three tables: (1) a Content Store (CS) that caches Data chunks, (2) a Pending Interest Table (PIT) that records the incoming interface for an Interest, and (3) a Forwarding Information Base (FIB) that records the outgoing interface for forwarded Interests.

There are other tables to handle flow of the packets introduced by NFD. Since we do not want to focus on a particular design, we just concentrate on the essential tables in the NDN architecture.

As shown in Figure 6-1 an arrived Interest packet to a router first searches CS for already cached Data chunk in the router. In case of finding the corresponding Data, the Data chunk will be returned to the requester. Otherwise in the second step PIT table will be examined by the Interest packet. In case of PIT hit, Interest packet just updates the PIT entry by adding its incoming interface address to the existing PIT entry and then the packet will be dropped. PIT hit means this Interest packet had already been forwarded from this router and by receiving the Data chunk, it will be

forwarded to the all interfaces found from PIT. Finally if the Interest packet address does not match with any entry in PIT (miss), the packet will be sent out to proper interfaces using routing information provided in FIB. To do that, the Interest packet goes through a small and fast on chip memory (FIBon) to obtain longest prefix match (LPM) first. Using the LPM the main FIB table which is a large and slow memory and placed in an off-chip memory (FIBoff) determines the interface(s) which gets this Interest packet closer to the source of its Data [41].

Note that although there are ongoing argument about NDN naming system, we assumed the hierarchical naming system proposed in [41]. Using hierarchical naming system, the routing table size significantly reduces by holding the routing information only for the longest prefix name of contents.

When the idea of content networking is reintroduced in [41], it was assumed that everything (all tables and their indices) are implemented in a single memory (i.e., in the buffer of the interfaces). Since the forwarding architecture is prone to memory latency, in other proposals [5], [62] the delay of memory lookup tried to compensate by using different memory technologies. A hybrid memory technology was introduced to reduce the delay while keeping the price of the memory affordable. We choose the hybrid memory technology to show that even with such complicated designs, still the forwarding pipeline of NDN remains troublesome.

Another concern is the time complexity of processing the tables. In the latest document on how NDN forwarding architecture works [3], PIT is the first table to search before searching CS (contrary to the original design suggested in [41]). In addition, there are two other tables (Strategy Choice table and Measurement table) introduced. The detailed pipeline path for Interest and Data packets are described as well.

Both PIT and CS can filter part of the passing traffic through the router. PIT can aggregate similar Interests initiated from different clients. On the other hand CS can satisfy some Interests looking for a cached data inside the router.

Which table is better to be probed first is not known yet however, in case of using a hybrid memory technology it is trivial to use the faster memory as the first table to probe.

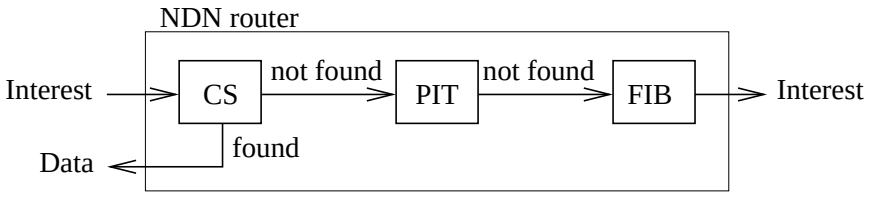


Figure 6-1: The sequence of memory unites in an NDN router.

Memory technology and its latency defines the forwarding delay. However, the memory lookup delays can be overlapped using a careful pipeline design. Nevertheless, the performance of a pipeline highly depends on the nature of the task. To design a perfect pipeline, a task should acquire two main properties:

- The task should be able to be broken to completely disjoint subtasks.
- The processing delay of each subtask should be relatively equal.

NDN pipeline refers to looking up at each functional tables. Therefore, it can be assumed that they are disjoint. The forwarding pipeline of NDN forwarder is explained in [3] in details. The big picture of forwarding pipeline and strategy of NFD forwarder is presented in Figure 6-2. Each blue box represents a pipeline and the white boxes represent decision points. The diagram in Figure 6-2 consists of Interest and Data incoming pipelines. Pipeline is a series of functions on a received packet which activates by an event. NFD separates Interest processing into the following pipelines:

- incoming Interest: processing of incoming Interests
- Interest loop: processing incoming looped Interests (the same packet meets a router again)

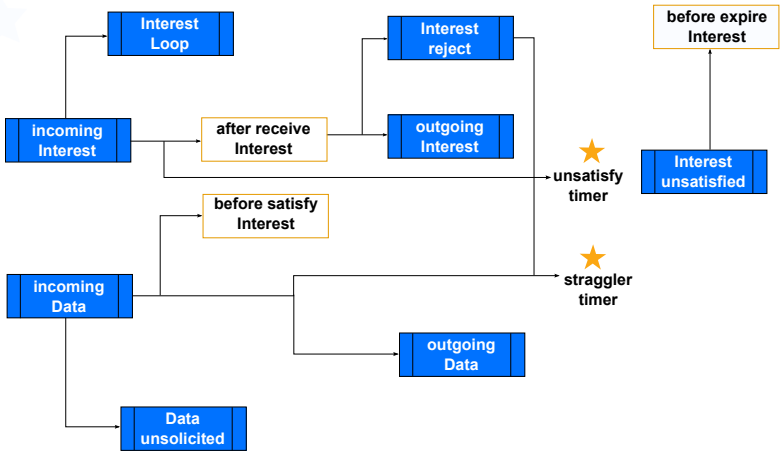


Figure 6-2: Forwarding pipeline of NFD [3]

- outgoing Interest: preparation and sending out Interests
- Interest reject: processing PIT entries that are rejected by the strategy
- Interest unsatisfied: processing PIT entries that are unsatisfied before all downstream entries for the same Interest timeout

Based on the predefined rules in strategy plan, different events might be trigged for different Interests.

Data processing in NFD is split into these pipelines:

- incoming Data: processing of incoming Data packets
- Data unsolicited: processing of incoming unsolicited Data packets
- outgoing Data: preparation and sending out Data packets

Unsolicited Data refers to the received Data packet from an interface which does not have any track in PIT. Usually such Data packets are considered as threat and are dropped. For detailed information read NFD guidelines [3].

One problem with this system is that although it seems there are two disjoint pipeline systems for incoming Data and Interests, but actually both Data and Interest

packets are accessing the same tables to lookup or update an entry there. That increases the dependency among the pipeline stages.

Before looking at the next problem of the pipeline design of NDN forwarder, note that basically a pipeline is for increasing the throughput of a system by increasing the throughput of each functional unit and by overlapping their latencies. If a pipeline stage waits for an input (which is the output of the previous stage) the efficiency of the pipeline is largely compromised and it is considered as one of the major issues of a pipeline design.

Therefore, in an optimum pipelined design all pipeline stages should work without any stalls. In other words, all pipeline stages should have the same incoming rates. Figure 6-3 shows the details of the pipeline for incoming Interest in Figure 6-2. We present this pipeline as an example of how the design is actually ineffective. Only Interests received from one of the interfaces is valid. If an Interest received from local-host it should be dropped. For a valid Interest, an entry will be added to the PIT. If it is a looped Interest, the Interest Loop pipeline will be triggered. Otherwise the Nonce value which is an increasing number to detect loops, is recorded and the entry in PIT is checked whether this is a new entry or pending entry. CS will be checked for each new entry of PIT. If Data can be found from CS then the outgoing Data pipeline will be trigged and the entry removes from PIT. Otherwise and if the entry in PIT is pending, the PIT timer will be reset and FIB will be searched for proper outgoing interfaces.

We see that each pipeline stage requires access to different tables with (perhaps) different lookup time complexity. Even if we assume that all tables are implemented in the same memory technology with a universal lookup algorithm (their lookup time complexity are the same), still there are many decision points before an Interest packet can reach to FIB. Meaning the incoming rate to the FIB is much less than the incoming rate to the interface.

We deal with these inefficiencies in following sections. The main ideas are:

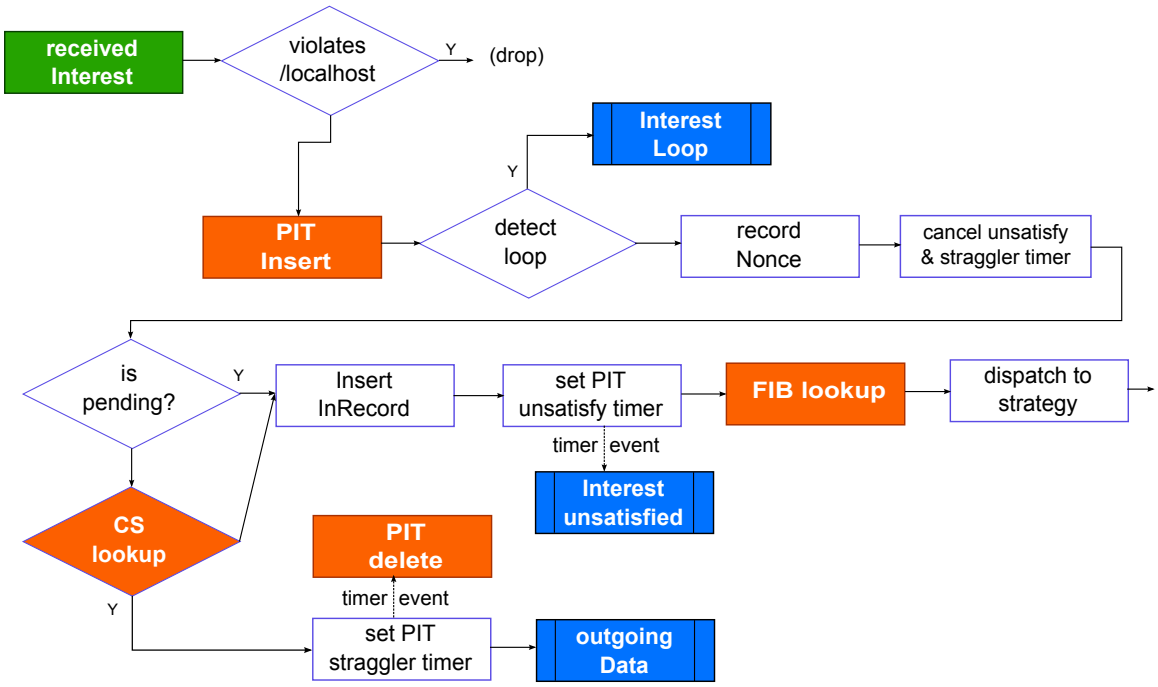


Figure 6-3: The Interest Pipeline [3]

1- Make a better decision on searching CS by using CCndnS.

2- In a parallel fashion, we inject a packet to all related functional units.

Doing so the result of the table lookup would be ready regardless of decision points. Meaning that if a packet (for instance an Interest packet) needs the result of later stages, the result would be ready otherwise the result can be discarded.

In this chapter we try to meet the goal of pipelining which is having an output after each d time slots where d is the processing time of the slowest pipeline stage. To do that we need to analyze the system for its bottlenecked component:

- It is believed that obtaining data from a router's memory improves the packet latency and so users' satisfaction. This might be true for cacheable contents. Nevertheless, many applications like live streaming or voice and video communications do not generate cacheable contents. Even for cacheable contents, access data at the routers' cache costs searching the entire cache for all Interest packets. If data chunks can be found from a closer node to the requester, total

download time might be reduced. However, cache misses impose cache search delay on the packets. Therefore, search time for accessing a data chunks from a further router (or source) might have even negative impact (larger total download time). Our experiments in Section 6.3 show that the CS hit probability is insignificant and cache misses adds huge delay on each packet (probably this is even worse in a real network as the requests are much more diverse). Two main influential factors for CS hits are:
(1) popularity of Data chunks and
(2) their life time duration in CS.
Other parameters like replacement policy (selection of a data chunk from CS to be replaced with a newly arrived data), cache policy (which data should be cached in which router), distance to the requesters etc. directly or indirectly affect the above factors.

In order to increase the cache hit probability, their life span in CS should be prolonged. However, considering the memory size of the current routers and the amount of data available on web, Data chunks might not get any hit before replacement. Thus, a large memory size for CS is essential. On the other hand, the memory technology for CS should be fast enough to prevent performance reduction in case of cache miss. Hence, to maintain an acceptable level of cache performance, CS should be fast and large. Considering the cost of the fast memories, this solution might be unrealistic. Using slower memory technology makes CS as one of the major bottlenecks in the NDN router.

- PIT table is the only table which is not size-oriented. PIT is going to keep track of Interest packets whilst they are waiting for Data chunk. So the maximum life time of an entry in PIT is equal to the round trip time of an Interest which left the router until the corresponding Data arrives (however, PIT entry might be dropped by a timer for congestion control). Therefore, PIT size can be calculated for a particular router with a specific link bandwidth.

If the average round trip time of a packet in the network and the total incoming rate to a router is known the PIT size can be found from Equation 6.1 and 6.2.

$$Average\ round\ trip\ time = \bar{t}$$

$$Aggregateincoming\ rate\ from\ a\ link = \lambda$$

$$Number\ of\ links\ to\ a\ router = l$$

If there is only one PIT in a router (as another implementation, each interfaces can have a separate PIT), maximum PIT size is:

$$Maximum\ PIT\ size = \sum_{i=1}^{l} \lambda_i * \bar{t} \tag{6.1}$$

If there is one PIT at each interface then:

$$Maximum\ PIT\ size\ for\ interface\ i = \lambda_i * \bar{t} \tag{6.2}$$

Since the size of PIT can be easily engineered, we assume that it can be implemented by fast memories. Considering that, we try to match the other tables' (i.e., CS and FIB) latency close to PIT.

- FIB collects routing information. It has to be large to match the enormous number of content names on the Web. Therefore, slow memory technologies might be used in its design for affordable implementation. That makes it a potential bottleneck.

Table 6.1 shows that the most suitable memory technology for CS and FIB is DRAM which is large and cheap. Non-volatile memory (like flash) can be larger and cheaper, but we exclude them since they will not last when written at line rate [36].

technology	access time (ns)	maximum size	cost ($/MB)	power (W/MB)
SRAM	0.45	∼210Mb	27	0.12
RLDRAM	15	∼2Gb	0.27	0.027
DRAM	55	∼10GB	0.016	0.023

Table 6.1: Summary of memory technologies [62]

However, calculations show that DRAM latency may not match line speed [62], thus making CS and FIB the router bottleneck.

6.2 ndn‖mem

Thus, we propose ndn‖mem (ndn with the parallel memory architecture shown in Figure 6-4), a redesigned model of the NDN router to cope with the line speed problem whilst using the slower memory technology. There are four ideas in our ndn‖mem design:

(1) A parallel search of the three memories (CS, PIT, FIB).

(2) A FIB cache for the Interest's longest prefix match (LPM) [106].

(3) Using CCndnS (see Chapter 3) cache policy.

(4) A droptail replacement policy for chunks from the same file.

In the following, we describe each idea in corresponding subsections.

6.2.1 Parallel Search

Unlike conventional TCP/IP protocol in which a packet needs to be processed only once in each router, in NDN architecture a packet must be processed at least three times (considering LPM makes it four times) using three different memory units. Therefore, forwarding algorithm of an NDN router is prone to memory latency.

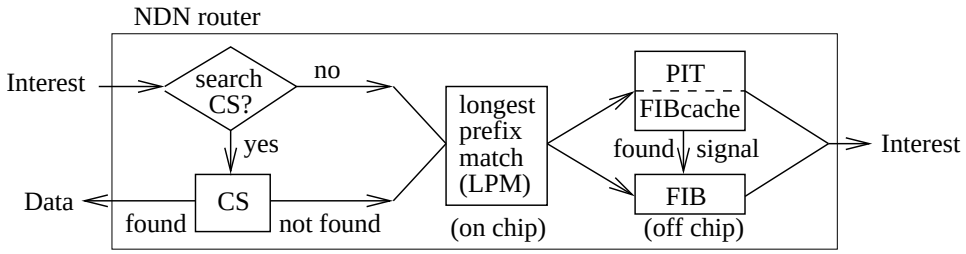


Figure 6-4: The ndn||mem architecture. It allows Interests to bypass CS, and possibly abort a visit to FIB.

The packet latency in a router like Figure 6-1 is the summation of all the three memory latencies ($packetdelay = CSdelay + PITdelay + FIBdelay$). To rectify the problem we suggest the parallel search on the three memory units. In this way, the packet latency would be equal to the slowest memory unit (the bottleneck component). Of course in this way we compromise the throughput of the non-bottleneck components. Since all the three memories are going to be processed at the same time for one packet, the memory units that finish their job earlier should wait for the slower memory unit before they can process the next packet.

We have seen that CS and FIB are the two major bottlenecks for an NDN router. In the following Sections by introducing new techniques we improve the parallel idea to reduce the effect of the bottleneck components.

Section 6.6.1 will present a performance comparison of the two architectures (serial and parallel).

6.2.2 PIT/FIBcache

As mentioned earlier, FIB should be huge enough to cover as many content name as possible (the best case is equal to the all of the names on the web). Therefore, to keep the price low, it has to be implemented with slow memory.

In order to maintain the FIB search time low, we introduce a faster FIBcache. FIBcache can be faster as it is smaller. Entries in FIBcache need to remain in the memory for shorter time (i.e. equal to the time needed to download a complete file).

Assuming that there is at least one entry in PIT for all ongoing downloading files, the maximum number of entries in FIBcache could be equal to the maximum number of entries in PIT table. Of course PIT hit does not count in measuring FIBcache size as those packets just update PIT and are then dropped (they do not need routing information).

Since the size of FIBcache is not larger than PIT, it can be implemented with the same memory technology as PIT. Duration of entries in PIT (which is equal to the round trip time of a packet) is small so it can be implemented with a fast memory (e.g., RLDRAM).

Using the same memory technology, we could merge the two FIBcache and PIT tables into one table and perform a parallel search on both for an arriving Interest packet (Figure 6-5). There are other researches trying to reduce FIB latency like [87] which can be applied here as well. Reducing FIB latency definitely improves the performance of the system by improving one of the bottleneck components.

Correlation between chunk sequences of a file is a good opportunity to cache the routing information of the first chunk of the file for the use of the subsequent chunks of it. One of the main assumptions we make is that most likely chunks of a file can be found from the same node [68]. Hence, routing information of the chunks of a file should be the same. Therefore, by keeping the routing information of the first chunk of a file in FIBcache we save ($number\ of\ packets\ in\ a\ file\ -1$) FIB searches.

The PIT/FIBcache in Figure 6-4 has a PIT table and a FIBcache — see Figure 6-5. PIT/FIBcache acts upon received Interest packet as follows:

- An Interest arrival prompts a parallel search of these two tables: the longest prefix match in FIBcache, and the remainder of the address in PIT.

- If an LPM entry is found in FIBcache, then the Interest is forwarded via the indicated outgoing interface if and only if the remainder of the address is not found in PIT. Either way, a signal is sent to stop the off-chip FIB search.

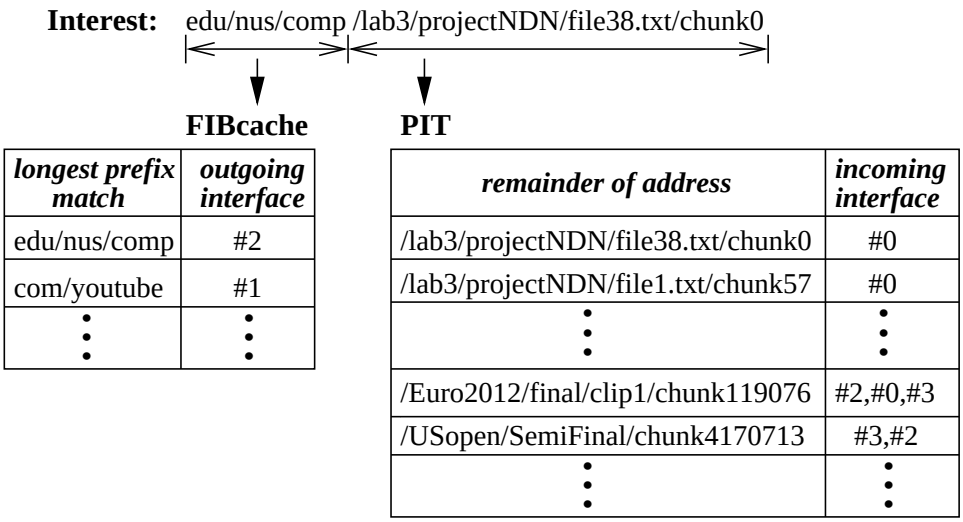


Figure 6-5: PIT/FIBcache structure: Each longest prefix match (LPM) is in FIBcache, and the remaining addresses for the Interests are in PIT. The two tables are searched in parallel; a FIBcache hit sends a signal to stop the concurrent FIB search.

