ABSTRACT

WANG, YAOGONG. Caching, Routing and Congestion Control in a Future Information-Centric Internet. (Under the direction of Dr. Injong Rhee.)

Information-Centric Networking (ICN) has recently drawn the attention of the research community as a fundamental design principle shared by many future Internet architecture proposals. The key vision of ICN is that, the primary application of the current Internet is large-scale content distribution but IP, the core protocol of the Internet, was designed for host-to-host conversation. This model mismatch is the root cause of such problems as flash crowd, traffic redundancy and denial-of-service (DoS) attack. ICN proposes to change the basic primitive of the Internet from "sending packets to named hosts" to "fetching named contents to users". By using named contents rather than host identifiers as the central abstraction of the architecture, ICN is able to support today's and future Internet applications more efficiently.

Existing research into ICN is at an early stage and many designs are still sketchy. To make ICN a practically working system at Internet scale, there are still many missing pieces to be filled in. In this dissertation, we study three most important design issues in ICN, namely caching, routing and congestion control. By proposing specific cache deployment strategies, routing protocols and congestion control mechanisms, we make ICN a more viable architecture for the future Internet.

Pervasive in-network caching is a feature advocated by almost all ICN architectures. Since caching is inherently supported by ICN, many existing studies automatically assume that caches should be deployed on every ICN router. In this dissertation, we re-examine this implicit assumption and evaluate the performance of pervasive in-network cache deployment via theoretic analysis as well as large-scale simulation. Both real and synthesized network topologies and traffic traces have been used to drive our analysis and simulations. Our conclusion is that pervasive caching is far from the optimal cache deployment strategy and would not bring significant performance improvement over traditional network caching mechanisms even if nearest replica

routing is used. Hence, future ICN proposals should not automatically assume its presence.

Routing is another important issue in ICN. Given the exceptionally large number of contents in the Internet and the appeal to use completely location-independent names, it is very challenging to design a scalable and efficient ICN routing protocol. Existing proposals either resort to a global name resolution service or rely on hierarchical aggregation, both of which may incur high routing stretch. In this dissertation, we propose to use limited-scope Bloom filter advertisement to assist ICN routing and demonstrate that it could significantly improve the routing performance at minimum cost. The possibility to extend this idea to a standalone routing protocol at Internet scale has also been discussed. We believe that Bloom filter-based routing is a feasible and valuable performance enhancer to existing ICN routing protocols and has the potential to be a competitive standalone solution to scalable ICN routing in future.

Congestion control in ICN differs significantly from traditional TCP congestion control due to its pull-based model and inherent support for multipoint retrieval. In this dissertation, we propose a novel congestion control scheme for ICN that incorporates hop-by-hop proactive interest shaping, timely and reliable congestion signaling, proper client reaction, as well as a multipath forwarding strategy to utilize available bandwidth from multiple paths in an efficient and fair manner. Our simulation results show that the proposed scheme is able to effectively avoid congestion collapse, achieve near-optimal throughput and maintain reasonable fairness among competing flows across a large number of different scenarios, including those with multiple paths or in-network caching.

Caching, Routing and Congestion Control in a Future Information-Centric Internet

by
Yaogong Wang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2013

APPROVED BY:

Dr. Rudra Dutta	Dr. George Rouskas
Dr. David Thuente	Dr. Injong Rhee Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

Yaogong Wang was born in Chengdu, Sichuan, China. He received a bachelor's degree in Communication Engineering from University of Electronic Science and Technology of China in 2005 and a master's degree in Electronic Engineering from Shanghai Jiao Tong University in 2008. Since August 2008, he pursued his Ph.D. degree in Computer Science at North Carolina State University. His research interest lies in the design and implementation of networking protocols.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Injong Rhee, for his constant encouragement and generous support throughout my Ph.D. journey. Without his guidance I would never acquire the skill set I currently possess and grow to the engineer I am today. I would also like to thank other members in my advisory committee, Dr. Dutta, Dr. Rouskas and Dr. Thuente, for their insightful comments on my dissertation.

During my doctoral research, I was lucky to have worked with many great colleagues. Many thanks to Kyunghan Lee, Dave Oran, Ashok Narayanan, Won So, Mark Stapp, Natalya Rozhnova, Yuanjie Li, Balakrishna Venkataraman, Ravi Shamanna and Sunhee Yang who have contributed to the research projects in this dissertation. I am also grateful to other members of our lab, including Sangtae Ha, Hyungsuk Won, Seongik Hong, Jeongki Min, Sankararaman Janakiraman, Sungro Yoon, Haiqing Jiang, Seungeun Chung, Nithyananthan Poosamani, Arpit Gupta, Sohee Han, Shuiqing Wang, Puneet Arora, Alphonse Selvanayagam and many others.

Last but not least, I would like to thank my friends, Yongmin Tan, Xing Wu, Zhengzhang Chen and Zhenhuan Gong among many others, who shared the precious memories of my Ph.D. life. I'm also indebted to Pingwu Xu, Zao Wang, Biwen Zhu and Haoming Cai who helped me settle down in the United States during the first few days of my arrival. Finally, my very special thanks to Dr. Wu, without whose inspiration I may never consider pursuing a Ph.D. degree, and my beloved parents and wonderful wife, without whose support I could never finish this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The current Internet architecture was developed in the 1960s and 70s, way before the World Wide Web (WWW) or online social networks ever exist. Back then, the primary application of the Internet was to share expensive and scarce *physical resources* (e.g. telnet to a mainframe computer). People cared about *where* the resources are and hence designed the Internet in a host-centric fashion to facilitate point-to-point conversation between named hosts. IP, the core protocol of the Internet, exactly reflects this design philosophy: hosts are first level objects in this architecture and are given globally unique names (IP addresses). Each IP datagram has a source IP address and a destination IP address. All communications are about sending packets from a named host to another named host.

The world has changed dramatically since then. With the proliferation of personal computers and more recently portable devices like tablets and smart phones, computing resources are becoming cheaper and cheaper. The primary use of the Internet today is no longer to share physical resources but to share *digital information*. People care about *what* the information is rather than *where* it is fetched from. To bridge the gap between the information-centric applications and the host-centric Internet architecture, clients have to map the contents that the user wants to the machine that hosts them. This mapping is the root cause of a number of problems with the current Internet, such as flash crowd, traffic redundancy and denial-of-service

(DoS) attack.

Take Web browsing for instance. When a user types a URL (Uniform Resource Locator) into the address bar of his/her browser, the browser identifies the domain name part of the URL and uses DNS (Domain Name System) to resolve it into the IP address of the Web server. It then establishes a TCP connection to that server at port 80 and fetches a specified file under a specified directory on that server. This type of operation is so common in today's Internet applications that people take it for granted. In fact, this is exactly the cause of problems like flash crowd or DoS attack: everyone has to visit a specific server to obtain the content and this creates a single point of failure. This also introduces a lot of redundant traffic in the network since each content is wrapped in an end-to-end connection. An intermediate router cannot identify that two hosts are fetching the same content unless it does costly deep packet inspection (DPI).

Fundamentally, these problems are due to the host-centric nature of the current Internet: contents are second level objects in this architecture and are named relative to their hosts. A number of techniques have been proposed to mitigate these issues, including Web caching, Content Delivery Networks (CDN), Dynamic DNS, HTTP redirection and peer-to-peer (P2P) systems. However, these add-ons are often ad-hoc and application-specific. They also have plenty of side effects that bring new issues to the Internet (e.g., [102, 95, 35]).

Instead of adding another patchwork, Information-Centric Networking (ICN) tries to solve this problem at the architectural level. By making contents first level objects and giving them globally unique, location-independent and persistent names, ICN profoundly changes how the Internet works and is a promising candidate for the future Internet architecture. There are numerous different instantiations of the ICN idea [48, 64, 57, 117, 5, 9, 4, 44, 45] and we will survey them in Section 3.1. But the following features are shared by most ICN architectures:

- **Publish/Subscribe Primitive:** the primitive operation in an ICN architecture is to fetch a named content. The client specifies the name of the content he/she wants and the network retrieves the content data for him/her. Since the content name is location-

independent, the network can fetch the content from anywhere. This extra degree of freedom is the key to facilitate large-scale content distribution and to solve many existing problems in the current Internet. For instance, the network may fetch the content from the nearest cache so as to minimize user-perceived latency. The network may also balance the load among multiple content sources to avoid server overload in case of flash crowd. In contrast, the client specifies a specific destination in IP and the network just delivers the packet to that destination. The unnecessarily *early binding* of a content with a location is the root cause of such problems as flash crowd and difficulty in mobility.

Another notable implication of the Publish/Subscribe primitive is that all traffic are receiver-driven. This nature has profound impacts on the design of many ICN mechanisms such as traffic prioritization, spam avoidance and in our case, congestion control. Especially, some ICN architectures (e.g., [57, 117]) enforce symmetric routing on the receiver-driven traffic. This opens a lot of new possibilities in the design of effective congestion control algorithms for ICN. Most notably, proactive congestion control may be possible, i.e, by controlling the rate of the requests, we may ensure that the returning content data will not overload the links. We will further discuss this in Chapter 6.

- **Name-based Routing:** in ICN, routers forward requests based on the name of the requested content rather than any address. This is the key enabler of many ICN benefits but also creates severe scalability problems. First, the number of contents in the Internet is much more than the number of hosts. Second, hosts can be assigned location-dependent names (e.g., IP addresses) that can be hierarchically aggregated while content names are assumed to be location-independent. Finally, host movement or network topology change happens at a much larger time scale than content movement. The combination of these three factors makes routing on location-independent names at Internet scale extremely difficult. We will address this problem via Bloom filter-based routing in Chapter 5.

- **Content Caching/Multihoming/Migration:** each content in ICN is given a globally

unique and persistent name. The content data along with its name and other meta information is signed by its publisher. This makes each content a *self-contained* entity that could reside anywhere and be served to anyone. No matter where the content resides, users always refer to it in the same way since its name is location-independent. No matter where the content is fetched from, users can independently verify its integrity via the digital signature. Hence, content caching/multihoming/migration are intrinsically supported by the ICN architecture.

One corollary of this inherent caching support, which many existing studies (e.g., [73, 30, 26, 106, 68, 91]) take for granted, is that caches could be deployed pervasively on every ICN router across the entire network. At first sight, this appears to be an obviously good idea since it could potentially reduce the load on the servers that host popular contents, eliminate redundant traffic in the network and decrease the latency perceived by end users. In Chapter 4, we will quantitatively evaluate such benefits and re-examine the automatic assumption of pervasive in-network cache deployment.

- **Content-based Security:** in ICN, each content is digitally signed by its publisher. The client or any network entity could verify the integrity of the content by this signature. This is completely different from most existing security mechanisms who aim to secure the communication channel between the sender and the receiver. This content-based security model is also a key component of ICN. Of course, a basic digital signature does not address all the security requirements of all applications. More complicated security mechanisms are required to provide more sophisticated services (e.g., who can access whose contents in an online social network). We do not address any specific ICN security issue in this dissertation but we always bear this content-based security model in mind when designing our schemes.

1.1 Contributions

The contributions of this dissertations can be summarized as follows:

1. We formulated the cache deployment problem in ICN as an Integer Linear Program (ILP) and developed a greedy heuristic algorithm to solve it in polynomial time. Based on the numerical results obtained over a series of realistic problem instances, we concluded that provisioning cache pervasively on all nodes in a network is sub-optimal.

2. We complemented the theoretic analysis with large-scale simulation to evaluate the performance of pervasive in-network caching. After exploring a large number of different scenarios, we concluded that pervasive caching brings little (if any) performance improvement over simple edge caching and nearest replica routing adds limited help on top of that. Hence, future ICN proposals should not blindly assume pervasive in-network cache deployment.

3. We proposed Bloom filter-assisted routing to reduce the routing stretch of existing named-based ICN routing protocols. By advertising Bloom filters within limited scope (which is perfectly feasible with today's technology), we were able to discover nearby content copies and avoid the high stretches incurred by name resolution or ignorance of such copies due to aggregatability constraints. We evaluated the performance improvement brought by Bloom filter assistance quantitatively via simulation.

4. We explored the possibility to extend Bloom filter-based routing to an Internet-scale routing protocol for ICN and evaluated the resulting scalability implications. We pointed out a number of scalability advantages of Bloom filter-based routing and also pinpointed the potential difficulties in adopting it. We believe that, although extending Bloom filter-based routing to the Internet scale may be challenging today, it is a promising direction to pursue in future.

5. We designed a novel hop-by-hop interest shaper for ICN congestion control that takes into

account the unique interdependence between interest packets and returning data packets, and provably achieves proportional fairness between two-way traffic.

6. We complemented the shaping algorithm with state-of-the-art active queue management (AQM) schemes to generate congestion signals timely and to avoid standing queues and unnecessary delay. We also showed that the congestion signals will never be lost due to congestion in the presence of hop-by-hop shaping. When coupled with proper client reaction, the entire congestion control scheme achieves efficient and fair utilization of the network resources.

7. We proposed a multipath forwarding strategy to improve the throughput of multipath flows in the absence of competing traffic and ensure fairness in the presence of competing flows. It also serves to balance congestion and achieve resource pooling across multiple paths [109].

1.2 Dissertation Roadmap

The rest of the dissertation are organized as follows:

Chapter 2 gave an overview of our ICN architecture to set the context for our later discussions. Among the numerous ICN instantiations, we picked Named Data Networking (NDN [117], also known as Content-Centric Networking or CCN [57]) as the basis of our architecture due to its popularity and readiness (open source prototype [3] and ns-3-based simulator [8] are both available). We detailed the NDN architecture in Section 2.1 and identified the missing pieces of the picture in Section 2.2. We also showed how our work fills in these missing pieces and makes ICN a step further toward a practically working system.

Chapter 3 discussed the related work. We started by surveying alternative ICN architectures proposed by researchers all around the world (Section 3.1). We then moved on to discuss papers related to the three specific topics we studied in this dissertation, namely ICN caching (Section 3.2), ICN routing (Section 3.3) and ICN congestion control (Section 3.4).

Chapter 4 detailed our evaluation of pervasive in-network caching in ICN. Section 4.1 described how we formulated the problem as an ILP and proposed a greedy heuristic algorithm to solve it. We then generated a few realistic problem instances and solved them using the proposed algorithm. The numerical results demonstrated the sub-optimality of pervasive caching. Section 4.2 elaborated our simulation setup and results. By comparing the performance of edge caching and pervasive caching under various cache replacement and routing algorithms, we further confirmed the limited value of pervasive cache deployment. The implications of our findings on ICN were discussed in Section 4.3 and we concluded the chapter in Section 4.4.

Chapter 5 presented our Bloom filter-based ICN routing proposal. We motivated our work by pointing out the problems with existing ICN routing algorithms that rely on either a global name resolution service or hierarchical aggregation (Section 5.1). Bloom filter assisted routing as a complement to the existing routing schemes was elaborated in Section 5.2 and evaluated by simulation in Section 5.3. In Section 5.4 we discussed the possibility to extend Bloom filter-based routing to a full-fledged ICN routing scheme at Internet scale. Finally, we summarized our work in Section 5.5.

Chapter 6 provided our complete solution to the congestion control problem in ICN. We motivated our work by pointing out the unique characteristics of ICN that make traditional TCP congestion control mechanisms unsuitable (Section 6.1). Then we described our hop-by-hop interest shaping algorithm as well as the theoretic background behind it in Section 6.2. How the congestion is signaled back to the clients and how the clients react to the signals were elaborated in Section 6.3. Section 6.4 then discussed the design of our multipath forwarding strategy. We evaluated each component of our congestion control scheme as well as the overall performance in Section 6.5 and concluded in Section 6.6.

Finally, Chapter 7 summarized the entire dissertation and pointed out a few possible directions for further research.

Chapter 2

Architectural Overview

The previous chapter has outlined the basic idea of ICN and its most important features. However, there is still a considerable gap between these conceptual principles and a practically working system. In this chapter, we first provide a detailed introduction to the NDN architecture, which is a popular instantiation of ICN proposed by Jacobson et al. [57, 117]. NDN defines specific name conventions, packet formats, forwarding engine models etc. for ICN and is arguably the most practical ICN proposal. Hence, we use it as the basis of our ICN architecture and build our proposals on top of it.

Despite the relative completeness of the NDN architecture, there are still quite a few missing pieces in this picture. We identified three most important missing pieces and addressed them one by one in this dissertation. By filling in these missing pieces in the global picture, we further bridge the gap between ICN theory and practice and make it a more viable architecture for the future Internet.

2.1 A Global Picture

The basic networking primitive of ICN is to fetch a named content. This is reflected in NDN by the definition of two basic packet types, namely *Interest* and *Data*.. An interest packet expresses a client request by specifying the name of the data packet he/she wants along with

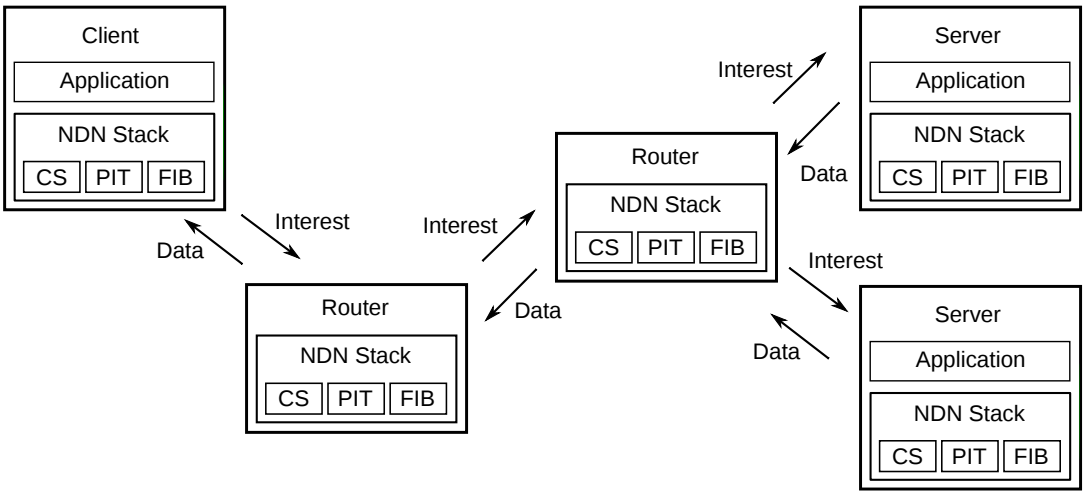


Figure 2.1: Overview of the NDN architecture

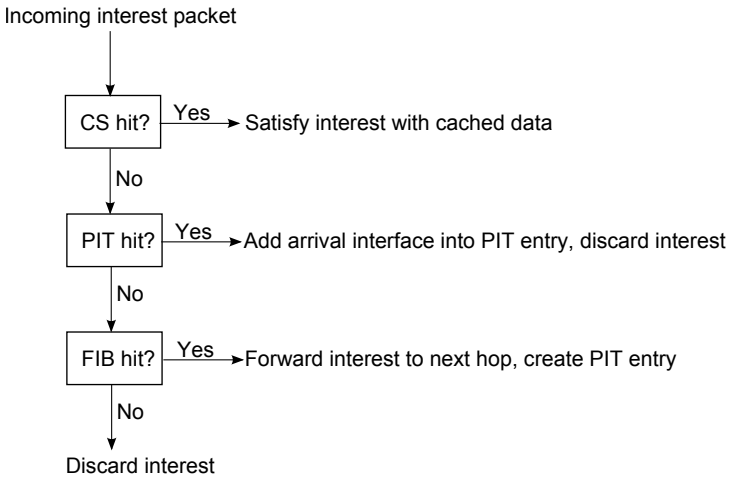
some additional control information. Any node who hears the interest and has the requested data can respond to it with a data packet. Data packets are transmitted only in response to interests and cannot be sent voluntarily. There is a strict one-for-one mapping between an interest packet and the data packet it fetches.

Note that, although all ICN architectures adopt such a receiver-driven communication paradigm, NDN differentiates itself from other proposals by enforcing this primitive at *packet level*. A large content item needs to be segmented into packet-sized chunks and each chunk (i.e., data packet) is given a unique and persistent name. The client issues a series of interest packets, each for one data packets, to fetch all the chunks and obtain the entire content item.

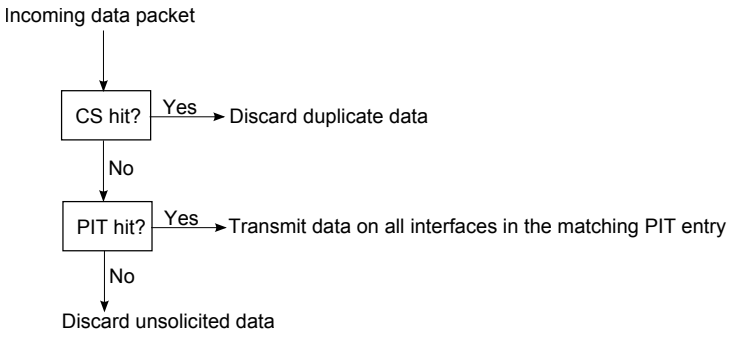
NDN uses hierarchical names for the data packets. Routing is carried out directly on the names or name prefixes without explicitly resolving them into addresses. Chunks of a large content item may be named under the same prefix and aggregated in the routing domain.

Figure 2.1 gives an overview of the NDN architecture. The core packet forwarding engine in the NDN stack has three major components: the Content Store (CS), the Pending Interest Table (PIT) and the Forwarding Information Base (FIB). They work as follows:

Upon receiving an interest, the router first looks up CS to see if the requested data packet is



(a) Processing interest packets



(b) Processing data packets

Figure 2.2: Forwarding process of an NDN node

cached locally. If so, the interest is satisfied immediately with the locally cached copy. Otherwise, we look up PIT to see if the same data packet has been requested before by someone else and that previous interest is still pending. If so, the router adds the arrival interface of this interest into the matching PIT entry and discards the interest (since a request has already been sent upstream). If no PIT entry match is found, the router looks up the FIB and forward the interest to the next hop. A new PIT entry is created for this interest. In case no FIB entry is found for this interest, the router simply discards the packet.

Upon receiving a data packet, the router first looks up CS to see if we already have a copy of it. If so, this packet can be safely discarded. Otherwise, the router may choose to cache this new

data packet based on its cache replacement policy. Next, a lookup is performed on PIT. If there is a match, the data packet is transmitted on all interfaces that have requested it. Otherwise, the data packet is unsolicited and is discarded.

Figure 2.2 depicts the packet processing flow described above. Compared to IP nodes who mainly rely on the FIB, NDN nodes have the additional components of CS and PIT. CS is easy to understand: it is basically a cache. However, PIT needs some more explanation as it provides a number of important functions: first, data packets do not look up the FIB at all. They follow the "bread crumbs" (i.e., the PIT entries) left by the interest, thus ensuring symmetric routing of each interest/data pair. Second, PIT implicitly supports multicast functionality. When multiple interests for the same data packet go through the same router at roughly the same time, the router will not forward the interests except the first one. Since the data packet goes through the same path as the interest, the router will receive the requested data as long as one interest is forwarded upstream. It then duplicates the data packet and satisfy all the interests. This reduces redundant traffic in the Internet. Third, by letting data packets follow the PIT entries, NDN removes the need for source addresses or host identifiers, alleviating privacy concerns. Finally, PIT enables stateful forwarding [112] which could help detect routing loops, react faster to link failures or congestion and so on. Someone may be concerned with the overhead incurred by PIT and hence its scalability. We refer the readers to [83, 70] that examined this issue and [69, 114, 32] that proposed corresponding solutions.

Note that the introduction above only covers part of the NDN architecture that is most relevant to our research in this dissertation. For more details such as name conventions and packet formats, please refer to [57, 117, 3].

2.2 The Missing Pieces

As can been seen from the previous section, NDN has a very clear and detailed definition of its data plane but is brief in defining the control plane and the transport layer. For the control plane, the authors only propose to extend existing link state routing protocols such

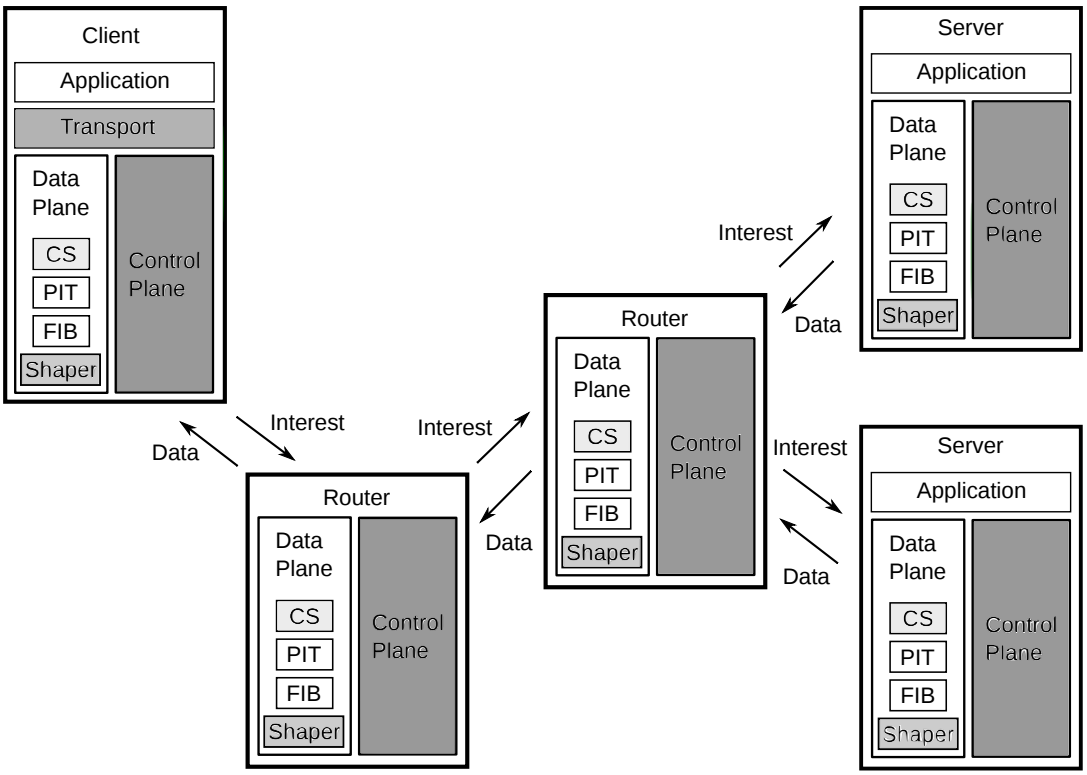


Figure 2.3: How our work fits into the global picture

as OSPF [75] and IS-IS [78] and advertise name prefixes in addition to adjacencies. However, the scalability and efficiency of this approach is in question [76]. As for the transport layer, it is almost unspecified in the original proposal. Most notably, a congestion control mechanism for NDN is missing. Without an effective congestion control scheme to ensure efficient and fair utilization of the network resources, NDN can hardly be considered as a fully working system. Finally, the introduction of CS in the core NDN stack implicitly equips every NDN node with caching capability. Whether this pervasive cache deployment is necessary and how we should dimension the cache on each node is also an open question.

Figure 2.3 depicts how our work in this dissertation fits into the global picture. Compared to Figure 2.1, our contributions have been highlighted by the shaded areas. Chapter 4 looked at the content stores (CS) and evaluated how pervasive in-network cache deployment may impact the content delivery performance. We found that, although ICN intrinsically supports

caching, deploying caches at every router does not seem to be an efficient strategy. Hence, we rectified a potential misconception in ICN. Chapter 5 then looked into the control plane issues and proposed a Bloom filter-based approach that can either complement existing NDN routing protocols as a performance enhancer or run independently as a standalone routing protocol. Finally, Chapter 6 tackled the congestion control problem in NDN. Note that, instead of a TCP-like end-to-end congestion control mechanism where transport layer protocols run at the two end points, our congestion control scheme consists of interest shapers at every hop and proper reactions on the client side only (as depicted in the figure).

Obviously, there are still other missing pieces in this picture that are not addressed in this dissertation. However, we believe that the three topics we picked are central to take ICN toward a practically working system. We also want to emphasize that, despite using NDN as our target platform, many findings and proposed schemes in this dissertation have a broad application scope and can be used in other ICN architectures as well.

A final note on the terminologies used in this dissertation: we use ICN to refer to the general idea and use NDN and CCN interchangeably to refer to the specific architecture proposed in [57, 117]. Interests are sometimes also referred to as requests while data packets may be called contents or responses.

Chapter 3

Related Work

3.1 ICN Architectures

ICN is a new way to engineer computer networks with a top-down approach. We make contents the first level objects in the network and design a network that facilitates large-scale content distribution. In contrast, the current Internet was built with a bottom-up approach: we first built an IP network that provides global host-to-host packet delivery service and then developed all the information-centric applications on top of it. Due to the mismatch of the application model and the networking model, we have to employ a lot of inelegant translations and mappings which lead to such problems as flash crowd, traffic redundancy and DoS attack.

In this section, we provide a survey of the most prominent ICN architectures in the literature. Please refer to [31] and [10] for more complete surveys of ICN. For a broader survey on future Internet architecture proposals (including those that are not ICN-based), please refer to [80].

3.1.1 TRIAD

The basic idea of ICN dates back to the TRIAD project [48] as early as 2001. To the best of our knowledge, it is among the first to realize the architectural difficulty of the current Internet to support large-scale content distribution and the inadequacies of ad-hoc patchworks

like dynamic DNS or Web caching. It then proposed a new Internet architecture where contents are identified by names rather than addresses and the name-to-address resolution is done by the content routers deployed all over the Internet rather than specific DNS servers. Although TRIAD routing still operates at the granularity of server names rather than full URL of each individual content, it contains many characteristics of an ICN architecture such as location-independent naming and low-stretch name-based routing.

3.1.2 DONA

Data-Oriented Networking Architecture (DONA [64]) first extended the ICN idea to content (or datum) granularity. In DONA, each datum is given a self-certifying flat name and name resolution is done by resolution handlers (RH). Each domain or administrative entity has one logical RH and these RHs form a hierarchy according to the provider/customer/peer relationships among these domains or entities. Content providers publish content names by sending $REGISTER$ messages to its local RH. Upon receiving a $REGISTER$ message from its customer, an RH forwards the message to its provider and peers. The $REGISTER$ messages set up the registration table on each RH so that $FIND$ messages (which look up content names) can be routed effectively later on. If a $FIND$ message matches an entry in the registration table, it is forwarded to the next hop indicated by that entry. Otherwise, the $FIND$ message is forwarded to the provider RH until it reaches tier-1 ISPs. The DONA proposal is a clean-slate redesign of the naming and name resolution part of the Internet. However, it does not change the narrow waist of the hourglass Internet architecture. Name-based routing is done at resolution speeds rather than line speeds. Once we have located the content by the $FIND$ message, the actual data packet exchange is done by standard IP.

3.1.3 CCN/NDN

More recently, Jacobson et al. proposed the Content-Centric Networking (CCN [57]) architecture which puts named contents at the thin waist of the protocol stack. This proposal has

ignited wide-spread interest in the research community and became one of the Future Internet Architecture (FIA) projects funded by National Science Foundation (NSF) under the new name of Named Data Networking (NDN [117]). We have already elaborated this architecture in detail in Chapter 2 since it is the basis of our ICN architecture. However, we want to reiterate that most of the findings and schemes in this dissertation (especially those in Chapter 4 and Chapter 5) are also applicable to other ICN architectures.

