ADAPTIVE FORWARDING IN NAMED DATA NETWORKING

by

Cheng Yi

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2014

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Cheng Yi
entitled Adaptive Forwarding in Named Data Networking
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Christopher Gniady — Date: 20 May 2014

John Hartman — Date: 20 May 2014

Richard Snodgrass — Date: 20 May 2014

Beichuan Zhang — Date: 20 May 2014

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director: Beichuan Zhang — Date: 20 May 2014

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

SIGNED: Cheng Yi

ACKNOWLEDGEMENTS

First, I would like to sincerely and wholeheartedly thank my advisor, Dr. Beichuan Zhang, for providing invaluable guidance and support during my Ph.D. study. This dissertation would not have been possible without his patience and persistent help. His rigor and passion in research will have a profound impact on my future career.

Next gratitude goes to my dissertation committee members, Dr. Chris Gniady, Dr. John Hartman, and Dr. Richard Snodgrass. Their constructive suggestions and comments are very helpful for improving this dissertation. It is my honor and privilege to have worked with every one of them.

I am deeply grateful to our collaborators from the NDN project team, especially Dr. Alexander Afanasyev and Dr. Lixia Zhang from UCLA, and Dr. Lan Wang from University of Memphis. I benefited tremendously from the stimulating and insightful discussions with them.

I also want to thank my colleagues from the Network Research Lab: Yifeng Li, Jerald Abraham, Junxiao Shi, Yi Huang and Varun Khare. Their support and cooperation helped me immensely in the completion of this dissertation work. It is a great pleasure to have worked with them.

Many thanks to the good friends I met at the University of Arizona: Mingsong Bi, Rui Zhang, Lei Ye and Jinyan Guan. They made my Ph.D. experience enjoyable and memorable. Thanks also to Tom Lowry for his patient assistance and Bridget Radcliff for being a great academic advisor.

Finally, I owe my utmost gratitude to my family who have given me their unconditional love and support throughout my whole life. Their trust and understanding helped me overcome many obstacles during the Ph.D. process. None of my accomplishments would have been possible without them.

DEDICATION

I dedicate this dissertation work to my dear family. A special feeling of gratitude to my beloved wife Fengqiong Huang who is always encouraging and supportive, my newborn angel Jayden Yi who gives me immense motivation and strength, and my amazing parents Tongxiu Chen and Mengsheng Yi who love me more than themselves.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF FIGURES – *Continued*

LIST OF TABLES

LIST OF PSEUDO-CODES

ABSTRACT

Named Data Networking (NDN) is a recently proposed new Internet architecture. By naming data instead of locations, it changes the very basic network service abstraction from "delivering packets to given destinations" to "retrieving data of given names." This fundamental change creates an abundance of new opportunities as well as many intellectual challenges in application development, network routing and forwarding, communication security and privacy.

The focus of this dissertation is a unique feature introduced by NDN: its adaptive forwarding plane. Communication in NDN is done by exchanges of Interest and Data packets. Consumers send Interest packets to request desired Data, routers forward them based on data names, and producers answer with Data packets, which take the same path of Interests but in reverse direction. During this process, routers maintain state information of pending Interests. This state information, coupled with the symmetric exchange of Interest and Data, enables NDN routers to detect loops, observe data retrieval performance, and explore multiple forwarding paths, all at the forwarding plane. Since NDN is still in its early stage, however, none of these powerful features has been systematically designed, evaluated, or explored.

In this dissertation, we present a concrete design of NDN's forwarding plane to make the network *resilient* and *efficient*. First, we design the basic adaptation mechanism and evaluate its effectiveness in circumventing prefix hijack attacks. Second, we propose a novel NACK mechanism for fast failure detection and evaluate its benefits in handling network failures. We also show that a resilient forwarding plane makes routing more stable and more scalable. Third, we design a congestion control mechanism, Dynamic Interest Limiting, to adapt traffic rate in a hop-by-hop and multipath fashion, which is effective even with a large number of flows in a large network topology.