- If an LPM entry is not found in FIBcache, the creation of a new FIBcache entry for LPM must wait for the FIB search to finish with a decision on the outgoing interface(s).

- In any case, PIT records the Interest's incoming interface (in an existing entry or a new entry).

In this design, PIT hit translates to hit entries at both FIBcache and PIT tables. The entry in FIBcache remains longer than entry in PIT table. PIT entry will be removed by arrival of the corresponding Data packet (after round trip time $[\overline{t}]$). However, FIBcache entry needs to stay there for the entire download time of the file (i.e., $\overline{t} * file\ size$).

Search on FIBcache always finishes before FIB as we assumed that memory technology used for FIBcache is much faster than FIB. Having FIBcache with slower or even the same memory speed spoils the idea of having FIBcache.

Section 6.6.2 will present an experiment to demonstrate the efficacy of the FIBcache.

6.2.3 Using CCndnS to Decide CS Search

In order to have reasonable performance from CS, it should be large and therefore slow to keep the cost low. Consequently, cache misses would be so expensive for packet latencies. In vanilla NDN design, an Interest packet searches all the CSs during the journey to the source of the content. If an Interest packet would not be satisfied from one of the CSs, it suffers huge delay from searching all the CSs in the path. Even in the parallel architecture, CS is the major bottleneck in the routers. Although the memories will be searched concurrently, still all the packets should go through the bottleneck component.

Note that there is no guarantee to implement CS index table with a faster memory. It is possible that the CS index table would be placed in the same memory as the CS table. In an alternative approach, CS index table might be implemented with a faster memory (this is what we considered in our experiments). Although CS index table might not be the bottleneck, it is an extra unnecessary delay to those Interest packets that get miss from CS (it will be shown that CS miss ratio is huge).

Hence, to do better a router should decide to search its CS for an arrived Interest packet or just bypass the CS. This means, CS is not going to be searched by all arriving Interest packets. This idea forced us to remove the CS from the parallel search and make it pipelined with the other two tables. Doing that, whilst PIT/FIBcache and FIB are going to be concurrently searched for an Interest packet (which does not need to search CS), in a pipelined fashion CS can be examined by another Interest packet (which has been decided to search CS).

The key issue here is that the decision algorithm must be fast to not impose another delay to the packet processing time. On the other hand, the algorithm should be precise enough to not reduce the cache efficiency by bypassing the CS whilst data was there.

Our basic decision idea is that an Interest is more likely to find its Data in CS if previous Interests for the same file $\mathcal{F}$ succeeded in doing so. This is equivalent the

CCndnS idea proposed in Chapter 3. In short CCndnS does:

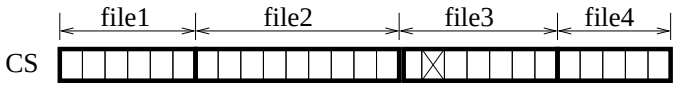
- For chunk 1, always search CS;
- for all chunks in segment k, search CS if the CS is supposed to cache segment k (of $\mathcal{F}$);
- skip this CS if the CS does not cache segment k.

For more information on how a requester finds the hop distance of the router where a segment caches there, see Section 3.2.1.

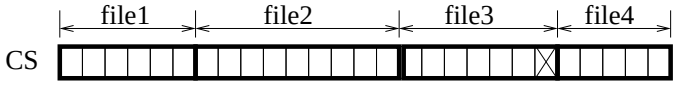
CCndnS implementation requires just a simple comparator, and two bytes in the chunk header (sufficient for 256 hops).

Since the proposed method is using hop count to find the node with the data, some doubts might raise:

- Forwarding an Interest packet to multiple interfaces in a router brings multiple different hop distance! This is not a problem as only the first arrived Data chunk (including hop distance) can consume PIT entry and the rest of the Data chunks for that particular Interest will be dropped on that router.
- A router might decide to send an Interest packet to a wrong interface which will end up reaching the source of the file whilst Data was there in another path. This might happen by a bad routing strategy. By the way if the router chooses a wrong path, even searching all CS in the path would not solve the problem.
- A content might get closer to the requester upon requesting by another client but our algorithm would fail to get that! We addressed this issue in Section 3.2.1. Briefly we can say, since the first Interest of all segments search all CSs along the path the chance of happening this is very small. However, out of order requesting of chunks of a file might confuse LRU replacement policy and make this problem happen. A tailored replacement policy for our architecture is introduced in Section 6.2.4.



(a) Arbitrary replacement of a chunk can create a gap.



(b) A droptail replacement policy preserves Data contiguity.

Figure 6-6: A cross indicates a replaced chunk. In (a), a replacement of an arbitrary chunk can cause Interest for subsequent chunks to skip this CS (although their Data are there). The droptail replacement in (b) avoids this problem.

Section 6.6.3 will present an experimental evaluation of this idea.

6.2.4 A $\langle \mathcal{P}_{\text{file}}, \mathcal{P}_{\text{chunk}} \rangle$ Replacement Policy for CS

The main intuition of skipping part of CCndnS is on how a router assumes chunk $k+1$ will not be in CS if chunk k was not there. The CS cache replacement policy should not contradict this policy. For example, if CS is full in Figure 6-6(a) and chunk2 of file3 were replaced, then an Interest that finds that chunk missing will reset the hop count $h_{\mathcal{F}}$ for subsequent Interests, causing them to skip this CS even though their Data are there.

CS performance therefore depends on the chunk replacement policy. We denote a replacement policy $\mathcal{P}$ by

$$\mathcal{P} = \langle \mathcal{P}_{\text{file}}, \mathcal{P}_{\text{chunk}} \rangle,$$

where $\mathcal{P}_{\text{file}}$ is a policy for choosing a file $\mathcal{F}$ that has chunks in the CS; $\mathcal{P}_{\text{chunk}}$ is a policy for choosing a chunk of $\mathcal{F}$ for replacement. For example, $\langle$ LRU, random $\rangle$ is a policy that

- uses the least recently used (LRU) policy to pick a file victim $\mathcal{F}$ from those in CS
- then replaces a randomly chosen chunk of $\mathcal{F}$.

To avoid creating a chunk gap illustrated in Figure 6-6(a), ndn‖mem adopts a **droptail** policy for $\mathcal{P}_{\text{chunk}}$, i.e. the last chunk of a file victim is chosen for replacement. This is illustrated in Figure 6-6(b).

In our experiments, we consider $\langle LRU, droptail \rangle$, $\langle random, droptail \rangle$ and $\langle random, random \rangle$.

6.2.5 Architecture Summary

We introduced the parallel memory architecture for an NDN router to reduce the packet forwarding delay.

To reduce the burden of searching slow CS for all Interest packets, we exploit the hop count algorithm to search the CS of a router only for the Interest packets with higher probability of finding Data chunk there. To do that, the CS is removed from parallel search.

As FIB is a huge table implemented with a slow memory, the delay of a parallel search of the two PIT and FIB tables, would be the delay of FIB. Note that duration of entries in PIT (which is equal to the round trip time of a packet) is small. So, the probability of PIT hit is negligible (except for live streaming applications). Thus, most of those Interest packets that got miss from CS or bypass it will go through FIB table.

Therefore, an extra memory table named *FIBcache* which is smaller and faster than FIB table was added to a router to reduce the delay of FIB search. FIBcache is in the size of PIT table and can be merged with it.

The overall footprint of a packet in an NDN router would be as follows:

- Based on the hop distance information provided in the header of the packet, router decides whether to search the CS for the packet or not.

- If the router decides on searching the CS, presence of data will be checked by looking at CS index table.

- In case of CS miss or the router decides on not searching the CS, the Interest packet leads to the LPM table.

- Then the LPM of the name is sent it to the parallel tables (PIT/FIBcache and FIB).

- If the Interest packet finds a same entry in PIT, PIT will be updated with the packet's interface name and the packet will be dropped.

- In case of PIT miss, if the routing information is found from FIBcache, search on FIB will be aborted and the packet will be sent out using the obtained routing information.

- In case of FIBcache miss, FIB will decide (based on its routing information or defined routing strategy) on forwarding interfaces.

Ignoring the FIBon delay (SRAM is so fast), the worst case search latency (which is rare and less than 4% in our experiments) is CS index plus FIB. In the best case, the Interest bypasses CS (see Section 6.2.3) and the search time is just PIT/FIBcache latency.

6.3 Evaluation of the New Architecture

In this chapter we evaluate the effectiveness of our ideas in improving router forwarding delay. To serve this purpose, we implemented our event-driven simulator written in Java. Since this is the first study at the memory queue level for NDN routers, we could not find any convenient NDN simulator for this purpose.

We first specify the set-up for the simulation analysis of ndn‖mem and CCndn, then discuss the performance metrics.

6.4 Simulator Parameters

We designed the topology in Figure 5-1 so it has cross traffic, multipathing, and a mix of edge and core routers. This topology may look small but, in fact, we have found no other paper that simulates memory events at multiple routers. (Some experiments take days to finish because they simulate a huge number of events at nanosecond granularity.) In any case, Section 6.7 will present results from simulating a larger Abilene-like topology.

To help the reader remember the topology, we label 4 routers X_1, X_2, X_3 and X_4 along an "x-axis", and 2 routers Y_1 and Y_2 along a "y-axis". X_2 and X_3 are **core** routers, and the others are **edge** routers. To keep simulation time tractable, we fix the number of clients at 50 for each edge router.

The main control variable is λ_{file}, the file request rate per client, which is often pushed beyond 2000 files/sec. Each client is thus multiplexing requests from many users in some subnetwork, and we change the number of users by changing λ_{file}. The inter-file request time is exponentially distributed with rate λ_{file}. We consider two kinds of traffic:

(CBR) For constant bit rate traffic (CBR), a client $\mathcal{C}$ sends Interests for consecutive chunks of a file $\mathcal{F}$ with constant inter-request time (1500ns); i.e. $\mathcal{C}$ requests for chunk $k+1$ regardless of whether it has received chunk k.

(nonCBR) For nonCBR traffic, $\mathcal{C}$ sends Interest for chunk $k+1$ of $\mathcal{F}$ only after $\mathcal{C}$ has received Data for chunk k.

For nonCBR, Y_1 forwards the Interest for the first chunk of a segment to both X_2 and X_3; if Y_1 receives the Data from X_j, Y_1 will forward subsequent Interests for that segment to X_j only. For CBR, Y_1 may receive multiple Interests for a segment before it gets the first Data chunk; in that case, Y_1 forwards them all to both X_2 and X_3, until it receives the first Data. Similar remarks applies to Y_2, X_2 and X_3.

A client that has not finished downloading a file $\mathcal{F}$ will not again request $\mathcal{F}$; otherwise, caching at the edge router will hide network performance. Clients do not

cache chunks, so their Interests are always sent to edge routers. To keep the simulation time tractable, we fix the file size at 100 chunks.

For ease of implementation, each client (recall: it represents many users) is the source for one file, and can request a file from any client not attached to the same edge router. File popularity has an obvious impact on caching performance [73], and the commonly used model is Zipf. Unfortunately, there is disagreement in the literature on what the exponent α should be. We therefore use $\alpha = 1$, i.e. the n-th most popular file is accessed with probability proportional to $1/n$,

CS size is a fraction of catalog size (200×100chunks). Again, there is no agreement in the literature on what this fraction should be [73], so we fix it as 0.015 (i.e. 300 chunks for CS size); we will see (in Figure 6-21) that ndn‖mem's router latency is robust with respect to this choice. $SegSize$ is 20 chunks, so each CS can contain 15 segments. An Interest is 500Bytes, and a Data chunk is 9KBytes, so the default file size is 0.9MBytes.

We will later examine the performance impact of CS size and file size.

We do not limit the PIT and FIB sizes, to avoid introducing some packet dropping scheme that may influence the results.

Our focus is on the memory bottleneck, so the most crucial parameters are the memory latencies, as specified in table 6.1. For the simulation, we use SRAM for LPM, DRAM for CS and (offchip) FIB, and RLDRAM for CS index, PIT (for serial) and PIT/FIBcache (ndn‖mem). Link bandwidth is 300Gbps.

Our design for ndn‖mem and CCndn is orthogonal to how memory is accessed. An index for CS search [89], say, can be used for both ndn‖mem and the serial architecture. In the simulation, we assume that an Interest or a Data chunk accesses each of PIT, FIB and CS at most once. We will examine how this assumption affects the simulation results.

The default parameter values above will be changed in some of the following experiments.

6.5 Performance Metrics

ndn‖mem aims to relax the slow memory bottleneck, so the main performance metric is **router latency**, i.e. the time between when an Interest enters and leaves a router. This latency is unaffected by pipelining of Interests: it sums all delays that an Interest experiences at the CS index, PIT and FIB in a router.

For Interests that find their Data in the CS, **CS delay** measures all the relevant delays which include their queuing plus access time at CS index and CS itself. An Interest leaves a router because it skips checking the CS index (and skips CS), or the CS index shows its Data is not in CS, so router latency does not include CS delay.

These two measurements are going to be used for both serial and parallel architectures. The only difference is in serial architecture the router delay consists of CS index delay for all Interest packets whilst for parallel many Interest packets just bypass CS and do not have CS index delay (the delay for them is only PIT delay or FIB delay for few of them).

Router latency and CS delay are single-router metrics. For multi-router performance, we use the average **hop count**, i.e. the number of hops taken by an Interest before it finds its Data, and source hit rate (the rate that a source of a file is going to respond to Interest packets; it shows the effectiveness of in-network caching).

We use hop count to decide whether an Interest should bypass a CS without checking if the CS contains the named Data (Section 6.2.3). If n Interests enter a router and, among them, n' do not check the CS, we call n'/n the **skip fraction**. We really like to skip checking CS if data is not there and we will show that in our experiments the amount of skipping the CS happens in a parallel router is similar to the amount of cache miss happen in a serial architecture. That means we do a good job in not searching CS for that amount of Interest packets.

Among the n' Interests that skip CS checking, the Data for some n'' Interests may in fact be in the skipped CS. We measure this likelihood by the **skip error** n''/n'. Of course we want to reduce skip error as much as possible.

Some papers use the probability of **cache hits** to measure router performance [18, 25, 73]. In our setting, the input traffic to CS is different for the serial and parallel architectures, so it is *not meaningful* to compare their CS hits.

Even for ndn‖mem itself, it is not meaningful to compare, say, hit rates for $\langle LRU, droptail \rangle$ and $\langle random, droptail \rangle$, since these replacement policies have different skip fractions and skip errors.

The intuition for using CS hit rates is to compare effectiveness of caching. Since there is more than one router that can cache a desired Data, hop count is a better metric for comparing caching effectiveness.

6.6 Evaluation: Validating the Ideas

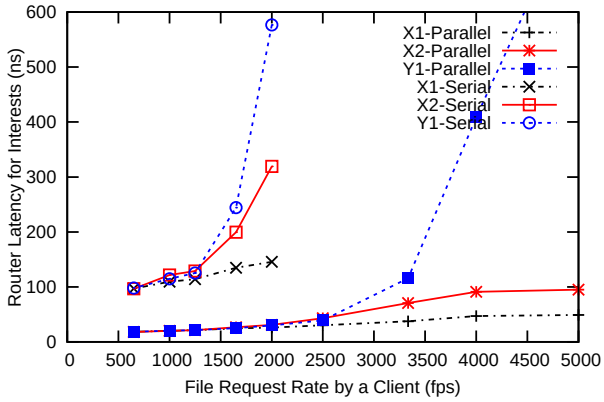
We now validate the effectiveness of the five ideas in relaxing the NDN router memory bottleneck. For single router performance, it suffices (by symmetry) that we present results for X_1, X_2 and Y_1.

Section 6.6.1 first compares router latencies of ndn‖mem and serial architectures. We then validate the key ideas of FIBcache (Section 6.6.2), CS skipping (Section 6.6.3), droptail replacement (Section 6.6.4) and distributed caching (Section 6.6.5). We also examine the sensitivity of the results to cache size, file size and number of memory accesses (Section 6.6.6).

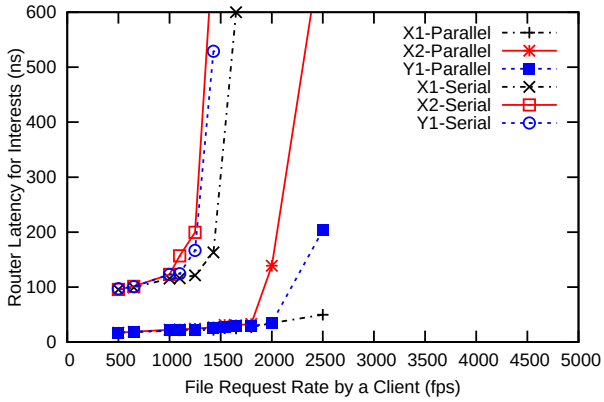
6.6.1 Parallel Search: ndn‖mem vs Serial

Figure 6-7 shows that ndn‖mem has much lower latency than serial, for both nonCBR and CBR traffic. With nonCBR, latency for serial grows rapidly around $\lambda_{\text{file}} =$ 2000fps, thus indicating **router saturation**.

With CBR, saturation for serial occurs earlier, around 1400fps. This is because, for nonCBR, congestion along a route delays receipt of the Data, and thus delays transmission of the next Interest by the client. In contrast, a CBR client sends



(a) nonCBR



(b) CBR

Figure 6-7: ndn‖mem has much lower router latency than serial, and postpones router saturation. The replacement policy is $\langle random, droptail \rangle$.

Interests at a constant rate regardless of delays between consecutive Data chunks. CBR clients are thus not moderated by feedback, so congestion is worse than for nonCBR.

For both nonCBR and CBR, Figure 6-7 shows that ndn‖mem saturates at a higher λ_{file} than serial, thus increasing router capacity. Moreover, Figure 6-12 shows that X_2's link queue to Y_1 and Y_2 grows faster for ndn‖mem than serial. Our design thus shifts the bottleneck from memory to the links as traffic grows.

We now take a closer look at the other parameters. Figure 6-8 reveals that Interest packets never get stuck in an ndn‖mem CS. Increasing the load of the network has

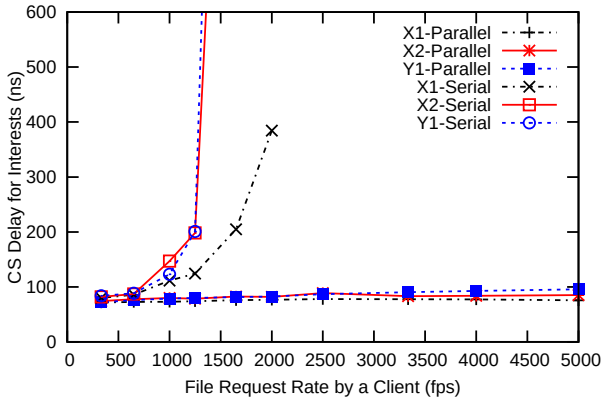


Figure 6-8: CS delay for ndn‖mem keeps constant whilst serial saturates very soon (The replacement policy is $\langle random, droptail \rangle$; nonCBR traffic).