3.1.4 PSIRP/PURSUIT

There are quite a few ICN projects supported by European Union FP7 (The Seventh Framework Programme) as well. Publish-Subscribe Internet Routing Paradigm (PSIRP), later succeeded by the Publish-Subscribe Internet Technology (PURSUIT) project is one of them [44]. In PSIRP/PURSUIT, a rendezvous system is used to match publications and subscriptions. Scoping and filtering functions are added to address the routing scalability problem [27]. A subscription request is resolved into a forwarding identifier (FI) by the rendezvous system and sent to the content source. This FI consists of a Bloom filter that encodes the information of a multicast tree to distribute the content to all its subscribers. The content source then puts this FI in the packet header and sends the packet out. Routers can utilize the FI to make forwarding decisions to multicast the content to the subscribers [59].

3.1.5 4WARD/SAIL NetInf

Network of Information (NetInf [9]) was initially conceived in the FP7 4WARD project [5]. Later on, it has evolved further under FP7 SAIL project [4]. One interesting aspect of NetInf is its multi-level representation of information: an information object (IO) represents the high-level semantics of the information (e.g., a song). It may refer to multiple data objects (DO) which are actual bits that contain the information (e.g., .mp3 version of the song, or .wav version of the same song). This extra level of indirection gives the system more expressive power to represent the users' intentions. As for routing, NetInf adopts a hybrid approach that supports

both name resolution [33] and name-based routing.

3.1.6 COMET

COntent Mediator architecture for content-aware nETworks (COMET [45]) is another FP7-funded project on ICN. It features a mediation plane provided by the Internet Service Providers (ISPs) that lies between the physical networks and the upper content layer. This mediation plane is aware of content locations, server and network conditions and routing paths information so that it can ensure content delivery with guaranteed Quality-of-Service (QoS). COMET proposes two implementation approaches, namely decoupled and coupled. They roughly correspond to the resolve-and-retrieve and route-by-name approaches which will be discussed in Section 3.3.

3.1.7 Content-based Networking

Content-based Networking [24] differentiates itself from all above-mentioned ICN architectures by its unique naming scheme. Each content is identified by a set of attribute/value pairs in this architecture rather than a hierarchical name or a flat label. Accordingly, a user request is expressed as a selection predicate which are a logical disjunction of conjunctions of elementary constraints over the values of individual attributes. Thus, the matching of publication and subscription is not longer a prefix match or an exact match. Instead, it is a search process to find contents that match the selection predicates declared by the client. Due to the completely different semantics, routing and forwarding in content-based networking require completely different techniques from other ICN architectures. The proposals in this dissertation do not apply to this unique variant of ICN.

3.2 ICN Caching

Standalone Web caches have been extensively studied in the literature [104, 84]. A number of different cooperative Web caching mechanisms have also been proposed [72, 28, 36]. However, none of them studied the problem in the context of pervasive caching at the network layer.

With the recent interest in ICN, pervasive in-networking caching has become a hot topic in the literature [86, 23, 60, 100, 91]. Many content replacement algorithms have been proposed in this new context [73, 30, 26, 106, 68, 85, 96, 97]. A few papers also tried to address the problem of nearest replica routing under pervasive caching [89, 67, 107]. In this dissertation, we are not comparing the performance of these proposals. Instead, we are questioning the presumption of pervasive in-network cache deployment which is implicitly assumed in most of these studies.

[47] was the first to question this premise but its arguments were based on indirect evidences in past literature [111, 49, 13]. More recently, [38] also looked into this problem and we arrived at very similar conclusions independently. One potential limitation of [38] is that it used Least Recently Used (LRU) replacement across all its simulations. This may bias the results since LRU replacement has been shown to be especially inappropriate for pervasive caching due to cache redundancy issues. Furthermore, we also formulate an ILP to derive the optimal cache deployment scheme in addition to running simulations. Most recently, [108] has also studied the cache allocation problem in ICN and formulated the problem as an optimization problem. Their formulation differs from our model in that they assume on-path caching and always use shortest path routing. In contrast, our model assumes nearest replica routing.

Formulating caching as an optimization problem is a commonly used technique. It has been applied in many different contexts such as Video-on-Demand [12], CDN [15] and ICN [106, 68]. However, they all focused on the content placement problem under a given per-router cache size. [66] was among the few papers that investigate optimal cache sizing under a given total budget. It formulated the problem as an ILP similar to our model but was limited to hierarchical topologies. In contrast, our formulation is applicable to arbitrary topology and is more suitable in the context of ICN. [92] also studies the cache deployment problem in ICN but only takes topological information into consideration. In contrast, we also considered the impacts of many other traffic-related factors such as content popularity, original server location distribution, client request rate distribution, etc.

3.3 ICN Routing

Two most important goals in ICN routing are scalability and low stretch. Unfortunately, these two goals are often contradictory and each design has to make a trade-off between them. At one end of the spectrum is DHT-based (Distributed Hash Table) routing which is great in scalability but often incurs high stretch. At the other end of the spectrum is link state routing (similar to IP routing) which ensures shortest path but has severe scalability problems in ICN. Existing proposals typically lie somewhere in between, leaning closer to one end or the other. They can be roughly classified into two categories: resolve-and-retrieve or route-by-name. The former first resolves content names into addresses and then use the address to make actual forwarding decisions. Instead, the later routes directly on location-independent names.

3.3.1 Resolve-and-retrieve Approaches

Multi-level DHT (MDHT [33]) tries to reduce the routing stretch of DHT by building a hierarchy of DHTs that roughly reflects the actual network topology. Names that can be resolved in a lower-level local DHT do not need to go to the global DHT and hence have lower resolution latency and routing stretch. DMap [103] instead creates multiple global $O(1)$ DHTs with the help of IPv4 reachability information already available in today's routers. Since we need only one overlay hop for each name resolution and can always pick the one with minimum actual latency out of all DHTs, DMap is able to finish 95% of resolutions within 86ms when there are 5 DHTs.

DHTs incur long resolution latency and high routing stretch because they are topology-agnostic. DONA [64] and CURLING [27] try to reduce the stretch by building a hierarchical and distributed database that matches the network topology. In their schemes, each Autonomous System (AS) maintains a resolution server (called Resolution Handler in DONA and Content Resolution Server in CURLING) and these resolution servers form a hierarchy according to the provider/customer/peer relationships among the ASes. Content publishing follows similar path to prefix advertisement in today's IP routing: each AS advertises its customers' contents names

(equivalent of address prefixes in IP) to its provider and peers. Name resolution/routing follows the FIB set up by the advertisements with the default route to the provider AS. This type of ICN routing lies in the middle of DHT-based routing and link state routing. It reduces the routing stretch of DHT since it is topology-aware (at AS level). It is also feasible with today's technology since there is one logical resolution server per AS and can hence be implemented as a server farm or even data center which can easily sustain the scalability requirements.

3.3.2 Route-by-name Approaches

OSPFN [105], which extends the existing link state routing protocol of the Internet (OSPF [75]) with name prefix advertisement, is the ICN routing scheme that strives for minimum stretch at the cost of scalability. By pushing topology-awareness to the router level (as compared to AS level in DONA and CURLING), OSPFN is able to find the shortest path to a content. However, this puts heavy burden on the routers as they need to keep track of a large number of content names. Considering the difficulty to apply hierarchical aggregation in ICN, this scheme does not seem to be feasible at Internet scale with today's technology [76]. The authors of OSPFN later extended the protocol to NLSR [53] which is still a link state protocol but runs on top on NDN itself. NLSR has a number of practical advantages over OSPFN but does not change the scalability properties of the routing protocol. Hence, it may suffer from the same issue mentioned earlier.

There are other more "wild" proposals for ICN routing. [101] examines blind routing in ICN while [82] proposes greedy routing in a hyperbolic metric space. Both schemes are topology-oblivious and require few (if any) routing table entries in each router. Hence, they are completely free from scalability concerns. However, the applicability of these futuristic proposals to ICN routing remains to be verified.

3.3.3 Bloom Filters

In this dissertation, we proposed Bloom filter-based routing for ICN. Bloom filter was originally proposed in 1970 by [14]. It has a number of variants such as counting Bloom filter [36], compressed Bloom filter [74], dynamic Bloom filter [50], scalable Bloom filter [11] and incremental Bloom filter [37]. For a complete survey of their applications in networking, please refer to [19].

Bloom filter has been widely used to facilitate packet *forwarding*. [34] uses it in the longest prefix match operation of IP forwarding. By performing parallel queries on Bloom filters to determine the *length* of the longest matching prefix and then looking up the corresponding hash tables, it provides better performance and scalability than TCAM-based IP lookup. BUFFALO [115] instead uses Bloom filter for packet forwarding in enterprise and data center networks with flat address spaces. It stores one Bloom filter for each next-hop and puts all the addresses that should be forwarded via that next-hop into the filter. Packet forwarding is simply Bloom filter lookups with some false positive mitigation techniques. LIPSIN [59] also utilizes Bloom filters in multicast forwarding. It encodes the link IDs of the multicast delivery tree into a Bloom filter and places it in the packet header. Routers can simply AND the ID of each outgoing link with the Bloom filter in the packet header and make fast forwarding decisions. Note that all these schemes are pure data plane solutions. Control plane issues are assumed to be handled out of band. In contrast, our scheme uses Bloom filter for both routing and forwarding.

Bloom filter has been used in *routing* as well. [88] used it in OceanStore [65], a global-scale distributed peer-to-peer storage system. They proposed the attenuated Bloom filter which is similar to our scheme: each node maintains an array of Bloom filters for each interface. The ith Bloom filter is the merger of all Bloom filters for the nodes that are i hops away through any path starting with that interface. However, their routing update algorithm uses differential compression which is not as robust as our scheme. SCAN [67] also proposed to use Bloom filters to exchange information on cached contents in information-centric networks. It differs from our scheme in that SCAN routers advertise aggregated Bloom filter in a way similar to distance-vector routing algorithms and maintain only one Bloom filter for each interface. Since

aggregated Bloom filters have higher fill ratio, they cannot be heavily compressed. In contrast, the link state advertisements in our scheme are typically very sparse Bloom filters that can be compressed to smaller sizes.

3.4 ICN Congestion Control

NDN congestion control proposals can be classified into two categories: purely receiver-based mechanisms and schemes that require network-assistance.

3.4.1 Receiver-based Approaches

ICP [20] was among the first receiver-based congestion control proposals for NDN. It is a window-based mechanism that closely mimics TCP AIMD behavior. The difficulty here is how to properly set the timeout value for the interest packets in the presence of extensive content multihoming and caching. ICP took a simple approach to set the timer to the mean of the minimum and maximum RTT despite of the potentially diverse content sources. This was later improved in [22] by adding route labels to the data packets so that the end client can identify multiple paths and maintain separate RTT estimation for each path. CCTCP [94] took this further by asking the client to list the subsequent data packets it intends to request in each interest packet and letting the routers indicate whether they have cached such data packets. Thus, the receiver could *predict* the locations where future interests will be satisfied and maintain separate congestion states for each. This predictive approach was also proposed in [17] with some refinements to reduce the complexity and overhead of the protocol.

[16] compared the performance of these receiver-based congestion control strategies and concluded that receiver-based, timeout-driven congestion control is generally *not* suitable for NDN due to the unpredictability of the content locations.

3.4.2 Network-assisted Approaches

Congestion control schemes that incorporate hop-by-hop interest shaping significantly alleviate the problem. Rather than letting the receiver to infer congestion by timeout, congestion is detected at the bottleneck link by the interest shaper and signaled to the end clients explicitly. A number of hop-by-hop interest shapers have been proposed in the literature.

[113, 112] proposed the simplest algorithm where the interest rate is shaped so that the returning data rate matches the reverse link capacity multiplied by a pre-determined coefficient between 0 and 1. A more sophisticated shaping algorithm was proposed in [93] where the shaping rate is dynamically adjusted based on the instantaneous queue length and available capacity to send data packets at time t.

[79] presented a *per-flow* bandwidth fair sharing mechanism. Interests exceeding the fair share are discarded with NACKs and simple AIMD schemes are used to adjust the client's interest rate. ICP was also extended in [21] by adding a hop-by-hop interest shaping scheme where each flow is allocated a fair share of the link bandwidth and shaped accordingly. Our proposal is unique in that: 1) we are the first to identify the interdependence between the interests and contents in reverse directions and our scheme achieves fair allocation of bandwidth in the presence of bidirectional traffic; 2) by adopting state-of-the-art AQM schemes in the shaper interest queues, we timely generate congestion signals to avoid excessive queuing delay and our shaping scheme ensures that NACKs have a very low probability of getting dropped, thus avoiding the traditional problem with congestion-generated NACKs exacerbating the congestion problem; 3) we develop a multipath forwarding strategy that effectively utilizes available bandwidths from multiple paths but still ensures fair sharing of the bandwidth with single-path flows.

3.4.3 Multipath Congestion Control

Multipath congestion control has been extensively studied in the current Internet. [62] and [51] laid out the theoretical foundation for multipath congestion control. [110] proposed a practical algorithm for multipath TCP [43] based on the theoretical results. [63] further improved [110] to

achieve optimal resource pooling and responsiveness simultaneously. Although these algorithms are not directly applicable to NDN due to its dramatically different transport model, we are able to leverage the principles behind these algorithms in the design of our multipath forwarding strategy for NDN.

Chapter 4

Pervasive In-network Caching: to Be or Not to Be?

ICN inherently supports network caching. In ICN, each content is a self-contained entity since it has a globally unique and persistent name and can be independently verified via its publisher's digital signature. Hence, the content could be cached anywhere in the network and be served to any user. As caching becomes so easy in ICN, why not do it pervasively across the entire network? At first sight, this appears to be an obviously good idea since it could potentially reduce the load on the servers that host popular contents, eliminate redundant traffic in the network and decrease the latency perceived by end users. In fact, pervasive in-network caching is advocated by almost all ICN architectures that many people take it for granted. They automatically assume that cache space should be deployed on every ICN router and propose a number of schemes under the unexamined assumption of pervasive in-network cache deployment [73, 30, 26, 106, 68, 91].

In this chapter, we re-examine this premise using both theoretic analysis and large-scale simulation. We first formulate cache deployment as an optimization problem, i.e., given a total cache budget, how should we divide it among all the nodes in the network so as to maximize the cache performance? By solving this optimization over realistic problem instances, we found

that the optimal cache deployment is highly uneven, thus arguing against pervasive cache provisioning. We complement the theoretic analysis with large-scale simulation to evaluate the performance of pervasive in-network caching. After exploring a large number of different scenarios, we conclude that pervasive caching brings little (if any) performance improvement over simple edge caching and nearest replica routing adds limited help on top of that. Hence, future ICN proposals should not blindly assume pervasive in-network cache deployment.

Note that this does not argue against the ICN idea itself. ICN is not just about pervasive in-network caching. ICN can stand by itself even if we do not deploy caches pervasively: it could greatly simplify the caching operation even for traditional proxy servers (no more deep packet inspection, no need to maintain end-to-end TCP semantics, application-agnostic). It could ensure content availability even if the data is moved to another site (no more broken links). It could provide better security by authenticating the data itself rather than the communication channel. None of these benefits are tied to the deployment of pervasive in-network caching.

4.1 Theoretic Analysis

The design of a network caching system consists of three sub-problems: 1) cache deployment, i.e., at which nodes should the caches be deployed and how much cache space should be provided? 2) content (re)placement, i.e., which content should be cached where? 3) routing, i.e., how should we forward a request to the cache or server that has the requested content? In this chapter, we focus on the first sub-problem of cache deployment. However, the performance of the system is also dependent on the content (re)placement and routing schemes being used. We solve this problem as follows: we first obtain the optimal cache deployment algorithm under optimal content placement and idealistic routing and compare it against pervasive caching. We then evaluate the performance of pervasive caching under practical content replacement and routing schemes via simulation.

4.1.1 Problem Formulation

We formulate the ICN cache deployment problem as an ILP. The *input parameters* to this problem include:

- A connected undirected graph $G = (V, E)$ that represents the topology of the network.
- The distance d_{ij} between every pair of vertices (i, j) that reflects the performance metric we would like to optimize (e.g., latency, hop count, cost).
- A set C of $|C|$ unit-sized contents. This assumption is reasonable since many ICN architectures segment contents into similar-sized chunks for flexible distribution and control.
- The original servers' location, denoted by a binary variable s_{jk} that equals to one if content k is served on vertex j and zero otherwise. Each content has exactly one server, i.e., $\sum_{j \in V} s_{jk} = 1, \forall k \in C$
- The popularity distribution p_k for each content $k \in C$, $\sum_{k \in C} p_k = 1$
- The average request rate $\lambda_i \geq 0$ for each vertex $i \in V$
- A total cache budget of B units where B is an integer greater than zero.

Note that, unlike many existing cache formulations where the cache size of *each vertex* is given [106, 68, 12, 15], we are only given a *total cache budget* and the division of the budget among all vertices is left to be optimized. The purpose of our model is to determine the optimal cache sizes rather that the optimal content placement.

As discussed before, we determine the optimal cache deployment strategy under optimal content placement and idealistic routing. Hence, we assume that contents are statically cached (prefetched) in the vertices and no dynamic content replacement is taking place. We also assume that a vertex (client) always fetches a specific content from the nearest vertex that has the content (either a cache or the original server). With these assumptions, the *decision variables* of the optimization problem can be defined as follows: let x_{ijk} denote a binary variable that

equals to one if vertex i fetches content k from vertex j and zero otherwise. Also let y_{jk} denote a binary variable that equals to one if content k is cached by vertex j and zero otherwise. These two variables are not independent and their relationship is depicted by Equation 4.1: a content can be fetched by a user from a node only if it is cached on that node or the node is the original server of that content.

$$y_{jk} = \begin{cases} 1 & \text{if } (1 - s_{jk}) \sum_{i \in V} x_{ijk} > 0; \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

With the above-mentioned input parameters and decision variables, the cache deployment problem can be formulated as follows:

Objective:

$$\min \sum_{i \in V} \lambda_i \sum_{k \in C} p_k \sum_{j \in V} d_{ij} x_{ijk} \tag{4.2}$$

Subject to:

$$\sum_{j \in V} x_{ijk} = 1, \forall i \in V, k \in C \tag{4.3}$$

$$(1 - s_{jk}) \sum_{i \in V} x_{ijk} \leq |V| \cdot y_{jk} \tag{4.4}$$

$$\sum_{j \in V} \sum_{k \in C} y_{jk} \leq B \tag{4.5}$$

The objective is to minimize the total cost for the clients to fetch contents from the servers or caches. This cost is weighted by the popularity of each content and the request rate from each client (Equation 4.2). Constraints to this problem include: 1) a client always fetches a specific content from the same unique vertex (Equation 4.3); 2) the relationship between x_{ijk} and y_{jk} (Equation 4.4, equivalent to Equation 4.1 but in a linear form); and 3) the total cache budget (Equation 4.5).

Note that the solution to this optimization problem will prescribe which contents should be cached where. But that is not the purpose of our formulation. Our purpose is to decide how the cache budget should be divided among all the vertices and the optimal content placement

prescription just implicitly sets the optimal cache size for each vertex. Hence, we would like to reiterate that, despite its similarity to many existing cache formulations that aim at optimal content placement under given *per-node* cache capacities, our model tries to derive the optimal cache deployment strategy under a constrained total cache budget with the assumption of optimal content placement and idealistic routing.

Directly solving the formulated ILP is in fact NP-hard. We hence propose a greedy heuristic algorithm to obtain the near-optimal solution. It works as follows: we start from empty caches on all vertices and place a content to a vertex in each iteration. Thus, after B iterations the cache budget is all spent and the algorithm terminates. The resulting content placement prescription and cache size for each vertex are the solution.

In each iteration the $(vertex, content)$ pair (caching this content in this vertex) is greedily picked based on the gain it brings. The gain of a $(vertex, content)$ pair (j, k) at each round is defined as:

$$gain_{jk} = \sum_{i \in V} \lambda_i p_k \cdot \max\{0, d_{is} - d_{ij}\} \tag{4.6}$$

where s is the nearest vertex from which i could obtain content k up until the end of the previous iteration. Initially, s should be the original server of each content but it changes as we progress. In each iteration, we calculate the gain for all the $(vertex, content)$ pairs that have not been picked in the previous rounds and choose the one with maximum gain. We then update the corresponding d_{is} and start the next iteration. Placement decisions made in previous iterations are not changed or affected by subsequent iterations. The complexity of this greedy heuristic algorithm is $O(B|V|^2|C|)$.

4.1.2 Problem Instance Generation

To exercise the formulation proposed above, we need some problem instances as input and would like them to realistically represent the workloads of the current and future Internet. This section elaborates how we generate such problem instances.

A problem instance for cache performance evaluation consists of two major components:

the network topology and the traffic running on top of it. To make our problem instances realistic, we use real ISP topologies and real traffic traces collected from the current Internet. The difficulty here is that we need a traffic trace that spreads across the entire ISP network so as to effectively evaluate the performance of pervasive in-network caching. However, most traffic traces available today are collected at a single point in the network (e.g., the access link of a university). The best we could find is the IRCache traces [2] which are simultaneously collected at eight geographically different locations in the U.S. (Boulder, CO; New York, NY; Palo Alto, CA; Research Triangle Park, NC; San Diego, CA; San Jose, CA; Silicon Valley, CA; Urbana-Champaign, IL). We map the IRCache traffic traces to the networks of three U.S. ISPs (Qwest, Sprint, XO) measured by [98] so as to obtain three realistic problem instances to drive our optimization.

The mapping of traffic traces to ISP topologies is conducted as follows. Each record in the traffic trace consists of such information as the client IP address and the requested URL. The client IP address is randomly mapped to a node in the ISP topology since it has been anonymized. However, the original server of each requested content is set to the node in the ISP network that is geographically closest to the actual server of the requested URL. This distance-based mapping is possible since the ISP maps provided by [98] not only show the topologies of the ISP networks but also the geographical location of each node. The location of the requested URL's server is determined by resolving the domain name (not anonymized) and looking up MaxMind's free GeoLite City database [1].

With this mapping, we can now replay the IRCache traffic traces over the three ISP networks and use them as the input to our formulation. However, since both the network topology and the traffic trace are collected from the current Internet, they only faithfully represent the workloads of the *current* Internet. In a *future* information-centric Internet, the traffic patterns may be dramatically different from today and we have no way to know how they will look like. To address this concern, we also generate a few synthesized problem instances based on the models of the current Internet. We then vary the parameters in these synthesized models across a wide

range so as to explore the parameter space that the future Internet may reach. We do not claim that our exploration is exhaustive. However, we believe it does provide us some useful insights into the problem.

The network topologies of the synthesized problem instances are generated by the transit-stub model of GT-ITM [116]. It has 3 transit domains with 3 transit nodes each. Attached to each transit node are 3 stub domains with 4 nodes in each domain. This results in a 117-node network. The transit domains are fully connected while nodes within each transit domain and stub domain are connected with probability 0.6 and 0.3 respectively. The scales for the top level, the transit domain and the stub domain are set to 20, 20 and 10 respectively. The resulting edge length is used as the link latency (in milliseconds). All links are assumed to have infinite bandwidths. We randomly generate 10 topologies and all experiments have been repeated over these 10 topologies to calculate 95% confidence intervals. Note that, clients and servers are assumed to be attached to the stub nodes only.

10,000 contents exist in this synthesized network and each content has exactly one original server. The distribution of the original servers of the contents follows Zipf distribution defined as below:

$$p_N(i) = \frac{C}{i^\alpha}, \text{ for } i = 1, 2, \cdots, N \tag{4.7}$$

where N is the total number of nodes in the network, i is the rank and

$$C = (\sum_{i=1}^{N} \frac{1}{i^\alpha})^{-1} \tag{4.8}$$

A larger α means most contents are served on a few popular nodes while a smaller α means more uniform distribution. This distribution is chosen based on our analysis of the IRCache traffic traces. As shown in Figure 4.1, in log-log scale, the head of the distribution is roughly a straight line although the tail decays much faster. Hence, we model the original server distribution as Zipfian and fit the head part of the curve. Table 4.1 shows the values of α for each of the 16 IRCache traces and we pick 0.75 as the baseline parameter setting. We will vary this parameter

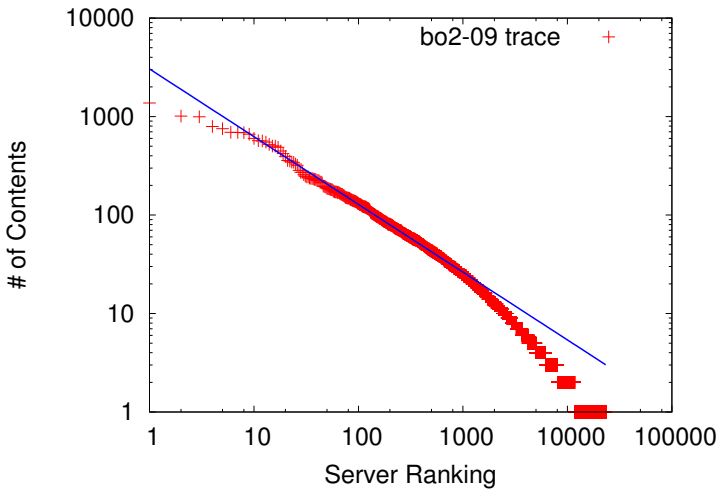


Figure 4.1: Number of contents per server vs. server ranking

Table 4.1: Values of α for each trace

bo2-09	bo2-10	ny-09	ny-10	pa-09	pa-10	rtp-09	rtp-10
0.688	0.664	0.631	0.633	0.648	0.668	0.770	0.770
sd-09	sd-10	sj-09	sj-10	sv-09	sv-10	uc-09	uc-10
0.807	0.755	0.779	0.783	0.719	0.756	0.764	0.765

in a large range in our evaluation.

The popularity of each content p_k also follows Zipf distribution with $\alpha = 0.75$. Although there are a large number of papers that debate on whether the popularity distribution of the current Internet traffic is Zipf-like or not [18, 49, 25, 54, 7], we believe using Zipf distribution is a reasonable starting point, especially considering that no one really knows what the traffic pattern would look like in future ICN networks. The popularity distribution will also be varied from uniform to highly skewed (Zipf with $\alpha = 1.25$) in our experiments.

Finally, the request rate of each node λ_i follows truncated geometric distribution with $p = 0.9$. The truncated geometric distribution is defined as:

$$p_N(i) = C \cdot p^i, \text{ for } i = 1, 2, \cdots, N \tag{4.9}$$

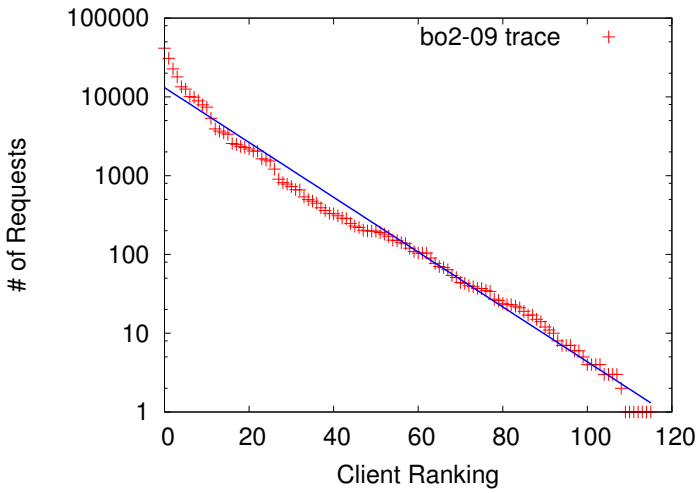


Figure 4.2: Number of requests per client vs. client ranking

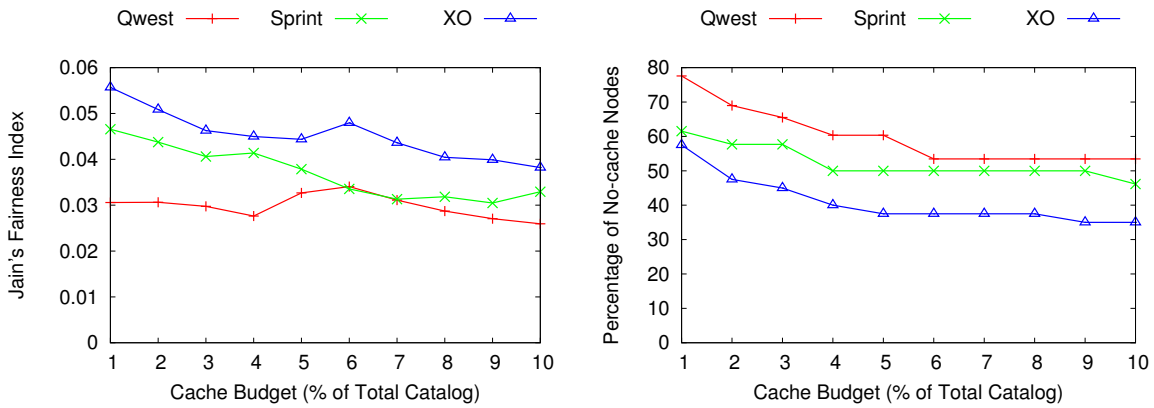
Table 4.2: Values of p for each trace

bo2-09	bo2-10	ny-09	ny-10	pa-09	pa-10	rtp-09	rtp-10
0.923	0.925	0.915	0.918	0.908	0.913	0.933	0.930
sd-09	sd-10	sj-09	sj-10	sv-09	sv-10	uc-09	uc-10
0.925	0.916	0.922	0.918	0.899	0.900	0.937	0.940

where N is the total number of client nodes, i is the rank and

$$C = (\sum_{i=1}^{N} p^i)^{-1} \tag{4.10}$$

Hence, smaller p means more skewed distribution while p close to 1 means more uniform distribution. This is also based on our analysis of the IRCache traces. Although the client IP addresses have been randomized to ensure anonymity, the randomization is consistent for each trace file. Hence, that does not affect our analysis of the request rate distribution. Figure 4.2 shows the request rate distribution from one trace file (bo2-09). Note that only the Y axis is in log scale and the curve fits a straight line well, indicating that the distribution decays exponentially. Similar trends are found in other trace files (not shown) and we use linear regression



(a) Jain's fairness index

(b) Percentage of no-cache nodes

Figure 4.3: The optimal cache deployment is highly uneven. A significant portion of the nodes are not allocated any cache space.

to fit the curves. Table 4.2 shows the values of p for each of the 16 trace files. Hence, we model the request rate as a truncated geometric distribution with $p = 0.9$ as the baseline and vary p from 0.5 to 1 in our experiments.