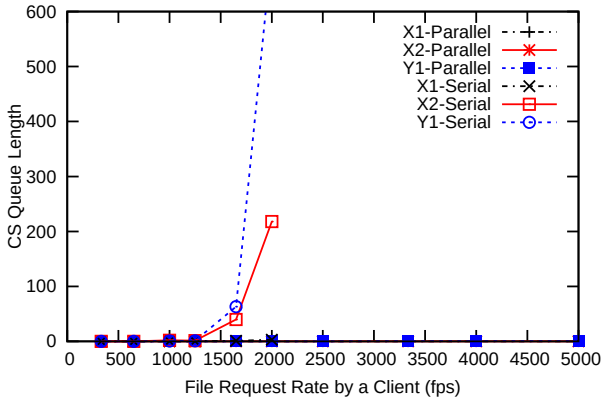


Figure 6-9: Unlike serial routers, CS queue length for ndn‖mem is almost zero for all three routers. (The replacement policy is $\langle random, droptail \rangle$; nonCBR traffic).

no effect on CS delay for parallel architecture. In other words CS is not a bottleneck for ndn‖mem.

Note that the parallel router goes to saturation in Figure 6-7 before CS become congested.

Figure 6-9 is another support for such claim. CS queue length for all parallel routers are almost zero but it is not the case for serial routers.

So far we realized that CS is not a bottleneck for parallel router and Interest packets can get their Data from CS with no obstacle. However, we still need to find the cause of saturation for parallel architecture (Figure 6-7) for those Interest packets

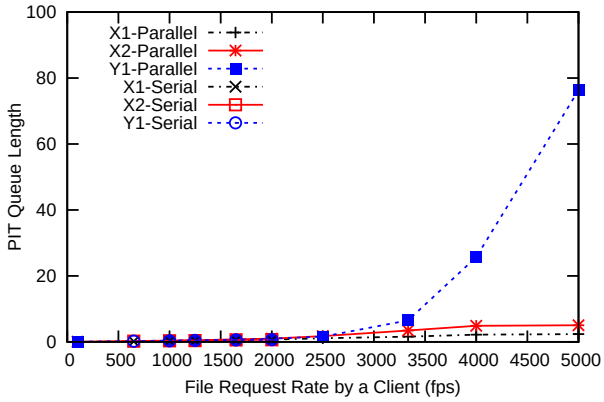


Figure 6-10: PIT is the bottleneck for parallel router. (The replacement policy is $\langle random, droptail \rangle$; nonCBR traffic).

that just need to pass through the router (router latency).

Since CS is not a bottleneck, its index table should not be a bottleneck as well (CS index table is implemented with faster memory technology in our experiments). Figure 6-10 reveals that PIT table is actually the bottleneck component as its queue is the longest.

So far we found out that PIT is the bottleneck for the **parallel** router. However, we still need to find the source of saturation for **serial** architecture. All Interest packets in serial architecture go through CS index table and PIT table. Figure 6-10 indicates that PIT is not the bottleneck for the serial router and as CS index is made of the same memory technology (in our experiments) then it is not the bottleneck either.

Figure 6-11 shows that even small queue at FIB table cause a considerable latency for Interest packets. That means that FIB is the bottleneck component for the serial router.

FIB queue length has no meaning for parallel router. FIB and PIT are parallel components with only one queue.

Figure 6-12 shows that relaxing the bottleneck component for the parallel router makes it fast enough to shift the bottleneck from memory to the output link.

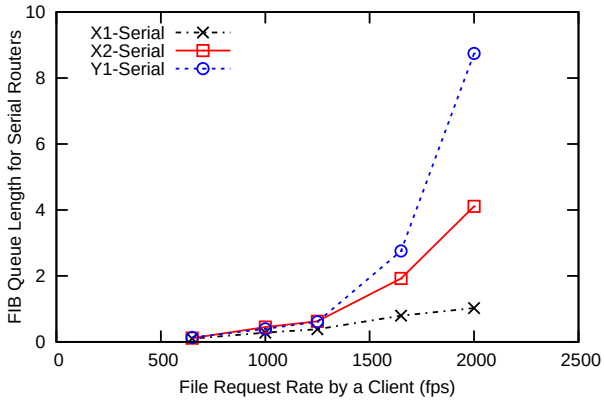


Figure 6-11: FIB is the bottleneck for serial router. (The replacement policy is $\langle random, droptail \rangle$; nonCBR traffic).

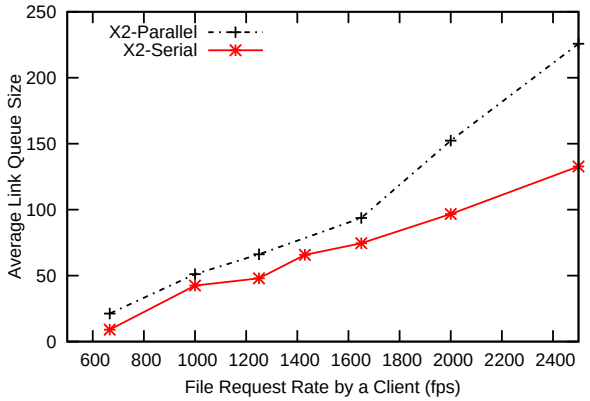


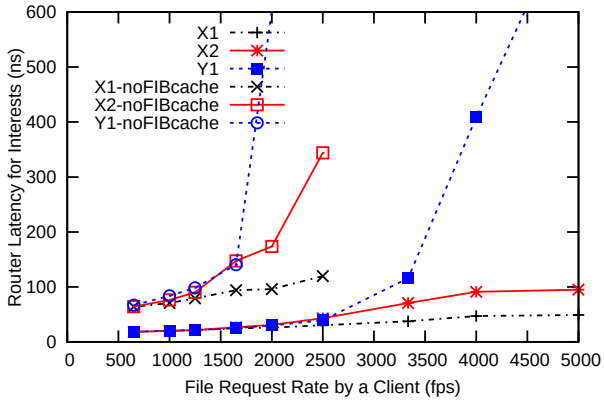
Figure 6-12: As request rate increases, ndn‖mem shifts X_2's bottleneck from its memory to its output (egress) links. (The replacement policy is $\langle random, droptail \rangle$; nonCBR traffic; 200Gbps link).

6.6.2 PIT/FIBcache: ndn‖mem Postpones Router Saturation

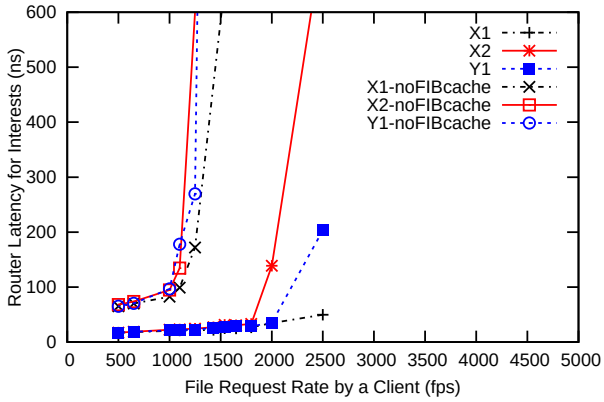
We next validate the idea of having a FIBcache. Figure 6-13 shows that a FIBcache reduces router latency significantly, and postpones saturation for both nonCBR and CBR traffic.

Henceforth, we only present results for nonCBR traffic.

As we have seen from Figure 6-10, PIT is the bottleneck for parallel routers. Since PIT and FIB are sharing one queue, when there is no FIBcache the delay for PIT/FIB



(a) nonCBR



(b) CBR

Figure 6-13: ndn‖mem's FIBcache postpones router saturation. (The replacement policy is $\langle random, droptail \rangle$.) Henceforth, we only present results for nonCBR traffic.

section would be the slower component which is FIB (as searches are performed on both tables concurrently).

Comparing Figure 6-13 with Figure 6-7 shows that router latency for parallel router without FIBcache is rather similar to serial router. It shows that removing FIBcache from the parallel router makes FIB the bottleneck component (the same as serial router).

6.6.3 Using Hop Count to Skip CS Search

The third idea is to use hop count to decide if an Interest should check a CS for its Data. As defined in Section 6.5, the skip fraction measures how much traffic bypass the CS without checking, and skip error measures the fraction of such Interests that miss their Data.

Figure 6-14(a) shows that the skip fraction is uniformly high at around 90% for both edge and core routers in our topology. In other words, some 90% of the Interests save the time they would have spent waiting for and accessing CS.

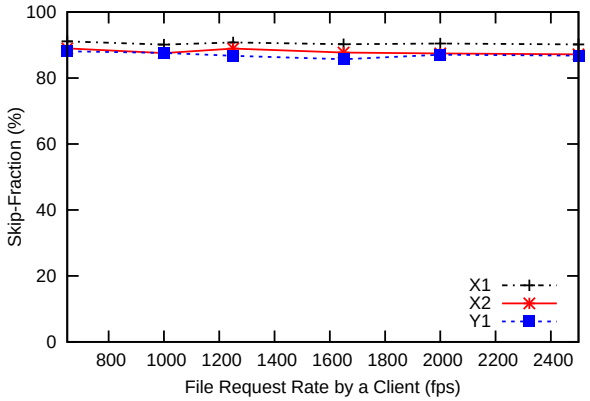
Figure 6-14(b) shows the decision to skip is mostly correct: skip errors are less than 3% for all routers. The small skip error must mean that skipping the CS check does not hurt the hop count. Indeed, Figure 6-15 shows that ndn‖mem has similar average hop count as serial.

Skip error is skipping CS when Data is there. What about checking CS when Data is not there, i.e. a cache miss? Figure 6-16 plots the CS hit rates for X_1 and X_2 in ndn‖mem and the serial architecture. (Recall from Section 6.5 that these are not directly comparable because their Interest traffic are different.)

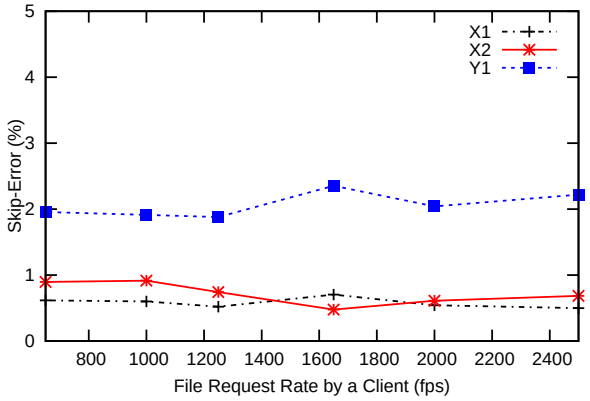
The plot shows that serial has less than 10% hits, i.e. 90% of the CS checks are misses that waste time. For ndn‖mem, Figure 6-14(a) shows a skip fraction of about 90%; for the 10% of Interests that do check CS, Figure 6-16 shows some 60% will be hits; thus, only about $10(1 - 0.6) = 4\%$ check CS when Data is not there — much better than serial. Only these 4% incur the worst case latency of searching both CS and FIB.

6.6.4 A Droptail Replacement Policy for CS

Since hop count for chunk k is used to guide CS skipping for chunk $k+1$, we proposed the droptail policy for chunk replacement (Figure 6-6). We now compare this policy to random chunk replacement.



(a) High skip fraction



(b) Low skip error ($< 3\%$)

Figure 6-14: Fraction of arriving Interests that (a) do not check the CS and (b) do not check the CS but CS contains the corresponding Data. (The replacement policy is $\langle random, droptail \rangle$.)

Our experiments show that, if file victims are randomly chosen, then droptail is better than random for chunk replacement, i.e. $\langle random, droptail \rangle$ is better than $\langle random, random \rangle$. This is best seen in Figure 6-17, which plots the **source hit rate**, i.e. arrival rate at sources for Interests that fail to find their Data en route (so a smaller source hit rate is better).

Figure 6-17 also shows that, with droptail chunk replacement, using LRU or a random choice of file victim has similar effect, i.e. $\langle LRU, droptail \rangle$ and $\langle random, droptail \rangle$ are similar.

This is why — for the rest of the report — we only present results for $\langle random, droptail \rangle$.

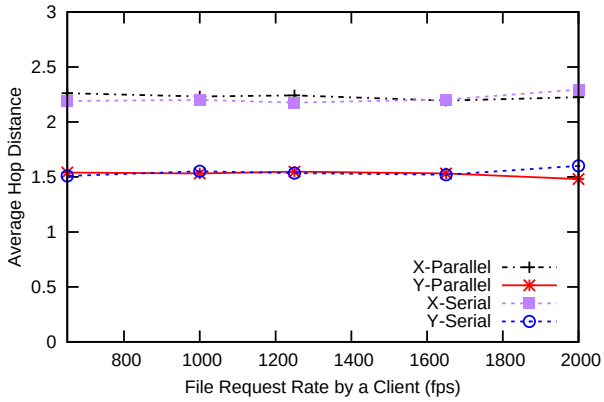


Figure 6-15: ndn‖mem (with skipping) has similar hop count as serial (without skipping) in both X and Y directions. (The replacement policy is $\langle random, droptail \rangle$.)

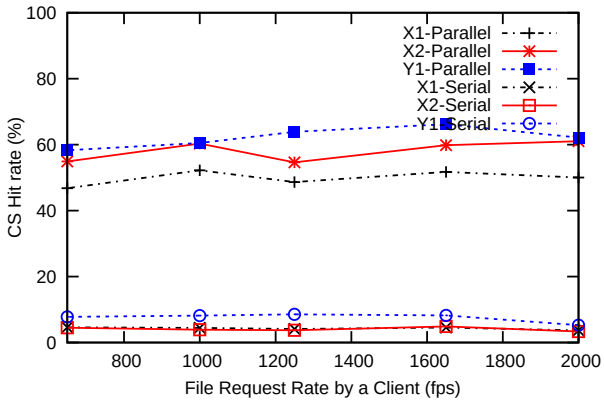


Figure 6-16: For serial, CS hits are less than 10%, i.e. more than 90% of the time, a CS check is wasted time. For ndn‖mem, CS hits are 50–60%, out of the 10% *not* skipped — see Figure 6-14(a). (The replacement policy is $\langle random, droptail \rangle$.)

(Note that the implementation of random replacement is far easier than LRU.)

6.6.5 CCndn's Distributed Content Caching

As it is described in Section 3.1 CCndn serves two main purposes:

- Increases the cache efficiency especially for core routers.
- Reduces the Data chunks traffic at CS queue of a router, which improves *CS delay* for Interests.

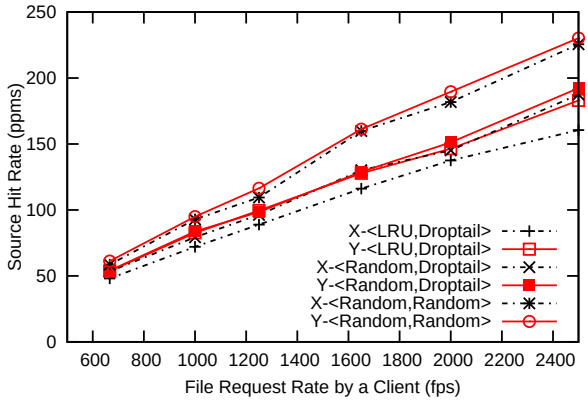


Figure 6-17: For chunk replacement, $\langle random, droptail \rangle$ is better than $\langle random, random \rangle$; for choosing a file victim, $\langle random, droptail \rangle$ and $\langle LRU, droptail \rangle$ are similar. These hold for traffic from both X and Y clients (ppms is packets/msec). Henceforth, we only present results for $\langle random, droptail \rangle$.

Figure 6-18 clearly shows the effect of cache redundancy when CCndn is not used. Since without CCndn all Data chunks are going to be cached in all routers, presence of the same data at the edge routers prevents caches at core routers from getting sufficient hits.

Figure 6-18 also shows that with CCndn core routers have slightly better cache hit rate. That is because in our experiments routers X1 and X4 are only serving X files. However, the other routers (core routers) serve both X and Y files. This shows that even increasing the catalog size will not affect the CS hit rate when using CCndn (larger catalog size in a fixed router cache size means more contention between data chunks to get a free space in a cache).

With CCndn, core routers get more hits (Figure 6-18) and thus reduce Interest arrival rate at the sources, as shown in Figure 6-19, i.e. CCndn's in-network caching is more effective.

As the second purpose of CCndn, it serves as a load balancing strategy by spreading a file along a path. We see this effect in Figure 6-20, where Data chunks at CS queues are negligible for CCndn at file request arrival rates that cause saturation without CCndn.

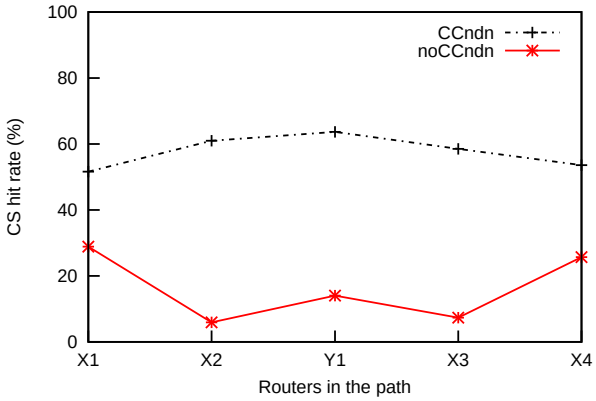


Figure 6-18: Without CCndn, CS hit rates at core routers are much less. (Y_1 is both edge and core.) ($\lambda_{\text{file}} = 1250$ fps.)

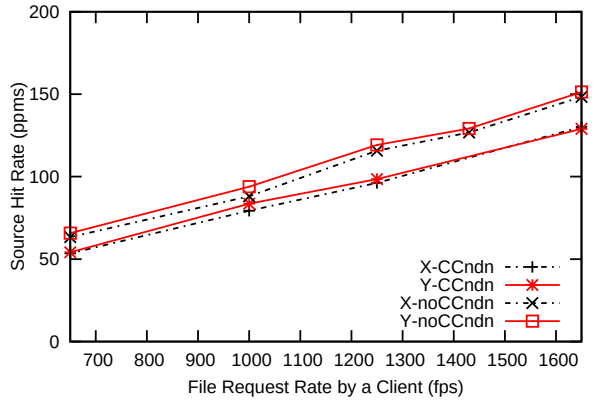


Figure 6-19: CCndn reduces the amount of Interest traffic to data sources, regardless of request origin (X or Y). (ppms is packets/msec.)

In their evaluation of NDN routers, Perino and Varvello do not consider Data chunks [62]. Their results can be misleading because Figure 6-20 shows that Data can, in fact, seriously delay Interests, since they share the CS queue.

6.6.6 Sensitivity of Simulation Results to Parameter Values

We now consider how cache size, file size and number of memory accesses can affect the simulation results.

First, Figure 6-21 shows that router latency is sensitive to CS size for the serial architecture. This is because a reduced CS size increases CS misses, so the router

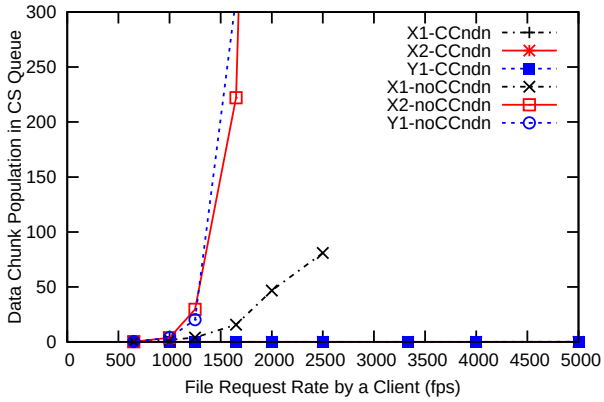


Figure 6-20: Without CCndn, CS queues at Y_1 and Y_2 would bear the brunt of the increase in Data traffic as λ_{file} increases. With CCndn, this load is spread out among edge and core routers, and queues do not build up.

must, in addition, check its FIB to decide how the Interest should be routed. From Section 6.6.5 we know that FIB is the bottleneck for serial router. So increasing the traffic at FIB means larger router latency.

In ndn‖mem, however, most Interests do not even check the CS (Figure 6-14(a)). For an Interest that does the checking, a miss will cause the router to check both FIB and the fast FIBcache; if there is a FIBcache entry for the least prefix match, the FIB search is aborted (see Figure 6-4), so the impact of CS misses on ndn‖mem is much less. We see from Figure 6-21 that, in this sense, ndn‖mem performance is robust with respect to CS size. This is not a surprise as it is already shown that CS is not a bottleneck for parallel router.

Second, the default file size is 0.9MBytes. Figure 6-7(a) shows that the saturation points for ndn‖mem and serial have a ratio of about $\frac{3400\text{fps}}{1700\text{fps}} = 2$. If file size is doubled, Figure 6-22 shows the same ratio $\frac{1500\text{fps}}{750\text{fps}} = 2$. In other words, ndn‖mem's advantage over serial holds for larger files.

Third, we assume that an Interest or Data chunk makes zero or one access each to PIT, CS or FIB; we now relax this assumption with an experiment where Interest and Data make zero or two accesses each to PIT, CS and FIB.

Figure 6-23 shows that the saturation pattern looks like Figure 6-7(a), except for

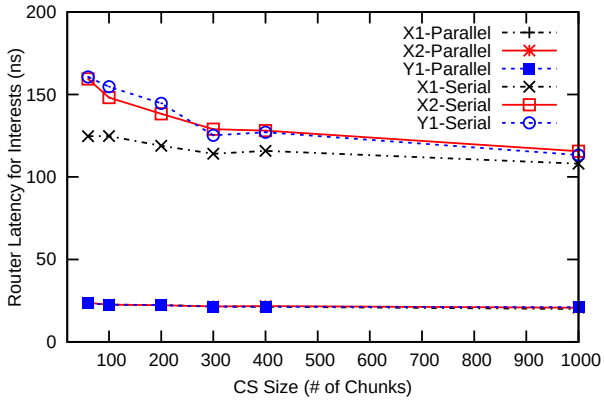


Figure 6-21: Router latency for ndn‖mem is robust with respect to a reduction in CS size. ($\lambda_{\text{file}} = 1250$ fps)

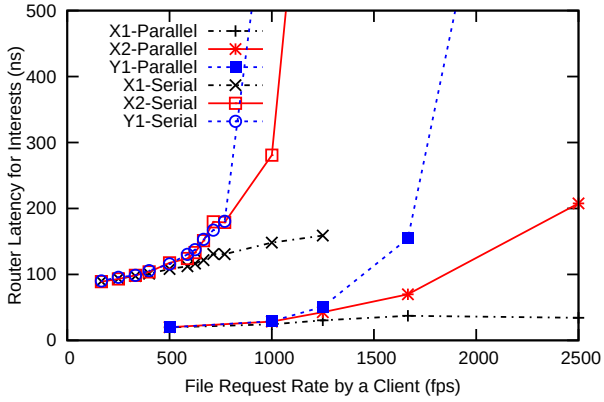


Figure 6-22: Although file sizes here are double those in Figure 6-7(a), the saturation gap between ndn‖mem and serial is similar.

X_2. This exception suggests a closer look, and Figure 6-24 shows that Interests in fact suffer no saturation at all in CS delay for ndn‖mem.

Results show that even with larger look up delays only PIT remains as bottleneck for the parallel router (Figure 6-25). Unlike CS, PIT performance affects all Interests that arrive at the router. However, compared to CS, PIT sizing is easier to engineer: The required PIT size can be estimated (using Little's Law [82]) by the product of Interest arrival rate and round trip time; a timer is used to delete old PIT entries anyway (in case their Data cannot be found); and the smaller PIT size makes it affordable to use faster memory (e.g. SRAM). Engineering PIT for line speed was

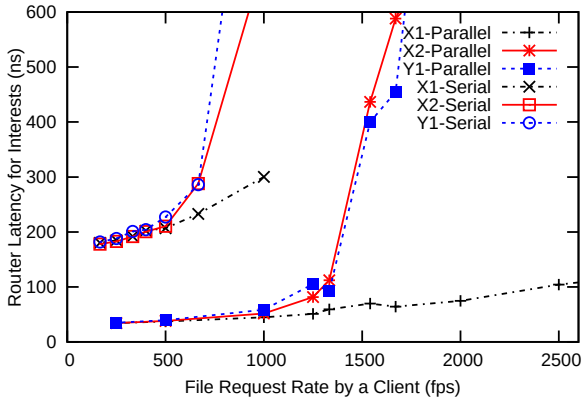


Figure 6-23: More memory accesses: Saturation pattern looks like Figure 6-7, except for X_2.

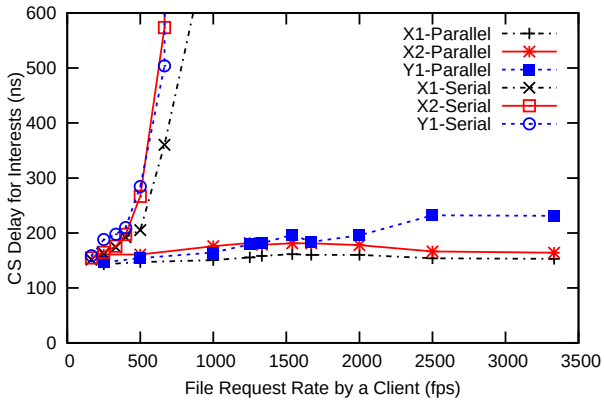


Figure 6-24: More memory accesses: Interests suffer no saturation at CS in ndn‖mem (cf. Figure 6-23).

recently examined by Varvello et al. [53]

6.7 Experiments: An Abilene-like Topology

The X-Y topology in Figure 5-1 is *designed* for analyzing ndn‖mem performance. For a realistic topology, we use the one in Figure 3-2, which is similar to Abilene[1]; it has routers $R1, \ldots, R11$, and we added a link from $R3$ (Denver) to $R4$ (Los Angeles) to increase the multipathing.

[1] *http* : *//nic.onenet.net/technical/category5/core_topology.htm*

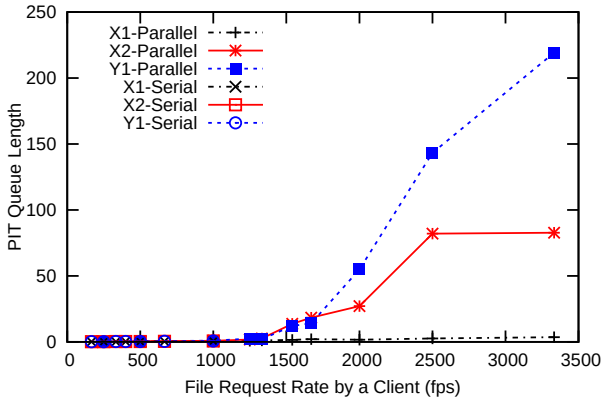


Figure 6-25: More memory accesses: Interests suffer saturation at PIT in ndn‖mem (cf. Figure 6-23).

Each router has 30 clients (330 altogether), and each client has 1 file. File popularity (Zipf) is randomly determined, and the 10 most popular files are located at $R7$, $R3$, $R2$, $R3$, $R7$, $R2$, $R6$, $R10$, $R2$, $R3$ (ordered by decreasing popularity). The rest of the experimental setup follows Section 6.3 (CS size is 0.015 of catalog size, $\langle random, droptail \rangle$, etc.).

We try to present the most useful measurements from important routers like $R3$ which 3 of the 10 most popular files are at the local clients of this router. The measurement for other routers are presented in Appendix A.

The simulation results for this realistic topology confirm our analysis with the X-Y topology.

Figure 6-26 compares the router latency for Interest packets for the two architectures. As expected the router latency for serial architecture is much higher than parallel architecture. Since serial router saturated at CS queue, we could not run any more experiments to show saturation point for its router latency. Appendix A presents some results from other routers which shows the saturation point for the router latency. CS and FIB both are bottleneck for a serial router and depends on CS hit rate, one or both of them (CS and FIB) go to saturate.

Some Figures in Appendix A show saturation for router latency for parallel router.

This and the results for PIT queue length again prove that the bottleneck shift from CS to PIT for the parallel architecture.

Figure 6-27 confirms that the parallel architecture is very effective to avoid having CS as the bottleneck. Whilst the serial router starts to saturate at around 700fps, we could not reach to the saturation point for parallel router (we could not afford any higher traffic density experiments due to very exhaustive simulation time). The result for parallel router without CCndn shows that this cache policy is the main reason of shifting the bottleneck off from CS.

Note the difference between router latency and CS delay. The router latency in Figure 6-26 is for Interests that leave the router. Since most Interests that leave an ndn‖mem router do not check CS (Figure 6-14(a)), they do not suffer the congestion at the CS queue. For Interests that do find their Data from CS and must wait to retrieve their Data, Figure 6-28 shows that the congestion causes a delay that is worse than for serial.

The parallel router surpasses serial in router throughput as well. Figure 6-29(a & b) show the throughput of the router $R3$ for the two architectures. Having better throughput allows the parallel router to pass packets much faster than serial and in absence of CCndn its CS queue will be exhausted (shown in Figure 6-27). That is the main reason that parallel-noCCndn saturates earlier than serial (Figure 6-28).

For multi-router performance, the average hop count depends on where the clients are (e.g. Seattle or Houston), and who are the sources for the popular files. Generally, ndn‖mem (with or without CCndn) has similar hop count as serial, as illustrated in Figure 6-30 for hop count measured from $R3$(Denver).

The small difference in hop distance between serial and parallel routers might question the effectiveness of CCndn cache policy. However, Figure 6-31 shows very small skip error caused by CCndn algorithm. By the way, the effect of skip error is to find data from a longer distance. From Figure 6-30 we see that this difference is negligible compared to other benefits (e.g., lower router latency for Interests and

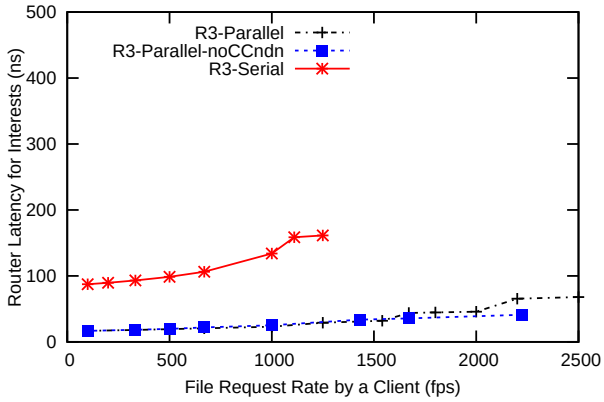


Figure 6-26: Router latency at $R3$(Denver) in Abilene topology: The comparison is similar to Figure 6-7(a). R3-Serial saturates at about 1500fps.

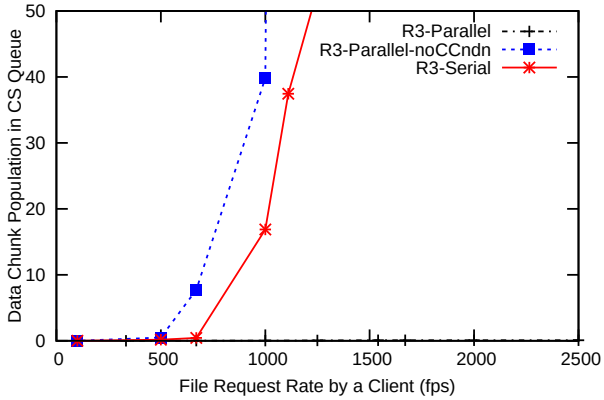


Figure 6-27: Without CCndn, the CS queue at $R3$ in Abilene topology can be saturated by Data chunks (cf. Figure 6-20).

avoid making CS a bottleneck) from ndn‖mem architecture.

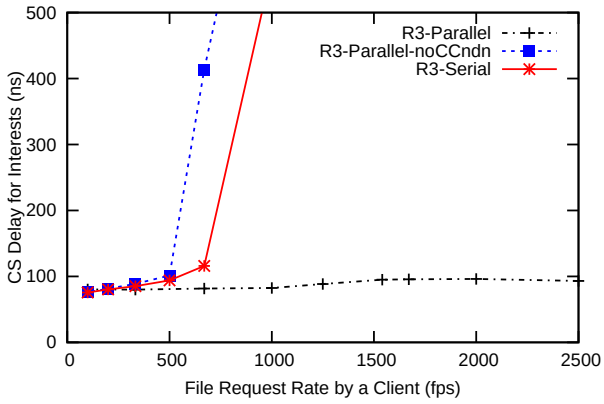
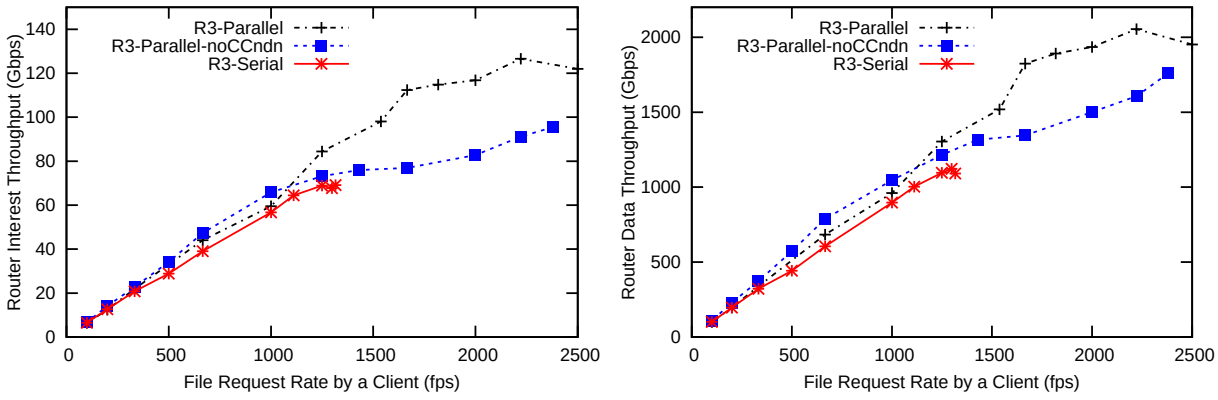


Figure 6-28: CS Delay at $R3$ in Abilene topology: Congestion of CS queue by Data chunks can cause big delays for Interests that find their Data in CS, in contrast to Interests that do not (see Figure 6-26).



(a) For Interest packets

(b) For Data chunks

Figure 6-29: Router throughput.

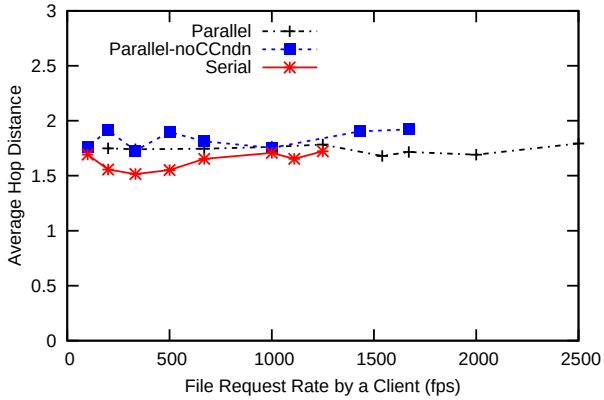


Figure 6-30: Average hop counts from $R3$ in Abilene topology are similar for serial and ndn‖mem, with or without CCndn (cf. figure 6-15).

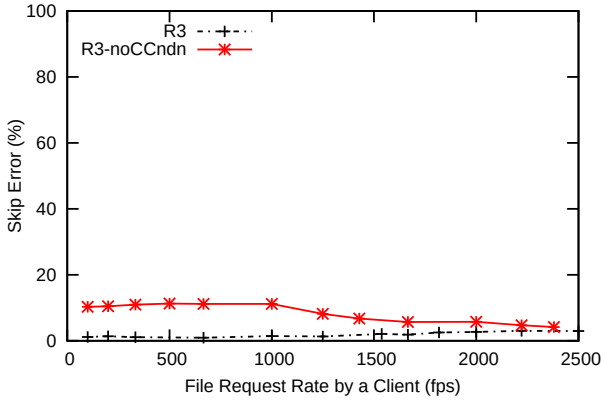


Figure 6-31: Skip error for Abilene network shows the effectiveness of CCndn cache policy.

Chapter 7

Conclusion and Future Work

Advancing the technology of mobile and tablet devices on one hand and rising applications like Facebook, Twitter, Youtube, IPTV, etc, on the other, increased the urge of using the Internet more than ever. The architecture of the Internet which was designed more than two decades ago is not suitable for the current mobile and application technologies.

It is apparent that overlay remedies to enhance the Internet functionality are reaching to their limits. That leaves no choice but to figure out a new architecture with a new communication paradigm that meets the current application requirements.

Information Centric Networking is a paradigm that has potential to overcome the current Internet's difficulties. NDN as one of the leading project in ICN domain claims for incremental deployment which makes it exceptional among all other proposals. However, in-network caching in general and forwarding architecture of NDN in particular still needs rigorous research to make it feasible.

The performance of in-network caching is a function of many parameters including, content popularity, catalog size, replacement policy, cache policy, topology, traffic pattern and more. That makes studying the behavior of network of caches extremely hard. Studies on CDN and web caches do not match with in-network caching. Larger incoming traffic to routers, smaller memory size, in-line processing of packets, depen-

dent on routing algorithm, data dependency between caches and many more make studying in-network caching a brand new research on distributed caches.

So far, coordination and collaboration among caches are the main solutions to scale the performance of the in-network caches. Due to their complexity and overhead, we suggest another approach for cache policy which is based on hop distance. The algorithm allows only a fraction (a segment) of a file to be cached in a router with a specific hop distance from the client. Interest packets in their upstream path to the publisher, search only the routers in the same hop distance from the client that match with the hop distance in the header of the packets and skip the others. In Chapter 3 through simulations we have shown that the idea could improve the efficiency of core routers and so the total efficiency of the network of caches.

Our cache policy (CCndnS) breaks the dependency between caches. That makes it possible to use a popularity distribution (such as Zipf) freely for caches at any level without concerning about the filtration effect. This simple improvement made the process of modeling hit rate of caches straightforward. Our presented equations in Chapter 3 can be used for further studies on the field or can be used in a real application like what we suggested for SLA application in Chapter 4.

With CCndnS, both Data and Interest packets incoming rate to a cache reduces as well. That helps to match the packet processing delay of slow memories with the incoming rate to the memory. In addition, skipping CS relaxes dependency between CS and the other two tables. Assuming CS as the first table to search, without using CCndnS, PIT and FIB have to wait for the packets to search CS first. In other words, the Interest input rate to them cannot exceed more than the throughput of CS. Chapter 6 explains the architecture including the idea of parallel search of PIT and FIB.

For further improvement on cache efficiency, we investigated a dynamic partitioning scheme using a cache miss equation to give a better chance to more cachable content to be cached in a router. Chapter 5 explains our algorithm to partition

memory of routers to increase performance or fairness.

7.1 Future Work

There are several possible areas that our research can be extended to:

- We are in the middle of developing a cache aware routing algorithm. In this thesis we only looked at en-route caching. En-route caching refers to the caching scheme where Interests only search CS on the path to the publisher. In an off-path caching, using the aggregated information from neighbors, a router might decide to forward an Interest packet to a node with possible cached content. There are several problems in such a cooperative caching which we addressed them in Chapter 2. In our new routing algorithm we will try to use the PIT entries as a temporary entry of FIB tables. When a Data packet exhaust a PIT entry, the information retrieved from PIT can be added to FIBcache for a limited time to serve immediate request for the same chunk. The algorithm requires a very precise timing to expire the temporary entries in FIBcache. Small timers might expire too soon and the scheme will just add extra complexity to the router without any gain. Longer timers might confuse requests by sending them to a wrong direction.