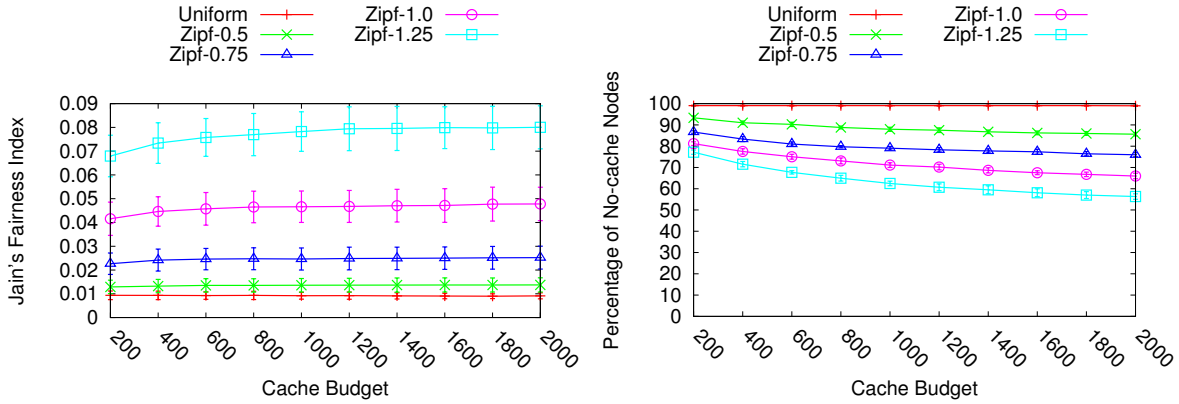
4.1.3 Numerical Results

We first run the greedy algorithm over the three realistic problem instances generated earlier with the cache budget varied between 1%∼10% of the total content catalog. This is in line with the fact that we can only cache a small portion of all the contents in the Internet. The numerical results after solving the formulated optimization problem are shown in Figure 4.3a and Figure 4.3b.

We use two different metrics to illustrate how the optimal cache deployment looks like. The first metric is the Jain's fairness index [58] over the cache sizes of each node in the network. Jain's fairness index is defined as:

$$f = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \cdot \sum_{i=1}^{N} x_i^2} \tag{4.11}$$

where N is the number of vertices and x_i is the cache size of vertex i. It is a normalized metric



(a) Jain's fairness index

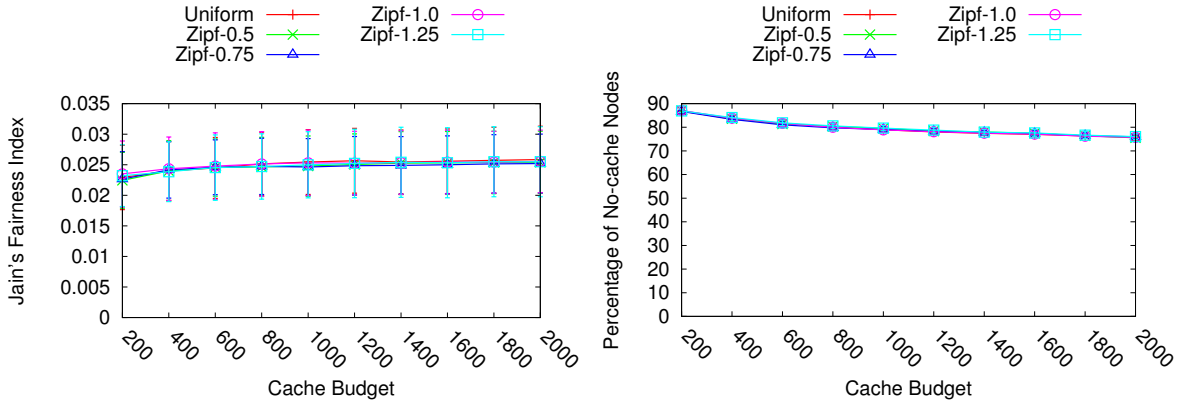
(b) Percentage of no-cache nodes

Figure 4.4: Numerical results under various content popularity distributions

between $1/N$ and 1. If the cache budget is evenly divided among all vertices (i.e., uniform pervasive caching), $f = 1$. Else if all cache budget is allocated to one vertex, $f = 1/N$. The lower this metric is, the more uneven the optimal cache deployment is, and vice versa. The second metric we use is the percentage of nodes that are not allocated any cache space in the optimal solution. This metric more directly reflects whether it is necessary to pervasively provision caches on every node.

The number of vertices N in the three ISP topologies are 58 (Qwest, AS 209), 52 (Sprint, AS 1239) and 40 (XO, AS 2828) respectively. As Figure 4.3a shows, the fairness indices in all three topologies are close to $1/N$, suggesting that the optimal cache deployment is highly uneven. This conclusion holds true even if we vary the cache budget size between 1% and 10% of the total content catalog. If we take a look at the percentage of nodes that are not allocated any cache space, even under the largest cache budget they still account for 53.45%, 46.15% and 35.00% for Qwest, Sprint and XO respectively. This again argues against pervasive cache deployment.

Let us now look at the numerical results out of the synthesized problem instances. As discussed earlier, we vary the content popularity distribution, the original server distribution and the request rate distribution in a considerable range to test the parameter sensitiveness of



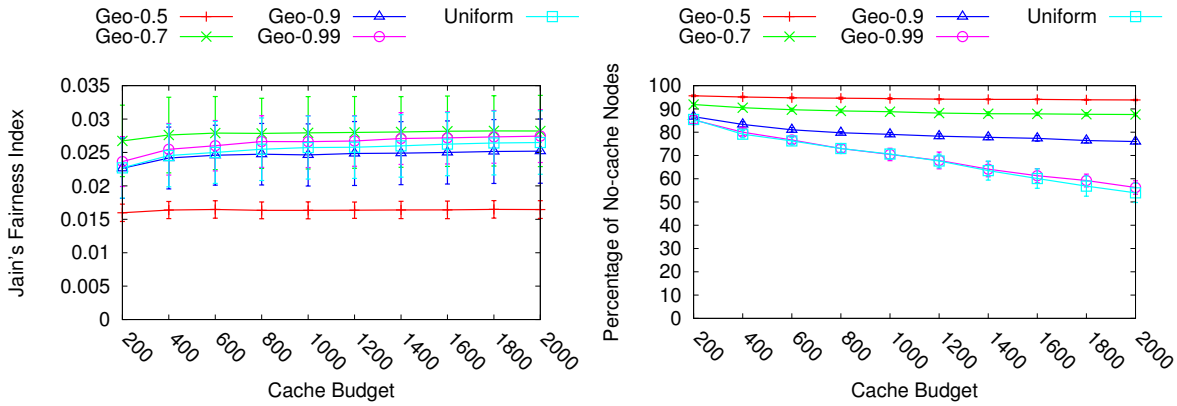
(a) Jain's fairness index

(b) Percentage of no-cache nodes

Figure 4.5: Numerical results under various original server distributions

our results.

Figure 4.4a and Figure 4.4b respectively depict the Jain's fairness index and the percentage of no-cache nodes under various cache budgets with varied content popularity distributions. As the figures show, the fairness index increases with the increase of the cache budget while the percentage of no-cache nodes diminishes with the increase of the cache budget. This is expected: if we have a very small budget, it is better to spend them only on the most important strategic points in the network. As we have a larger cache budget, we may want to provision caches on more nodes. However, the figures show that the slopes for both trends are very gentle. And even in the best case, the fairness index is still below 0.09 and there are still 56.32% of nodes that should not get any cache space in the optimal solution. Also note that more skewed content popularity distribution leads to higher fairness index and allocates cache space on more nodes. This is also intuitive: if the content popularity is highly skewed, those few nodes that serve the most popular contents become hot spots. It makes sense to provision caches more evenly across the network to counteract such imbalance and minimize the average latency for all users. We can also interpret the results from another angle: when the content popularity distribution is highly skewed, a small set of the most popular contents dominates the performance. In this case, it is best to cache this small set everywhere in the network and ignore the tail of the



(a) Jain's fairness index

(b) Percentage of no-cache nodes

Figure 4.6: Numerical results under various request rate distributions

distribution, yielding a higher fairness index.

The simulation results under various original server distributions are shown in Figure 4.5a and Figure 4.5b. Similar to the previous case, the absolute value of the fairness index is always very small across all cache budgets or original server distributions. And even under the largest cache budget, only around 25% of the nodes should be provisioned with cache space. This result argues against pervasive caching: to minimize user-perceived latency, it is better to place caches at some strategic points rather than provisioning them pervasively on all nodes. The original server distribution seems to have little impact on the optimal cache deployment solution.

Finally, Figure 4.6a and Figure 4.6b show the results under various request rate distributions. When the request rate is highly skewed (e.g., Geo-0.5), the fairness index of optimal cache sizing is very low and no matter how much cache budget we have, they are all spent on a very small fraction of the nodes in the network. That is because the severe imbalance in request rates requires that we provide almost all caches to those a few heavy users and ignore the clients with lesser demand. However, when the request rate distribution is uniform, the cache budget will be allocated to more nodes as it increases.

In summary, across all scenarios we have tested, the Jain's fairness index of the optimal solution to the cache deployment problem is always very small. And the percentage of nodes

that are not allocated any cache space is constantly above 50% (with only very few exceptions). Hence, we conclude that, pervasive in-network caching is far from the optimal cache deployment strategy and should not be automatically assumed in ICN networks.

4.2 Simulation

Although the theoretic analysis gives us a good understanding of how caches should be deployed to achieve optimal performance, it omits a few practical factors. For instance, we may not be able to prefetch all the contents according to the optimal solution and can only cache contents along the paths they traverse. A dynamic and distributed cache replacement strategy is typically applied to determine which contents stay in the cache. Further, in our modeling we assume perfect knowledge by the clients so that they can always locate the nearest copy of each content. This may be infeasible in practice since 1) cached contents are being replaced from time to time and 2) propagating such state information is costly. Finally, for a real network, we may not have a priori knowledge of such information as content popularity and request rates to calculate the optimal solution. Hence, we complement our analysis with simulation to address the practical factors in a network caching system.

4.2.1 Methodology

We use the same network topologies and traffic patterns in our simulations as the synthesized problem instances discussed earlier. However, in the simulation we generate actual traffic according to a Poisson process (at an average rate of 100 requests per second) and simulate how the system would evolve over time under dynamic content replacement algorithms and different routing schemes.

There are many proposals for an efficient content replacement strategy for ICN. Least Recently Used (LRU) was mentioned in the original CCN/NDN proposal [57, 117]. However, as pointed out by many studies, LRU is not the best choice under pervasive caching since it may result in redundant caching of the same content along the data path. Numerous alternatives

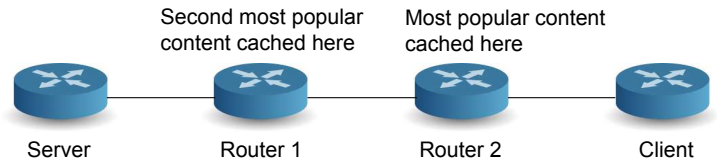


Figure 4.7: Illustrating the implicit cooperation induced by perfect LFU strategy that improves cache diversity.

have been proposed to rectify this behavior [73, 30, 26, 106, 68, 85]. However, we cannot simulate all of them. Instead, we use *perfect* Least Frequently Used (LFU) algorithm as a representative of them.

Perfect LFU differs from in-cache LFU in that the latter only tracks the access frequency of contents currently in the cache while the former keeps track of all past contents even if they are not in the cache any more. Perfect LFU may be a costly algorithm in practice but it is a good approximation of the alternative content replacement proposals mentioned above. That is because it achieves the following two desirable properties which all these proposals try to achieve: 1) avoid duplicate caching of the same content everywhere and increase cache diversity; 2) the more popular the content is, the closer it should be cached to the clients. Figure 4.7 illustrates this point by a simplified example: the client retrieves contents from the server. Assume that router 1 and router 2 each can cache only one content and they independently employ perfect LFU replacement. Obviously, the most popular content will be cached at router 2 since it sees most frequent requests for that content from the client. In contrast, router 1 will not see the requests for the most popular content since they are already satisfied by the cached copy at router 2 and will not hit router 1. Hence, router 1 will cache the second most popular content. A real system has more noise than is described here due to a number of reasons (e.g., multiple downstream nodes, flatness of the popularity curve, popularity learning time, short-term and long-term changes in content access patterns etc.). But we believe that most of the proposed improvements for large cache networks exist within this noisy space and perfect LFU is adequate to qualitatively gauge their performance (e.g., in [106], the proposed LB scheme outperforms LRU significantly but is marginally better than LFU). An even better content replacement

strategy may exist but it would most likely incur costly coordination among the routers (e.g. [68]) and is unlikely to be adopted in pervasive caching settings.

Also note that there are two different methods to present a content replacement algorithm. Some papers (e.g., [68, 26, 85]) divide the task into two steps. The first step is to determine whether a newly arrived content should be inserted into the cache. If so, the second step is to find a victim to be discarded. Otherwise no operation is needed. Other papers (including this one) always insert the new content into the cache and use a single algorithm to determine which content should be discarded. In this case, since it is perfectly possible to discard the content that has just been inserted (unless LRU is being used), it is equivalent to not caching this content in the first place. Hence, these two presentations are convertible but should not be confused.

With regard to routing, we mainly compare two schemes: shortest path (SP) routing and nearest replica (NR) routing. In SP, a request is forwarded along the shortest path towards the original server of the requested content. Only caches along this path can serve the request. This is clearly sub-optimal since off-path caches are not being utilized. However, given the incredible scalability challenges in ICN, we believe this is the most feasible one in the near future and will use it as the default routing scheme. Nearest replica routing is an idealistic routing scheme where a request is always forwarded towards the nearest copy the requested content. In our simulation of NR, we assume that every router has instantaneous knowledge of which node has what contents and makes the forwarding decision accordingly. This is highly impractical but serves well to gauge the upper bound of the cache performance. Content data, i.e. the responses, follow the same path as the corresponding requests so that no extra routing is needed for them. This mimics the behavior of CCN/NDN [57, 117] where Pending Interest Table (PIT) leaves a trail of "bread crumbs" for the returning content data.

Using the above-mentioned content replacement and routing schemes, we compare the performance of edge caching and pervasive caching. Edge caching evenly divides the cache budget (rounding to integers) among what we call "gateway" nodes in the transit-stub topology. They

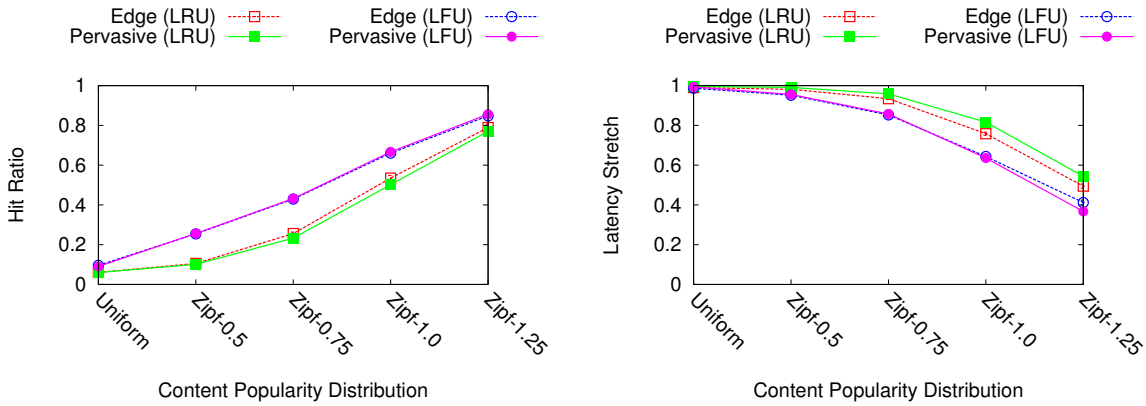


Figure 4.8: Edge caching vs. pervasive caching under various content popularity distributions

are the stub nodes that connect a stub domain to the transit domain. This is to simulate traditional Web caching where caches are deployed at access links. On the other hand, pervasive caching evenly divides the cache budget (rounding to integers) among all nodes in the network. The cache budget is fixed to 10% of the total content catalog.

All caches are empty when the simulation starts. This empty initial state does not affect the cache performance since the system we simulate is ergodic [90]. Metric measurements are not started until the cache space is filled. We periodically (every 1000 seconds) calculate the performance metrics and terminate the simulation only if the metrics stabilize (within 5% error range from the previous round). The performance metrics we use in our simulations are hit ratio and latency stretch (the ratio between the actual fetch latency and the latency to fetch from the original server).

4.2.2 Results

Let us first compare the performance of edge caching and pervasive caching under shortest path routing. As before, we vary the content popularity distribution, the original server distribution and the request rate distribution in a considerable range to test the parameter sensitiveness of the results. Figure 4.8 – Figure 4.10 show the simulation results under different parameter

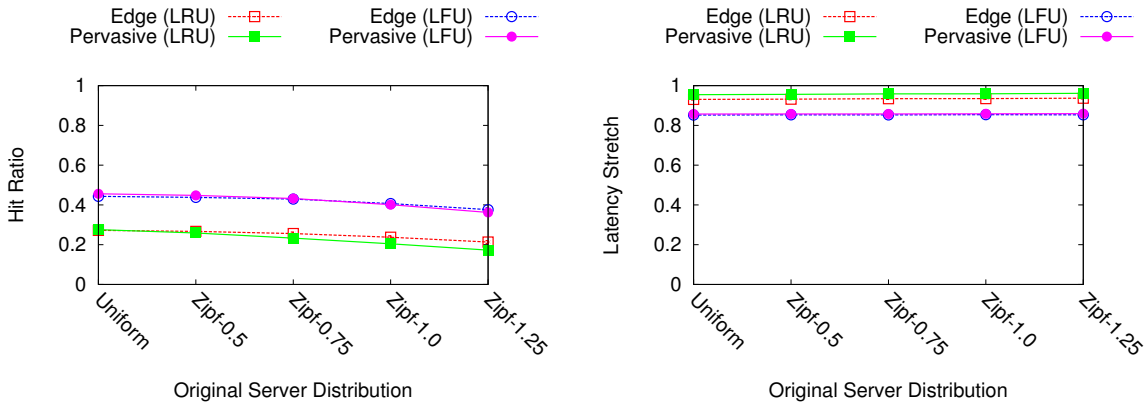


Figure 4.9: Edge caching vs. pervasive caching under various original server distributions

configurations respectively.

Figure 4.8 shows the performance of edge caching vs. pervasive caching under various content popularity distributions. We can observe the following phenomena from this figure: 1) no matter how the content popularity distribution changes (from uniform popularity to highly skewed popularity), the performance difference between edge caching and pervasive caching is very small (at most 6.5%) under the same content replacement policy. In fact, edge caching may perform even better than pervasive caching, especially under LRU replacement. 2) Perfect LFU replacement performs much better than LRU replacement, especially for pervasive caching. This is primarily due to the cache redundancy issue we mentioned earlier. When LRU replacement policy is used in conjunction with pervasive cache deployment, the same content may be cached at multiple nodes along the path. This is a waste of the cache space and degrades the performance of the system. Under perfect LFU replacement, such cache redundancies are largely eliminated and we observe a much higher hit ratio and lower latency stretch. 3) As the content popularity distribution becomes more skewed, the overall cache performance improves. This is easy to understand: more skewed popularity means we only need to cache a few most popularity contents to achieve very high hit ratio.

Figure 4.9 and Figure 4.10 show the simulation results under various original server or

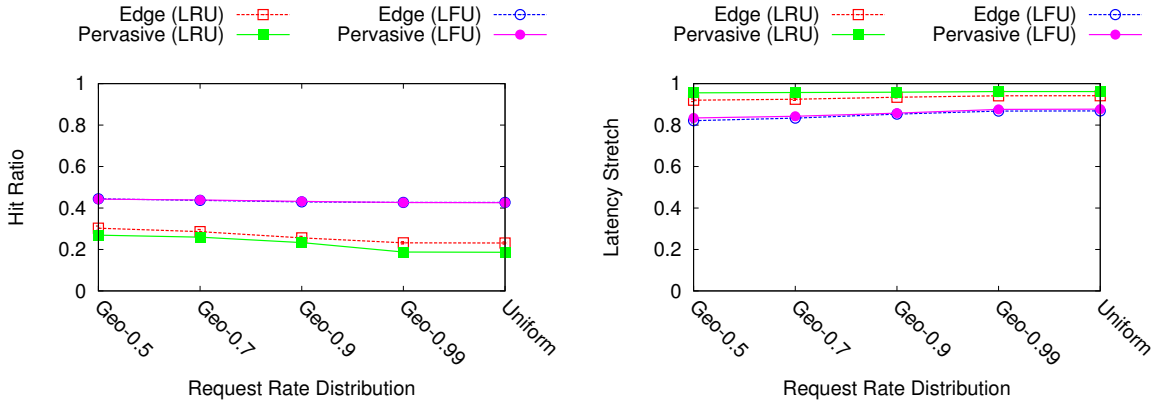


Figure 4.10: Edge caching vs. pervasive caching under various request rate distributions

request rate distribution. Again, we observe minimum performance gap between edge caching and pervasive caching across all cases. Moreover, it is typically edge caching that holds the slight advantage. Another observation is that, sometimes higher hit ratio does not necessarily lead to lower latency stretch. That is because the latency stretch depends not only on *whether* cache hit happens but also *where* it happens. In the case of pervasive caching, a cache hit in the middle of the network bring much less latency reduction than a cache hit close to the client.

In summary, across a wide range of parameter space, the performance of edge caching is almost as good as pervasive caching (if not better). Given the deployment cost, pervasive caching does not seem to make much sense. Another thing to note is that LRU is a very bad content replacement policy for pervasive caching as it creates lots of cache redundancies. We can improve the performance of pervasive caching by adopting a more intelligent replacement policy. This is also pointed out by many existing studies [73, 30, 26, 106, 85]. However, this does not demonstrate the necessity of pervasive cache deployment. As the simulation results show, we may get equally good performance with edge caching. Hence, future ICN proposals should not automatically assume pervasive cache deployment and propose new replacement strategies to beat LRU under that setting. Instead, we need to re-evaluate how we should provision cache space in ICN networks.

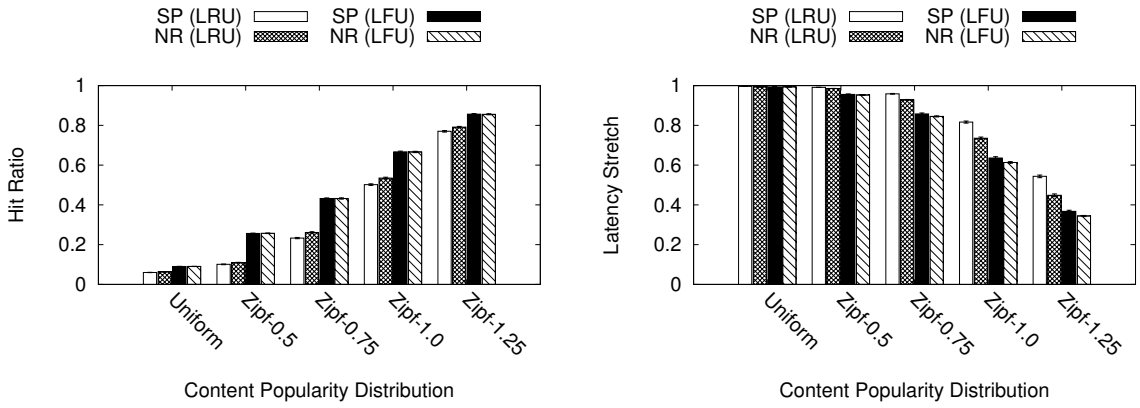


Figure 4.11: Shortest path routing vs. nearest replica routing under various content popularity distributions

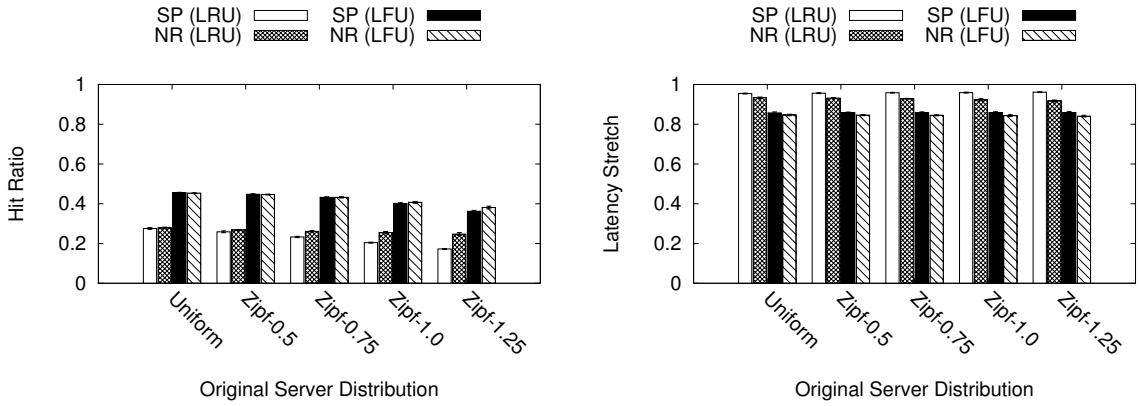


Figure 4.12: Shortest path routing vs. nearest replica routing under various original server distributions

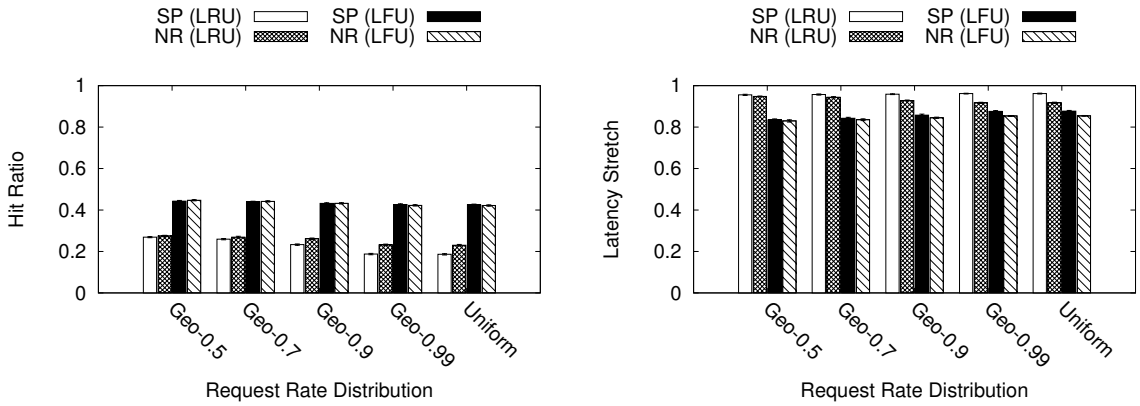


Figure 4.13: Shortest path routing vs. nearest replica routing under various request rate distributions

Someone may argue that shortest path routing does not fully unleash the potential of pervasive caching. Hence, we run a set of simulations to see how much performance gain can be added to pervasive caching by adopting nearest replica routing. Figure 4.11 – Figure 4.13 shows the simulation results. Observe that: 1) Nearest replica routing does improve the performance of pervasive caching. 2) The improvement is more prominent under LRU replacement than LFU replacement. This is also due to the cache redundancy issue. Since LRU may result in redundant caching of the same content along the path, checking off-path caches via nearest replica routing is very helpful. Under perfect LFU replacement, the effect of nearest replica routing is much weaker. 3) Even for LRU replacement, the maximum reduction in latency stretch brought by nearest replica routing is only 17.5%. Hence, we conclude that, even if nearest replica routing is feasible, it would bring limited performance improvement, especially under an intelligent content replacement policy similar to perfect LFU.

4.3 Discussion

With all the results shown in the previous sections, we would like to rethink about the relationship between ICN and pervasive caching. We want to point out that ICN simplifies the

cache operation but does not change the cache deployment problem itself. Given a network topology and a traffic matrix, how we should deploy caches to optimize the performance has nothing to do with whether this is a TCP/IP network or an ICN network. The difference lies only in the implementation: in the current Internet, we need deep packet inspection to identify the content being requested and we also need some packet header manipulation to maintain end-to-end TCP semantics. In contrast, ICN naturally supports caching in its primitive and makes pervasive caching much less costly.

Hence, by adopting a location-independent network primitive, ICN gives extra degrees of freedom to the network to optimize large-scale content distribution. It makes caching easy, to the extent that pervasive caching may be *possible*. However, how we actually want to utilize the provided degrees of freedom is an open question. ICN does not imply that pervasive caching is *necessary*. As demonstrated by our previous analysis and simulation, pervasive cache deployment is far from the optimal strategy and future ICN proposals should not automatically assume its presence. On the other hand, ICN is not just about pervasive in-network caching. Even if we stick to the existing caching architecture (e.g., CDN), ICN could still simplify its implementation. Moreover, ICN could also benefit in many other aspects such as content migration, multihoming and security.

4.4 Concluding Remarks

Pervasive in-network caching is a feature advocated by almost all ICN architectures. Many existing studies consider them as bundled and propose ICN caching or routing schemes under the implicit assumption of pervasive caching. In this chapter, we re-examine the assumption of pervasive in-network cache deployment and demonstrate that it may not be necessary. We back up our conclusion with both theoretic analysis and large-scale simulation under both realistic and synthesized workloads. Although we did not propose any new ICN caching or routing scheme here, we believe it is equally, if not more, important to understand the upper limit achievable by pervasive caching and nearest replica routing. We believe our findings rectify

some important misconceptions in ICN and are helpful to the design of future ICN caching or routing schemes.

Note that caching may serve other purposes in ICN besides server load relief and latency reduction. For instance, it may be considered as a reliability mechanism for adverse network environments like Disruption-Tolerant Networks (DTN). It may also improve user mobility support since a content lost during handover may be cached nearby so that the roaming user can get it quickly upon a second attempt. We leave the evaluation of these aspects of ICN caching as future work.

Chapter 5

Scalable and Efficient ICN Routing via Bloom Filters

Designing a scalable and efficient ICN routing scheme is very challenging due to a number of reasons. First, the number of contents in the Internet is exceptionally large. Back in 2008, Google reported hitting the milestone of one trillion (10^{12}) unique URLs on the Web [6]. And this number will be ever increasing in future, creating serious scalability concerns for ICN. Second, hierarchical aggregation, the key technique that makes IP scalable, is less applicable in ICN networks since it is effective only if the content names (or prefixes) have some degree of dependence on their locations, which is opposite to the design philosophy of ICN. Finally, content movement (due to caching, multihoming, migration, etc.) may happen at smaller time scales than IP routing updates, making the speed of convergence and the associated signaling overhead a major issue for ICN routing.

Existing proposals for ICN routing roughly fall into two categories. The first category, represented by MDHT [33] and DONA [64], resolves names into locators via a global Name Resolution Service (NRS) and uses the resolved locator for routing and forwarding. By separating name resolution from routing, NRS-based schemes could utilize existing Internet routing protocols and only handle the name resolution part whose scalability issue is easier to address with the

help of Distributed Hash Tables (DHT) and large-scale data centers. The drawback, however, is the resolution latency that is hard to be amortized for nearby contents. The other category, represented by OSPFN [105] and NLSR [53] (hereafter referred to as "CCN routing"), adopts a more radical approach where packets are directly routed by content names. It removes the resolution latency and the cost to deploy a global NRS infrastructure. However, it may still suffer from sub-optimal performance since it relies on hierarchical aggregation to scale. If a copy of a content is not aggregatable, it may be overlooked by the routing protocol. This is especially problematic for cached or replicated (multihomed) contents whose locations are not topologically regular.