- Cache placement is another interesting research topic which we tend to conduct using our cache hit equation. In all presented experiments in this thesis we assumed that the cache size of all nodes are the same. This is a very strong assumption and we need to relax it. But question is which cache arrangement is the most effective? As we have seen there are different opinion on cache placement but majority of researches opt for caching more at edge routers. That makes sense by considering the filter effect of edge routers. Since CCndnS relaxes the problem of filter effect, we might can find a more interesting result for cache placement. We have observed that cache hit is mainly influenced by

catalog size of the cache and content popularity. Most likely the traffic volume passing through a core router is higher than an edge router. That increases the probability of having larger catalog size at a core router. Moreover, it is possible that some content are not popular in any local ISP but aggregated request from different ISPs make it popular at a core router.

- We have seen in Chapter 5 that traffic separation is possible with partitioning. In the same way as CCndnS which provides different set of catalog for caches in the path, partitioning can be another way of traffic separation. However, the necessity of traffic partitioning for removing the filter effect of edge routers is that there are enough different type of traffics with popular files.

- We used the cache miss equation for a dynamic cache partitioning in Chapter 5. The idea could be extended to the whole network. Considering the entire ISP's network as a huge cache and apply the partitioning scheme. However, the idea needs to be supported by the routing algorithm. The routing algorithm should be able to obtain information about the location of partitions that cache each traffic type. We need to find a proper routing algorithm which is able to find the correct direction with minimum overhead.

- A more general research topic for the cache miss equation of Chapter 5 is to extend the equation itself for a larger area. It is shown that the equation can be used for a single cache. A better question is if we can modify the equation in a way that can estimate the cache miss for the whole network instead of a single node! Nevertheless, such extension might have lack of incentive. CCndnS is mainly introduced to overcome the filter effect of the edge routers. Filter effect makes core router cache worthless. Therefore, it might be better to study the cache miss equation for the larger domain when CCnsnS is applied as the cache policy.

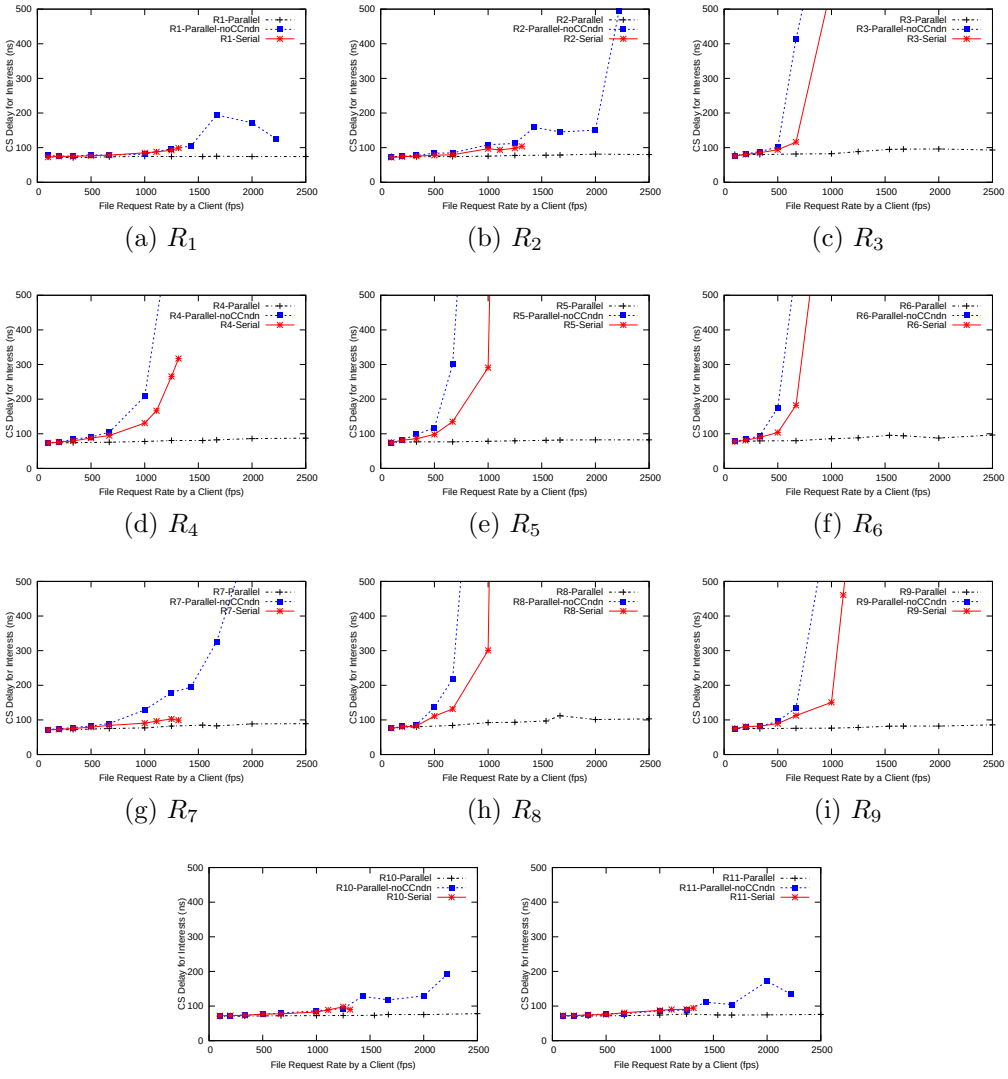
- The NDN router architecture definitely requires many upgrades. The proposed

forwarding pipeline is too complicated and inefficient. We tried to highlight the problem but our work is not a complete solution.

Appendices

Appendix A

Abilene Network Results for ndn‖mem Design



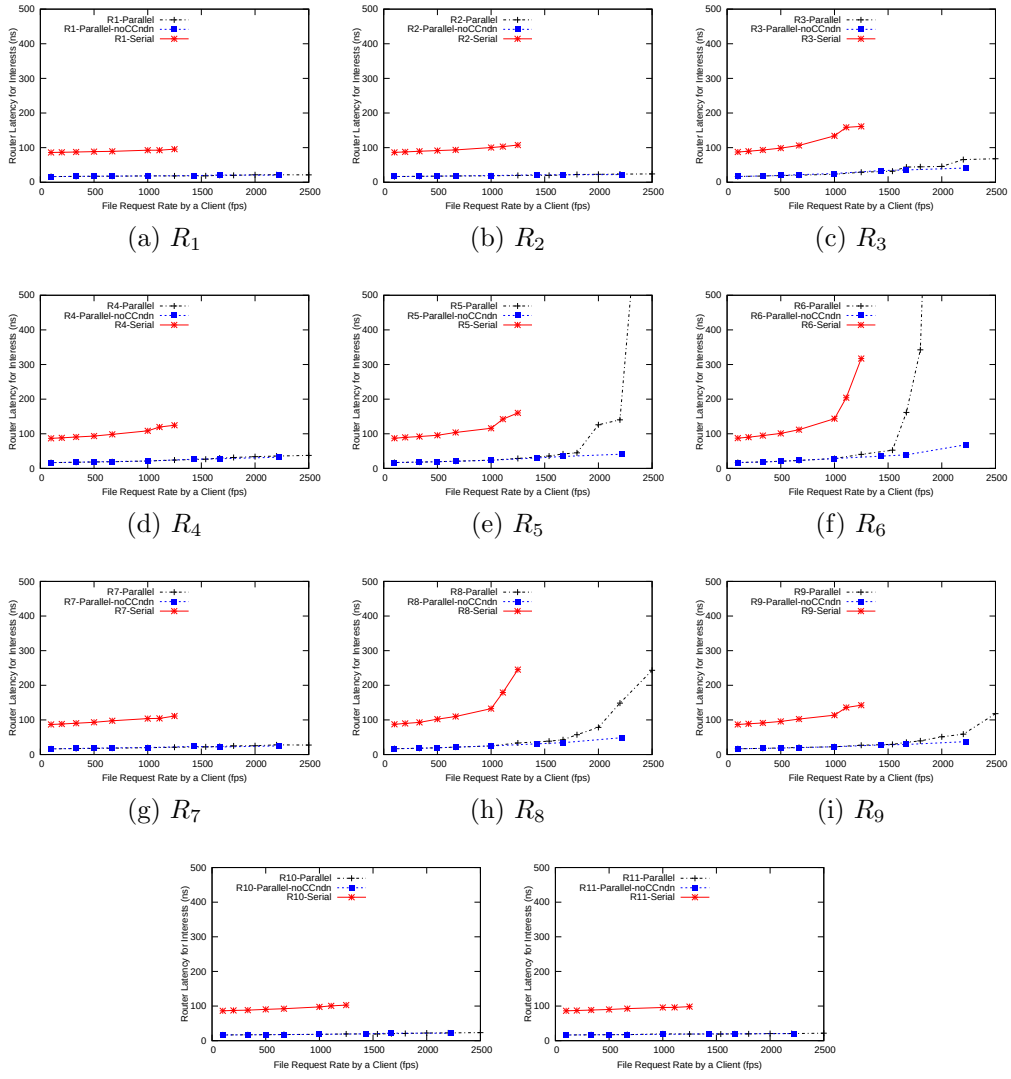
(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-1: CS delay for Interest packets.



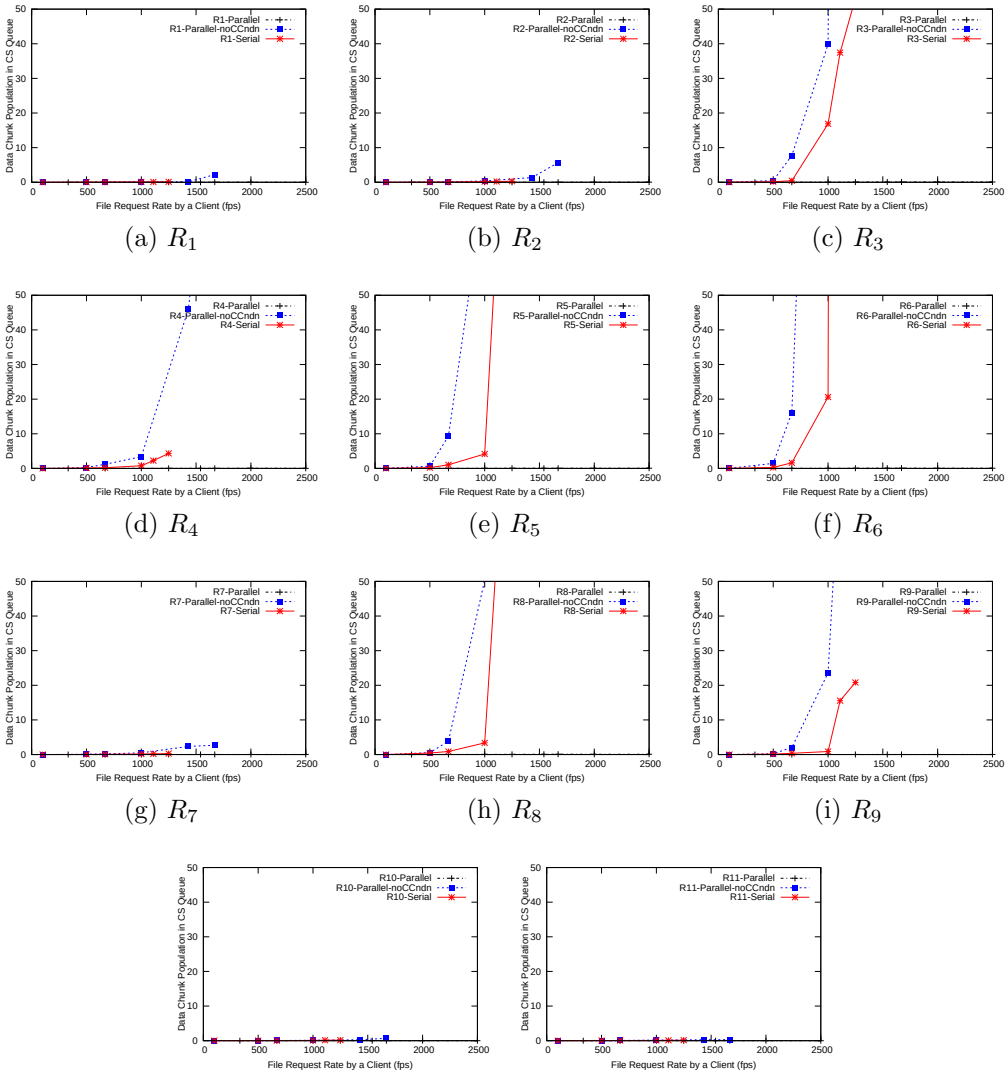
(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-2: Router latency for Interest packets.



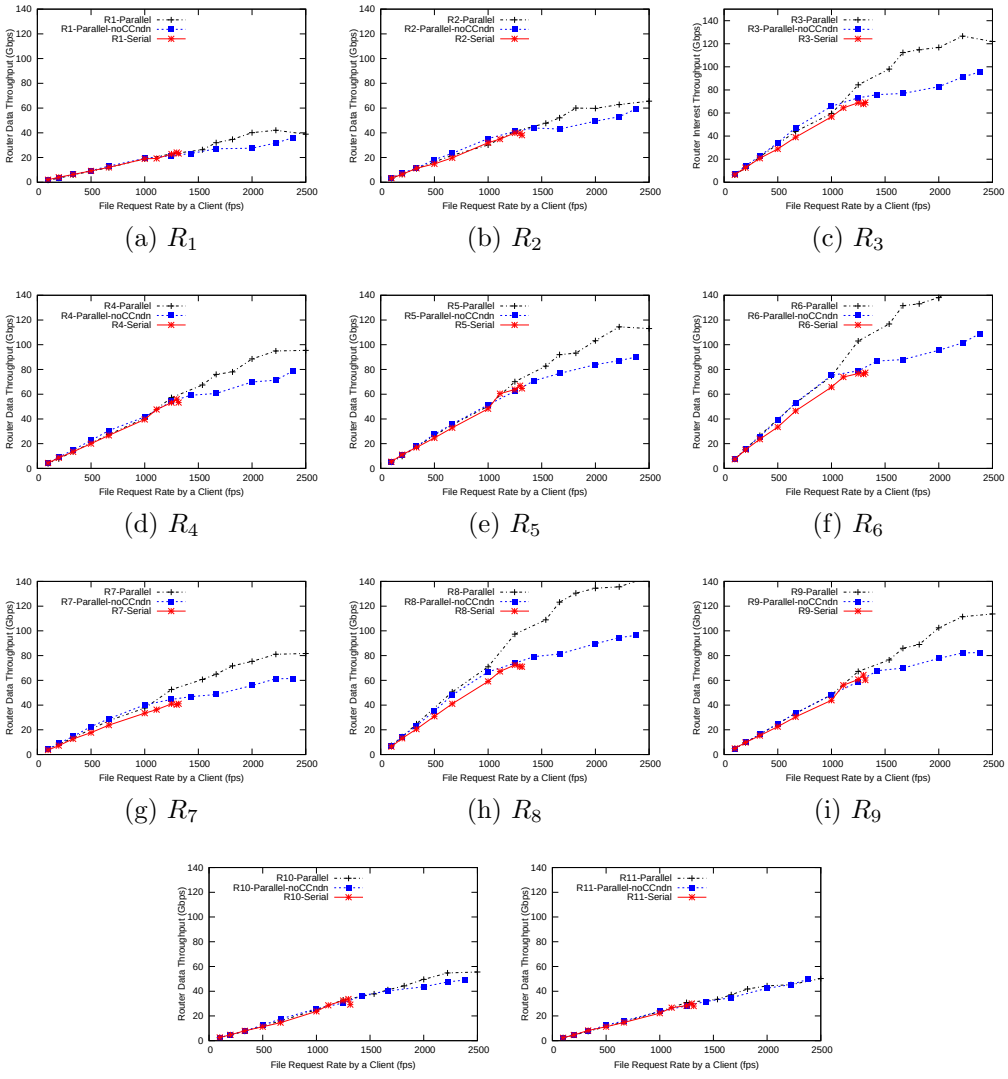
(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-3: Data chunk opulation in CS queue.



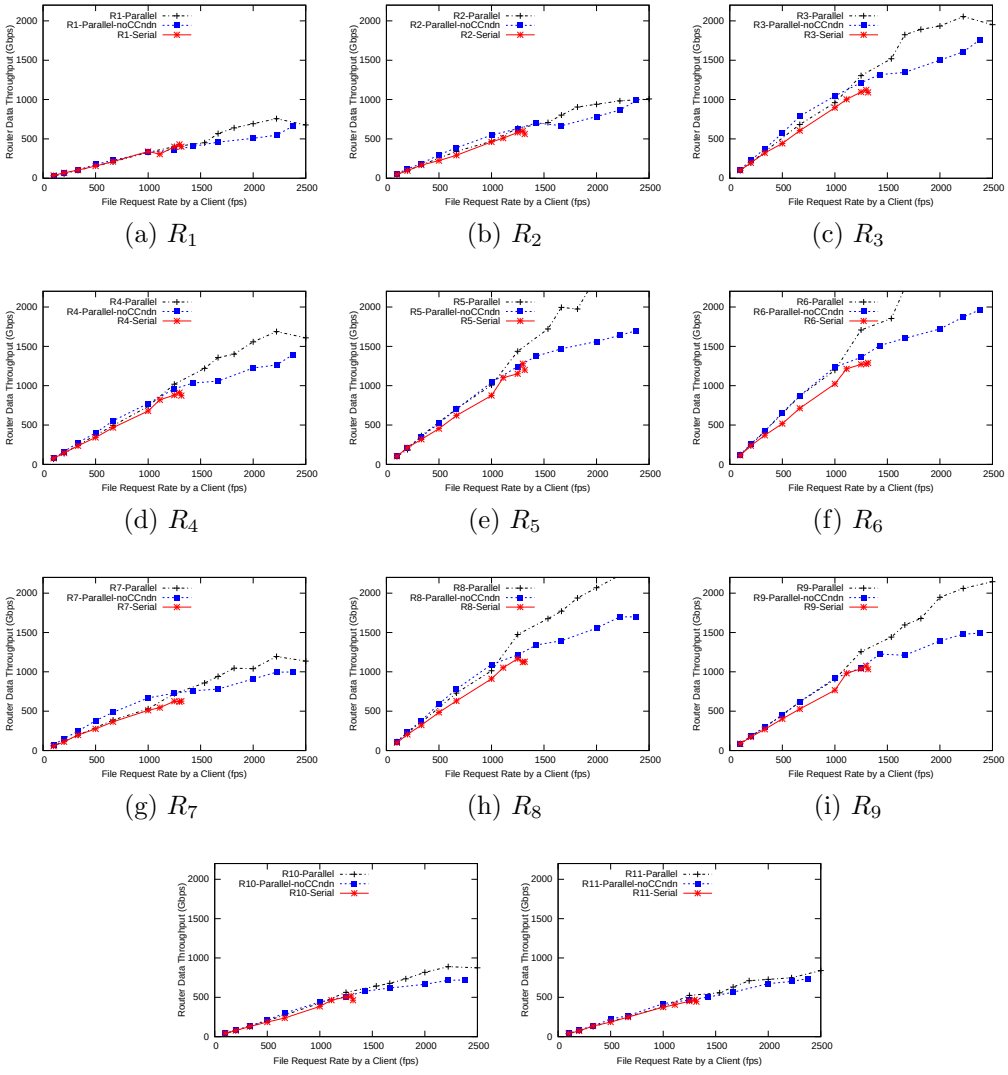
(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-4: Router throughput for Interest packets.



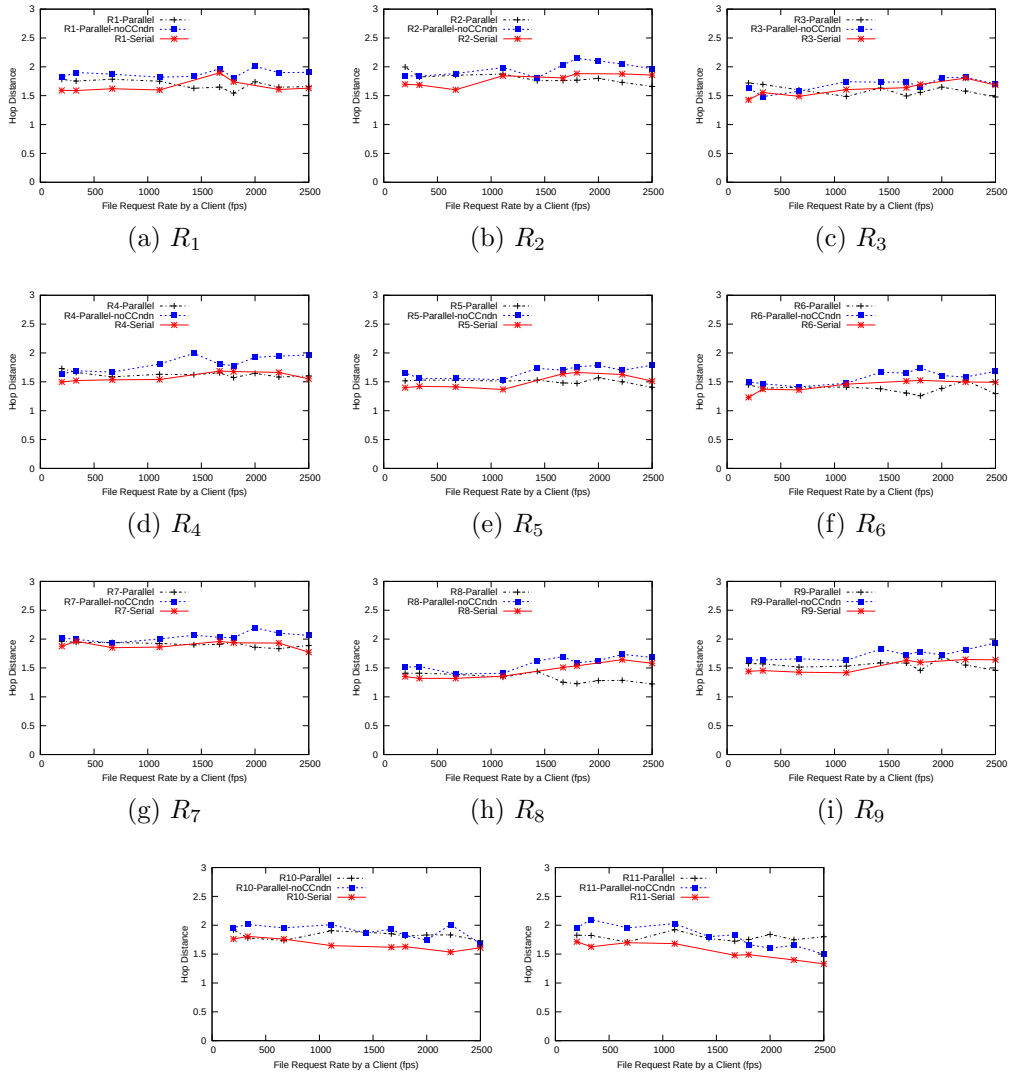
(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-5: Router throughput for Data packets.



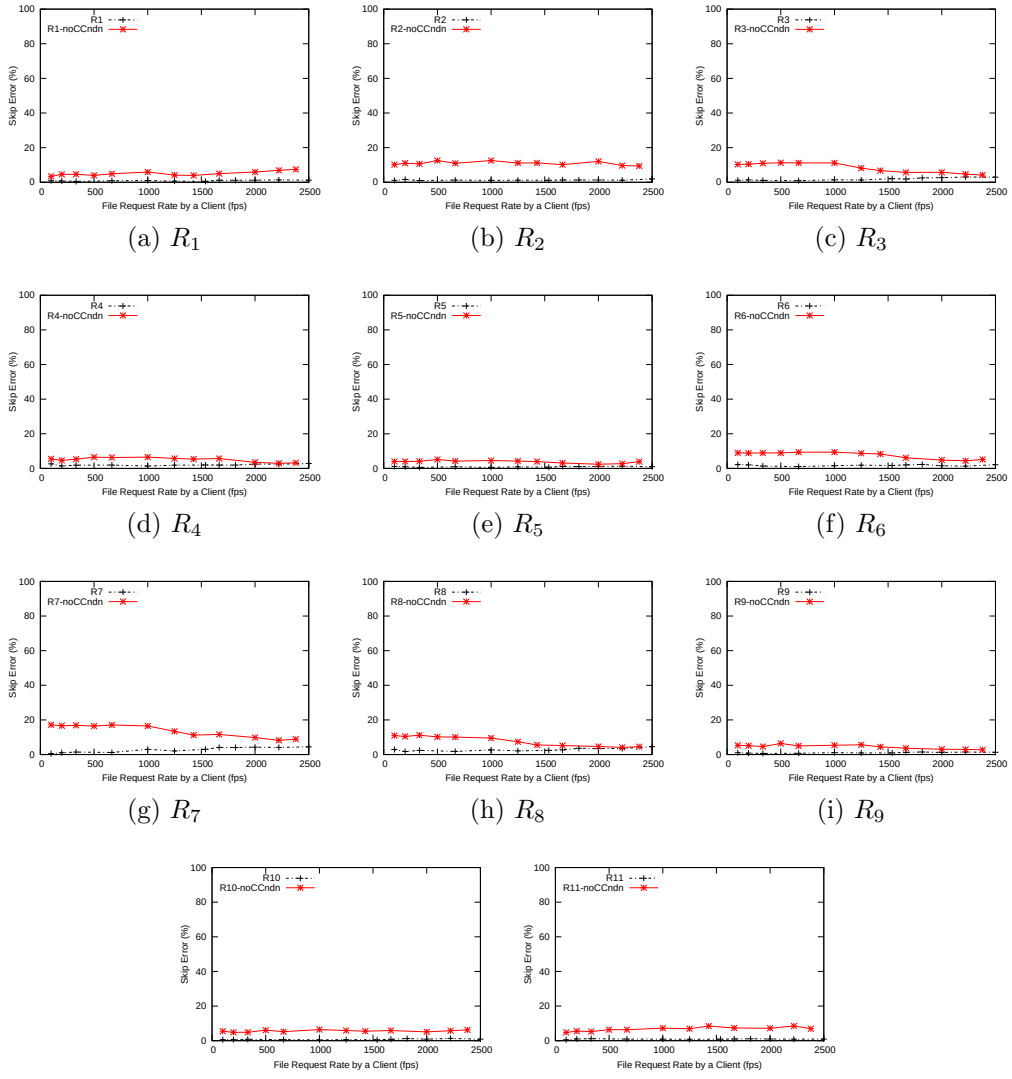
(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-6: Distance of content in hop based to each router.



(a) R_1 (b) R_2 (c) R_3

(d) R_4 (e) R_5 (f) R_6

(g) R_7 (h) R_8 (i) R_9

(j) R_{10} (k) R_{11}

Figure A-7: Skip Error of each router.

Appendix B

Trace Based Network Results for ndn‖mem Design

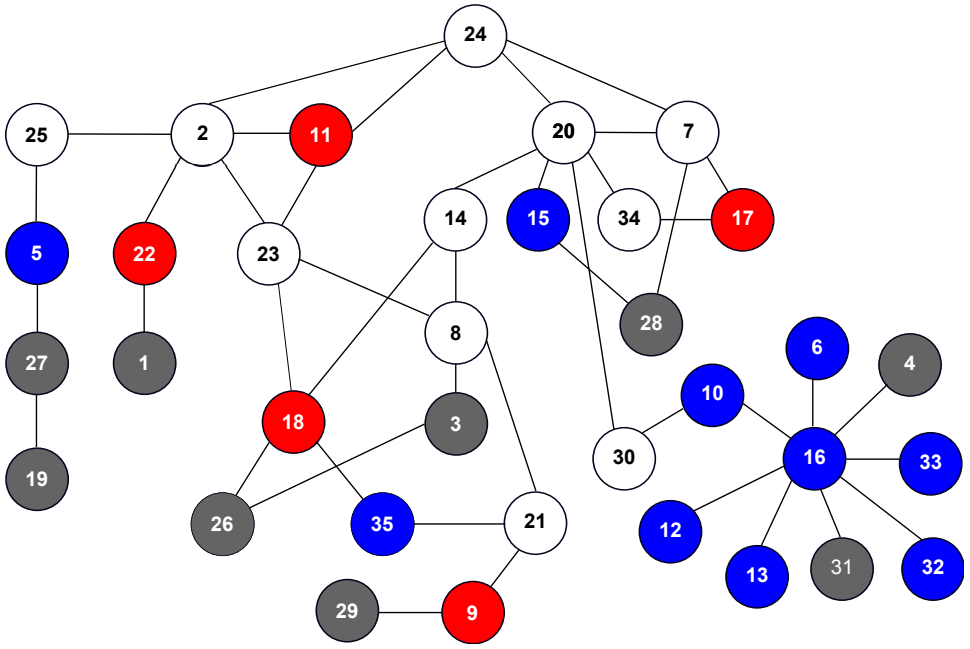
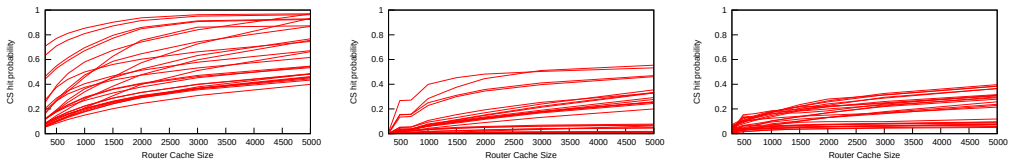


Figure B-1: The trace based topology with 35 routers. Gray nodes are routers without any traffic passing through them, Red nodes are routers which is connected to requesters, Blue nodes are connected to content producers.



(a) *CCndnS* (b) *LCE* (c) *LCD*

Figure B-2: Compare the cache hit rate for all routers for the three cache policies.

The network used in this section is presented in Figure B-1.

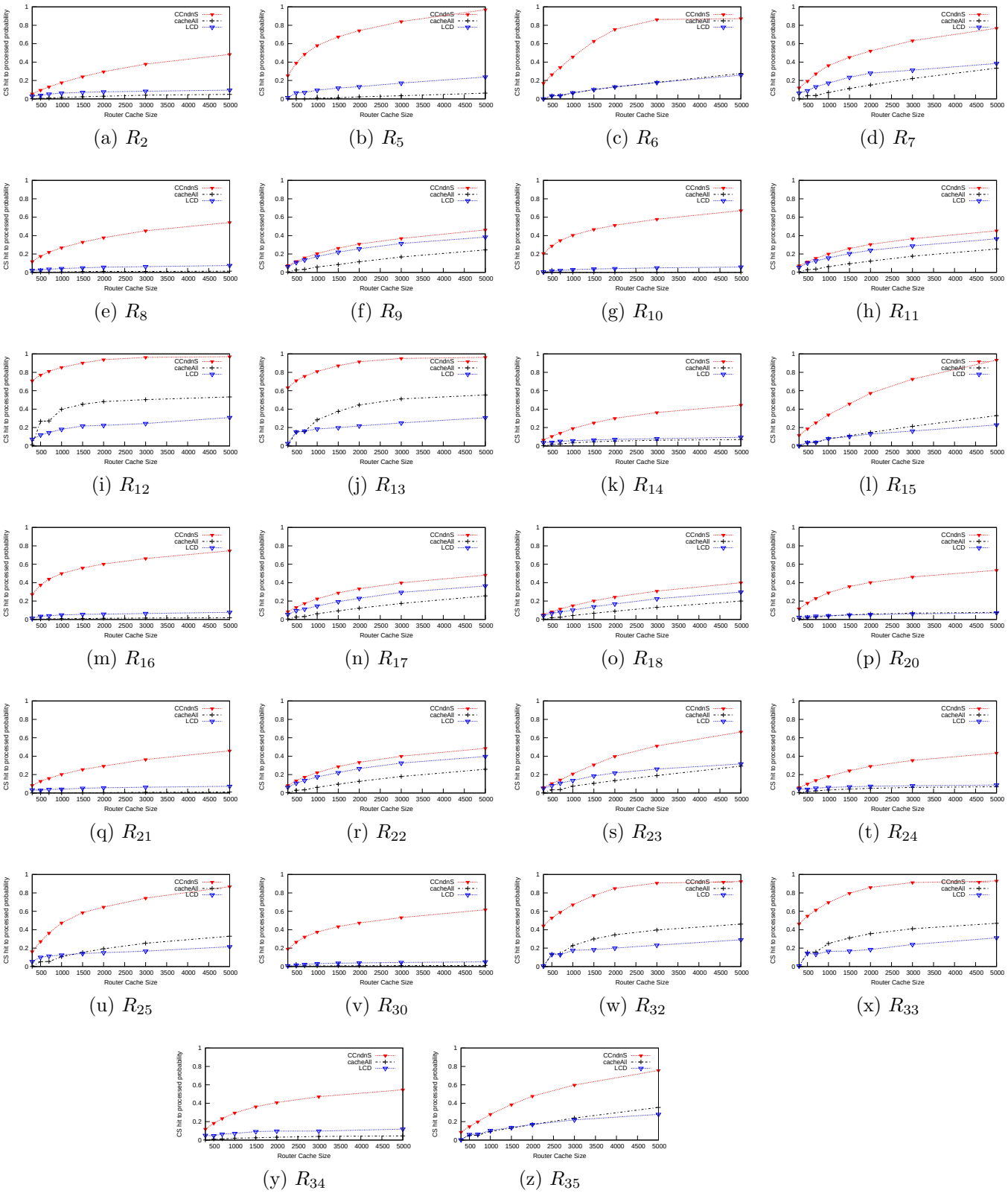
Figure B-2 shows the cache hit rate for all routers in Figure B-1 except the gray routers.

The result for each individual router is presented in Figure B-3.

Figure B-5 shows bad behavior of CCndnS for both network hit and average hop distance to content. First of all, CCndnS does not try to reduce hop distance as it already spreads content through the path between consumer and producer. However, we expect in a normal situation it reduces the distance. The reason is that CCndnS

improves the performance of core routers and almost all of the routers receive equal cache hit rate. Whereas, other cache policies have a good cache hit only at the edge router. So Interests will hit only edge router and the source of the file.

In the example of Figure B-1 many of the routers are just connected to a source file without any other traffic interferer with them (such as R_{12}, R_{13}, R_{32} and R_{33}). Such routers only bring the popular files attached to them only one hop closer to the clients (which is not much improvement in performance) but increases the average network hit rate anyway. Thats the main reason of having a good network hit rate for other cache policies. Nevertheless, using a better cache replacement policy like SLRU the performance of each single router with CCndnS will improve which results in a better network hit in Figure B-5(b).



(a) R_2 (b) R_5 (c) R_6 (d) R_7

(e) R_8 (f) R_9 (g) R_{10} (h) R_{11}

(i) R_{12} (j) R_{13} (k) R_{14} (l) R_{15}

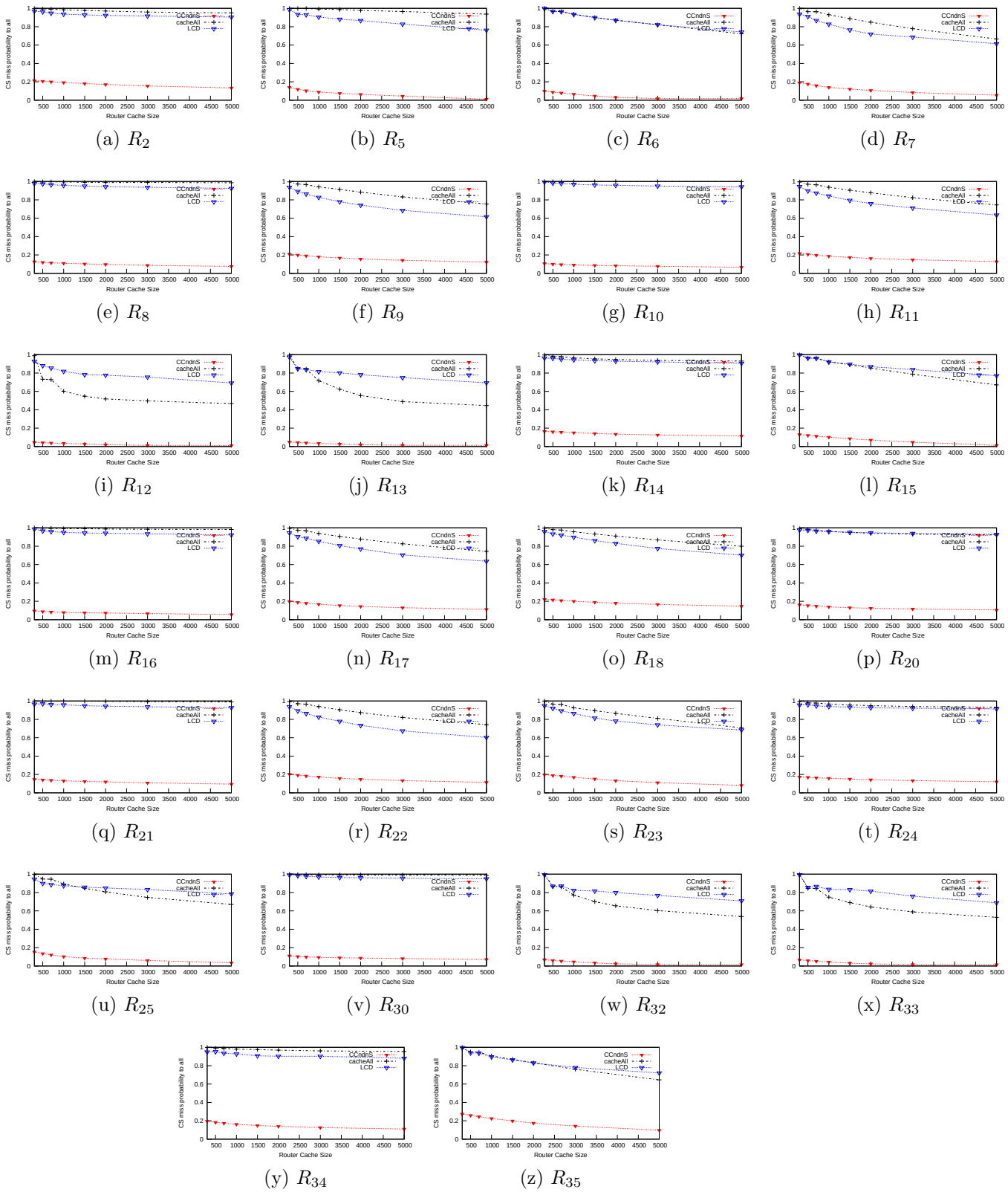
(m) R_{16} (n) R_{17} (o) R_{18} (p) R_{20}

(q) R_{21} (r) R_{22} (s) R_{23} (t) R_{24}

(u) R_{25} (v) R_{30} (w) R_{32} (x) R_{33}

(y) R_{34} (z) R_{35}

Figure B-3: Cache hit rate for all routers.