To address the above-mentioned issues on both sides, in this chapter we propose Bloom filter-based routing for ICN networks. In our scheme, each node uses Bloom filters to compactly represent the contents it possesses and advertises them in a limited scope (e.g., within three hops). Upon receiving such Bloom filter advertisements, a router could locate nearby contents without name resolution latency and is not subject to any aggregatablity constraints. This could bring substantial performance improvement (up to 91% reduction in routing stretch according to our simulation results in Section 5.3) since NRS and CCN routing have most problems when fetching off-path nearby contents. Moreover, due to the space efficiency of Bloom filters, limited-scope advertisement incurs minimum signaling overhead, making Bloom filter-based routing a cost-effective performance enhancer.

We also explore the possibility to extend Bloom filter-based ICN routing to Internet scale. Compared to NRS or CCN routing, Bloom filter-based routing has a number of advantages such as a much smaller number of entries in the routing table and a much simpler lookup operation that is easy to be parallelized and accelerated by hardware. However, the memory requirements and the Bloom filter update overhead need to be carefully taken care of. We propose several mitigation techniques to these scalability challenges.

5.1 Motivation

The most important advantage of ICN over the current Internet is that it inherently supports multihoming or migration of data (either topologically or logically). Data multihoming or migration is difficult in the current Internet because contents are named relative to their locations. The two global namespaces, IP addresses and DNS names, identify locations in a network topology or a domain hierarchy. When contents move to a new physical location or a new administrative domain, their names change, which is the source of many problems in the current Internet such as redundant traffic and broken links. To mitigate such problems, all sorts of add-ons to the existing Internet architecture have been proposed, including Web caching, P2P systems, CDNs, dynamic DNS and HTTP redirection. These patchworks are ad-hoc and application-specific while ICN solves the problem at architectural level.

To reap this benefit of ICN, a scalable routing algorithm that incurs low stretch is essential. Existing approaches in the literature can be roughly classified into two categories:

The first approach is to build a global name resolution service (NRS) that translates content names into routable locators (e.g., IP addresses) and use existing routing infrastructure to forward packets. The NRS can be constructed in different manners. DONA [64] and CURLING [27] advocate a hierarchical domain-level infrastructure mirroring the ISP relationships in the current Internet. Name resolution follows the provider/customer/peer relationships all the way up to Tier-1 ISPs if it cannot be resolved in the lower levels. The problem with such systems is the scalability issue at the top level where all names need to be kept track of. A more scalable NRS could be built via DHT. Traditional DHTs have very high stretch ($O(\log N)$ overlay hops where N is the number of nodes that constitute the DHT) and are unable to satisfy the requirements of ICN. MDHT [33] mitigates this problem by having multi-level DHTs so that locally registered names may be resolved locally. DMap [103] instead leverages existing BGP reachability information to build a $O(1)$ DHT with minimum overhead. Both approaches claim average resolution latencies in the order of 100ms, which may be acceptable for some services but non-negligible for others or for contents with nearby copies.

The second approach gets rid of name resolution and routes directly on names. This radical approach has incomparable benefits but also brings serious scalability concerns. For instance, OSPFN [105] extends existing link state routing protocols like OSPF [75] to advertise name prefixes in addition to adjacencies. ICN routers utilize such prefix advertisements to construct a name-based FIB and use it to forward packets. Since hierarchical names are used in CCN, CCN routing scales by aggregation in similar ways to IP. However, hierarchical aggregation works only if content names have certain degree of dependence on locations. When contents are multihomed in many different places across the Internet, the achievable degree of aggregation decreases. CCN routing has to ignore some unaggregatable copies of the contents in order to stay feasible. Another aspect of complexity in name aggregation is that the name space of CCN is infinite. There may be infinite name suffixes under a given prefix as compared to finite and known subnet address ranges in IP. This makes advertisement aggregation by intermediate routers extremely difficult since they never know whether they have all the suffixes under a prefix and can hence aggregate them into a single entry. Only the publisher knows such information and is the sole entity that can aggregate the routing advertisements. In fact, even if we only keep track of the contents on the publishers' servers and each publisher can aggregate its contents into a single entry in the advertisement, CCN routing still needs to handle around 10^8 entries (the number of second-level domains registered in DNS today) which is beyond the ability of today's high-end routers [76]. Further, by overlooking the unaggregatable caches and replicas, CCN routing may incur high stretch, especially for contents that are cached or replicated nearby.

Note that the gap between name resolution and name-based routing is not as huge as it appears. Name-based routing also implicitly resolves a name into a location. It just happens that this resolution path is the same as the routing path. Hence, the routing stretch is minimized. Name resolution can be also viewed as name-based routing. It just takes a detour to certain name resolution server first and then arrives at the actual content location. Therefore, there is only a trade-off between scalability and stretch rather than a wall that segregates these two approaches.

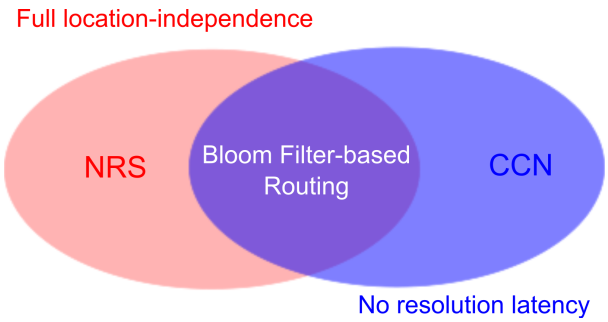


Figure 5.1: Bloom filter-based routing combines the benefits of NRS and CCN routing.

In light of the limitations of the existing two approaches, we crave an ICN routing scheme that requires no name resolution but still keeps track of all contents despite their locations (Figure 5.1). Our approach resembles CCN routing but does not rely on hierarchical aggregation for scalability. Instead, we make use of the space-efficiency and simple semantics of Bloom filters to be scalable. Our proposal can be applied in a limited scope or to the global Internet. Section 5.2 and 5.3 discusses the former case where limited-scope Bloom filter advertisement greatly reduces the stretch of nearby contents and improves the performance of NRS and CCN routing at minimum cost. Section 5.4 investigates the latter case and evaluates the benefits and cost of Bloom filter-based ICN routing for the entire Internet.

5.2 Bloom Filter-based Routing

5.2.1 Bloom Filter Primer

Bloom filter [14] is a space-efficient data structure that represents a set of elements from a universe U and supports $O(1)$ membership lookup. Compared to a list of unique identifiers which requires at least $n \log_2 |U|$ bits to represent n elements, a Bloom filter uses only $n\frac{|\ln p|}{(\ln 2)^2}$ bits where p is the target false positive probability. When the universe is large (e.g., all the contents over the Internet), Bloom filter is a much more compact way to represent the contents a router/server has (requiring 8–16 bits per content for a reasonably small false positive probability) than a full list of content names (requiring hundreds of bits per content).

Bloom filter achieves the above-mentioned properties as follows: a Bloom filter that represents a set $S = \{x_1, x_2, \cdots, x_n\}$ of n elements is a bit array of m bits associated with k independent hash functions $h_1(\cdot), h_2(\cdot), \cdots, h_k(\cdot)$ whose hash values are within the range of $\{1, 2, \cdots, m\}$. Initially, the bit array is set to all zero. As we insert each element x_i into the Bloom filter, we calculate its hash values out of the k hash functions $h_1(x_i), h_2(x_i), \cdots, h_k(x_i)$ and set the corresponding positions in the bit array to 1. Obviously, there may be hash collisions and a bit may be set multiple times but as long as it has been set to 1, we do not care how many times it has been set.

After the Bloom filter is constructed, we can check if an item y is a member of set S in constant time. It works as follows: we calculate the k hash values $h_1(y), h_2(y), \cdots, h_k(y)$ of y and verify whether the corresponding bits in the array are set. If any of $h_j(y), j = 1, 2, \cdots, k$ is not set, we are certain that $y \notin S$. However, if all $h_j(y), j = 1, 2, \cdots, k$ are set, we assume that $y \in S$ but we know we could be wrong. That is because the $h_j(y)$ bits could be set due to a combining effect of multiple elements that are not y but each sets a subset of $h_j(y), j = 1, 2, \cdots, k$. This is called the false positive probability p of a Bloom filter and it can be derived as:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k \tag{5.1}$$

Given the number of elements we intend to represent n and the Bloom filter size m, we can derive the optimal number of hash functions to minimize the false positive probability above:

$$k = \frac{m}{n} \cdot \ln 2 \tag{5.2}$$

And the false positive probability under this optimal k is:

$$p = \left(\frac{1}{2}\right)^{\frac{m}{n} \cdot \ln 2} \tag{5.3}$$

Hence, we could determine the parameters of a Bloom filter in the following order: first, we

need to estimate or predict n, i.e., the number of elements we intend to insert into the Bloom filter. Second, based on n and the false positive probability p that we can tolerate, we determine the size m of the Bloom filter based on Equation 5.3. Finally, we figure out the number of hash functions we need via Equation 5.2.

For the detailed derivation of the above-mentioned equations, please refer to [19, 11].

5.2.2 Routing Advertisement

Let us now look at how we utilize Bloom filters to help ICN routing. In our scheme, each content serving entity (be it the original server, a mirror site or an intermediate cache) generates Bloom filters for the contents it possesses and advertises them in similar ways to link state routing protocols like OSPF [75].

Note that we do not flood the advertisements over the entire network. Instead, we limit the advertisement scope to a few hops to minimize the signaling overhead. With this limited-scope advertisement, each router or host is aware of all the available contents a few hops away from itself and could utilize them without any name resolution. In contrast, neither NRS nor CCN routing could handle nearby contents very efficiently. NRS imposes a resolution latency which is a huge overhead for locally replicated or cached contents. On the other hand, CCN may not even know such off-path contents due to aggregatability constraints. Note that determining the optimal advertisement scope is a challenging optimization problem requiring detailed knowledge of the network topology and the traffic matrix. Solving this optimization for a general network is beyond our scope. However, we do study the impact of the advertisement scope on routing performance in our simulations (Section 5.3).

One important design parameter we need to choose carefully is the size of the Bloom filter since it determines how many contents can be accommodated without incurring unacceptable false positive probability. Note that the router or server should not advertise a Bloom filter that just fits the contents it possesses. That is because the advertisements will be aggregated later on to construct the routing table (discussed in the next subsection) and the Bloom filter should

ensure low false positive probability even after the aggregation. Unfortunately, the number of contents that will be aggregated is unknown to the routers or servers when they advertise the Bloom filter. But to ensure interoperability the Bloom filter size needs to be determined in advance and is global to the whole network.

In our scheme, we decide to set the Bloom filter to a relatively large size due to a number of reasons. First, it ensures a low false positive probability after aggregation. Second, since the Bloom filter size targets at the aggregated number of contents, it should be quite sparse when it only holds the contents of its advertiser. This sparse Bloom filter can be effectively compressed using standard coding techniques. In fact, it has been proved that a larger and hence sparser Bloom filter could be smaller after compression than a Bloom filter that just fits [74]. Third, if the Bloom filter is still too large and sparse after aggregation, we can shrink its size by "halving". Suppose the size of a Bloom filter is a power of two, we can halve the size of the Bloom filter by taking the union of its first half and the second half. This can be done recursively until the fill ratio (the number of 1's compared to the size of the Bloom filter), or the corresponding false positive probability of the Bloom filter reaches a certain threshold. Lookup can still be carried out at $O(1)$ speed on the shrunk Bloom filter: we simply need to take some modulo operations on the hash values.

Finally, in case the predefined size is too small to accommodate all the contents, there are two choices: bear with the false positive or bear with the false negative. In the former case, we simply insert contents or aggregate Bloom filters even if it leads to high false positive probability. This false positive will be taken into consideration when the routers make forwarding decisions (Section 5.2.4). The latter approach is also simple: if the fill ratio of a Bloom filter reaches certain threshold, we no long insert items or aggregate other Bloom filters into it. This may lead to "false negative" since existent contents are not reflected by the Bloom filter. But since our scheme mainly works as a performance enhancer with fallback mechanisms, this does not affect the ultimate routing correctness.

5.2.3 Routing Table Construction

An ICN router may receive plenty of advertisements from nearby routers or servers. To construct the routing table, Bloom filters that are equivalent in forwarding can be aggregated by taking their union. Specifically, if the router would 1) use the same interface, 2) forward to the same next-hop and 3) go across the same distance to fetch the contents advertised by two Bloom filters, they are aggregated. Note that "distance" might be an ambiguous metric here. Ideally, it could be the latency to fetch a content. However, to control the size of the routing table and simplify the operations, we use hop count as a rough estimate of the distance. That way, the constructed routing table would be a list of Bloom filters, each representing the contents that are reachable x hops away from an interface/next-hop, as shown in Figure 5.2.

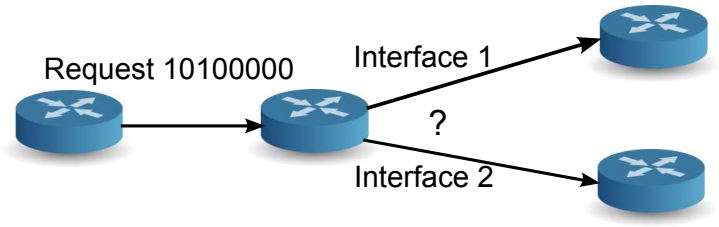
Note that the size of this routing table is upper bound by the number of active interfaces or next-hops a router has, multiplied by the advertisement scope (in hop count). This is a much smaller number than that of a routing table indexed by content names or prefixes. If we need an even smaller routing table, we can ignore the distance information and simply keep one Bloom filter for each interface/next-hop.

5.2.4 Forwarding

Once the routing table is constructed, request forwarding in the data plane may work as follows: the router walks through the list of aggregated Bloom filters in the routing table and calculates the Expected Fetch Time (EFT) for each interface/next-hop j (except the one this request comes from) according to the following equation:

$$\begin{aligned} EFT_j = {} & (1 - FP_{1j}) \cdot D_{1j} + FP_{1j} \cdot (1 - FP_{2j}) \cdot D_{2j} + \cdots \\ & + (\prod_{i=1}^{N-1} FP_{ij}) \cdot (1 - FP_{Nj}) \cdot D_{Nj} \\ & + (\prod_{i=1}^{N} FP_{ij}) \cdot (D_{Nj} + D) \end{aligned} \tag{5.4}$$

where FP_{ij} is the false positive probability of the ith nearest Bloom filter on interface j which matches the request and D_{ij} is the corresponding distance. D is the distance to fetch the requested content via the fallback routing algorithm (e.g., OSPFN). We only need to consider the N matched Bloom filters that are closer than D, i.e., $D_{ij} < D, i = 1, 2, \cdots, N$. The last item in Equation 5.4 needs some explanation. It represents the worst case if all Bloom filters turn out to be false positive: the request can be sent back to this router (D_{Nj}) and then forwarded via the fallback routing mechanism (D). EFT_j is set to infinity in case $N = 0$, i.e., no match is found in any of the Bloom filters that are closer than D on interface j.



Interface	Distance	Bloom Filter	False Positive
1	1	11001001	0.2500
	2	10100000	0.0625
	3	11001000	0.1406
2	1	00100000	0.0156
	2	10000001	0.0625
	3	10000100	0.0625
	4	11110000	0.2500

Figure 5.2: An example to illustrate our forwarding algorithm

With the expected fetch time of each interface/next-hop in hand, the router picks the one with minimum EFT and compare it with D. If minimum EFT $< D$, the request is forwarded to the corresponding interface/next-hop. Otherwise, we simply fallback to NRS or CCN routing. Figure 5.2 gives an example to illustrate this forwarding algorithm: the router in the middle

receives a request and needs to decide which interface to forward to. Assume the Bloom filter is 8-bit long and has 2 hash functions. The requested content name hashes into position 1 and 3 (10100000) and the distance to fetch this content from the original server is 7. Given the FIB shown in the figure, $EFT_1 = (1 - 0.0625) \times 2 + 0.0625 \times (2 + 7) = 2.4375$ while $EFT_2 = (1-0.25) \times 4 + 0.25 \times (4+7) = 5.75$. Hence, the request will be forwarded via Interface 1.

Note that this is just one of many possible ways to do forwarding based on the routing table constructed earlier. Alternatives include: 1) picking the matched Bloom filter with minimum distance, 2) picking the matched Bloom filter with minimum false positive probability (in case distance information is ignored and there is only one Bloom filter for each interface/next-hop) or even 3) random choice among all matched Bloom filters (this actually has some unique benefits which we discuss in Section 5.4). A specific forwarding algorithm may be chosen based on the computation and storage capability of the router. For instance, calculating EFT requires extra computation while keeping one Bloom filter per interface/next-hop may save memory space.

Forwarding of the response (content data) is simpler. We can either follow the trail of "bread crumbs" left by the request (as in [57, 117]) or use standard IP routing since the source and destination addresses are already known (as in [64]).

5.2.5 Routing Updates

Bloom filter in our scheme works similarly to Link State Advertisement (LSA) in OSPF. They need to be refreshed periodically and each ICN router needs to maintain a database to store and update these advertisements. Advertisement update needs to be considered on both the sender side and the receiver side. On the sender side, the Bloom filter needs to be updated when the contents stored by the router or server change. The original Bloom filter proposed in [14] are not deletable. If some contents are no longer available, the router/server needs to generate the Bloom filter from scratch, inserting each item one by one. An alternative is to use counting Bloom filter [36] which is larger than the original Bloom filter but supports deletion of items.

We keep a counting Bloom filter locally for fast Bloom filter update but advertise (compressed) original Bloom filters to reduce control traffic.

On the receiver side, namely the router who receives advertisements from other routers or servers and aggregates them, the router needs to store each individual advertisement (rather than just the aggregated ones) so that updating the routing table upon a refreshed advertisement is possible. That is because, once the Bloom filters are aggregated, the reverse operation is not possible. If one of them gets an updated version, it is hard to apply it to the already aggregated Bloom filter. An alternative is to advertise deltas instead of a new Bloom filter. However, this poses threats to robustness since the advertisement would be useless if the previous state is lost (advertisement packet loss, router crash, etc.). Therefore, we still opt to manage the advertisements similar to how OSPF manages its link state database. If the size of this database is a concern, we can store the individual advertisements on disks and only keep the aggregated ones in memory. We assume that the update of routing states happens at a larger timescale than packet forwarding. This will be further discussed in Section 5.4.

5.3 Performance Evaluation

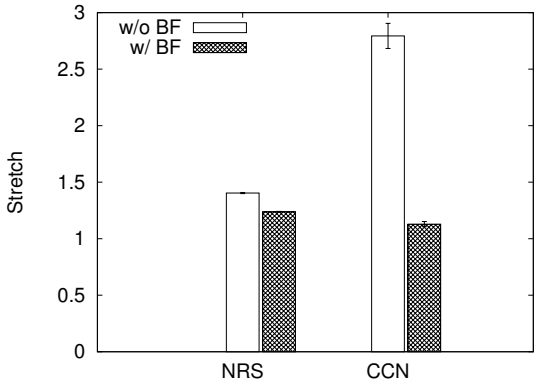
5.3.1 Simulation Environment

We have developed a custom simulator in Python to compare the performance of different ICN routing schemes under various conditions. To have a realistic input of network topology to our simulator, GT-ITM [116] has been used. We generate a 2048-node transit-stub network that has 4 transit domains with 8 transit nodes each. Attached to each transit node are 7 stub domains with 9 nodes in each domain. The transit domains are fully connected while nodes within each transit domain and stub domain are connected with probability 0.6 and 0.3 respectively. The scales for the top level, the transit domain and the stub domain are set to 20, 20 and 10 respectively. The resulting edge length is used as the link latency (in milliseconds). Bandwidths of all links are assumed to be infinite. Hence, the latency to fetch a content is the sum of the

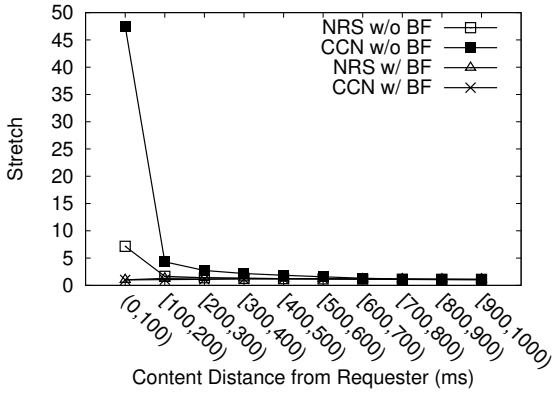
latencies of the links traversed by the request and response. We generate 10 different topologies and repeat all tests over these 10 topologies to calculate 95% confidence intervals.

10,000 contents of Zipfian popularity exist in this network. The original server of each content is picked uniformly among the 2016 stub nodes of the network (We assume that servers do not reside on transit nodes which represent backbone routers). The top 10% most popular contents are replicated in other nodes and we vary the number of replicas (default: 10) in our simulation to study its impact on performance. Requests arrive according to a Poisson process. The source of the request is uniformly drawn from the stub nodes and the destination (i.e., the requested content) is picked based on standard Zipf distribution. Similar to previous works (e.g., [18]), we adopt the independent reference model where consecutive requests are independent of each other but only dependent on the popularity distribution. We simulate both no caching and with caching cases (default: no caching). When caching is considered, we assume that all nodes (including transit nodes) have the same cache capacity and use LFU as the replacement strategy. Note that we have demonstrated in the previous chapter that such uniform pervasive caching may not be the best cache deployment strategy. However, this setup is the most difficult case for the routing system and is hence used to evaluate our proposal.

NRS and CCN routing with or without Bloom filter assistance are compared in our simulation. For NRS, a fixed resolution latency (default: 100ms) is incurred before the user sends out each request for a named content. The name resolution is assumed to be able to locate the closest replica of the requested content. On the other hand, CCN routing does not need any resolution but could only locate the content hosted by its publisher's server (This is a conservative assumption since some replicated contents may be aggregatable. But as discussed in Section 5.1, at Internet scale, even if CCN routing only handles publishers' servers, it is still beyond the capability of today's technology). With Bloom filter assistance, both schemes are able to peek all contents (original/replicated/cached) available within a few hops (depending on the advertisement scope of the Bloom filters, default: 3 hops) and utilize them without resolution. When caching is involved, on-path caches can be utilized by any scheme but off-path



(a) Average Routing Stretch Comparison



(b) Routing Stretch vs. Content Distance from Requester

Figure 5.3: Baseline comparison: (a) Average routing stretch of NRS and CCN with and without Bloom filter (BF) assistance under default parameter setting. (b) Most performance improvement provided by Bloom filter assistance stems from nearby contents.

caches (i.e., caches not on the default path to the original server but within the advertisement scope) are only visible to Bloom filter assisted schemes.

We use routing *stretch* as our primary performance metric. It is defined as the ratio of the actual time to fetch a content vs. the round-trip latency from the requester to the closest prepositioned *replica*. Note that when caching is involved, the stretch may be smaller than one since we may find a closer cached copy of the content than the closest replica. We do not define stretch based on the nearest cached content since caching is volatile while replicas are persistent.

5.3.2 Simulation Results

Figure 5.3a compares the average routing stretches of NRS and CCN with and without Bloom filter assistance under default parameter settings (Bloom filters advertised within 3 hops, 100ms resolution latency for NRS, 10 replicas, no caching). As the figure shows, Bloom filter assistance could reduce the stretch for both NRS and CCN. A closer look at the routing stretch for contents of different distances from the requester (Figure 5.3b) reveals that most of the performance enhancement comes from nearby contents whose routing stretch is very high without Bloom

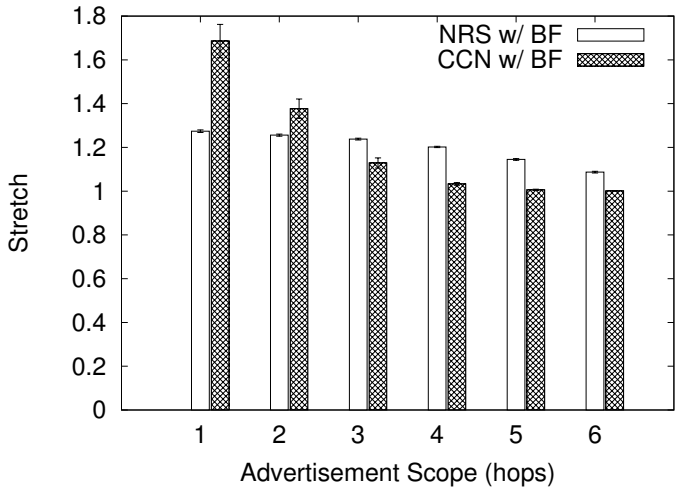


Figure 5.4: Routing stretch vs. advertisement scope: advertising Bloom filters within a few hops seems to be adequate to reap most of the benefits. Further increase of advertisement scope provides marginal performance improvement.

filter assistance. That is because the 100ms resolution latency in NRS is a huge overhead for contents that have a nearby copy and could have been retrieved in a few milliseconds. For CCN, these nearby copies may be overlooked by the routers since they are not aggregatable. The request goes to the publisher's server which may be far away. With Bloom filter assistance, even though we can only peek contents within 3 hops, significant performance improvement can be obtained. Actually, peeking contents further away has marginal impact on the performance as shown by Figure 5.4. Here we repeat the same tests as Figure 5.3 with different advertisement scopes. According to this figure, 3 or 4 hops of Bloom filter advertisement seem to be adequate to decrease the routing stretch to a level very close to optimal. At this scale, the overhead of Bloom filter advertisement is minimum and is definitely within the reach of today's technology.

Let us now vary the parameter settings in the baseline experiment to study their impacts on the performance. Figure 5.5 shows the impact of name resolution latency on the performance of NRS routing schemes. Obviously, longer name resolution latency leads to higher stretch either with or without Bloom filter assistance. But Bloom filter assisted NRS always performs better than NRS itself and the extent of improvement increases as resolution latency increases (7%

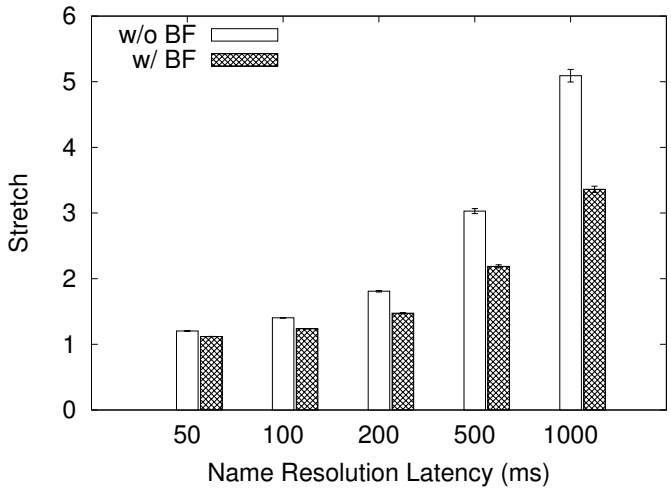


Figure 5.5: Impact of name resolution latency on the routing stretch of NRS schemes: longer resolution latency leads to higher stretch but Bloom filter assistance always helps.

stretch reduction at 50ms resolution latency vs. 34% stretch reduction at 1000ms resolution latency).

Another parameter we studied is the number of replicas in the network. Note that when we increase the number of replicas, users are more likely to have nearby copies of the requested contents. Hence the routing stretch of NRS and CCN will increase if they cannot locate these nearby copies. On the other hand, more replicas makes it more likely to find a copy of the content within in 3 hops, resulting in a stretch close to one (optimal) for Bloom filter assisted routing. Figure 5.6 demonstrates this phenomenon: the increased number of replicas has contrary impacts on the stretch for routing schemes with and without Bloom filter assistance. NRS and CCN without Bloom filter assistance both see a significant increase of their routing stretch while Bloom filter assisted routing has even lower stretch with more replicas. In the most extreme case (CCN under 100 replicas), Bloom filter assistance could reduce the routing stretch by as much as 91%.

Finally, we study the impact of caching by varying the cache capacity of all routers from 0 to 50 contents. Although these sizes seem to be small, they reflect the fact that the number of contents that can be cached by a single node is a very smaller portion of the total number of

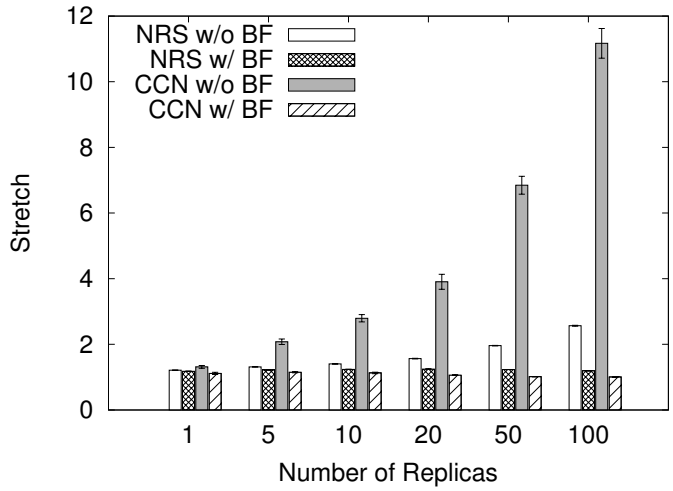


Figure 5.6: Routing stretch vs. number of replicas: more replicas cause significant increase of routing stretch for NRS and CCN but lowers the stretch of Bloom filter assisted routing schemes.

contents. As shown in Figure 5.7, increased cache capacity leads to decreased routing stretch for all schemes since on-path caches can be utilized by all of them. But Bloom filter assisted routing still provides considerable performance enhancements over NRS and CCN routing.

5.4 Extending to Internet Scale

In this section, we explore the possibility to extend Bloom filter-based routing to a Internet-scale routing protocol for ICN. We identify its unique advantages compared to existing approaches and also point out a few obstacles we need to clear to make it feasible.

5.4.1 Advantages of Bloom Filter-based ICN Routing

As discussed in Section 5.1, Bloom filter-based ICN routing combines the benefits of NRS and CCN. It removes the overhead of name resolution latency in NRS and could handle all kinds of contents without any aggregatability constraints. Moreover, the number of entries in a Bloom filter-based routing table is much smaller than that of NRS or CCN. That is because NRS or

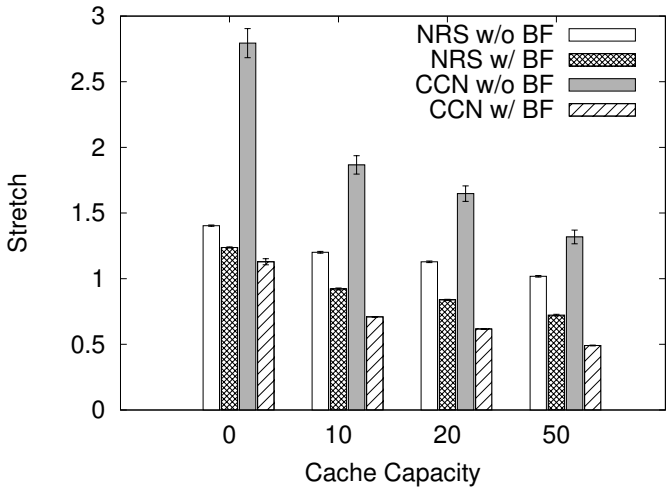


Figure 5.7: Routing stretch vs. cache capacity: increased cache size decreases routing stretches of all schemes but we can still observe clear performance improvements by Bloom filter assistance.

CCN routing tables are both indexed by content names or name prefixes. Each entry indicates that a specific content name or name prefix should be forwarded through a specific interface to a specific next-hop. Bloom filter-based routing table works the other way around. The table is indexed by the interfaces/next-hop and each entry indicates what contents are reachable via this interface/next-hop. Therefore, the number of entries in a Bloom filter-based routing table is independent of the number of contents we need to handle but depends on the number of interfaces/next-hop a router has. The latter is several order of magnitude smaller than the former. Finally, the lookup algorithm of Bloom filter-based routing is much simpler than longest prefix match used in IP or CCN. Each Bloom filter lookup is a simple $O(1)$ operation that can be easily accelerated by hardware circuits and multiple Bloom filter lookups can be parallelized as well.

5.4.2 Scalability Concerns and Mitigation Techniques

Memory Requirements

Although the number of entries in a Bloom filter-based routing table is independent of the number of contents, the size of each entry is dependent on it. The major component of each entry is the Bloom filter that represents the contents that are reachable via this interface/next-hop (There may be other metadata associated with each entry but they are small compared to the Bloom filter). It is the aggregation of all Bloom filters advertised by the routers/servers reachable via this interface/next-hop. This Bloom filter must be large enough to keep the false positive probability at an acceptable level after all the aggregations. Bloom filters typically require at least 8 bits for each item inserted into it to keep the false positive probability low. If we aim to handle 10^{12} contents, the Bloom filter size needs to be 1TB which is fairly large but feasible via solid-state drive (SSD). If even more contents need to be accommodated (e.g., 10^{15}), it may be infeasible with today's technology. However, we want to point out that this limitation is purely a size issue. Since Bloom filter lookup is a much simpler operation than longest prefix match, hardware acceleration and parallelization may be applied and our requirements on memory access speed may be relieved. We may use slower but larger storage devices and still achieve line speed forwarding. With the advancement in memory and storage technology, we hope a balance point between speed and capacity can be found so that Bloom filter-based ICN routing is feasible at Internet scale.

Signaling Overhead

The cost to exchange control messages (primarily Bloom filter advertisements) among ICN routers is another important issue we need to consider. This signaling overhead is dependent on three factors: the size, frequency and scope of the advertisements.

As discussed earlier, the size of the Bloom filter is very large if we want to handle all the contents over the Internet. To advertise Bloom filters as large as 1TB all over the Internet is

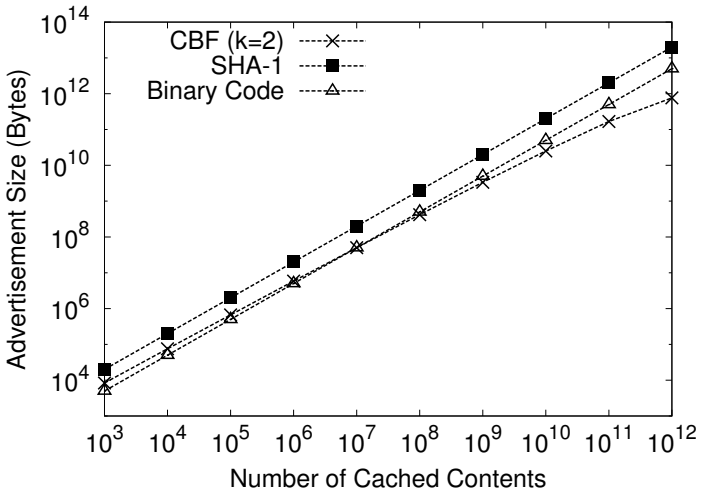


Figure 5.8: Comparison of the advertisement size: compressed Bloom filter (CBF) with 2 hash functions ($k = 2$) vs. 160bit SHA-1 hash vs. 40bit binary code (minimum required bits to uniquely represent 10^{12} contents).

obviously unrealistic. Luckily, that size is set to maintain a low false positive probability *after* Bloom filter aggregation. When the Bloom filters are initially constructed and advertised, they should be very sparse since the number of contents possessed by each router/server is only a very small portion of all contents over the Internet. Such sparse arrays can be losslessly compressed to much smaller sizes via standard coding techniques. For instance, let us assume a Bloom filter of 1TB ($m = 8 \times 10^{12}$ bits) with two hash functions ($k = 2$). If the router/server has one million ($n = 10^6$) contents, the resulting Bloom filter would have a fill ratio ρ around 2.5×10^{-7} since

$$\rho = 1 - (1 - \frac{1}{m})^{kn} \tag{5.5}$$

Under optimal compression, the 1TB Bloom filter can be compressed down to around 5.8MB as the size s of the compressed Bloom filter is

$$s = m(-\rho \log_2 \rho - (1 - \rho) \log_2(1 - \rho)) \tag{5.6}$$

Figure 5.8 shows a broader picture when this Bloom filter is used to represent different

number of contents (from 10^3 to 10^{12}). Compared to advertising a list of SHA-1 hash values for all cached contents (160 bits per content), the compressed Bloom filter is constantly smaller. Even when we compare it with 40bit binary coding which is the minimum number of bits required to uniquely identify 10^{12} contents, the compressed Bloom filter is still better or as good unless you only cache a really small amount of contents (less than 10^5).

In addition to reducing Bloom filter size via compression, we also need to control the frequency and scope of advertisement updates so that the control plane overhead is manageable. To do this, we introduce the concept of *cache commitment* associated with *adaptive advertisement*. In our scheme, each router/server makes commitments on how long it will cache each content. Depending on the committed cache duration, we decide how far we should advertise the existence of this copy of the content. The longer we commit to cache a content, the farther it can be advertised. We believe such coupling of caching and routing is necessary in ICN. Otherwise, the quickly replaced contents can never be effectively used by other users.

Ideally, we should calculate an optimal cache duration for each content. However, this operation may be too costly. To keep a good balance between flexibility and simplicity, we propose to classify the cached contents into a few categories with different cache duration. For example, an ICN router may have a hierarchical cache of three levels. The lowest level is a transient cache where replacement can happen at any time. Contents in this cache are used only locally and are not advertised. The next level could be a durable cache where cache duration is at minute scale and will be advertised to routers within a few hops. The highest level could be a semi-permanent cache where cache duration is at hour scale and will be advertised even farther away. That way, we allow the routers enough flexibility to replace its cache at lower levels but regulate the replacement at higher levels to facilitate tractability and usability.

All contents start in the lowest transient cache. They may be promoted to higher levels or demoted back later based on their popularity or fetch cost. The specific criteria to determine promotion or demotion, namely the cache replacement algorithm, is orthogonal to our routing and forwarding algorithms. As long as the router conforms to its cache commitments, any

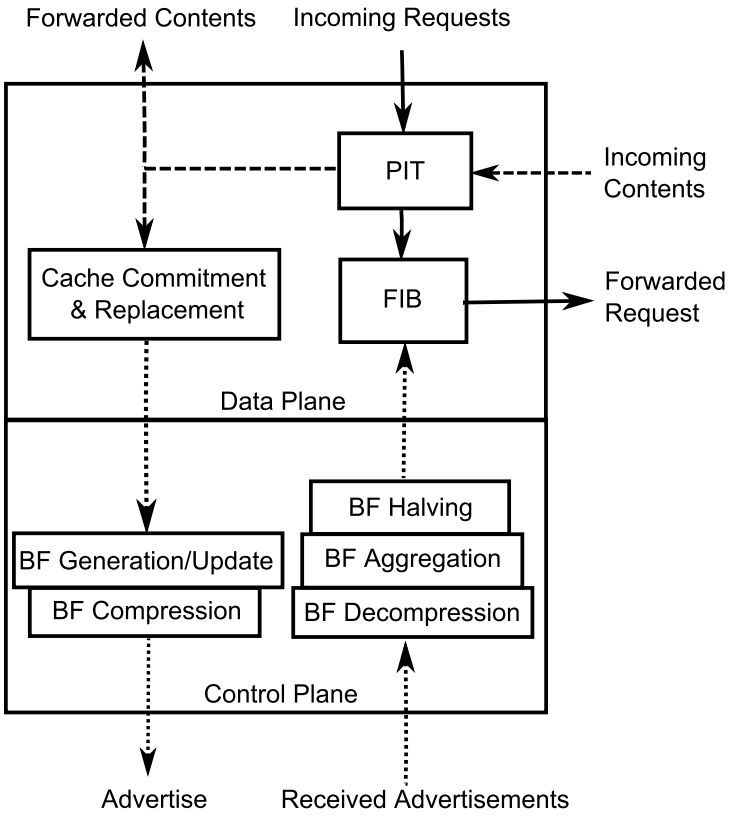


Figure 5.9: An architectural overview of an ICN router that combines all our routing proposals

replacement algorithm can be used.

Figure 5.9 shows an architectural overview of an ICN router that combines all our routing proposals: the caching module is not necessary for every router (as argued in Chapter 4). But if an router is equipped with caching capability, it manages the cache in a three-level hierarchy to balance between caching flexibility and routing scalability. For those contents that will be cached for a committed period of time, the router maintains counting Bloom filters to represent them locally (for easy update) and uses compressed Bloom filters to advertise them (for their smaller sizes). Upon receiving such advertisements, the router first decompresses them and then aggregate them as specified in Section 5.2. In case the aggregated Bloom filter is still very sparse, we use halving techniques to further reduce its size. This constructed routing table is then fed into the FIB for packet forwarding. Other data plane mechanisms follow the design of

CCN/NDN as introduced in Section 2.1.

Handling False Positive

False positive may happen in Bloom filters although we keep its probability very low. When false positive happens, we can mitigate its impact by employing a few techniques in the data plane. One way is to let the router forward the request to all matching interfaces/next-hops instead of only the best one (e.g., the one with minimum EFT). Since Bloom filter has no false negative, as long as the requested content is available, at least one request would finally reach it. If this operation is too expensive (as multiple copies of the requested content may be fetched simultaneously), another approach is to randomize the forwarding decision. Instead of always picking the best route, the router may randomly choose one of the matching interfaces/next-hops. If false positive happens and no content is retrieved by a request, the user will re-express its interest by issuing another request. With randomization, it is highly unlikely that the requests will repeatedly hit false positive as its probability is fairly low. Since the network layer anyway only provides best-effort service, we believe this is a satisfactory solution to handle false positive in Bloom filter based routing.

5.5 Concluding Remarks

In this chapter, we studied the ICN routing problem and proposed a Bloom filter-based approach to solve it either at a limited scale or full Internet scale. We quantitatively evaluated the performance improvement achievable by limited-scope Bloom filter advertisement via simulations. We then proposed a bold design to extend Bloom filter-based routing to full Internet scale and discussed its pros and cons. Our bottom line is that, limited-scope Bloom filter-based routing is easily feasible today and could bring considerable performance enhancements over NRS or CCN routing. Extending it to global scale may be challenging today but is a valuable direction to pursue in future.

Chapter 6

Proactive ICN Congestion Control with Multipath Support

Congestion control in NDN differs from traditional TCP-like congestion control due to its pull-based model and inherent support for multipoint retrieval. In this chapter, we argue that hop-by-hop congestion control along with backpressure signals and proper router/client reaction is a better fit for NDN than pure end-to-end congestion control, and present a complete solution of this form.

We first propose a novel hop-by-hop interest shaping algorithm to be executed at every NDN router. The interest shaper takes into account the unique interdependence between interests and returning data in NDN, and provably achieves proportional fairness between two-way traffic. We then propose a congestion signaling mechanism to timely and reliably notify downstream routers as well as end clients of the congestion. The clients react to the congestion signals in similar ways to relentless congestion control [71] to achieve efficient yet fair utilization of the available bandwidth. The downstream routers also utilize the congestion signals to make informed multipath forwarding decisions.

We implement our proposal in the ns-3-based NDN simulator (ndnSIM [8]) and evaluate its performance across a large number of different scenarios. The simulation results show that

our proposed scheme is able to effectively avoid congestion collapse, efficiently utilize available bandwidths from multiple paths, achieve near-optimal throughput with minimum delay and maintain reasonable fairness among competing clients.

6.1 Motivation

The unique communication paradigm of NDN requires a congestion control scheme that is significantly different from TCP. We hereby present an overview of our novel congestion control scheme which we believe is well-suited to NDN.

First, we control congestion by shaping interests rather than data packets. Since all data packets are sent in response to interest packets in NDN and they follow the same path, proper interest shaping can proactively prevents data packet loss. Although interest packet loss is still possible in this case, dropping interest packets early wastes fewer resources than dropping data packets late. We show that a correct interest shaping algorithm can efficiently use bandwidth without causing data packet loss under a wide variety of scenarios.

Second, we perform interest shaping in a hop-by-hop manner as opposed to end-to-end. That is because the data retrieval process is inherently point-to-multipoint. Many end-to-end mechanisms are not easily applicable in this case (e.g., maintaining a retransmission timer based on observed RTT). Hence, the first cornerstone of our NDN congestion control scheme is a hop-by-hop interest shaping algorithm to be executed at every NDN router. This shaping algorithm ensures that, for each individual link, the data packets retrieved by the interests will never overflow the link. Hence, the data packets will hardly queue up and congestion is reflected only in interest queue build-up. Section 6.2 will detail the design of the interest shaper.

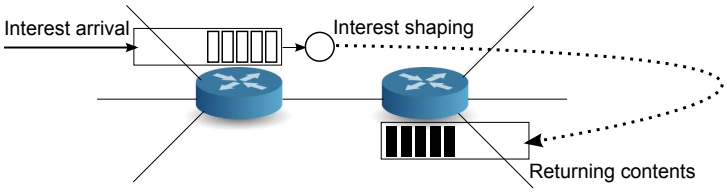
Enforcing interest shaping at every hop does not solve all the congestion control problem. Clients may still issue more interests than the network can handle and these excessive interests will queue up before the shapers at the bottleneck link and be discarded eventually. Although the cost of early interest packet loss is less than that of late data packet loss, the network resources consumed by those interests before being discarded are still wasted and this may

cause congestion collapse in the worst case. Hence, we need a mechanism for end clients to detect congestion so that they can react by reducing their request rates to the network.

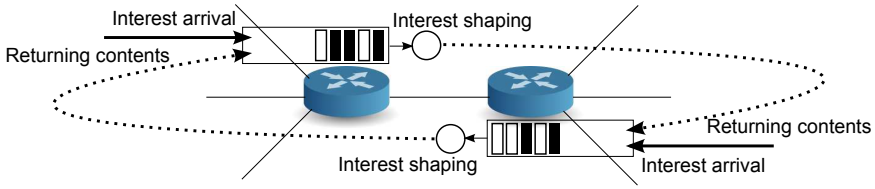
Traditional implicit congestion detection techniques that rely on end-to-end monitoring are not always applicable here. For instance, in NDN we can no longer use duplicate acknowledgements to infer packet loss. The presence of content caching or multihoming may also impact the accuracy of RTT-based timeout mechanisms. Additionally, such packet loss inference at end points causes confusion between congestive and non-congestive losses. Hence, an explicit congestion signaling mechanism is desired. Many such mechanisms (e.g., [87]) have been proposed in the past but they typically suffer from the problem that congestion signals themselves may exacerbate the congestion and get lost. In Section 6.3 we will show that our scheme will not increase congestion in the network under any circumstance and the congestion signals could be reliably delivered back to the clients.

The clients' reaction to the congestion signals along with the queuing mode at the interest shapers together determines the efficiency, fairness and convergence speed of our congestion control scheme. We opt for the relentless congestion control algorithm as our client reaction and complement it with state-of-the-art AQM scheme: PIE [81]. We will show that this combination is able to achieve efficient yet fair bandwidth utilization, converges reasonably fast, avoids long standing queues and unnecessary delay and is especially suitable for multipath scenarios.

Multipath scenarios are an important aspect of NDN congestion control and a key differentiator from TCP congestion control. Although congestion control algorithms exist for multipath TCP (MPTCP) as well [110, 52], they are designed under quite different assumptions from NDN. In NDN, end clients request for contents by location-independent names. The routers resolve such names into locations (either explicitly or implicitly) and take care of the potential multihoming of the requested contents or multiple paths to reach the contents. This procedure is largely opaque to the clients. In contrast, MPTCP requires the client to be fully aware of the multiple paths and impose window control on each individual path. Although there are proposals that advocate the use of routing labels to make clients multipath-aware [22], we do



(a) Simplified scenario



(b) Practical scenario

Figure 6.1: Interdependence between interest and data packets in reverse directions and its impact on hop-by-hop interest shaping

not think it conforms to the philosophy of ICN. In Section 6.4, we will propose a scheme that achieves similar efficiency and fairness goals as MPTCP but keeps the clients minimally aware of multipath.

6.2 Hop-by-hop Interest Shaping

Hop-by-hop interest shaping is the first building block of our NDN congestion control scheme. Its basic idea is very straightforward: each hop shapes the interests it sends upstream in order to control the returning data rate and avoid congestion in the downstream link. For instance, in the simplified scenario shown in Figure 6.1a where interests flow in only one direction and content data return in the reverse direction, if we assume that the link bandwidth is a known constant (c) and the interest and data packet sizes are also fixed (denoted as s_i and s_d respectively), then the returning data rate can be kept below c by shaping interest rate below c/r, where r is the size ratio between data and interest packets ($r = s_d/s_i$).

However, as shown in Figure 6.1b, a typical link within the network would see two-way traffic where interest and data packets flow in both directions simultaneously. This is especially

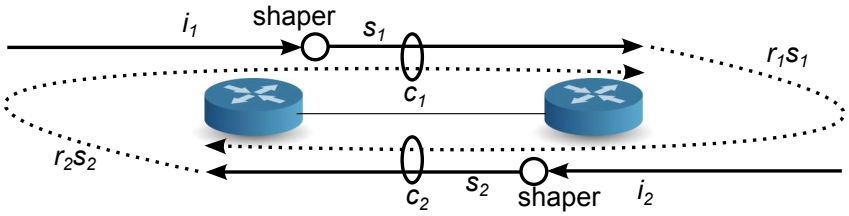


Figure 6.2: Modeling hop-by-hop interest shaping

true for NDN where the difference between clients and servers is obscure and every network entity can be a potential content source. Moreover, content names in interests can be long since it includes transactional information in many applications (e.g., [56]). As each interest packet fetches exactly one data packet in NDN, interests may consume a non-negligible fraction of the link bandwidth. Hence, it is non-trivial to determine the correct interest shaping rates since the interests in reverse directions are competing for bandwidth with data packets. The interest shaper in one direction can no longer assume that the entire reverse link bandwidth is available for returning contents. It needs to shape the interests properly so that enough room is left on the reverse link for interests in the reverse direction. The same logic applies to the interest shaper on the other side of a link and they form a recursive interdependence as depicted in Figure 6.1b. An imprudent shaping algorithm may cause congestion (by not accounting for reverse interests), or link under-utilization (by over-compensating for them). This problem is further complicated by other factors such as varying packet size, traffic burstiness and asymmetric link bandwidth. Hence, we need a more careful analysis of this unique interdependence between interest and data packets in reverse directions.

6.2.1 Theoretic Analysis

As depicted by Figure 6.2, let i_1 and i_2 denote the interest arrival rate in each direction. They are shaped down to s_1 and s_2 respectively by the interest shapers at the routers. r_1 and r_2 denote the average size ratio between data and interest packets in each direction and we assume that data packets are larger than interest packets (i.e., $r_1 > 1$ and $r_2 > 1$), which is typically true in

practice. Also let c_1 and c_2 denote the link capacity in each direction. Then we can formulate the interest shaping problem under steady state as follows:

Objective:

$$\max\ u(s_1) + u(s_2) \tag{6.1}$$

Subject to:

$$0 \leq s_1 \leq i_1 \tag{6.2}$$

$$0 \leq s_2 \leq i_2 \tag{6.3}$$

$$s_1 + r_2 s_2 \leq c_1 \tag{6.4}$$

$$r_1 s_1 + s_2 \leq c_2 \tag{6.5}$$

where $u(\cdot)$ is the utility function discussed later. The objective of this optimization is to maximize network utility (Equation 6.1) subject to demand (Equation 6.2 and 6.3) and bandwidth (Equation 6.4 and 6.5) constraints. As Figure 6.3 shows, the feasible region of this optimization problem is convex. Hence, if the utility function constitutes a concave objective function, the formulation is a convex optimization problem which is mathematically tractable.

Note that the utility function we pick has a profound impact on the solution. As mentioned above, reverse interests compete with forward data packets for bandwidth and the shaper needs to appropriately divide resources between them. The utility function must maximize data throughput while maintaining a degree of fairness between traffic in both directions. This is more complicated in asymmetric link bandwidth scenarios, where trying to maximize throughput in one direction can completely starve throughput in the opposite direction. We believe that *proportional fairness* is a proper fairness criterion between content flows in two directions and it has been shown in [61] that logarithmic utility functions can achieve proportional fairness. Hence, we hereby present a closed-form solution to the described optimization problem under a logarithmic utility functions (equivalent to maximizing the product of s_1 and s_2).

First, if we temporarily assume infinite demands in both directions, we can lift the con-

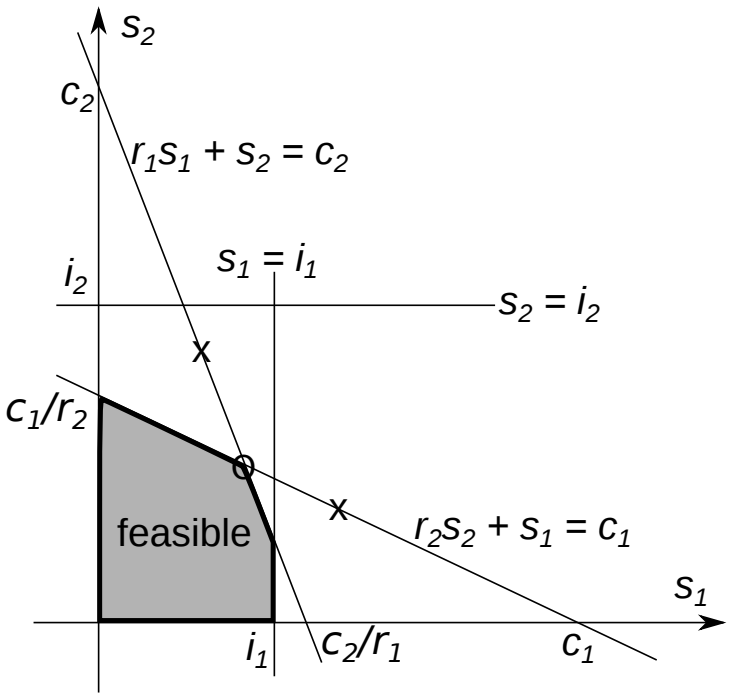


Figure 6.3: The feasible region of the optimization problem is convex.

straints in Equation 6.2 and 6.3. Second, from Figure 6.3 the optimal solution of the problem lies on the boundary of the feasible region, so we can convert the inequality constraints in Equation 6.4 and 6.5 into equality constraints. Now let us begin by solving the optimization problem under each equality constraint independently. Solving for Equation 6.4:

$$\begin{array}{ll} \text{maximize} & s_1 s_2 \\ \text{subject to} & s_1 + r_2 s_2 = c_1 \end{array}$$

The optimal solution for this problem can be determined using Lagrange multipliers:

$$\begin{cases} s_1 = \frac{c_1}{2} \\ s_2 = \frac{c_1}{2r_2} \end{cases} \tag{6.6}$$

This optimal point has been labeled by an "x" in Figure 6.3. Similarly, solving for Equation 6.5:

$$\begin{array}{ll} \text{maximize} & s_1 s_2 \\ \text{subject to} & r_1 s_1 + s_2 = c_2 \end{array}$$

The solution (also labeled by an "x" in Figure 6.3) is:

$$\begin{cases} s_1 = \frac{c_2}{2r_1} \\ s_2 = \frac{c_2}{2} \end{cases} \tag{6.7}$$

Now we consider the solution to the original problem under infinite demand (i.e., the optimization problem consisting Equation 6.1, 6.4 and 6.5). From Figure 6.3 we can see that the optimal solution primarily depends on how the two bandwidth constraint lines cross each other. If one of the optimal points is within the feasible region, then the optimal solution is just that point (Equation 6.6 or 6.7). If neither optimal point lies within the feasible region (the case shown in Figure 6.3), the optimal solution is given by the intersection of the two lines (labeled by a circle):

$$\begin{cases} s_1 = \frac{r_2 c_2 - c_1}{r_1 r_2 - 1} \\ s_2 = \frac{r_1 c_1 - c_2}{r_1 r_2 - 1} \end{cases} \tag{6.8}$$

The optimal solution to this infinite-load problem is summarized in Table 6.1.

Now we reintroduce the condition of finite load (Equation 6.2 and 6.3). In practice, it is possible that the traffic load in one direction is inherently low (e.g., the first hop link from a client may have lots of outgoing interests but few incoming interests). Under such cases, the reduced load in one direction should result in increased shaping rate in the opposite direction. The limiting case is unidirectional demand (Figure 6.1a) in which case:

$$i_2 = 0 \tag{6.9}$$

$$s_1 = \frac{c_2}{r_1} \tag{6.10}$$

Hence, the instantaneous shaping rate should be variable. We seek proportional fairness between the two-way traffic only if both directions have excessive demand and are competing for the link capacity in both directions. If the load in one direction is inherently low, our scheme is

Table 6.1: Optimal solution under infinite loads in both directions

case	s_1	s_2
$\frac{c_1}{c_2} < \frac{2r_2}{r_1r_2+1}$	$\frac{c_1}{2}$	$\frac{c_1}{2r_2}$
$\frac{2r_2}{r_1r_2+1} \leq \frac{c_1}{c_2} \leq \frac{r_1r_2+1}{2r_1}$	$\frac{r_2c_2-c_1}{r_1r_2-1}$	$\frac{r_1c_1-c_2}{r_1r_2-1}$
$\frac{c_1}{c_2} > \frac{r_1r_2+1}{2r_1}$	$\frac{c_2}{2r_1}$	$\frac{c_2}{2}$

work-conserving and will let the traffic in the other direction grab as much bandwidth as it can. An adaptive algorithm that adjusts the shaping rate between Table 6.1 and Equation 6.10 based on the fluctuating demand will be presented in the next section.

6.2.2 Practical Algorithm

As shown above, obtaining the optimal interest shaping rate is mathematically tractable if the shaper has knowledge of the data/interest size ratio (r_1 and r_2), link capacity (c_1 and c_2) and demand (i_1 and i_2) in both directions. Due to the symmetric routing of interests and contents in NDN, the shapers can independently measure r_1 and r_2 by observing the interests and contents arriving at and leaving the interface. If we assume that link bandwidths c_1 and c_2 are static parameters (we leave the study of links with varying bandwidths such as wireless links as our future work), they can also be made known to the shapers easily. However, the offered demand (i_1 and i_2) is constantly varying and cannot be accurately measured. Further, the shaper at one end of a link cannot know the interest load on the other side without additional message exchange. Hence, we have designed an adaptive algorithm that does not require accurate knowledge of the offered demand on both sides, which we present below.

From Equation 6.10, we can compute the *maximum* interest shaping rate max_s_1 (which occurs when $i_2 = 0$). We can also determine the *minimum* interest shaping rate min_s_1 from Table 6.1 (which occurs when i_2 is infinite). To determine the actual shaping rate, we measure the incoming interest rate and use it as an estimation of the load on the other side. Assuming a similar shaper is running on the other side of the link, if there is sufficient demand in the reverse direction, then the observed incoming interest rate (obs_s_2) should be no less than the

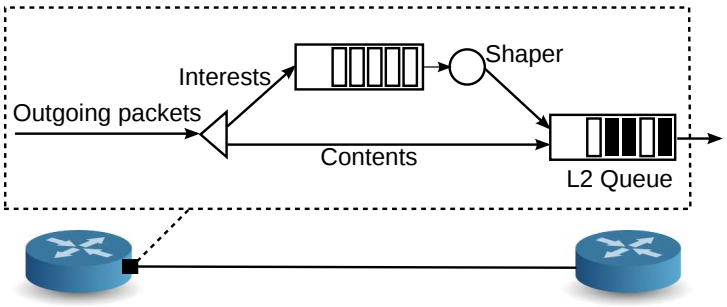


Figure 6.4: Shaper implementation

rate given by the s_2 column in Table 6.1 (we call it the expected minimum incoming interest $expmin\_s_2$). Hence, if $obs\_s_2 \geq expmin\_s_2$, the shaping rate is set to min_s_1. Otherwise, the shaping rate is calculated as follows:

$$min\_s_1 + (max\_s_1 - min\_s_1)(1 - \frac{obs\_s_2}{expmin\_s_2})^2 \tag{6.11}$$

This equation adjusts the outgoing interest rate between min_s_1 and max_s_1. If $obs\_s_2 = 0$, the shaping rate become max_s_1. As we observe higher incoming interest rate, we reduce the shaped outgoing interest rate until it hits min_s_1. In our simulations, we observed that the quadratic control used here is more conservative and robust than a linear control.

Figure 6.4 shows how the proposed interest shaper is implemented on each interface of an NDN router. The outgoing packets are first classified into interests and contents. Content packets are passed directly to the link output queue without shaping. Interest packets join a separate shaper queue, the output of which is fed into the link output queue. The shaping rate of this queue is dynamically computed as per Equation 6.11.

Note that our formulation gives the proportionally fair *average* shaping rate in steady state. It does not account for the transient fluctuation of the data rates. In practice, a perfectly shaped/paced interest flow at rate s_1 will never bring back a data flow at a constant rate of $r_1 s_1$, due to heterogeneous packet sizes or RTTs. Hence, we always leave some headroom (by default, 3%) in the actual shaper to accommodate such traffic burstiness. This headroom

together with proper provisioning of buffers in the Layer 2 (L2) queues is able to absorb the burstiness in the data flow and make the system converge to the optimal steady state. We will demonstrate this in Section 6.5 where we show that the L2 queues in all our simulations never overflow and our scheme effectively transfers congestion into the interest shaper queues.

6.3 Congestion Signaling and Client Reaction

As explained in Section 6.1, hop-by-hop interest shaping alone is inadequate to solve the entire congestion control problem in NDN. Interest shaping simply transfers data congestion into interest congestion (i.e., L2 queues will never overflow now, but interest queues before the shapers may overflow). This is helpful as early interest packet drop is much cheaper than late data packet loss. But it is not sufficient. In a multihop network, perfect interest shaping at each hop does not prevent clients from issuing excessive interests that will be discarded later by the shapers within the network core (unless the bottleneck link is always at the first hop) and this may lead to congestion collapse in the worst case. Hence, instead of letting the interest shapers silently discard excessive interests, congestion signaling is required and the clients should react upon such signals.

We propose to let the shapers reject excessive interest packets with negative acknowledgments (NACK). Such explicit congestion notification has been proposed in the current Internet [87]. A similar NACK mechanism for NDN has also been proposed in [113, 112]. However, in the following sections, we will explain how we utilize state-of-the-art AQM schemes to generate NACKs in a timely manner and why our congestion signaling mechanism does not suffer from the traditional problem that congestion signals may aggravate congestion or get lost due to congestion. We will also describe how the clients react to the NACKs to achieve fair and efficient network utilization.

6.3.1 Timely NACK Generation

With proper interest shaping, data packets should saturate with available bandwidth without overloading it. Hence, there should be no standing queues for the data packets. However, interest packets may still queue up at the shaper and a simple DropTail policy may lead to long queuing delay. Hence, we learn a lesson from the bufferbloat problem in the current Internet [46, 77] and adopt PIE [81] at the shaper interest queues to generate NACKs timely.