(a) R_2 (b) R_5 (c) R_6 (d) R_7

(e) R_8 (f) R_9 (g) R_{10} (h) R_{11}

(i) R_{12} (j) R_{13} (k) R_{14} (l) R_{15}

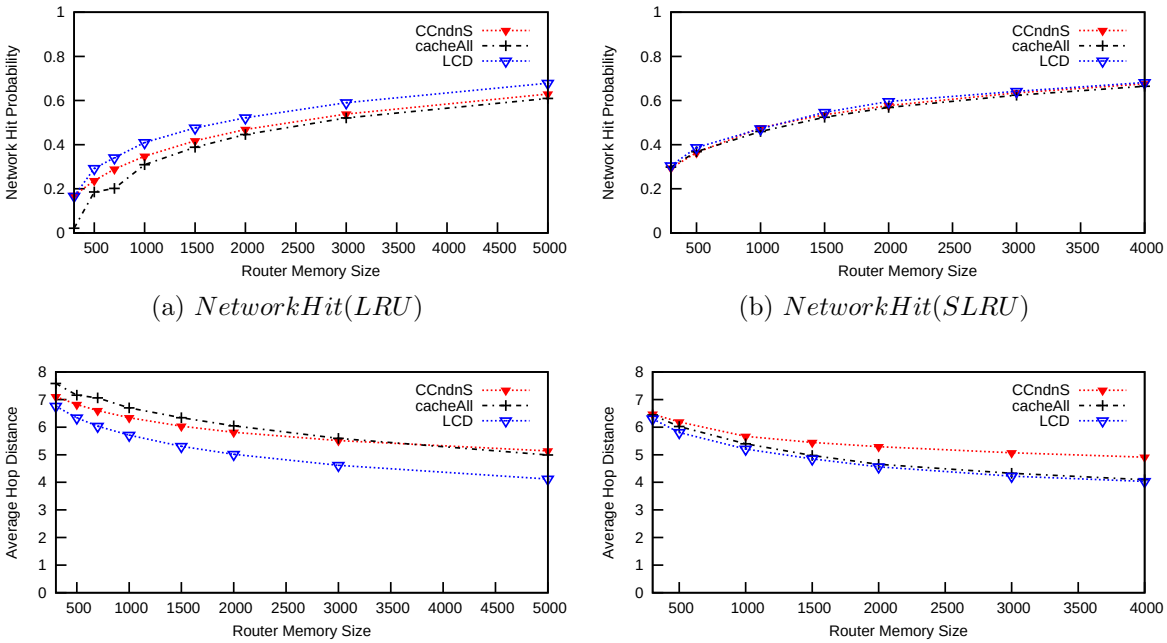
(m) R_{16} (n) R_{17} (o) R_{18} (p) R_{20}

(q) R_{21} (r) R_{22} (s) R_{23} (t) R_{24}

(u) R_{25} (v) R_{30} (w) R_{32} (x) R_{33}

(y) R_{34} (z) R_{35}

Figure B-4: Cache hit rate for all routers.



(a) $NetworkHit(LRU)$ (b) $NetworkHit(SLRU)$

(c) $AverageHopDistance(LRU)$ (d) $AverageHopDistance(SLRU)$

Figure B-5: Network Parameters.

Appendix C

SLA for 5-level Tree Topology

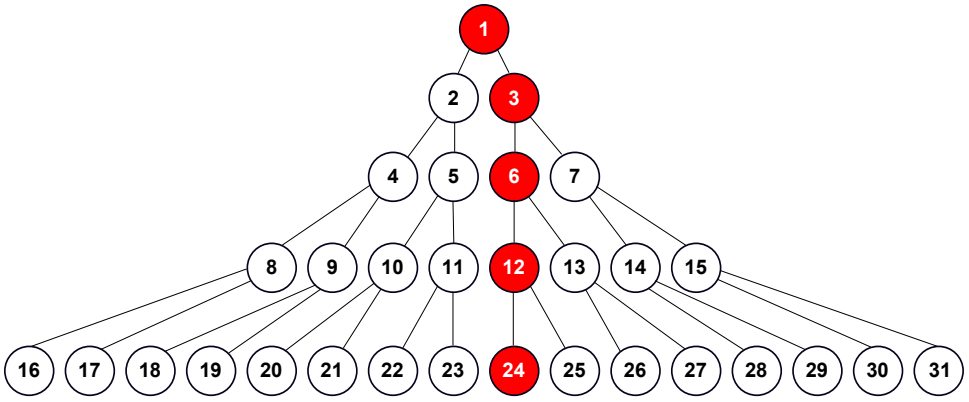
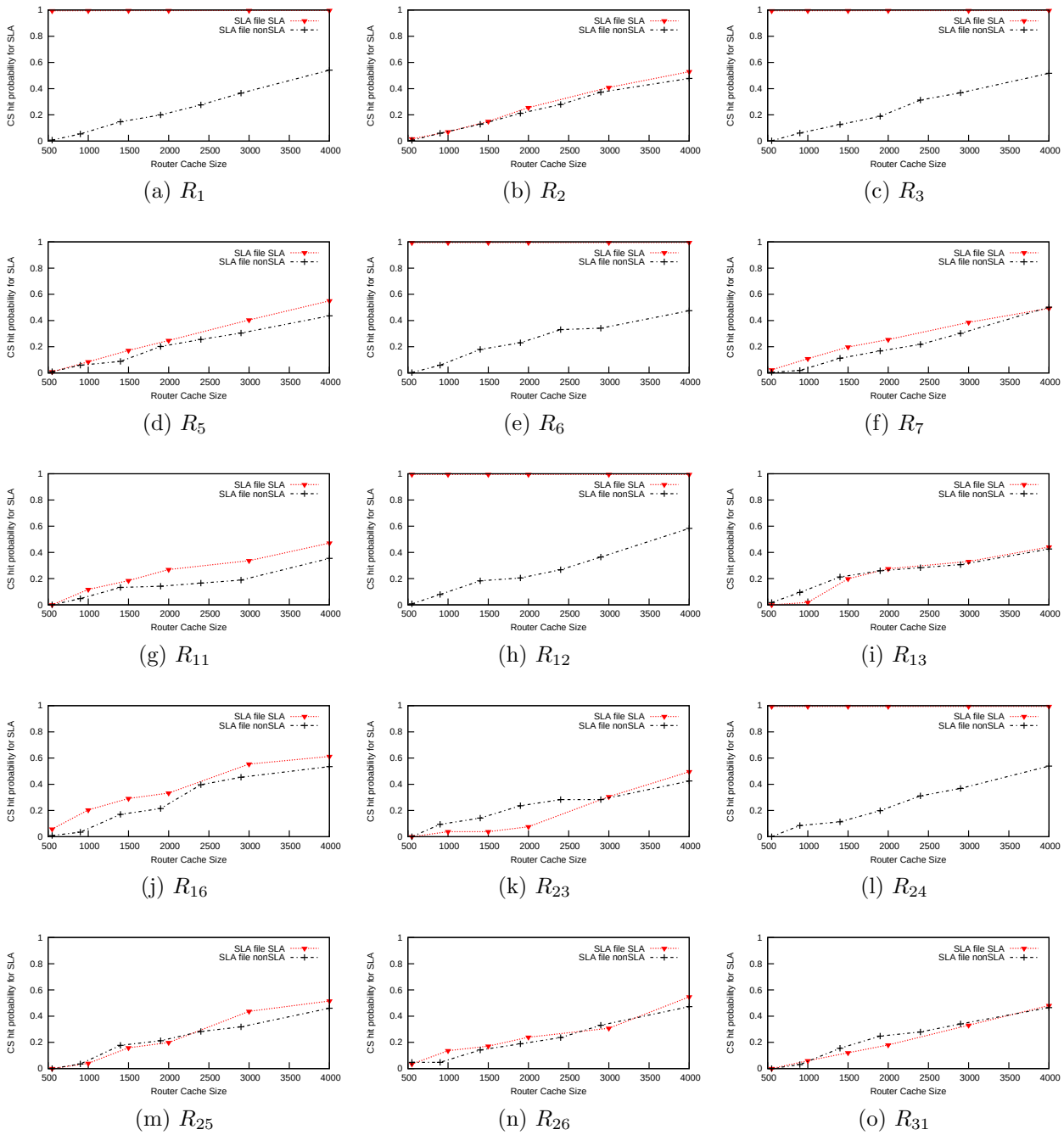


Figure C-1: All 500 files are attached to router $R1$ and each leaf is attached to 20 requesters. Clients on router $R24$ are the target of the SLA contract. The SLA file is the rank 11th file in popularity.

Results presented in this chapter is for a tree network presented in Figure C-1.



(a) R_1 (b) R_2 (c) R_3

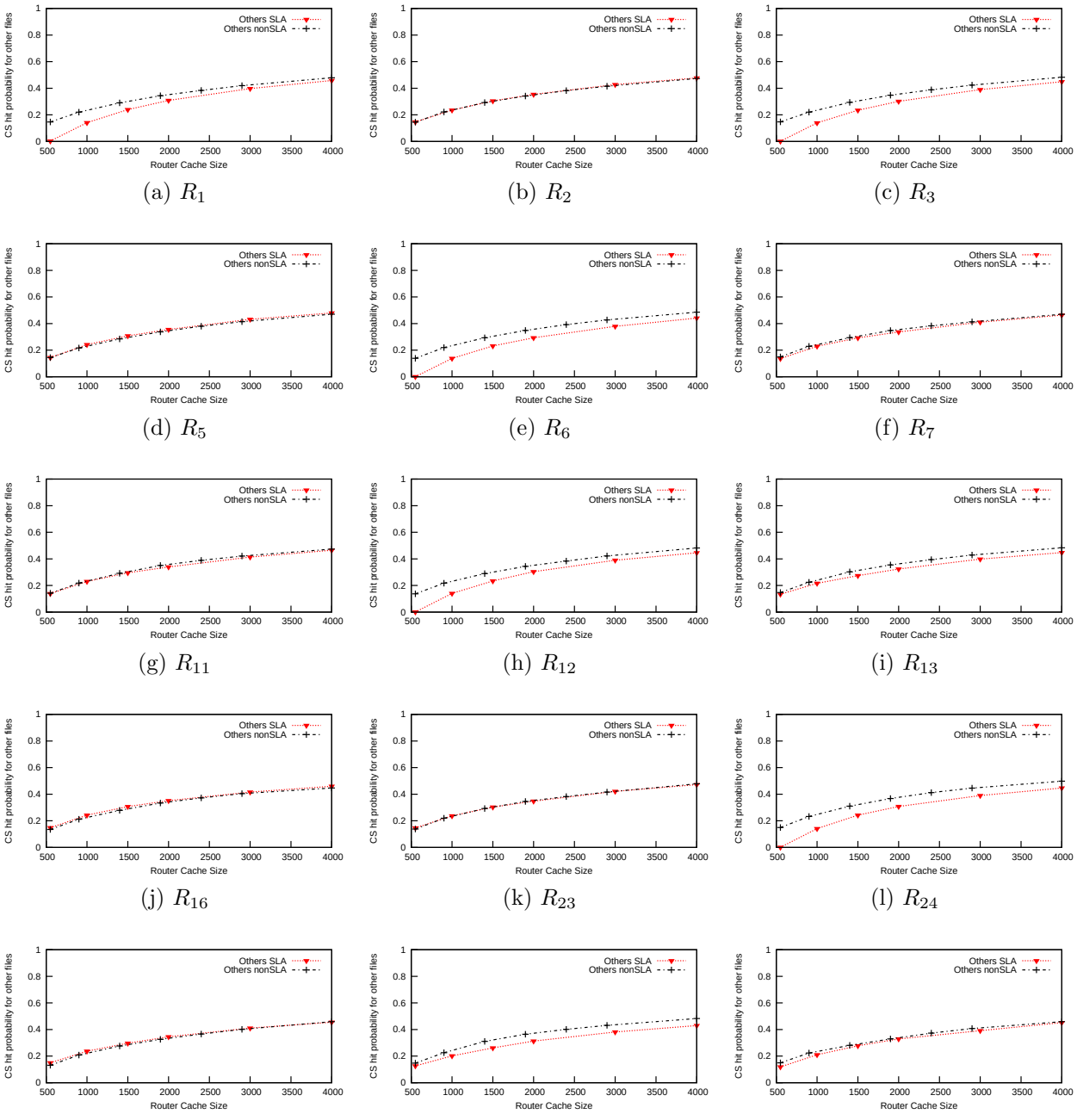
(d) R_5 (e) R_6 (f) R_7

(g) R_{11} (h) R_{12} (i) R_{13}

(j) R_{16} (k) R_{23} (l) R_{24}

(m) R_{25} (n) R_{26} (o) R_{31}

Figure C-2: Compare cache hit rate for the selected file as SLA when there is and there is not SLA agreement.



(a) R_1 (b) R_2 (c) R_3

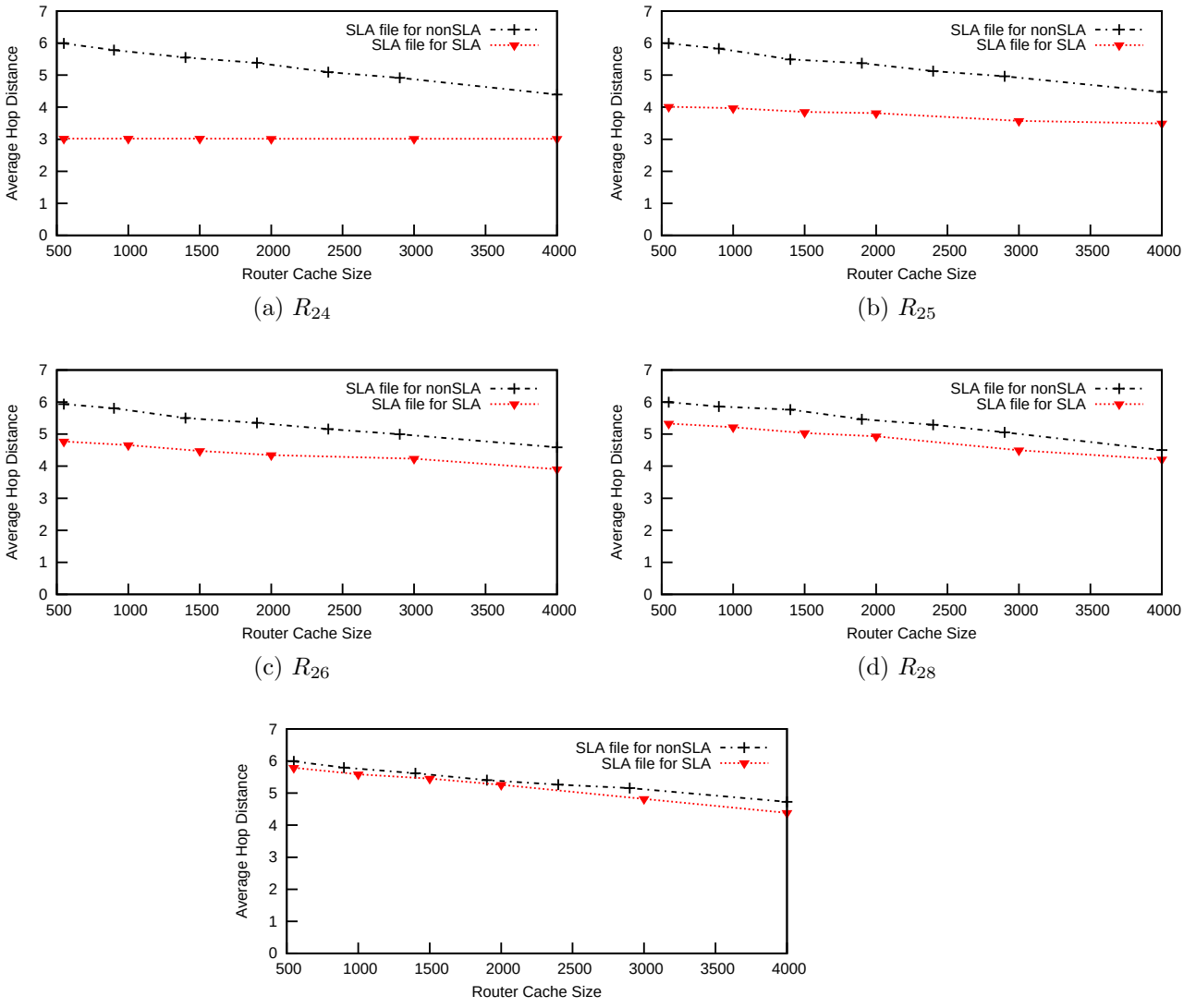
(d) R_5 (e) R_6 (f) R_7

(g) R_{11} (h) R_{12} (i) R_{13}

(j) R_{16} (k) R_{23} (l) R_{24}

(m) R_{25} (n) R_{26} (o) R_{31}

Figure C-3: Compare cache hit rate for the other files except the selected file as SLA when there is and there is not SLA agreement.

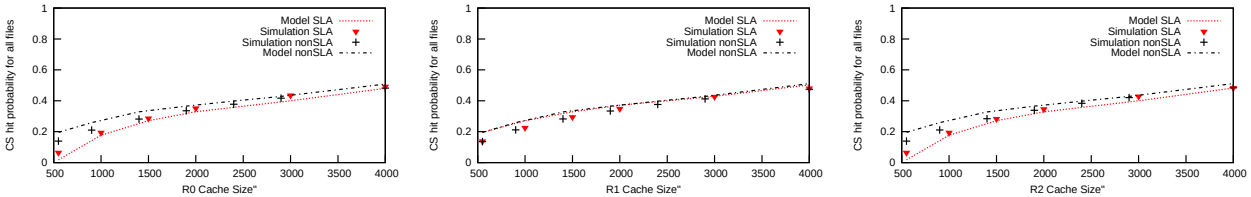


(a) R_{24} (b) R_{25}

(c) R_{26} (d) R_{28}

(e) R_{23}

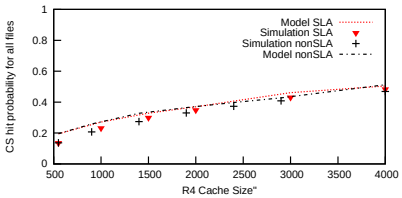
Figure C-4: Compare the average hop distance for the selected file as SLA when there is and there is not SLA agreement. Average hop distance of attached to some of the routers to the source are presented here. SLA agreement for a domain, relatively reduces the hop distance for other domains depends on their distance to the SLA path.



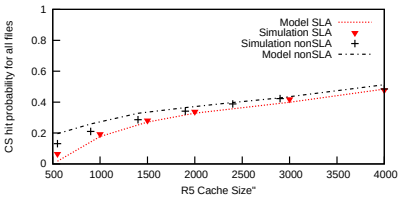
(a) R_1

(b) R_2

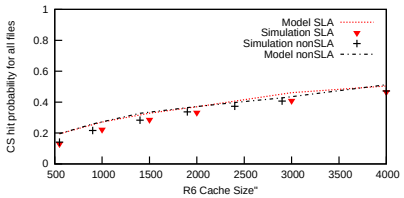
(c) R_3



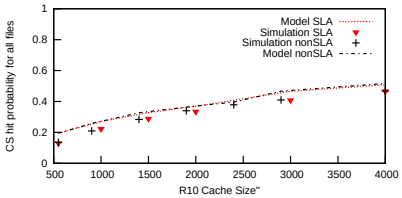
(d) R_5



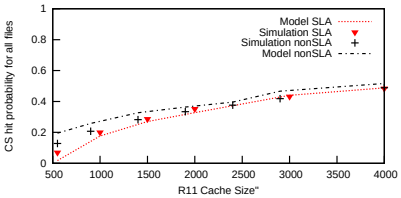
(e) R_6



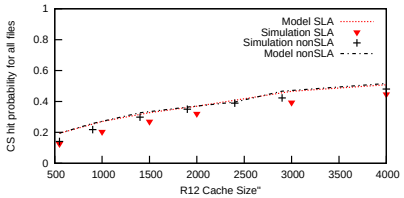
(f) R_7



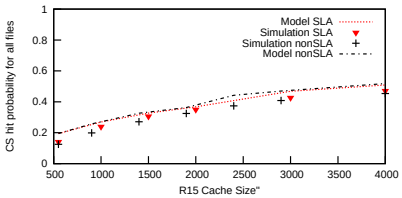
(g) R_{11}



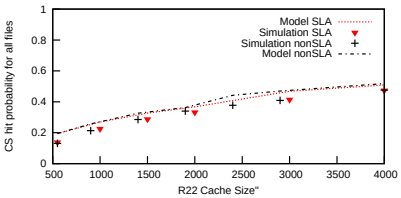
(h) R_{12}



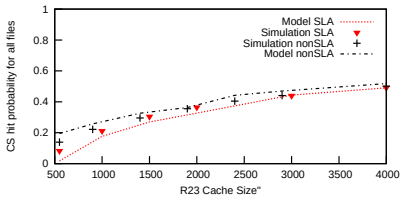
(i) R_{13}



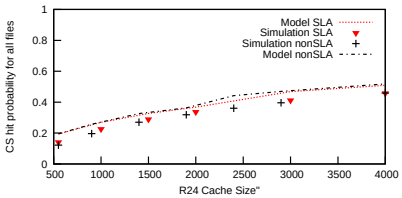
(j) R_{16}



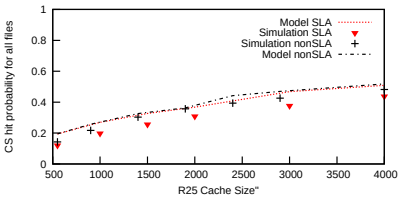
(k) R_{23}



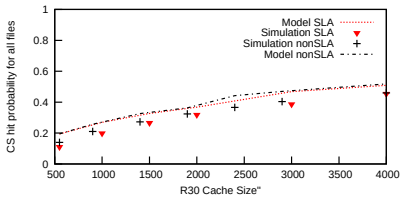
(l) R_{24}



(m) R_{25}



(n) R_{26}



(o) R_{31}

Figure C-5: The model accurately matches with experiments.