PIE periodically measures the experienced queuing delay of a queue and uses a proportional-integral controller to adjust the packet drop probability of this queue so as to keep the queuing delay close to a given target value (20ms by default). Incoming packets are randomly dropped based on the computed probability before being enqueued. This random early drop not only reduces the queuing delay but also improves the fairness and convergence speed among competing flows, which is a known advantage of AQM schemes over DropTail policy. Please refer to [81] for more details of the PIE algorithm. We literally adopt the same algorithm except that 1) the algorithm is executed at the interest shaper queues rather than the L2 queues and 2) interests are NACKed as opposed to dropped.

6.3.2 Reliable NACK Propagation

After a NACK is generated, it is forwarded downstream towards the client just as if it is the returning data packet (following the same path as the interest packet but in reverse order). A common problem for such explicit congestion notifications is that the congestion signals may exacerbate the congestion since they generate extra traffic. Also, they themselves may get lost due to congestion. However, we find that, with hop-by-hop interest shaping in place, such problems are automatically solved in NDN. When a neighboring router forwards an interest across a link, enough bandwidth has been accounted for in the reverse direction of this link to accommodate the expected returning data packet. If this interest is then rejected with a NACK towards the client, this NACK takes the place of the accounted-for data packet. As long as NACKs are smaller on average than data packets (which is the case in practice), there should

be enough bandwidth in all the downstream links for the NACKs to reach the clients and they should never get lost due to congestion.

6.3.3 Client Reaction

Let us now look at how the client should react when it receives a NACK packet. Obviously, the client should somehow reduce its interest sending rate. But how exactly should we do it? Note that when we say we should reduce the interest sending rate, we can do it only over a group of interests. With a group of interests, we can control its sending rate by adjusting the time gaps between consecutive interests. If we have only a single interest, there is not way to enforce rate control. Thus, a client must divide the interests it is going to send into groups and impose certain kind of rate/window control on each group. We call such interest groups "flows" and a flow is the basic operating unit of our congestion control scheme.

The definition of a flow is fairly straightforward in TCP/IP networks. A TCP connection identified by the 4-tuple (source/destination IP address and port number) naturally constitutes a flow. It is much harder to define a flow in NDN since the "destination" of an interest packet is a content name rather than a location. This location-independent semantic is the key to NDN's unique advantages over the current Internet but it also brings considerable difficulties for congestion control. That is because, congestion is fundamentally location-dependent: it is a specific link within the network that is congested and only the packets that may traverse that congested link should back off. Other packets should be unaffected. Hence, a flow should be a group of packets that may traverse the same set of links and its rate can be controlled to avoid congestion on these links. A TCP connection easily fits this condition since all packets with the same 4-tuple follow the same path. In NDN, a more careful definition of flow is required. We define an NDN flow as a group of interests (as well as the corresponding date packets fetched by them) that are issued by the same client and destined to the same prefix, where this prefix is *indivisible* in the routing/forwarding domain.

Note that the difficulty in defining an NDN flow is not due to the potential multihoming

of the contents and hence the multipath nature of an NDN flow. A flow with multiple paths is perfectly fine as demonstrated by multipath TCP [43] and SCTP [99]. The real difficulty is that, in theory, each NDN interest can be routed independently. If that is the case, no two interests can ever be group into a flow since they may traverse different set of links and hence the NACK of one interest should not slow down the rate of the other. In that case, it is almost impossible to define a flow and NDN congestion control would be extremely difficult. Also, routing each content name independently may have serious scalability problems as discussed in Chapter 5. Hence, in this chapter we assume that interests are not routed/forwarded individually. Interests under the same prefix (NDN uses hierarchical names) are grouped together and forwarded equivalently. Note that this aggregation is hierarchical. For instance, all interests under "/edu/ncsu/csc/*" may be grouped under one FIB entry in a core router. But at an edge router, it may have separate FIB entries for "/edu/ncsu/csc/yaogong/dissertation/*" and "/edu/ncsu/csc/courses/*". By indivisible prefixes, we mean the lowest level of the hierarchy where all interests under that prefix are guaranteed to be treated equivalently across all routers. This indivisibility can be guaranteed by the content publisher when it initially advertises or registers the prefix. The routers can freely aggregate the prefixes further up but cannot divide them further down. Under this assumption, our previous definition of an NDN flow makes sense: if a prefix is indivisible in the routing/forwarding domain, all interests issued by a client under this prefix will form a "Forwarding Equivalence Class (FEC)". They may span multiple paths but it still makes sense to control the overall rate of this flow since all its interests share the same set of congested links.

Also note that the hop-by-hop interest shapers proposed in the previous section are unaware of the traffic flows. They operate over the aggregated interests on each link and ensure that they do not overflow the reverse link. Only end clients are flow-aware and impose rate/window control on each flow. This is similar to the current Internet where the routers are typically oblivious to TCP flows and only clients keep track of end-to-end TCP connections. In contrast, some other NDN congestion control proposals (e.g., [21, 79]) require routers to identify traffic

flows, which is both costly and difficult (NDN interest packets do not contain the identity of the client).

Now that we have a clear definition of flow, let us see how the client controls the interest issuing rate of each flow. We adopt the traditional window-based control due to its technical maturity and simpler implementation (no high-precision timers needed). However, instead of the typical additive-increase-multiplicative-decrease (AIMD) algorithm used by many existing proposals [20, 21], we adopt a slightly different window control algorithm to better utilize the NACK packets. Our algorithm works as follows: upon receiving a data packet for a flow, we increase the window size W of this flow as

$$W \leftarrow W + \frac{\alpha}{W} \tag{6.12}$$

where α is a parameter that tunes the aggressiveness of the algorithm. By default, α is set to one. Upon receiving a NACK packet, the window size adjusts as

$$W \leftarrow W - 1 \tag{6.13}$$

Note that the window increase algorithm is additive and is in fact the same as AIMD. But the window decrease algorithm is different. Instead of applying a multiplicative reduction to the window upon a loss event, we reduce the window size by one for each received NACK. For instance, if the current window size is 10, 10 interests will be sent in flight. If the network can only accommodate 7 interest packets at this time, 3 NACKs will be generated and sent back to the client. Then the client will reduce the the window size to exactly 7. This is similar to relentless congestion control [71] where the window is reduced by the number of lost segments after each loss event. The advantage of this algorithm is that, in steady state, the window size will be very smooth (one packet above or below the network capacity) as opposed to AIMD's sawtooth behavior. This is especially helpful in multipath scenarios as will be shown in Section 6.4.

We have also incorporate the slow-start algorithm [55] where the window size is increased by one for each received data packet (equivalent to doubling the window size every RTT) at the initial phase of a flow to quickly approach the steady state. The flow exits slow-start after receiving the first NACK and switches to congestion avoidance phase where the window size is adjusted according to Equation 6.12 and 6.13.

As pointed out by [71], the relentless congestion control is not AIMD-friendly. Fortunately, in NDN, we do not need to worry so much about backward compatibility as TCP variants do (e.g., [40]). Instead, we are only concerned with the intra-protocol fairness. Let us now analyze this aspect of our algorithm: assume that two NDN flows are competing for a bottleneck link. The current window sizes of the two flows are W_1 and W_2 respectively. If the bottleneck capacity is C, when $W_1 + W_2 < C$, the link is under-utilized and both flows will increase their window size additively. At certain point, $W_1 + W_2 > C$. Then $W_1 + W_2 - C$ NACKs will be generated and sent by to the clients. Assume that the number of NACKs a flow receives is proportional to its current window size (roughly true under AQM as will be shown in the evaluation), then the number of NACKs received by Flow 1 will be

$$(W_1 + W_2 - C) \cdot \frac{W_1}{W_1 + W_2} = W_1 - \frac{W_1 \cdot C}{W_1 + W_2} \tag{6.14}$$

Then the window size of Flow 1 will be adjusted to

$$W_1 \leftarrow W_1 - (W_1 - \frac{W_1 \cdot C}{W_1 + W_2}) = \frac{C}{W_1 + W_2} \cdot W_1 \tag{6.15}$$

Similarly, the window size of Flow 2 will be

$$W_2 \leftarrow \frac{C}{W_1 + W_2} \cdot W_2 \tag{6.16}$$

Hence, the window sizes of both flows are multiplicatively decreased by $\frac{C}{W_1+W_2}$. Thus, this algorithm can be also viewed as AIMD except that, instead of decreasing by a magic fraction

of β, the decrease factor is varying. According to [29], this AIMD variant will still converge to efficiency and fairness although the process may take longer in practice. We will evaluate the convergence speed of this algorithm in Section 6.5.

Lastly, let us briefly derive the response function of relentless congestion control. Following the simplified method in [110], in equilibrium the window increase and decrease should balance out, that is

$$(1-p)\cdot\frac{\alpha}{W} = p\cdot 1 \tag{6.17}$$

where p is the NACK probability. Thus

$$W = \frac{\alpha\cdot(1-p)}{p} \approx \frac{\alpha}{p} \tag{6.18}$$

when p is small. This result roughly agrees with the analysis in [71]. It will be used later in our analysis of the multipath scenario. As a side note, this congestion control algorithm also has better scaling properties in long fat networks than traditional AIMD whose average window size is proportional to $1/\sqrt{p}$ rather than $1/p$.

Finally, note that NACKs can also be used for reliability management. If an interest is NACKed, the client may or may not want to reissue the interest, depending on the reliability requirements of the application. However, our congestion control scheme is full decoupled from reliability management. The former determines *whether* an interest can be sent at this time. The latter determines *which* interest to be sent. This is different from TCP where these two mechanisms are entangled. This decoupling is highly desirable since congestion control is a network requirement which should be imposed on all clients, while reliability is an application requirement which should be determined by each application individually.

6.4 Multipath Forwarding Strategy

The mechanisms proposed above (hop-by-hop interest shaping, NACK generation via PIE, NACK propagation and client reaction) already constitute a complete solution to NDN congestion control under single-path forwarding. Let us now look at how multipath forwarding may impact the picture. First, let us consider the design goals of a multipath congestion control scheme and how multipath flows should coexist with single-path flows. We learn from the design goals of multipath TCP congestion control [110] and come up with the following slight different design goals for NDN multipath congestion control:

1. **Improve throughput:** in the absence of competing flows, the multipath flow should be able to grab available bandwidth from all paths.

2. **Do no harm**: in the presence of competing flows, the multipath flow should get the same total throughput as a single-path flow would on the best path under the same RTT. Further, the capacity taken by the multipath flow from any of its paths should be no more than a single-path flow would on that path under the same RTT.

3. **Balance congestion**: a multipath flow should move as much traffic as possible off its most congested paths to achieve resource pooling.

Note that, a multipath flow should behave differently depending on whether there are competing single-path flows on each path it uses. In the absence of such competing flows, the primary goal of multipath congestion control is to improve throughput by utilizing the bandwidths on all the paths available to it. However, in the presence of competing flows, the primary goal is to ensure fairness among all flows no matter they are single-path or multipath. Specifically, we have two sub-goals as stated above: 1) the multipath flow should get the same total throughput as a single-path flow would on the best path under the same RTT. It should get no more than that otherwise it would be unfair for the other flows. It should get no less than that, either. Otherwise, the flow would rather use the single best path as opposed to utilizing multiple paths.

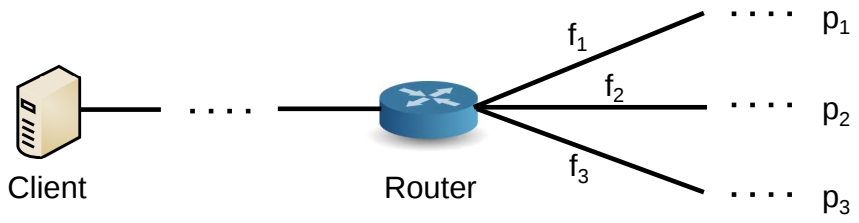


Figure 6.5: A simplified model of multipath flow.

2) The capacity taken by the multipath flow from any of its paths should be no more than a single-path flow would on that path under the same RTT. This imposes additional fairness constraints on the multipath flow: not only should the total throughput of the multipath flow be fair, the throughput fraction it gets on each individual path should be fair as well. Finally, balancing congestion is another important functionality of multipath flows: it achieves resource pooling across different paths within a network [109].

With the above-mentioned design goals in mind, let us now analyze how our current congestion control scheme would perform under multipath forwarding and how we can further polish our scheme for multipath scenarios.

6.4.1 Theoretic Analysis

Let us first formulate a simple model to analyze the performance of multipath flows. As shown in Figure 6.5, assume a client has a flow that may utilize n paths in the network. The client itself is unaware of the multiple paths and simply issues a series of interests under certain prefix. The router partitions these interests over the multiple paths that could serve this prefix and each path i gets a fraction of f_i. Obviously,

$$\sum_{i=1}^{n} f_i = 1 \tag{6.19}$$

Note that we assume the paths/interfaces that can be used for a prefix is pre-determined in the routing domain. The forwarding strategy treats them as given and only decides the traffic fraction of each path. It could not freely try out all interfaces as suggested in some NDN

proposals [112].

Now assume that there are single-path flows competing on each path and the NACK probability of path i is p_i. Then utilizing the results at the end of Section 6.3, the steady-state window size of a single-path flow on path i would be

$$W_i = \frac{1}{p_i} \tag{6.20}$$

if the default α value of 1 is used.

Now, let us think about the steady-state window size of the multipath flow. As discussed earlier, in NDN the client is typically unaware of the existence of multipath. It is the intermediate routers that make multipath forwarding decisions. Hence, the client adjusts the window of a flow in exactly the same way no matter it is single-path or multipath. Hence, the steady-state window size of the multipath flow should follow the same equation as single-path flows:

$$W_m = \frac{1}{p_m} \tag{6.21}$$

except that the NACK probability p_m experienced by the multipath flow is a combination of the NACK probability of each path. Assuming that NACKed interests are not retried on alternative paths by the router, then

$$p_m = \sum_{i=1}^{n} f_i p_i \tag{6.22}$$

With these definitions, let us look back at the design goals we mentioned earlier, especially the fairness goals. It is trivial to show that the fairness goals can be represented as:

$$W_m = \max_i W_i \tag{6.23}$$

$$f_i W_m \leq W_i, \forall i = 1, 2, \cdots, n \tag{6.24}$$

Now the question is: how should the multipath forwarding strategy adjust the traffic fraction

f_i of each path so as to achieve these goals? In fact, there is already an answer to this question in the literature. Both [62] and [51] have shown that, the router should adjust the traffic fraction of each path such that the NACK probability of all paths are equalized. In this ideal case,

$$p_i = p, \forall i = 1, 2, \cdots, n \tag{6.25}$$

$$p_m = \sum_{i=1}^{n} f_i p_i = \sum_{i=1}^{n} f_i p = p \tag{6.26}$$

Hence,

$$W_m = W_i, \forall i = 1, 2, \cdots, n \tag{6.27}$$

That is, all flows will achieve the same throughput (assuming homogeneous RTT) no matter they are single-path or multipath. In case it is not possible to equalize the NACK probability of all paths, the multipath flow should only use the least congested path(s). This is similar to the COUPLED algorithm in [110].

Although this algorithm is theoretically optimal in steady state, it has two problems in practice. First, in highly symmetric scenarios such as two paths with similar NACK probabilities, this algorithm will randomly oscillate between the two paths which is undesirable. Second, in highly asymmetric scenarios, the algorithm will let the multipath flow use only the least congested path. This is problematic under dynamic traffic where a congested flow may become uncongested. Under this scenario, the multipath flow will be tardy in picking up the bandwidth of the formerly congested path since it didn't send enough traffic to that path to constantly probe its congestion level.

To mitigate these issues, we adopt an alternative multipath forwarding strategy in our congestion control scheme: we set the traffic fraction of each path to

$$f_i = \frac{\left(\frac{1}{p_i}\right)^k}{\sum_{j=1}^{n} \left(\frac{1}{p_j}\right)^k} \tag{6.28}$$

Basically, this multipath forwarding strategy sets the traffic fraction of each path proportional to $\left(\frac{1}{p_i}\right)^k$ where $k(k \geq 1)$ is a parameter that tunes the trade-off between optimal resource pooling and responsiveness. This algorithm still favors less congested path as the COUPLED algorithm does, but it does not try to equalize the NACK probability of all paths. In fact, as we increase k, the algorithm approaches the COUPLED algorithm to achieve better resource pooling. When we decrease k, we try to make the algorithm more responsive to traffic load changes. When $k = 1$, the algorithm roughly corresponds to the SEMICOUPLED algorithm in [110]. We will find the optimal k via simulation in Section 6.5.

Note that there is a problem with this forwarding strategy: since the NACK probability on each path is not perfectly equalized, the NACK probability of the multipath flow will be somewhere between the least congested path and the most congested path:

$$p_{min} \leq p_m = \sum_{i=1}^{n} f_i p_i \leq p_{max} \tag{6.29}$$

In that case, the multipath flow may experience less throughput than a single-path flow on the best path, which violates our fairness goal (Equation 6.23). This is inevitable no matter how the router adjusts the traffic fractions (unless all interests go through the best path). To compensate for this throughput loss, we propose to adjust the additive increase factor α of the multipath flows. Instead of the default value of 1, we set it to

$$\alpha = \frac{p_m}{p_{min}} \tag{6.30}$$

With this compensation, the steady-state window size of the multipath flow would be

$$W_m = \frac{\alpha}{p_m} = \frac{1}{p_{min}} = \max_i W_i \tag{6.31}$$

Thus, we ensure that the multipath flow should get the same total throughput as a single-path flow would on the best path under the same RTT (Equation 6.23). Let us now check the other

fairness goal:

$$f_i W_m = \frac{\left(\frac{1}{p_i}\right)^k}{\sum_{j=1}^{n}\left(\frac{1}{p_j}\right)^k} \cdot \frac{1}{p_{min}} = \frac{\left(\frac{1}{p_i}\right)^{k-1} \cdot \frac{1}{p_{min}}}{\sum_{j=1}^{n}\left(\frac{1}{p_j}\right)^k} \cdot \frac{1}{p_i} = \frac{\left(\frac{1}{p_i}\right)^{k-1} \cdot \frac{1}{p_{min}}}{\left(\frac{1}{p_{min}}\right)^k + \sum_{j \neq min}\left(\frac{1}{p_j}\right)^k} \cdot W_i \leq W_i \quad (6.32)$$

Hence, Equation 6.24 is also satisfied.

In summary, by partitioning the traffic among multiple paths proportionally to $\left(\frac{1}{p_i}\right)^k$ and adjusting the α parameter of multipath flows to $\frac{p_m}{p_{min}}$, we achieve the design goals outlined by Equation 6.23 and Equation 6.24. A practical algorithm to implement our design will be discussed in the next section and evaluated across a wide range of scenarios in Section 6.5.

Before we move on, we would like to make two comments on our multipath forwarding strategy. First, we do not retry NACKed interests over alternative paths. This appears unintuitive since retrial of NACKed interests may improve the performance of the multipath flow. Indeed, if the router retries a NACKed interest on alternative paths and will not send the NACK packet back to the client unless all paths have failed, then the NACK probability of the multipath flow would be

$$p_m = \prod_{i=1}^{n} p_i \quad (6.33)$$

That is, the multipath flow will experience a much lower NACK probability than the single-path flows and will hence grab an unfairly higher throughput (violating Equation 6.23). Hence, we do not retry NACKed interests over alternative paths in our scheme. Second, the throughput fairness properties we derived earlier hold only under homogeneous RTTs. When the RTTs of the competing flows are heterogeneous, our current scheme cannot guarantee fairness. This issue has been studied in the context of multipath TCP and the RTT mismatch can be compensated there [110, 63]. But the same technique is not applicable in NDN since the NDN clients are multipath-unaware and do not know the RTT of each path. [22] has proposed the use of route labels to make multipath explicit to the client. But that is costly and does not agree with the ICN philosophy. Hence, we stick to the current design and will evaluate the impact of RTT

unfairness in Section 6.5.

6.4.2 Practical Algorithm

With the theoretic analysis in the previous section as our guideline, we now elaborate the actual algorithms to implement our multipath forwarding strategy and the α compensation for multipath flows.

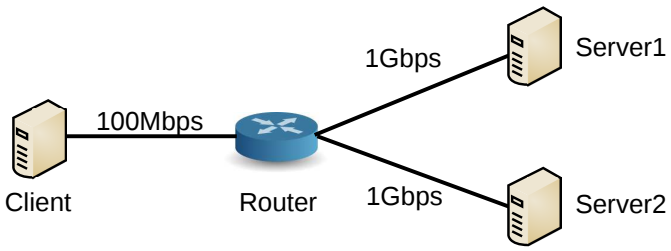
Let us look at the multipath forwarding strategy first: every NDN router has a FIB. Each FIB entry indicates the interface(s)/path(s) that can be used to forward the interests under a specific prefix. For those indivisible prefixes (defined in Section 6.3) that have multiple paths, the router estimates the NACK probability of each path by periodically (every 100ms by default) counting the number of received data packets and NACK packets. Then the router calculates the instantaneous NACK ratio in each period and use an Exponentially Weighted Moving Average (EWMA) filter to calculate the NACK ratio for that path. This smoothed NACK ratio is used as an estimate of the NACK probability p_i of that path and the router partitions the incoming interest packets under this prefix among all paths proportionally to $\left(\frac{1}{p_i}\right)^k$.

Two things need to be noted here. First, the router needs frequent feedback from each path to have an accurate estimation of its congestion level. The relentless congestion control at the client helps achieve this. As discussed earlier, in steady state, the window size of relentless congestion control should be very close to the network capacity (one packet above or below it). Hence, it will generate frequent NACK packets at the bottleneck link (ideally every 3 RTTs [71]). In contrast, AIMD control will halve its window size upon a loss event and will take a much longer time to generate the next NACK. Hence, relentless congestion control is adopted by our scheme as the client reaction algorithm in preference to AIMD. Second, it is possible that the NACK ratio measured in an interval is zero. Obviously, a zero NACK probability poses the "division by zero" problem to the forwarding strategy where the traffic fraction is proportional to $\left(\frac{1}{p_i}\right)^k$. We have mitigated this issue by using the EWMA filter so that a zero NACK ratio sample will only gradually bring down the NACK probability of a path as opposed

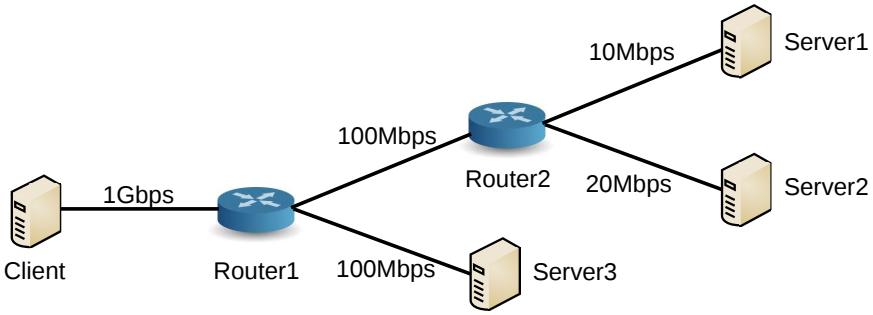
to setting it to zero. However, it is still possible for the smoothed p_i to be zero (e.g., before the first NACK is received or after many rounds of zero NACK ratio observed). Hence, we lower bound p_i to a very small value (10^{-6} by default) to avoid division by zero. The idea is that, if we observe no NACK packets on a path, it implies that we did not probe this path enough (the client window control algorithm will always generate NACKs eventually no matter how much bandwidth there is on that path). A very small p_i means a large traffic fraction will be shifted to this path so that we can probe it better. After that, we should have a good estimation of the congestion levels on all paths. Hence, this lower bound should never be hit in the steady state. It is activated only infrequently in the initial phase or during transient traffic change. We will evaluate the performance of this scheme in practice in Section 6.5.

Let us now move on to the α compensation mechanism for multipath flows. As shown in Equation 6.30, the client needs to know both p_m and p_{min} to calculate the correct α value. p_m is easy to know. The client could use the same mechanism as the router does to measure its own NACK ratio. The difficulty is how to let the client know p_{min}. Since we assume that the client is multipath-unaware, it cannot measure p_{min} by itself. p_{min} needs to be measured by the routers who make multipath forwarding decisions and piggybacked to the client via NACK packets. In the simplified scenario shown in Figure 6.5, this is straightforward. Since the router is already measuring the NACK ratio for each path, it simply finds the minimum NACK ratio among all paths and insert that information into the NACK packets it generates or forwards to the client. However, the case can be much more complicated (Figure 6.6). For instance, there may be multiple routers that are making multipath forwarding decisions. The bottleneck link may also appear before the path split. Hence, a more careful design is required.

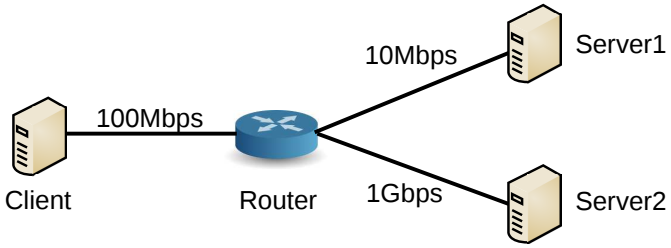
Algorithm 6.1 shows our p_{min} piggybacking algorithm executed by each NDN router before sending out a NACK packet. Note that this NACK could have been generated by this router itself (some local link is congested) or somewhere upstream. Either way, the router first needs to find the locally minimum p_i. This $local\_p_{min}$ may not exist (i.e., $local\_p_{min} = 2$) either because there is only a single path or there are multiple paths but none of them is a bottleneck link



(a) The bottleneck link may appear before the path split.



(b) There may be multiple routers that are making multipath forwarding decisions.



(c) The bottleneck link for each path may be different.

Figure 6.6: Piggybacking the correct p_{min} is non-trivial.

($p_i = 0$ because the bottleneck link of path i is downstream of this router). In these cases, the router simply sends or forwards the NACK packet along without any manipulation. However, if a $local\_p_{min}$ does exist (i.e., $local\_p_{min} \neq 2$), the router needs to piggyback this $local\_p_{min}$ in the NACK packet either when the current NACK packet does not piggyback any p_{min} or when it does, $local\_p_{min} < p_{min}$.

Algorithm 6.2 shows how a client should react upon receiving a NACK packet with p_{min} piggybacking. This is straightforward: if p_{min} exists and $p_{min} < p_m$, then $alpha$ is set to $\frac{p_m}{p_{min}}$.

Otherwise $\alpha = 1$. The case where $p_{min} > p_m$ may appear in the scenario shown in Figure 6.6c. Here, the bottleneck link of the upper path is the 10Mbps link while the bottleneck link of the lower path is the 100Mbps link. From the routers point of view, it should never receive any NACK on the lower path and will hence piggyback the NACK ratio of the upper path back to the client. But the client may experience an even lower overall NACK ratio than the upper path. Hence, we should not do any α compensation in this case.

Algorithm 6.1 p_{min} piggybacking by the router

Before sending out a NACK packet:
$local\_p_{min} \leftarrow 2$
if number of paths for this prefix > 1 **then**
  for each path i **do**
    if $p_i \neq 0$ **and** $p_i < local\_p_{min}$ **then**
      $local\_p_{min} \leftarrow p_i$
    end if
  end for
end if
if $local\_p_{min} \neq 2$ **then**
  extract p_{min} from the NACK packet
  if p_{min} does not exist **then**
    insert $local\_p_{min}$ into the NACK packet
  else if $local\_p_{min} < p_{min}$ **then**
    replace p_{min} with $local\_p_{min}$ in the NACK packet
  end if
end if

Algorithm 6.2 α compensation by the client

Upon receiving a NACK packet:
extract p_{min} from the NACK packet
if p_{min} does not exist **or** $p_{min} > p_m$ **then**
  $\alpha \leftarrow 1$
else
  $\alpha \leftarrow \frac{p_m}{p_{min}}$
end if

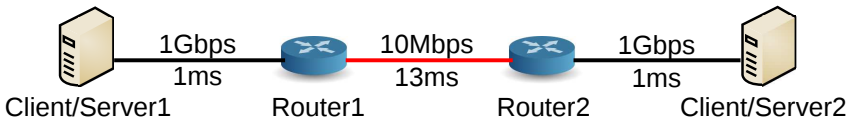


Figure 6.7: The baseline chain topology (we vary the parameters of the link in red)

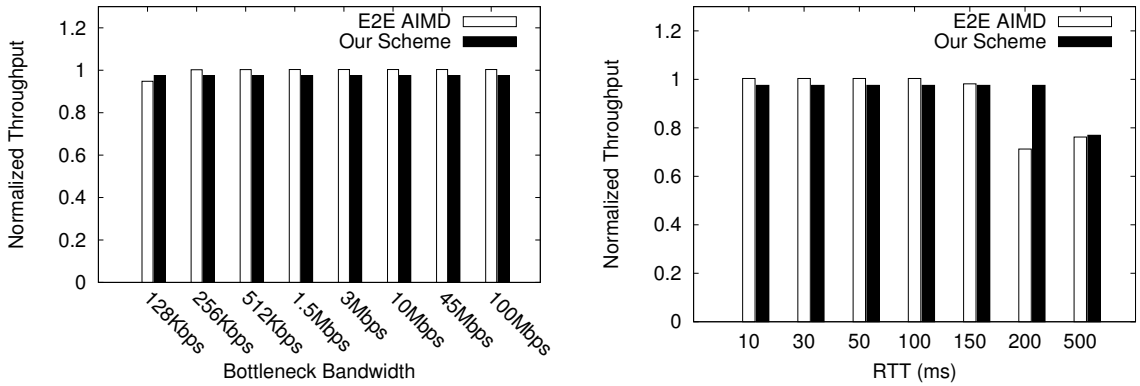
6.5 Performance Evaluation

We have implemented all our previous proposals in ndnSIM [8] and evaluated their performance in various scenarios. Unless otherwise noted, the data packet size is fixed to 1KB and no in-network caching is in use. The buffer size (for both L2 queue and interest shaper queue) is set to the bandwidth-delay product (BDP) of the network unless the BDP is too small (buffer size is set to 40 packets if BDP $<$ 40 packets). Each flow starts at a random time uniformly distributed between 0s and 5s. All simulations last for 70s and metric measurements start after 10s to eliminate the initial warm-up period. Every test case has been repeated 12 times and 95% confidence intervals are calculated.

6.5.1 Performance of the Hop-by-hop Interest Shaper

We first conduct a unit test of our hop-by-hop interest shaper and compare it against a pure end-to-end AIMD scheme (E2E AIMD) under the baseline chain topology shown in Figure 6.7. In the pure end-to-end scheme, the routers do not impose any interest shaping or generate any explicit congestion signal. Instead, the clients infer congestion via a retransmission timeout mechanism similar to TCP and use AIMD window control to avoid congestion. Relentless congestion control is not used along with the end-to-end scheme because there is no NACK packets to indicate the number of lost interests and unlike TCP you cannot infer that number via sequence numbers.