Bibliography

[1] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.

[2] B. Ager, F. Schneider, J. Kim, and A. Feldmann. Revisiting Cacheability in Times of User Generated Content. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pages 1–6, March 2010.

[3] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang , Lixia Zhang, and others. NDN, Technical Report NDN-0021. Technical report, University of California, Los Angeles The University of Arizona Colorado State University University Pierre and Marie Curie, Sorbonne University Beijing Institute of Technology Washington University in St. Louis The University of Memphis, Augest 2014.

[4] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in Network Traffic: Findings and Implications. *SIGMETRICS Perform. Eval. Rev.*, 37(1):37–48, June 2009.

[5] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *Proc. ReArch*, pages 5:1–5:6, 2010.

[6] J. Aweya. IP Router Architectures: An Overview. *Journal of Systems Architecture*, 46:483–511, 1999.

[7] A. Badam, K. Park, V. S. Pai, and L. L. Peterson. HashCache: Cache Storage for the Next Billion. In *Proceedings of the 6th USENIX Symposium on Net-*

worked Systems Design and Implementation, NSDI'09, pages 123–136, Berkeley, CA, USA, 2009. USENIX Association.

[8] L. Breslau, P. Cao, L. Fan, et al. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. INFOCOM*, pages 126–134, 1999.

[9] G. Carofiglio, M. Gallo, and L. Muscariello. ICP: Design and evaluation of an Interest control protocol for content-centric networking. In *Proc. INFOCOM Workshops*, pages 304–309, 2012.

[10] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling data transfer in content-centric networking. In *Teletraffic Congress (ITC), 2011 23rd International*, pages 111–118, 2011.

[11] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Evaluating per-application storage management in content-centric networks. *Comput. Commun.*, 36(7):750–757, Apr. 2013.

[12] G. Carofiglio, V. Gehlen, and D. Perino. Experimental Evaluation of Memory Management in Content-Centric Networking. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–6, 2011.

[13] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "Less for More" in Information-centric Networks. In *Proc. IFIP Networking*, May 2012.

[14] H. Che, Y. Tung, and Z. Wang. Hierarchical Web caching systems: modeling, design and experimental results. *JSAC*, 20(7):1305–1314.

[15] H. Che, Z. Wang, and Y. Tung. Analysis and design of hierarchical Web caching systems. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1416 –1424 vol.3, 2001.

[16] Cheriton, D. R. Cheriton and Gritter, Mark. TRIAD: a Scalable Deployable NAT-based Internet Architecture. Technical report, Stanford University, 2000.

[17] L. Cherkasova and M. Gupta. Characterizing locality, evolution, and life span of accesses in enterprise media server workloads. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '02, pages 33–42, New York, NY, USA, 2002. ACM.

[18] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack. WAVE: Popularity-based and collaborative in-network caching for content-oriented networks. In *INFOCOM Workshops*, pages 316–321, 2012.

[19] H. Dai, J. Lu, Y. Wang, and B. Liu. A two-layer intra-domain routing scheme for named data networking. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2815–2820, 2012.

[20] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li. Collaborative hierarchical caching with dynamic request routing for massive content distribution. In *INFOCOM, 2012 Proceedings IEEE*, pages 2444–2452, March 2012.

[21] L. Dong, D. Zhang, Y. Zhang, and D. Raychaudhuri. Optimal Caching with Content Broadcast in Cache-and-Forward Networks. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5, June 2011.

[22] W. Eatherton, G. Varghese, and Z. Dittia. Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, Apr. 2004.

[23] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.

[24] Fayazbakhsh, Seyed Kaveh and Lin, Yin and Tootoonchian, Amin and Ghodsi, Ali and Koponen, Teemu and Maggs, Bruce and Ng, K.C. and Sekar, Vyas and Shenker, Scott. Less Pain, Most of the Gain: Incrementally Deployable ICN. *SIGCOMM Comput. Commun. Rev.*, 43(4):147–158, Aug. 2013.

[25] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *Proc. INFOCOM Workshops*, pages 310–315, 2012.

[26] N. Fujita, Y. Ishikawa, A. Iwata, and R. Izmailov. Coarse-grain replica management strategies for dynamic replication of Web contents. *Computer Networks*, 45(1):19 – 34, 2004. The Global Internet.

[27] J. Garcia, J. Corbal, L. Cerda, and M. Valero. Design and implementation of high-performance memory systems for future packet buffers. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 372–384, 2003.

[28] J. Garcia, M. March, L. Cerda, J. Corbal, and M. Valero. On the design of hybrid DRAM/SRAM memory schemes for fast packet buffers. In *High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on*, pages 15–19, 2004.

[29] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker. Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, pages 1–6. ACM, Aug. 2011.

[30] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: seeing the forest for the trees. In *Proc. HotNets-X*, pages 1:1–1:6, 2011.

[31] G. Grassi, D. Pesavento, G. Pau, R. Vuyyuru, R. Wakikawa, and L. Zhang. VANET via Named Data Networking. In *Computer Communications Work-*

shops (INFOCOM WKSHPS), 2014 IEEE Conference on, pages 410–415, April 2014.

[32] P. Gupta, S. Lin, and N. McKeown. Routing lookups in hardware at memory access speeds. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1240–1247 vol.3, 1998.

[33] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 147–160, New York, NY, USA, 1999. ACM.

[34] J. Hasan, S. Chandra, and T. N. Vijaykumar. Efficient Use of Memory Bandwidth to Improve Network Processor Throughput. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 300–313, New York, NY, USA, 2003. ACM.

[35] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. NLSR: Named-data Link State Routing Protocol. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[36] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *Proc. SYSTOR 2009*, pages 10:1–10:9, 2009.

[37] D. Huang, X. Zhang, W. Shi, et al. LiU: Hiding Disk Access Latency for HPC Applications with a New SSD-Enabled Data Layout. In *Proc. MASCOTS*, pages 111–120, 2013.

[38] H. Hwang, S. Ata, and M. Murata. Realization of name lookup table in routers towards content-centric networks. In *Network and Service Management (CNSM), 2011 7th International Conference on*, pages 1–5, 2011.

[39] Hyesook Lim, Soohyun Lee and Earl E. Swartzlander Jr. A new hierarchical packet classification algorithm. *Computer Networks*, 56(13):3010 – 3022, 2012. Challenges in High-Performance Switching and Routing in the Future Internet.

[40] S. Iyer, R. Kompella, and N. McKeown. Designing Packet Buffers for Router Linecards. *Networking, IEEE/ACM Transactions on*, 16(3):705–717, 2008.

[41] V. Jacobson, D. K. Smetters, J. D. Thornton, et al. Networking named content. In *Proc. CoNEXT*, pages 1–12, 2009.

[42] Z. Jia, P. Zhang, J. Huang, et al. Modeling Hierarchical Caches in Content-Centric Networks. In *Proc. ICCCN*, pages 1–7, 2013.

[43] R. Karedla, J. Love, and B. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, March 1994.

[44] A. Z. Khan, S. Baqai, and F. R. Dogar. QoS Aware Path Selection in Content Centric Networks. In *Next Generation Network Symp.*, 2012.

[45] D. Kulinski and J. Burke. NDN Video: Live and Prerecorded Streaming over NDN. Technical Report NDN-0007, September 2012.

[46] L. Wang, A. K. M. M. Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF-based routing protocol for NDN. Technical Report NDN-0003, July 2012.

[47] N. Laoutaris, H. Che, and I. Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Perform. Eval.*, 63(7):609–634, 2006.

[48] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *Proc. IPCC*, pages 445–452, 2004.

[49] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, and L. Dong. Popularity-driven Coordinated Caching in Named Data Networking. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '12, pages 15–26, New York, NY, USA, 2012. ACM.

[50] Z. Li and G. Simon. Time-Shifted TV in Content Centric Networks: The Case for Cooperative In-Network Caching. In *ICC*, pages 1–6, 2011.

[51] Y. Lu, T. Abdelzaher, and A. Saxena. Design, implementation, and evaluation of differentiated caching services. *Parallel and Distributed Systems, IEEE Transactions on*, 15(5):440–452, 2004.

[52] M. March, J. García, L. Cerdá, and M. Valero. Analysis of a high performance DRAM/SRAM memory scheme for fast packet buffers, 2004-02-14 2004. Madrid, Spain.

[53] Matteo Varvello, Diego Perino, Leonardo Linguaglossa. On the design and implementation of a wire-speed pending interest table. In *IEEE INFOCOM Workshop on Emerging Design Choices in Name-Oriented Networking*, 2013.

[54] M. Meisel, V. Pappas, and L. Zhang. Ad hoc networking via named data. In *Proc. MobiArch*, pages 3–8, 2010.

[55] S. Michel, K. Nguyen, A. Rosenstein, et al. Adaptive Web Caching: Towards a New Global Caching Architecture. *Comput. Netw. ISDN Syst.*, 30(22-23):2169–2177, Nov. 1998.

[56] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and Storage Sharing Performance in Information Centric Networking. In *Proc. ICN*, pages 26–31, 2011.

[57] A. Narayanan and D. Oran. Content Routing using Internet Routing Protocols: Can it scale?, 2011. IETF-82, ICNRG BAR BOF.

[58] V. A. Niels L.M. and F. A. Kuipers. Globally accessible names in named data networking. In *The 2nd IEEE International Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN 2013)*, 2013.

[59] S. Nilsson and G. Karlsson. Fast Address Look-up for Internet Routers. In *Proceedings of the IFIP TC6/WG6.2 Fourth International Conference on Broadband Communications: The Future of Telecommunications*, BC '98, pages 11–22, London, UK, UK, 1998. Chapman & Hall, Ltd.

[60] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, Aug. 2010.

[61] C. Partridge, S. Member, P. P. Carvey, I. Castineyra, T. Clarke, J. Rokosz, J. Seeger, M. Sollins, S. Starch, B. Tober, G. D. Troxel, D. Waitzman, and S. Winterble. A 50-gb/s ip router. *IEEE/ACM Transactions on Networking*, 6:237–248, 1998.

[62] D. Perino and M. Varvello. A reality check for content centric networking. In *Proc. ICN*, pages 44–49, 2011.

[63] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic In-network Caching for Information-centric Networks. In *Proc. ICN*, pages 55–60, 2012.

[64] I. Psaras, R. G. Clegg, R. Landa, et al. Modelling and Evaluation of CCN-caching Trees. In *Proc. IFIP Networking*, pages 78–91, 2011.

[65] M. Rabinovich, J. S. Chase, and S. Gadde. Not all Hits are Created Equal: Cooperative Proxy Caching Over a Wide-Area Network. *Computer Networks*, 30(22-23):2253–2259, 1998.

[66] G. Rétvári, Z. Csernátony, A. Körösi, J. Tapolcai, A. Császár, G. Enyedi, and G. Pongrácz. Compressing IP forwarding tables for fun and profit. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 1–6, New York, NY, USA, 2012. ACM.

[67] M. Rezazad and Y. C. Tay. A cache miss equation for partitioning an NDN content store. In *Asian Internet Engineering Conference, AINTEC '13, Chiang Mai, Thailand, November 13-15, 2013*, pages 1–8, 2013.

[68] M. Rezazad and Y. C. Tay. ndn||mem: An Architecture to Alleviate the Memory Bottleneck for Named Data Networking. In *Proc. CoNEXT Student Workshop*, pages 1–3, 2013.

[69] M. Rezazad and Y. C. Tay. Ccndns: A strategy for spreading content and decoupling ndn caches. In *IFIP Networking, Toulouse, France May 20-22, 2015, appear to*, 2015.

[70] E. Rosensweig and J. Kurose. Breadcrumbs: Efficient, Best-Effort Content Location in Cache Networks. In *INFOCOM 2009, IEEE*, pages 2631 –2635, april 2009.

[71] E. J. Rosensweig, J. Kurose, and D. Towsley. Approximate Models for General Cache Networks. In *Proc. INFOCOM*, pages 1100–1108, 2010.

[72] D. Rossi and G. Rossini. Caching performance of content centric networks under multi-path routing (and more). Technical Report Telecom ParisTech, 2011.

[73] D. Rossi and G. Rossini. On sizing CCN content stores by exploiting topological information. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 280–285, March 2012.

[74] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang. Securing building management systems using named data networking. *Network, IEEE*, 28(3):50–56, May 2014.

[75] A. Sharifi, S. Srikantaiah, A. K. Mishra, M. Kandemir, and C. R. Das. METE: meeting end-to-end QoS in multicores through system-wide resource management. *SIGMETRICS '11*, 39(1):13–24, June 2011.

[76] A. Sharma, A. Venkataramani, and R. K. Sitaraman. Distributing Content Simplifies ISP Traffic Engineering. In *Proc. SIGMETRICS*, pages 229–242, 2013.

[77] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable Routing on Flat Names. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 20:1–20:12, New York, NY, USA, 2010. ACM.

[78] W. So, A. Narayanan, and D. Oran. Named Data Networking on a Router: Fast and DoS-resistant Forwarding with Hash Tables. In *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '13, pages 215–226, Piscataway, NJ, USA, 2013. IEEE Press.

[79] V. Srinivasan and G. Varghese. Faster IP Lookups Using Controlled Prefix Expansion. *SIGMETRICS Perform. Eval. Rev.*, 26(1):1–10, June 1998.

[80] H. S. Stone, J. Turek, and J. Wolf. Optimal partitioning of cache memory. *IEEE Transactions on Computers*, 41(9):1054–1068, 1992.

[81] G. Suh, L. Rudolph, and S. Devadas. Dynamic Partitioning of Shared Cache Memory. *The Journal of Supercomputing*, 28(1):7–26, 2004.

[82] Y. C. Tay. *Analytical Performance Modeling for Computer Systems*. Morgan & Claypool, 2013.

[83] Y. C. Tay and M. Zou. A page fault equation for modeling the effect of memory size. *Perform. Eval.*, 63(2):99–130, Feb. 2006.

[84] D. N. Tran, P. C. Huynh, Y. C. Tay, and A. K. H. Tung. A new approach to dynamic self-tuning of database buffers. *Trans. Storage*, 4(1):3:1–3:25, May 2008.

[85] D. N. Tran, W. T. Ooi, and Y. C. Tay. SAX: A Tool for Studying Congestion-Induced Surfer Behavior. In *Passive and Active Measurement Conference*, 2006.

[86] G. Tyson, S. Kaune, S. Miles, et al. A Trace-Driven Analysis of Caching in Content-Centric Networks. In *Proc. ICCCN*, pages 1–7, 2012.

[87] M. Varvello, D. Perino, and J. Esteban. Caesar: a content router for high speed forwarding. In *Proc. ICN*, pages 73–78, 2012.

[88] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable High Speed IP Routing Lookups. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '97, pages 25–36, New York, NY, USA, 1997. ACM.

[89] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen. Scalable name lookup in NDN using effective name component encoding. In *Proc. ICDCS*, 2012.

[90] Y. Wang, K. Lee, B. Venkataraman, R. L. Shamanna, I. Rhee, and S. Yang. Advertising cached contents in the control plane: Necessity and feasibility. In *INFOCOM Workshops'12*, pages 286–291, 2012.

[91] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie. Optimal cache allocation for Content-Centric Networking. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10, Oct 2013.

[92] A. Wolman, M. Voelker, N. Sharma, et al. On the Scale and Performance of Cooperative Web Proxy Caching. In *Proc. SOSP*, pages 16–31, 1999.

[93] W. Wong, M. V. L. Giraldi, M. F. Magalhes, and J. Kangasharju. Content routers: Fetching data on network path. In *ICC'11*, pages 1–6, 2011.

[94] W. Wong, L. Wang, and J. Kangasharju. Neighborhood search and admission control in cooperative caching networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2852–2858, Dec 2012.

[95] H. Wu, J. Li, Y. Wang, and B. Liu. EMC: The Effective Multi-Path Caching Scheme for Named Data Networking. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–7, July 2013.

[96] H. Xie, G. Shi, and P. Wang. TECC: Towards collaborative in-network caching guided by traffic engineering. In *INFOCOM, 2012 Proceedings IEEE*, pages 2546–2550, March 2012.

[97] G. Xylomenos, C. N. Ververidis, V. A. Siris, et al. A Survey of Information-Centric Networking Research. *IEEE Commun. Surveys and Tutorials*, 16(2):1024–1049, 2014.

[98] Yi Wang, Keqiang He, Huichen Dai, Wei Meng, Junchen Jiang, Bin Liu, Yan Chen. Scalable Name Lookup in NDN Using Effective Name Component Encoding. In *2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, June 18-21, 2012*, pages 688–697, 2012.

[99] Yi Wang, Yuan Zu, Ting Zhang, Kunyang Peng, Qunfeng Dong, Bin Liu, Wei Meng, Huicheng Dai, Xin Tian, Zhonghu Xu, Hao Wu, Di Yang. Wire Speed Name Lookup: A GPU-based Approach. In *NSDI 2013*, pages 199–212, 2013.

[100] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proc. EuroSys*, pages 333–344, 2006.

[101] H. Yuan and P. Crowley. Scalable Pending Interest Table design: From principles to practice. In *INFOCOM, 2014 Proceedings IEEE*, pages 2049–2057, April 2014.

[102] H. Yuan, T. Song, and P. Crowley. Scalable NDN Forwarding: Concepts, Issues and Principles. In *Proc. ICCCN*, 2012.

[103] Z. Zhu, C. Bian, A. Afanasyev, V. Jacobson, and L. Zhang. Chronos: Serverless multi-user chat over NDN. Technical Report NDN-0008, October 2012.

[104] M. Zec, L. Rizzo, and M. Mikuc. DXR: Towards a Billion Routing Lookups Per Second in Software. *SIGCOMM Comput. Commun. Rev.*, 42(5):29–36, Sept. 2012.

[105] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR)*, 44(3):66–73, Jul 2014.

[106] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, et al. Named Data Networking (NDN) Project. Technical Report NDN-0001, PARC Tech Report 2010-003, PARC, October 2010.

[107] Z. Zhu, J. Burke, L. Zhang, P. Gasti, Y. Lu, and V. Jacobson. A New Approach to Securing Audio Conference Tools. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, pages 120–123, New York, NY, USA, 2011. ACM.