We start with the simplest one-way traffic scenario where one client is issuing interests to the server on the other side of the network and data packets flow in the reverse direction to satisfy the interests. We vary the bandwidth and latency of the bottleneck link (the link in



(a) With varied bottleneck bandwidth

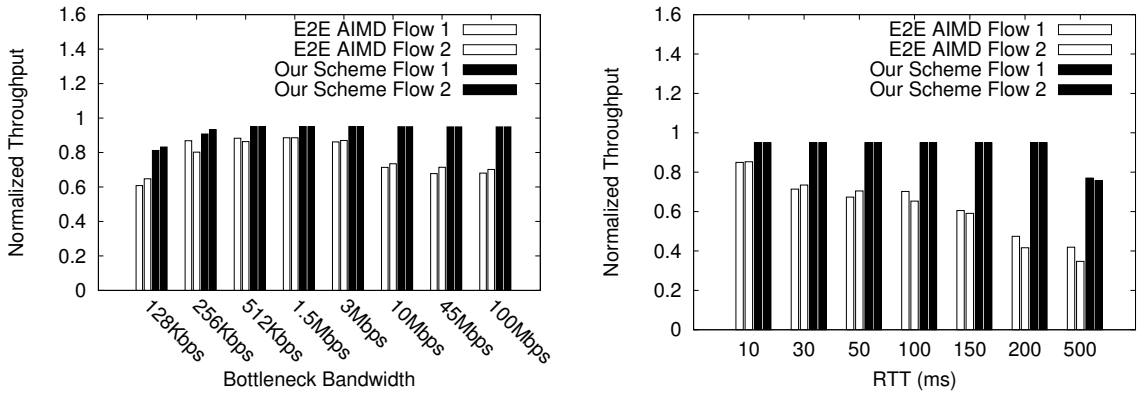
(b) With varied RTT

Figure 6.8: Throughput performance under one-way traffic

red in Figure 6.7) in a wide range to study their impacts on the performance. Figure 6.8a and Figure 6.8b show the throughput performance results (normalized over the bottleneck bandwidth).

As can be seen from Figure 6.8a, even though we vary the bottleneck bandwidth from 128Kbps to 100Mbps (while fixing the RTT to 30ms), both the E2E AIMD scheme and our scheme are always able to fully utilize the link under one-way traffic. In fact, the normalized throughput of our scheme is actually a little bit lower than that of E2E AIMD. That is because we always leave 3% of headroom in our interest shapers to accommodate traffic burstiness. Hence, our scheme will get around 97% utilization under one-way traffic while the E2E AIMD scheme can achieve full 100%.

Figure 6.8b shows the performance when we vary the bottleneck link latency so that the RTT ranges from 10ms to 500ms while the bottleneck bandwidth is fixed to 10Mbps. As the figure shows, both schemes suffer from throughput degradation under long RTT. This scalability problem of window-based control algorithms is well-known [39] and we do not plan to tackle this side issue in this dissertation. However, we can see from the figure that our scheme still achieves near-optimal throughput at 200ms RTT while E2E AIMD drops to around 70% utilization. This confirms our analysis in Section 6.3 that relentless congestion control has better scalability than



(a) With varied bottleneck bandwidth

(b) With varied RTT

Figure 6.9: Throughput performance under two-way traffic

AIMD.

Let us now consider the two-way traffic scenario. Here the two clients are simultaneously requesting for contents on the other end of the network and competing for the bottleneck link bandwidth in the middle. We vary the bottleneck bandwidth and RTT in the same way as before and show the results in Figure 6.9a and Figure 6.9b. As the figures show, our scheme manages to achieve near-optimal throughput in most scenarios while E2E AIMD clearly suffers from throughput degradation. This demonstrates the necessity to consider the interdependence between interest and data packets in reverse directions and adopt the hop-by-hop interest shaping algorithm we propose. We can also see that our scheme has also maintained good fairness between the two flows in opposite directions. Finally, we have verified that across all the test cases, our hop-by-hop interest shaper effectively prevents congestion so that the data packet loss rate (due to L2 queue overflow) is always zero.

Let us now try three test cases to further evaluate our interest shaping algorithm under special settings. The first test case randomizes the data packet size to be uniformly distributed between 600B and 1400B instead of a fixed 1KB size. This causes the size ratios r_1 and r_2 in our interest shaper to fluctuate over time. We would like to see whether the proposed shaping algorithm can still perform well under such fluctuations. The second test case sets the data

Table 6.2: Throughput performance under two-way traffic with special settings

	E2E AIMD		Our Scheme	
	Flow 1	Flow 2	Flow 1	Flow 2
Random Pkt Size	7.089 ± 0.090 Mbps	7.081 ± 0.068 Mbps	9.394 ± 0.005 Mbps	9.393 ± 0.004 Mbps
Asym. Size Ratio	8.853 ± 0.037 Mbps	5.814 ± 0.118 Mbps	9.249 ± 0.002 Mbps	9.760 ± 0.001 Mbps
Asym. Bandwidth	3.004 ± 0.222 Mbps	0.900 ± 0.010 Mbps	9.735 ± 0.001 Mbps	0.711 ± 0.001 Mbps

packet size at one server to be 500B while leaving the other at 1KB. This creates an asymmetric data/interest packet size ratio. Finally, the third test case sets the bandwidth of the bottleneck link in one direction to 1Mbps while the reverse direction is left at 10Mbps. This evaluates an asymmetric link bandwidth scenario common in today's residential networks and tests whether our shaping algorithm could achieve proportional fairness as predicted.

Table 6.2 shows the simulation results out of these three special cases. As we can see, even under randomized data packet size, our scheme still manages to achieve around 9.4Mbps in both directions. This is close to the optimal data throughput we may obtain in such cases. Beware that the optimal data throughput under two-way traffic is not 10Mbps since we need to leave some room in both directions for the interest packets. In contrast, E2E AIMD only gets around 7Mbps of throughput in both directions.

As for the asymmetric size ratio scenario, under our scheme, some slight throughput difference is observed between Flow 1 and Flow 2. This is expected since a smaller data packet size requires a higher interest rate to achieve the same data throughput. Hence, more bandwidth is consumed by interest packets in one direction and our hop-by-hop shaping algorithm has correctly adapted to it. This is not the case under E2E AIMD where one flow gets around 8.8Mbps while the other flow gets only 5.8Mbps.

Finally, the asymmetric link bandwidth scenario is the most interesting case. Here, the data throughput of Flow 2 under our scheme is 0.711Mbps out of 1Mbps, leaving the rest of the bandwidth to interest packets in the reverse direction to fetch 9.735Mbps of data. This is close to the theoretically optimal throughput under proportional fairness and highlights the importance of our formulation in Section 6.2. If no such interest shaping is conducted (as in

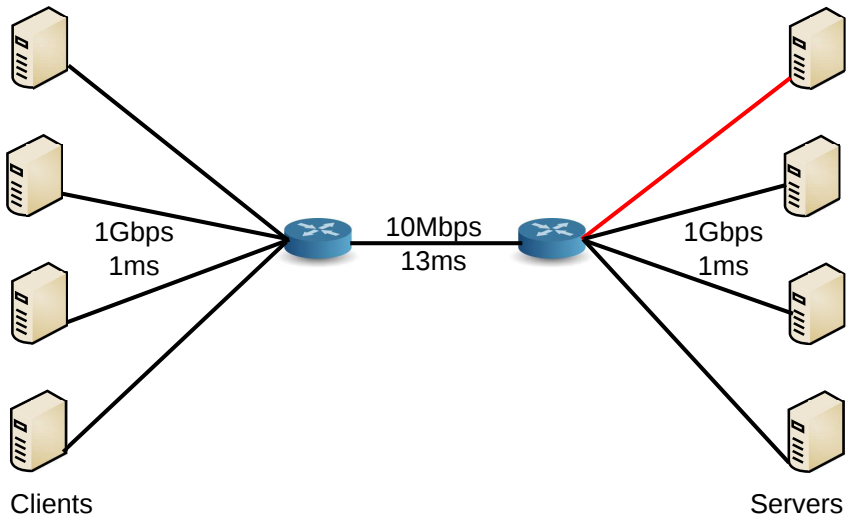


Figure 6.10: The dumbbell topology (we vary the parameters of the link in red)

E2E AIMD scheme), we could see that Flow 2 will try to grab as much data throughput as it can. Although it does get a higher throughput of 0.9Mbps (compared to 0.711Mbps in our scheme), it leaves little room for the interest packets of Flow 1. Hence, Flow 1 gets only 3Mbps of data throughput as compared to the 9.735Mbps in our scheme.

In summary, our hop-by-hop interest shaper has effectively controlled data congestion, achieved near-optimal data throughput and proportional fairness over the baseline chain topology under various bandwidth, RTT and traffic settings.

6.5.2 Performance under Single Path

Now let us test our congestion control scheme in broader scenarios with different topologies. For all the scenarios in this section, we consider single-path forwarding only. That is, the contents under a prefix reside on a single server and all clients fetch from it via the shortest path. The multipath case where contents are multihomed or clients can reach the server from multiple paths (not only the shortest path) will be studied in the next section.

First, we evaluate the fairness properties of our congestion control scheme in the dumbbell topology shown in Figure 6.10. In this scenario, two clients on the left side of the bottleneck link

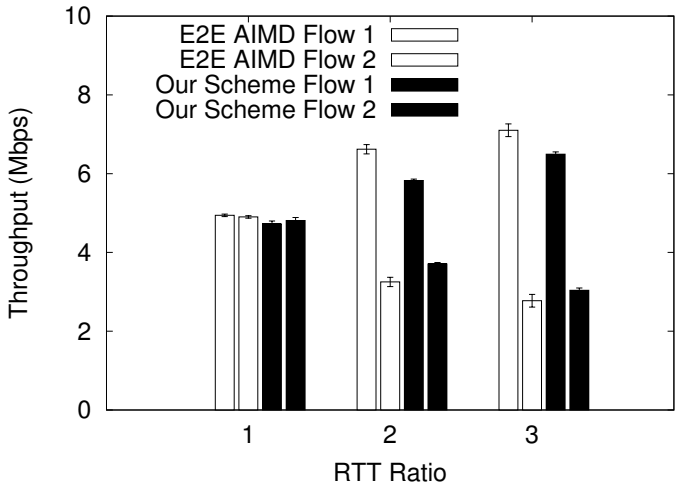


Figure 6.11: Throughput performance under dumbbell topology with various RTT ratios

request contents from two servers on the right (Flow 1 and Flow 2). The bottleneck link has a fixed bandwidth of 10Mbps and 13ms latency. All other links are 1Gbps and have 1ms latency except the link in red. For that link, we set its latency to 1ms, 16ms and 31ms respectively. Hence, the RTT of Flow 1 is fixed at 30ms while the RTT of Flow 2 varies from 30ms, 60ms to 90ms. This evaluates the fairness property of our proposal under various RTT ratios (1:1, 1:2 and 1:3). We also add some background traffic in both forward and backward directions on the remaining clients and servers. The background flows are Poisson with very low arrival rates so that they consume little bottleneck bandwidth but add randomness to the simulation to eliminate phase effects [41, 42].

Figure 6.11 shows the simulation results. As we can see, under homogeneous RTT (i.e., RTT ratio of 1), both our scheme and E2E AIMD achieve fair and full utilization of the bottleneck link. As we increase the RTT of Flow 2, we observe that the client with shorter RTT obtains obvious throughput advantage. Such RTT unfairness is well-known in TCP and it is unclear if we want perfect fairness between flows of different RTT. Hence, we do not try to solve this issue in our scheme. However, we point out that the degree of RTT unfairness in our scheme is less than that of E2E AIMD. We believe that this is related to our hop-by-hop interest shaping.

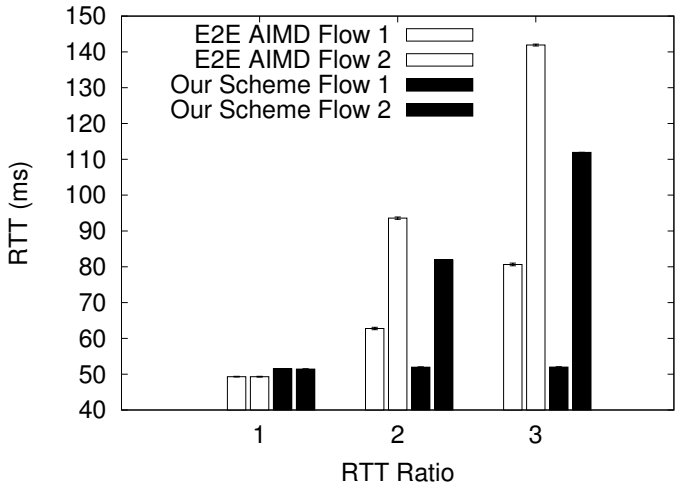


Figure 6.12: RTT performance under dumbbell topology with various RTT ratios

NACKs generated by interest shapers at intermediate routers do not experience as much RTT difference as data packets would. Hence, the RTT unfairness problem will be alleviated although not completely eliminated.

Figure 6.12 shows the RTT performance of the same tests. We observe that PIE perfectly controls the queuing delay to the 20ms target value. Hence, under our scheme, the RTTs experienced by Flow 2 in three cases are around 50ms, 80ms and 110ms (i.e., 20ms plus the raw RTT of 30ms, 60ms and 90ms) and the RTT of Flow 1 always stays at around 50ms. In contrast, the RTT under E2E AIMD scheme is up to the buffer provisioning (since DropTail queue is used) and may be much longer.

We have also experimented with a slight variant of the dumbbell topology. In this test, we change the bandwidth of the middle link to 100Mbps while setting the link in red to 10Mbps. This creates an interesting scenario where the bottleneck link of Flow 1 is the link in the middle while the bottleneck link of Flow 2 is the link in red. Our simulation results show that the proposed congestion control scheme handle this slightly more complicated scenario well. Flow 1 achieves 87.447Mbps of throughput while Flow 2 gets 9.754Mbps. Hence, Flow 2 has correctly adapts to the bottleneck bandwidth of 10Mbps of the red link, leaving the rest 90Mbps of the

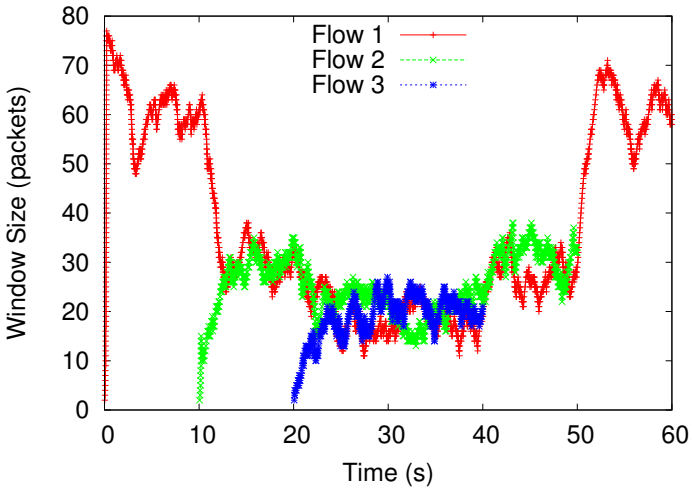
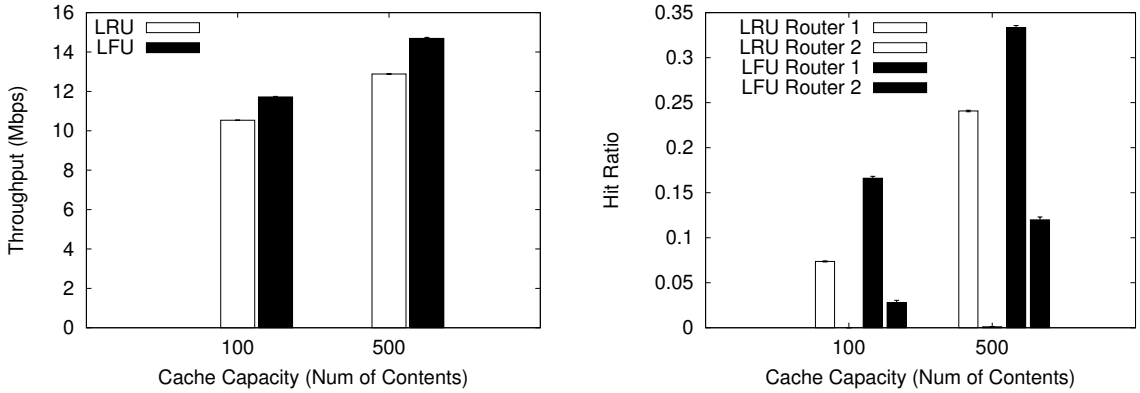


Figure 6.13: Window evolution as flows arrive and depart at different time

middle link to Flow 1.

The previous results only show the average throughput obtained by each flow. We would like to see how the clients adjust their windows over time and how fast they can converge to a fair sharing of the bottleneck bandwidth. Hence, we have designed the following test: in the same dumbbell topology as Figure 6.10, we have three flows that arrive and depart at different time. Flow 1 starts at 0s and ends at 60s. Flow 2 starts at 10s and ends at 50s. Flow 3 starts at 20s and ends at 40s. Figure 6.13 shows how the window size of each flow evolves over time. As can be seen from the figure, our congestion control scheme is quite responsive and the convergence speed is fast.

All previous test cases assume no in-network caching of the contents. As argued in Chapter 4, pervasive in-network cache deployment may not be a good idea for ICN. However, it does not mean that we should not do any caching at all. In fact, proper cache provisioning at strategic points within the network has been proven very effective in the current Internet. Hence, we still want to evaluate the potential impact of in-network caching on our congestion control scheme, although we believe it is a second-order effect to the efficiency and fairness properties of our scheme.



(a) Throughput (b) Hit Ratio

Figure 6.14: Performance under chain topology with varied cache capacity and replacement policy

In all the following test cases of in-network caching, we assume that 10,000 contents with Zipf-0.75 popularity distribution reside on the server. Clients fetch the contents from the server along the shortest path and cached contents are utilized opportunistically along the way. Each router makes independent cache replacement decisions (using LRU or LFU policy) without any coordination. The cache capacity of each router is varied to study its impact on the cache performance.

We first study a simple case under the baseline chain topology in Figure 6.7. Here Client 1 is fetching contents from Server 2 and the two intermediate routers provide in-network caching to improve the performance. Figure 6.14a shows the throughput obtained by the client under different cache capacities and replacement policies. As expected, with the help of caching, the client is able to get a throughput that is more than the bottleneck link bandwidth of 10Mbps and this improvement grows with the cache capacity. With regard to the comparison between LRU and LFU replacement, LFU is clearly better than LRU. This agrees with our previous results in Chapter 4 and can be better understood via Figure 6.14b. As we can see here, the hit ratio of Router 2 under LRU policy is almost zero no matter how large the cache space is. This is due to the cache redundancy issue discussed in Figure 4.7. Under LRU replacement, Router

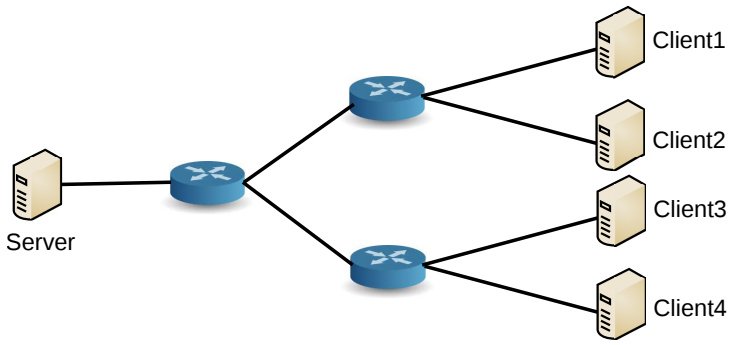
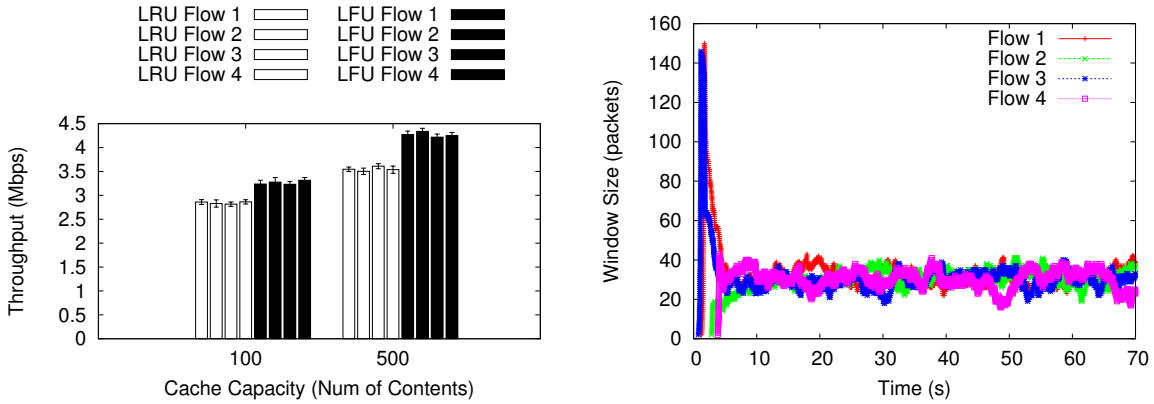


Figure 6.15: The tree topology

2 almost caches the same contents as Router 2 and add little help to the cache performance. In contrast, perfect LFU replacement improves cache diversity and has better hit ratio on Router 2 than LRU. Anyhow, the in-network caching improves the throughput performance as expected but does not confuse our congestion control scheme. It might have introduced some random noise to our window control algorithm due to the unpredictability of cache hit/miss. But our scheme still manages to converge to the steady state.

Let us now examine a more complicated scenario for caching. As shown in Figure 6.15, we have a tree topology where four clients are fetching contents from the same server. All links have 10Mbps of bandwidth and 10ms of latency. Due to the traffic aggregation in this topology, in-network caching at the intermediate routers should be very effective. However, it also casts questions on whether our congestion control scheme can still converge to a fair steady state in this scenario. Figure 6.16 gives the answer.

As can be seen from Figure 6.16a, the four flows get roughly the same throughput despite of the caches. As before, larger cache capacity achieves better performance and LFU always outperforms LRU. Let us take a deeper look into the best case where the cache capacity is 500 contents and perfect LFU is used as the replacement policy. Under this setup, each flow get around 4.2Mbps of throughput which sum up to 16.8Mbps, way beyond the 10Mbps bandwidth of the server's access link. The window sizes of the four flows are shown in Figure 6.16b. As the figure shows, although in-network caching does incur some fluctuation in the window sizes, they still converge to a relatively stable state where the bottleneck bandwidth is fairly shared



(a) Throughput (b) Window

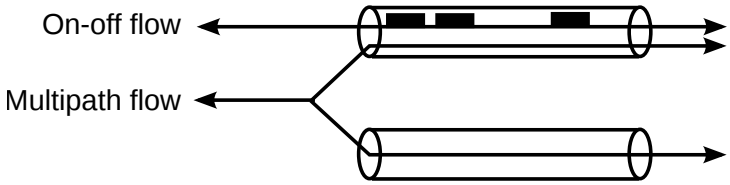
Figure 6.16: Performance under tree topology with varied cache capacity and replacement policy

among the competing flows.

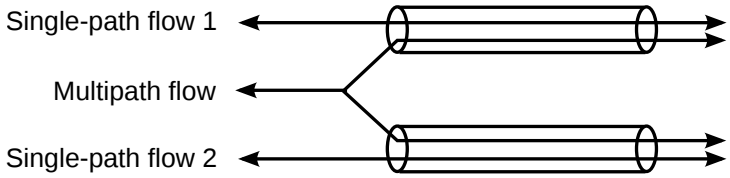
6.5.3 Performance under Multipath

Let us know evaluate our congestion control scheme in the presence of multipath forwarding. As discussed in Section 6.4, our multipath forwarding strategy partitions the traffic among all paths proportionally to $\left(\frac{1}{p_i}\right)^k$ where p_i is the NACK probability of path i. k is a parameter that tunes the trade-off between optimal resource pooling and responsiveness. Hence, let us first study how k affects the responsiveness and resource pooling properties of a multipath forwarding strategy and find a value that achieves a reasonable trade-off between the two.

We have designed two test cases as shown in Figure 6.17 to evaluate the two aspects of the problem respectively. The scenario in Figure 6.17a tests the responsiveness of the multipath forwarding strategy. Here the multipath flow has two paths, each traversing a 10Mbps bottleneck link. On the upper bottleneck link, there is an on-off flow that is competing with the multipath flow. The on-off flow issues interest packets in batches. The inter-arrival time between two batches is uniformly distributed between 0ms and 200ms. The number of interests issued in each batch is uniformly distributed between 0 and 40. This on-off flow evaluates how the multipath



(a) Responsiveness



(b) Balance congestion

Figure 6.17: Two scenarios to evaluate the responsiveness and resource pooling properties of a multipath forwarding strategy (all flows have the same RTT)

flow responds to bursty traffic and whether it is able to quickly pick up the varying available bandwidth in the upper bottleneck link.

The scenario in Figure 6.17b tests the multipath forwarding strategy's ability to balance congestion between the two bottleneck links and achieve optimal resource pooling. Here we have three flows: a multipath flow that traverses both bottleneck links and two single-path flows each competing on one bottleneck link. The bandwidth of the upper bottleneck link is fixed at 10Mbps while the bandwidth of the lower bottleneck link is varied from 10Mbps to 20Mbps, 40Mbps and 80Mbps. When both bottleneck links have 10Mbps of bandwidth, the multipath forwarding strategy should evenly distribute the traffic between the two paths and all three flows should achieve around 6.67Mbps of throughput ($20/3 \approx 6.67$). However, as the bandwidth of the lower bottleneck link increases, the multipath forwarding strategy should shift more traffic to the lower path to balance the congestion and achieve resource pooling. For instance, when the lower bottleneck link has 20Mbps, ideally the multipath flow should only use the lower path. Under this optimal resource pooling, each flow will get 10Mbps of throughput and achieve max-min fairness. However, in practice the multipath flow has to send a small fraction of the traffic to the upper path to constantly probe its congestion level and k determines how much probe

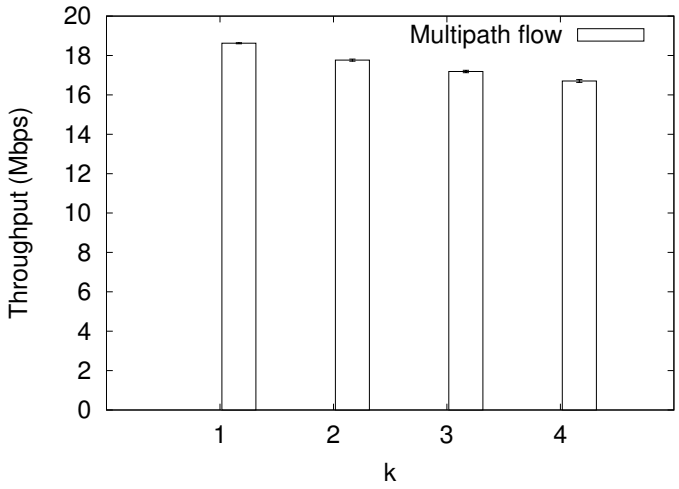
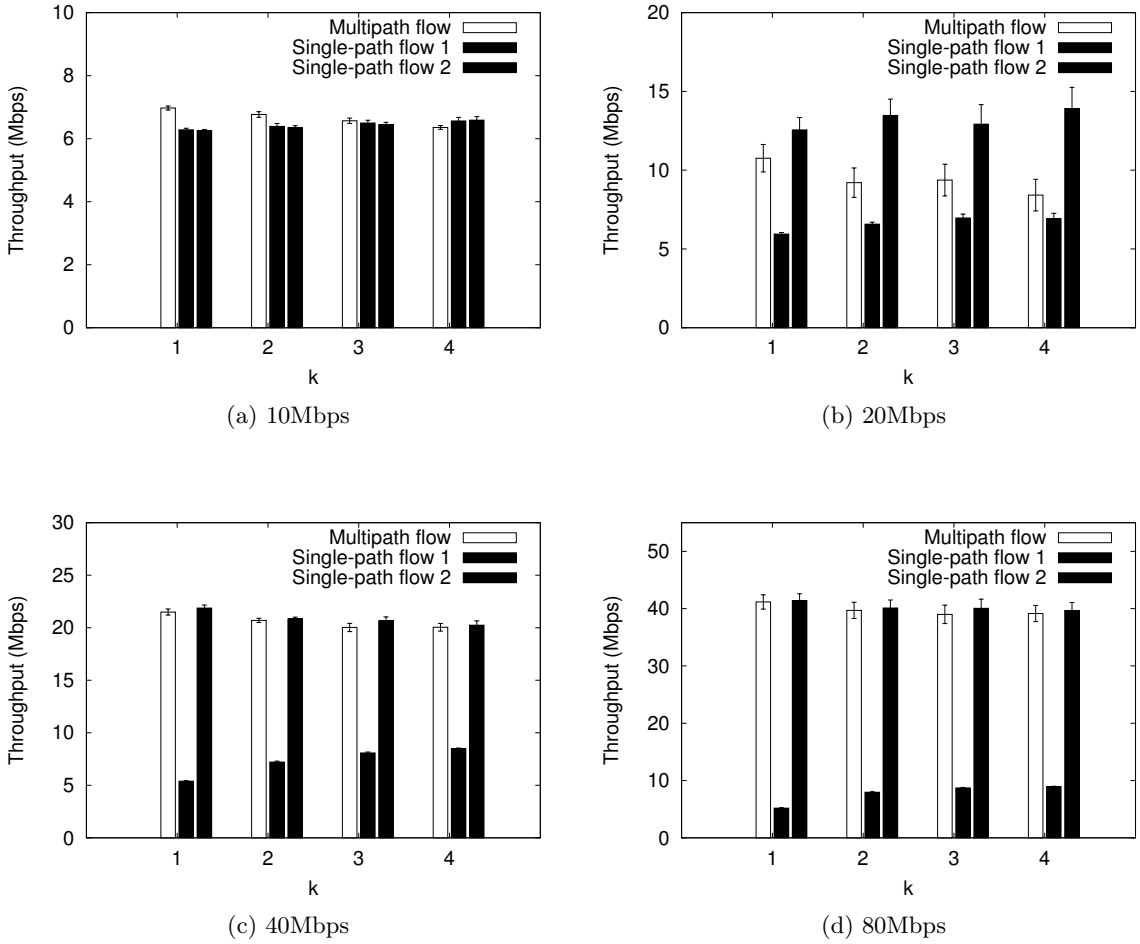


Figure 6.18: Throughput performance when competing with an on-off flow under different k settings

traffic should be left on the more congested path. In theory, the forwarding strategy with a smaller k would leave more probe traffic and hence is more responsive. A larger k value would be closer to optimal congestion balance and resource pooling.

We have evaluated our congestion control scheme in the two scenarios mentioned above under different k settings ($k = 1, 2, 3, 4$). Let us first look at the responsiveness results shown in Figure 6.18. As expected, with the increase of parameter k, the multipath flow becomes less responsive and hence achieves less throughput when competing with the on-off flow.

Now let us look at the results out of the resource pooling test. Figure 6.19 shows the performance of our congestion control scheme with varied bandwidth on the lower bottleneck link (10Mbps, 20Mbps, 40Mbps and 80Mbps) under different k settings. Observe the following phenomena: 1) when both bottleneck links have the same bandwidth of 10Mbps (Figure 6.19a), the three flows achieve roughly the same throughput in spite of the k value. 2) As the bandwidth of the lower bottleneck link increases, unfairness starts to appear. This is expected since single-path flow 1 can only utilize the 10Mbps upper bottleneck link while single-path flow 2 can utilize the lower bottleneck link who has more bandwidth. The multipath flow is suppose to balance such unfairness and k tunes the degree of such balancing effect. As we can see, a larger



(a) 10Mbps (b) 20Mbps

(c) 40Mbps (d) 80Mbps

Figure 6.19: Throughput performance with varied bandwidth on the lower bottleneck link under different k settings

k means higher throughput for single-path flow 1 and hence better resource pooling. But we can never achieve the optimal case where single-path flow 1 gets all 10Mbps of the upper bottleneck link while the multipath flow and single-path flow 2 share the lower bottleneck link unless k approaches infinity [63]. 3) The throughput of the multipath flow is usually on par with the best single-path flow. That means our p_{min} piggybacking and α compensation mechanism has taken effect. 4) Despite the inflated α, the multipath flow takes no more capacity from the upper bottleneck link than a single-path flow would. Hence, we see that single-path flow 1 always

gets more than 5Mbps of throughput in any scenario. Thus, our multipath congestion control scheme has satisfied the design goal of "do no harm".

In summary, the value of k tunes the trade-off between optimal resource pooling and responsiveness. A larger k means better balancing of congestion and hence better fairness. A smaller k means more responsiveness and hence better throughput. Based on our previous experiment results, we set the default value of k to 2 in our scheme and use it in all remaining test cases.

Now that we have picked the value of k and demonstrated that our scheme does no harm under homogeneous RTT, we examine the other two design goals of our multipath congestion control scheme, i.e., the ability to grab bandwidth from multiple paths in the absence of competing flows and the ability to balance congestion. We will also evaluate the performance of our scheme under heterogeneous RTT cases.

Let us first evaluate our scheme's ability to grab bandwidth from multiple paths with varied capacities and RTTs. For this test, we use the same scenario as Figure 6.17b but remove the two single-path flows. Hence, we can evaluate whether the multipath flow is able to fully utilize both bottleneck links in the absence of competition. The upper bottleneck link has a fixed bandwidth of 10Mbps and a fixed latency of 10ms. However, the bandwidth of the lower bottleneck link is again varied from 10Mbps to 20Mbps, 40Mbps and 80Mbps. We further vary its latency from 5ms to 10ms, 20ms, 40ms. Figure 6.20 shows the normalized throughput of the multipath flow (normalized by the total available bandwidth, i.e., the sum of the bottleneck bandwidth of the upper link and the lower link). As can be seen from the figure, the normalized throughput is around 90% across a wide range of bandwidth and latency settings. We can further improve the throughput by retrying NACKed interests over alternative paths. However, as discussed in Section 6.4, that would incur hard-to-solve fairness issues in the presence of competing flows. Hence, we stick to our current design and believe that its ability to grab bandwidth is acceptable.

Now let us look at an example of how the multipath flow help balance the congestion among multiple paths and achieve resource pooling. We again use the set up in Figure 6.17b. In this test case, we begin with two single-path flows only, each traversing a bottleneck link. After they enter

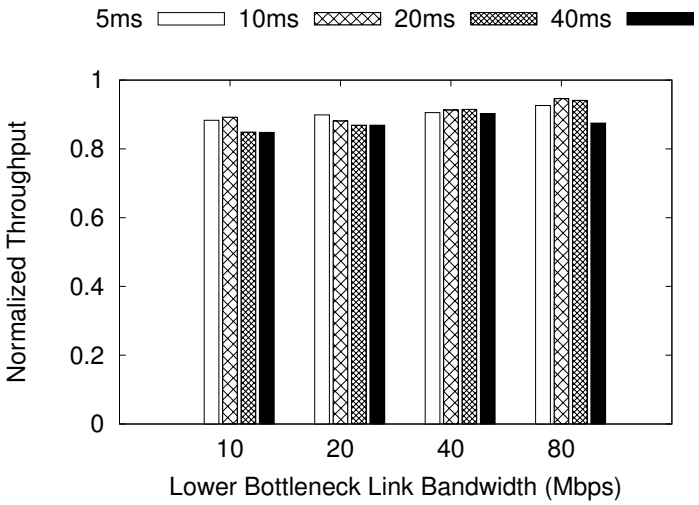


Figure 6.20: Throughput performance in the absence of competing flows with varied bandwidth and latency on the lower bottleneck link

into a steady state, we start a multipath flow that traverses both bottleneck links and observe its impact on the two single-path flows. The upper bottleneck link has 12Mbps of bandwidth and the lower bottleneck link has 18Mbps. Figure 6.21 shows the instantaneous throughput of the three flows over time. As can be seen here, initially both single-path flows fully utilize their bottleneck links, achieving close to 12Mbps and 18Mbps of throughput respectively. At around 30s, the multipath flow joins. Ideally, the multipath flow should perfectly balance the congestion between the two bottleneck links and each flow should get 10Mbps of throughput (the multipath flow should take 2Mbps out of the upper link and 8Mbps out of the lower link). This optimal resource pooling is not achieved. However, the throughput of the two single-path flows are still made closer to each other (around 8Mbps and 11Mbps). That means the multipath flow has taken around 4Mbps from the more congested upper link and 7Mbps from the less congested lower link. By sending more traffic to the less congested path, the multipath forwarding strategy does try to balance the congestion, although it is not perfectly accomplished. This is for the sake of responsiveness as we have discussed before.

Finally, we study the RTT fairness property of our multipath congestion control scheme. As shown in Section 6.4, our proposal has been proved to achieve all the fairness goals under

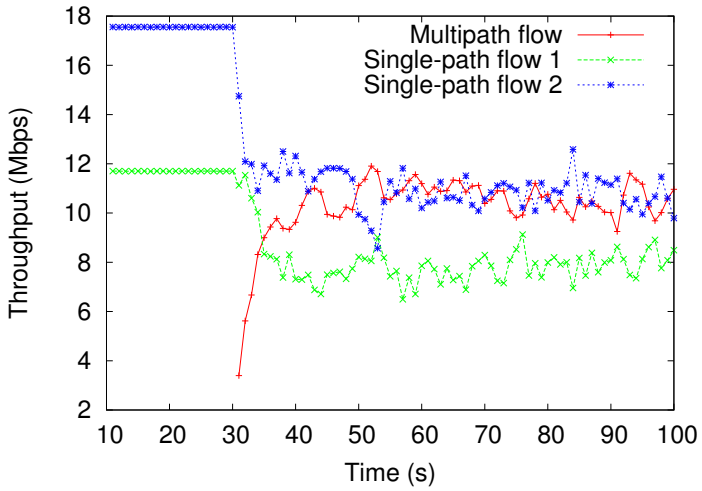
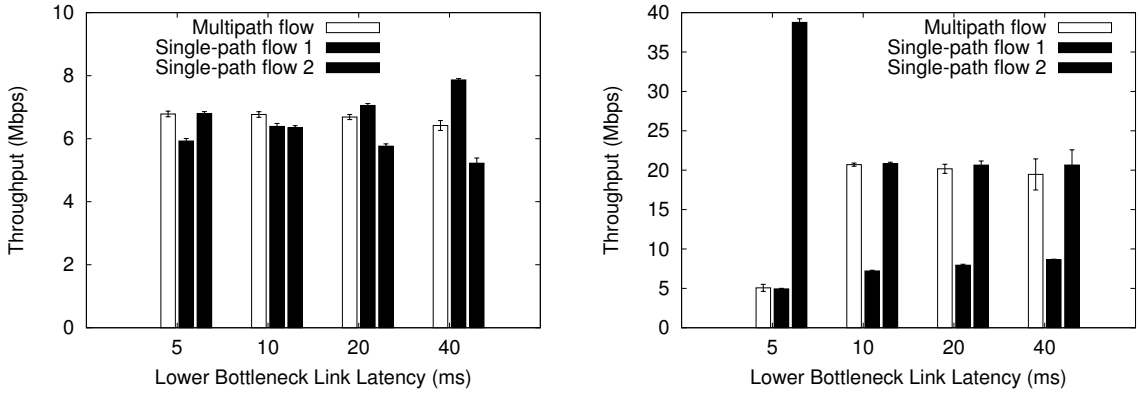


Figure 6.21: The addition of a multipath flow helps balance the congestion between two bottleneck links.

homogeneous RTT. However, we could not provide such theoretic guarantee when the RTTs of the multiple paths are heterogeneous. In this test case, we evaluate the potential impact of RTT heterogeneity on the performance of our scheme via simulation. Figure 6.22a shows the baseline scenario. Here, three flows compete for two bottleneck links as depicted in Figure 6.17b. Both bottleneck links has 10Mbps of bandwidth. We fix the latency of the upper link to 10ms while varying the latency of the lower link from 5ms to 10ms, 20ms and 40ms. As can be seen from the Figure, under homogeneous RTT (the 10ms case), the three flows have achieved fair sharing of the bottleneck links, each getting around 6.67Mbps of throughput. Now if we increase the latency of the lower bottleneck link, both the multipath flow and single-path flow 2 will be affected since they traverse that link. As we all know, higher RTT means lower throughput. Hence we can see that both flows achieve lower throughput while single-path flow 1 gets more throughput. On the other hand, if the latency of the lower link is decreased (the 5ms case), single-path flow 1 will suffer from throughput degradation while the other two flows gain ground. But despite of the high heterogeneity of the latencies (ranging from 1:2 to 4:1), the degree of RTT unfairness appears acceptable to us in this test case.

We now look at a more extreme case. In this scenario, the upper bottleneck link has 10Mbps



(a) 10Mbps (b) 40Mbps

Figure 6.22: Throughput performance with varied bandwidth and latency on the lower bottleneck link

of bandwidth while the lower one has 40Mbps. The latency settings are the same as before: we fix the upper link at 10ms while varying the lower link from 5ms to 10ms, 20ms and 40ms. Figure 6.22b shows the results. Under homogeneous RTT (the 10ms case), the multipath flow and single-path flow 2 both get around 20Mbps of throughput while single-path flow 1 gets roughly 7.2Mbps. This is close to the optimal fair sharing of the bandwidth because the three flows can never achieve the same throughput due to the inherent limitation of single-path flow 1 (it cannot utilize the 40Mbps lower link). When the latency of the bottleneck link with more bandwidth (the lower one) increases, it actually helps in terms of fairness. We can see that the throughput of single-path flow 1 slight increases in this case. The worst case is when the bottleneck link with more bandwidth also has lower latency (the 5ms case). In this case, single-path flow 2 will get an unfairly high throughput than the others and the multipath flow is unable to offset it. Note that this is not necessarily the fault of our multipath forwarding strategy. Here, single-path flow 2 will get a much higher throughput than single-path flow 1 even if the multipath flow does not exist. The throughput difference is due to the fact that single-path flow 2 traverses a bottleneck link with higher bandwidth and lower latency. The addition of a multipath flow is supposed to balance it and our scheme does try to do that. But it only

compensates for the difference in NACK probability and does not compensate for the RTT heterogeneity due to the technical difficulties discussed in Section 6.4. Since RTT unfairness has existed in the current Internet for a long time and whether flows with different RTTs should get the same throughput is in fact a controversial topic by itself, we believe it is not a deal breaker for our congestion control proposal. Our scheme does not exacerbate the problem. It just does not fully compensate for it as some multipath TCP proposals did [110, 63].

6.6 Concluding Remarks

In this chapter, we presented a complete congestion control scheme for NDN, built around a) an optimal and proportionally fair hop-by-hop interest shaper, b) use of NACKs for explicit congestion notification, c) a window-based client reaction algorithm similar to relentless congestion control, and d) a multipath forwarding strategy with p_{min} piggybacking and α compensation. Through extensive simulation, we demonstrated that our scheme can efficiently utilize and fairly share the bandwidth among multiple flows under two-way traffic over a variety of topologies and load scenarios. We also showed that our multipath congestion control schemes is able to quick grab bandwidth from multiple paths in the absence of competing flows, balance congestion among multiple bottleneck links while maintaining good responsiveness, and do no harm to competing single-path flows.

Chapter 7

Conclusion

7.1 Summary

ICN is a promising future Internet architecture that can support large-scale content distribution more efficiently. However, to engineer it into a practically working system, many pieces are still missing. In this dissertation, we fill in three most important missing pieces in the picture, namely caching, routing and congestion control, and make ICN a more viable architecture for the future Internet.

ICN changes caching by making the network primitive location-independent and hence greatly simplifies cache operations. Caching is such an integral part of ICN architectures that many people automatically assumed that caches should be deployed on every ICN router. In this dissertation, we re-examined this assumption via both theoretic analysis and large-scale simulation. The theoretic analysis showed that the optimal cache deployment is highly uneven and far from pervasive. The simulations showed that pervasive caching would bring limited improvement over edge caching even if nearest replica routing is used. Hence, we rectified an important misconception in ICN: ICN does not imply the necessity of pervasive caching and we need to re-evaluate the cache deployment strategy in ICN.

ICN routing is different from IP routing since we cannot rely on hierarchical aggregation any

more. Our proposed Bloom filter-based routing protocol has been shown via simulation that it can reduce the routing stretch at minimum cost if it is used as a complement to an existing ICN routing protocol. We further analyzed its scalability as a standalone protocol at Internet scale with back-of-the-envelop calculation. Despite the technical challenges we may face today, we believe that it is a valuable direction to pursue in future.

Finally, congestion control in ICN is completely different from that in TCP/IP due to ICN's pull-based model and inherent support for multipoint retrieval. Traditional TCP congestion control mechanisms are either inapplicable or may incur serious performance issues. We proposed a complete solution to ICN congestion control by incorporating proactive hop-by-hop interest shaping, timely and reliable congestion signaling, proper client reaction, as well as a multipath forwarding strategy. Through extensive simulation, we have demonstrated the effectiveness of our scheme in both single path and multipath scenarios.

7.2 Future Work

We hereby point out a few possible directions to further extend the current work:

We have demonstrated the sub-optimality of pervasive in-network caching in Chapter 4 and have also proposed an optimal cache deployment strategy. However, due to its computation complexity and the lack of required input information in practical settings, we do not expect it to be used by real network operators for cache deployment planning. An efficient and practical cache deployment strategy for ICN is highly desired. Furthermore, the optimal cache allocation plan may change over time due to changes in client population or content popularity. How frequent such re-planning should take place and how we can deploy such changes in the most efficient manner in practice is still an open question.

The effectiveness of Bloom filter-assisted ICN routing has already been proven in Chapter 5. However, extending it to Internet scale as a standalone routing protocol is really pushing the limits of today's technology. To demonstrate it in practice requires costly high-end hardware as well as considerable implementation efforts. We are unable to carry this out by ourselves in

an academic environment but would be really excited if someone with such resources could try this idea out in real routers.

As discussed in Chapter 6, one of the most difficult task in ICN congestion control design is the definition of flows and fairness. Our current definition is just one out of many possible definitions. We do not claim it is optimal but believe that it is the most natural definition derived from our previous experiences in TCP/IP. We realize that there are some limitations to our current definition and would like to seek a novel definition that better fits ICN. We have also assumed that the clients are always cooperative in our congestion control design and did not consider misbehaving clients. This is similar to the case of today's TCP congestion control. But a mechanism that could prevent uncooperative clients from hurting other cooperative clients is desirable.

REFERENCES

[1] GeoLite City Database. http://dev.maxmind.com/geoip/geolite.

[2] IRCache Traces. ftp://ircache.net/Traces/DITL-2007-01-09/.

[3] Project CCNx. https://github.com/ProjectCCNx/ccnx.

[4] Scalable and Adaptive Internet Solutions (SAIL). http://www.sail-project.eu/.

[5] The FP7 4WARD Project. http://www.4ward-project.eu/.

[6] We knew the web was big. http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

[7] H. Abrahamsson and M. Nordmark. Program Popularity and Viewer Behaviour in a Large TV-on-Demand System. In *IMC'12*, November 2012.

[8] A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, October 2012.

[9] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone. Design Considerations for a Network of Information. In *ACM CoNEXT Workshop on Re-Architecting the Internet*, December 2008.

[10] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A Survey of Information-Centric Networking. *IEEE Communications Magazine*, 50(7):26–36, July 2012.

[11] P. Almeida, C. Baquero, N. Preguiça, and D. Hutchison. Scalable Bloom Filters. *Information Processing Letters*, 101(6):255–261, March 2007.

[12] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan. Optimal Content Placement for a Large-scale VoD System. In *ACM CoNEXT*, December 2010.

[13] D. Beaver, S. Kumar, H. Li, J. Sobel, and P. Vajgel. Finding a Needle in Haystack: Facebook's Photo Storage. In *USENIX OSDI*, October 2010.

[14] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[15] S. Borst, V. Gupta, and A. Walid. Distributed Caching Algorithms for Content Distribution Networks. In *IEEE INFOCOM*, March 2010.

[16] S. Braun, M. Monti, M. Sifalakis, and C. Tschudin. An Empirical Study of Receiver-based AIMD Flow-Control Strategies for CCN. In *IEEE ICCCN'13*, July 2013.

[17] S. Braun, M. Monti, M. Sifalakis, and C. Tschudin. CCN and TCP co-existence in the Future Internet: Should CCN be compatible to TCP? In *The Fifth IFIP/IEEE International Workshop on Management of the Future Internet*, May 2013.

[18] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *IEEE INFOCOM*, March 1999.

[19] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509, 2004.

[20] G. Carofiglio, M. Gallo, and L. Muscariello. ICP: Design and Evaluation of an Interest Control Protocol for Content-Centric Networking. In *IEEE NOMEN'12*, March 2012.

[21] G. Carofiglio, M. Gallo, and L. Muscariello. Joint Hop-by-hop and Receiver-driven Interest Control Protocol for Content-Centric Networks. In *ACM ICN'12*, August 2012.

[22] G. Carofiglio, M. Gallo, L. Muscariello, and M. Papalini. Multipath Congestion Control in Content-Centric Networks. In *IEEE NOMEN'13*, April 2013.

[23] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling Data Transfer in Content-Centric Networking. In *Proceedings of the 23rd International Teletraffic Congress*, September 2011.

[24] A. Carzaniga and A. Wolf. Forwarding in a Content-Based Network. In *ACM SIGCOMM*, August 2003.

[25] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *IMC'07*, October 2007.

[26] W. Chai, D. He, I. Psaras, and G. Pavlou. Cache Less for More in Information-Centric Networks. In *Proceedings of IFIP International Conference on Networking*, May 2012.

[27] W. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G.G. de Blas, F.J. Ramon-Salguero, Lei L., S. Spirou, A. Beben, and E. Hadjioannou. CURLING: Content-Ubiquitous Resolution and Delivery Infrastructure for Next-Generation Services. *IEEE Communications Magazine*, 49(3):112–120, March 2011.

[28] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of USENIX 1996 Annual Technical Conference*, January 1996.

[29] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, June 1989.

[30] 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 *The 1st Workshop on Emerging Design Choices in Name-Oriented Networking (IEEE NOMEN)*, March 2012.

[31] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi. A Survey on Content-Oriented Networking for Efficient Content Delivery. *IEEE Communications Magazine*, 49(3):121–127, March 2011.

[32] H. Dai, B. Liu, Y. Chen, and Y. Wang. On Pending Interest Table in Named Data Networking. In *ACM/IEEE ANCS*, October 2012.

[33] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone. MDHT: a Hierarchical Name Resolution Service for Information-Centric Networks. In *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking*, August 2011.

[34] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor. Longest Prefix Matching Using Bloom Filters. In *ACM SIGCOMM*, August 2003.

[35] M. Dischinger, A. Mislove, A. Haeberlen, and K. Gummadi. Detecting BitTorrent Blocking. In *IMC'08*, October 2008.

[36] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: a Scalable Wide-area Web Cache Sharing Protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[37] H. Fang, M. Kodialam, and T.V. Lakshman. Incremental Bloom Filters. In *IEEE INFOCOM*, April 2008.

[38] S. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, K. C. Ng, T. Koponen, B. Maggs, V. Sekar, and S. Shenker. Less Pain, Most of the Gain: Incrementally Deployable ICN. In *ACM SIGCOMM*, August 2013.

[39] S. Floyd. HighSpeed TCP for Large Congestion Windows. IETF RFC 3649, December 2003.

[40] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. IETF RFC 5348, September 2008.

[41] S. Floyd and V. Jacobson. Traffic Phase Effects in Packet-switched Gateways. *SIGCOMM Comput. Commun. Rev.*, 21(2):26–42, April 1991.

[42] S. Floyd and E. Kohler. Internet Research Needs Better Models. *SIGCOMM Comput. Commun. Rev.*, 33(1):29–34, January 2003.

[43] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. IETF RFC 6182, March 2011.

[44] N. Fotiou, G. Polyzos, P. Nikander, and D. Trossen. Developing Information Networking Further: From PSIRP to PURSUIT. In *The 7th International ICST Conference on Broadband Communications, Networks and Systems*, October 2010.

[45] G. Garcia, A. Beben, F. J. Ramon, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Sliwinski, S. Spirou, S. Soursos, and E. Hadjioannou. COMET: Content Mediator Architecture for Content-aware Networks. In *Future Network and Mobile Summit*, June 2011.

[46] J. Gettys. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing*, 15(3):96, May-June 2011.

[47] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox. Information-Centric Networking: Seeing the Forest for the Trees. In *ACM HotNets-X*, November 2011.

[48] M. Gritter and D. Cheriton. An Architecture for Content Routing Support in the Internet. In *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*, March 2001.

[49] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-peer File-sharing Workload. In *ACM SOSP*, October 2003.

[50] D. Guo, J. Wu, H. Chen, and X. Luo. Theory and Network Applications of Dynamic Bloom Filters. In *IEEE INFOCOM*, April 2006.

[51] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley. Multi-path TCP: a Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet. *IEEE/ACM Trans. Netw.*, 14(6):1260–1271, December 2006.

[52] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda. Multipath Congestion Control for Shared Bottleneck. In *The 7th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT'09)*, May 2009.

[53] A K M Mahmudul Hoque, S. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. NLSR: Named-data Link State Routing Protocol. In *ACM ICN'13*, August 2013.

[54] S. Ihm and V. Pai. Towards Understanding Modern Web Traffic. In *IMC'11*, November 2011.

[55] V. Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, August 1988.

[56] V. Jacobson, D. Smetters, N. Briggs, M. Plass, P. Stewart, J. Thornton, and R. Braynard. VoCCN: Voice over Content-Centric Networks. In *ACM ReArch'09*, December 2009.

[57] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking Named Content. In *ACM CoNEXT*, December 2009.

[58] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical Report DEC-TR-301, Digital Equipment Corporation, September 1984.

[59] P. Jokela, A. Zahemszky, C. Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: Line Speed Publish/Subscribe Inter-Networking. In *ACM SIGCOMM*, August 2009.

[60] K. Katsaros, G. Xylomenos, and G. Polyzos. MultiCache: An Overlay Architecture for Information-Centric Networking. *Computer Networks*, 55(4):936–947, March 2011.

[61] F. Kelly. Charging and Rate Control for Elastic Traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[62] F. Kelly and T. Voice. Stability of End-to-end Algorithms for Joint Routing and Rate Control. *SIGCOMM Comput. Commun. Rev.*, 35(2):5–12, April 2005.

[63] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J. Le Boudec. MPTCP is Not Pareto-optimal: Performance Issues and a Possible Solution. In *ACM CoNEXT*, December 2012.

[64] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and beyond) Network Architecture. In *ACM SIGCOMM*, August 2007.

[65] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. OceanStore: an Architecture for Global-Scale Persistent Storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[66] N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. On the Optimization of Storage Capacity Allocation for Content Distribution. *Comput. Netw.*, 47(3):409–428, February 2005.

[67] M. Lee, K. Cho, K. Park, T. Kwon, and Y. Choi. SCAN: Scalable Content Routing for Content-Aware Networking. In *IEEE ICC*, June 2011.

[68] 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 *ACM/IEEE ANCS*, October 2012.

[69] Z. Li, J. Bi, S. Wang, and X. Jiang. Compression of Pending Interest Table with Bloom Filter in Content Centric Network. In *Proceedings of the 7th International Conference on Future Internet Technologies (CFI)*, June 2012.

[70] G. Marchetto, R. Sisto, and M. Virgilio. PIT Overload Analysis in Content Centric Networks. In *ACM ICN'13*, August 2013.

[71] M. Mathis. Relentless Congestion Control. In *The 7th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT'09)*, May 2009.

[72] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web Caching: towards a New Global Caching Architecture. *Computer Networks and ISDN Systems*, 30(22-23):2169–2177, November 1998.

[73] Z. Ming, M. Xu, and D. Wang. Age-based Cooperative Caching in Information-Centric Networks. In *The 1st Workshop on Emerging Design Choices in Name-Oriented Networking (IEEE NOMEN)*, March 2012.

[74] M. Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Transactions on Networking*, 10(5):604–612, October 2002.

[75] J. Moy. OSPF Version 2. IETF RFC 2328, April 1998.

[76] A. Narayanan and D. Oran. Content Routing Using Internet Routing Protocols: Can It Scale? In *ICNRG side meeting in IETF-82*, November 2011.

[77] K. Nichols and V. Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5):20:20–20:34, May 2012.

[78] D. Oran. OSI IS-IS Intra-domain Routing Protocol. IETF RFC 1142, February 1990.

[79] S. Oueslati, J. Roberts, and N. Sbihi. Flow-aware Traffic Control for a Content-Centric Network. In *IEEE INFOCOM*, March 2012.

[80] J. Pan, S. Paul, and R. Jain. A Survey of the Research on Future Internet Architectures. *IEEE Communications Magazine*, 49(7):26–36, July 2011.

[81] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *IEEE HPSR'13*, July 2013.

[82] F. Papadopoulos, D. Krioukov, M. Boguñá, and A. Vahdat. Greedy Forwarding in Dynamic Scale-free Networks Embedded in Hyperbolic Metric Spaces. In *IEEE INFOCOM*, March 2010.

[83] D. Perino and M. Varvello. A Reality Check for Content Centric Networking. In *ACM ICN'12*, August 2011.

[84] S. Podlipnig and L. Böszörmenyi. A Survey of Web Cache Replacement Strategies. *ACM Comput. Surv.*, 35(4):374–398, December 2003.

[85] I. Psaras, W. Chai, and G. Pavlou. To Cache or Not To Cache: Probabilistic In-Network Caching for Information-Centric Networks. In *Proceedings of ACM SIGCOMM Workshop on Information-Centric Networking*, August 2012.

[86] I. Psaras, R. Clegg, R. Landa, W. Chai, and G. Pavlou. Modelling and Evaluation of CCN-Caching Trees. In *Proceedings of the 10th International IFIP TC 6 Conference on Networking*, May 2011.

[87] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. IETF RFC 3168, September 2001.

[88] S.C. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *IEEE INFOCOM*, June 2002.

[89] E.J. Rosensweig and J. Kurose. Breadcrumbs: Efficient, Best-Effort Content Location in Cache Networks. In *IEEE INFOCOM Mini-Conference*, April 2009.

[90] E.J. Rosensweig, D.S. Menasche, and J. Kurose. On the Steady State of Cache Networks. In *IEEE INFOCOM*, April 2013.

[91] D. Rossi and G. Rossini. Cache Performance of Content Centric Networks under Multipath Routing (and More). Technical report, Telecom ParisTech, 2011.

[92] D. Rossi and G. Rossini. On Sizing CCN Content Stores by Exploiting Topological Information. In *The 1st Workshop on Emerging Design Choices in Name-Oriented Networking (IEEE NOMEN)*, March 2012.

[93] N. Rozhnova and S. Fdida. An Effective Hop-by-hop Interest Shaping Mechanism for CCN Communications. In *IEEE NOMEN'12*, March 2012.

[94] L. Saino, C. Cocora, and G. Pavlou. CCTCP: A Scalable Receiver-driven Congestion Control Protocol for Content Centric Networking. In *IEEE ICC 2013 - Next-Generation Networking Symposium (ICC'13 NGN)*, June 2013.

[95] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *IEEE INFOCOM*, April 2001.

[96] V. Sourlas, P. Flegkas, L. Gkatzikis, and L. Tassiulas. Autonomic Cache Management in Information-Centric Networks. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012.

[97] V. Sourlas, P. Flegkas, G. Paschos, D. Katsaros, and L. Tassiulas. Storage Planning and Replica Assignment in Content-Centric Publish/Subscribe Networks. *Computer Networks*, 55(18):4021–4032, December 2011.

[98] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, February 2004.

[99] R. Stewart. Stream Control Transmission Protocol. IETF RFC 4960, September 2007.

[100] G. Tyson, S. Kaune, S. Miles, Y. El-khatib, A. Mauthe, and A. Taweel. A Trace-Driven Analysis of Caching in Content-Centric Networks. In *IEEE ICCCN*, July 2012.

[101] M. Ul Haque, K. Pawlikowski, A. Willig, and L. Bischofs. Performance Analysis of Blind Routing Algorithms over Content Centric Networking Architecture. In *International Conference on Computer and Communication Engineering (ICCCE)*, July 2012.

[102] P. Vixie. What DNS Is Not. *ACM Queue*, 7(10):10–15, November 2009.

[103] T. Vu, A. Baid, Y. Zhang, T. Nguyen, J. Fukuyama, R. Martin, and D. Raychaudhuri. DMap: A Shared Hosting Scheme for Dynamic Identifier to Locator Mappings in the Global Internet. In *IEEE ICDCS*, June 2012.

[104] J. Wang. A Survey of Web Caching Schemes for the Internet. *SIGCOMM Comput. Commun. Rev.*, 29(5):36–46, October 1999.

[105] L. Wang, A K M Mahmudul Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. Technical Report NDN-0003, July 2012.

[106] S. Wang, J. Bi, J. Wu, Z. Li, W. Zhang, and X. Yang. Could In-Network Caching Benefit Information-Centric Networking? In *Proceedings of the 7th Asian Internet Engineering Conference*, November 2011.

[107] 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 *The 1st Workshop on Emerging Design Choices in Name-Oriented Networking (IEEE NOMEN)*, March 2012.

[108] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie. Optimal Cache Allocation for Content-Centric Networking. In *IEEE ICNP*, October 2013.

[109] D. Wischik, M. Handley, and M. Braun. The Resource Pooling Principle. *SIGCOMM Comput. Commun. Rev.*, 38(5):47–52, October 2008.

[110] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *USENIX NSDI*, March 2011.

[111] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the Scale and Performance of Cooperative Web Proxy Caching. In *ACM SOSP*, December 1999.

[112] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang. A Case for Stateful Forwarding Plane. Technical Report NDN-0002, July 2012.

[113] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. Adaptive Forwarding in Named Data Networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):62–67, June 2012.

[114] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon. DiPIT: A Distributed Bloom-Filter Based PIT Table for CCN Nodes. In *21st International Conference on Computer Communications and Networks (ICCCN)*, July 2012.

[115] M. Yu, A. Fabrikant, and J. Rexford. BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations. In *ACM CoNEXT*, December 2009.

[116] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *IEEE INFOCOM*, March 1996.

[117] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, D. Smetters, B. Zhang, G. Tsudik, kc claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh. Named Data Networking (NDN) Project. Technical Report NDN-0001, October 2010.