CHAPTER 1

INTRODUCTION

The *hourglass* architecture of today's Internet (see Figure 1.1) has lead to a great success by allowing both upper and lower layers of the middle thin waist to innovate independently. As the universal network layer at the thin waist, IP was originally designed for communication between two endpoints. With the rapid and continued growth of e-commerce, digital media, social networking and smartphone applications, however, Internet communication is becoming increasingly dominated by content distribution and retrieval. The Internet is getting exceedingly complex in its attempt to address the distribution problem with a point-to-point communication protocol. Therefore, researchers have started to investigate new architectures to accommodate this emerging communication pattern. As a result, different architecture designs (e.g., [6, 10, 11, 37, 39]) have been proposed under a general network research approach called *information-centric networking* (ICN).

Named Data Networking (NDN) [36, 73] is the leading design of recently proposed ICN architectures. It inherits the hourglass shape of the IP architecture, but replaces the end-to-end data delivery model at the thin waist by a receiver-driven data retrieval model (see Figure 1.1). As a result, NDN shifts the network service semantics from "delivering packets to given addresses" to "retrieving data of given names." NDN packets carry data names rather than source and destination addresses. Data consumers express *Interests* in the form of desired data names without specifying where the data may be located. Routers satisfy the Interests by retrieving the *Data*, which are bound to the names by cryptographic signatures, from router caches, intermediate data repositories, or original data producers. The NDN architecture offers a wide range of benefits, including built-in multicast data delivery, in-network caching, as well as data-centric security (see Chapter 2). This dissertation studies *adaptive forwarding*, a unique and powerful feature introduced

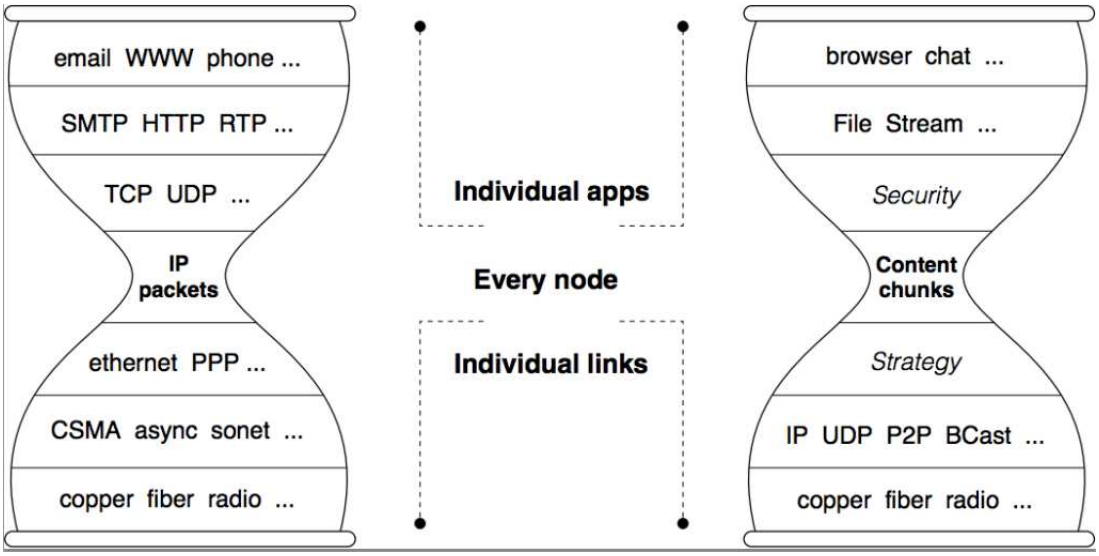


Figure 1.1: IP and NDN Hourglass Architecture, reused from [36]

by NDN.

A network's architecture design determines the shape and form of its forwarding mechanism. A typical router today contains a routing plane and a forwarding plane. The routing plane defines the part of the router architecture that constructs the routing table, while the forwarding plane makes forwarding decisions for incoming packets. Today's IP Internet accomplishes packet delivery in two phases. At the *routing plane*, routers exchange routing updates and select the best routes to construct the forwarding table (FIB); at the *forwarding plane*, routers forward packets strictly following the FIB. Thus, IP routing is stateful and adaptive, while IP forwarding is stateless and has no adaptability of its own. This *smart routing, dumb forwarding* approach places the responsibility of robust data delivery solely on the routing system. Consequently IP's routing plane is also referred to as the *control plane*, and its forwarding plane the *data plane*.

NDN has a smart and adaptive forwarding plane as opposed to IP. While routing in an NDN network serves the same purpose as in an IP network, i.e., computing routing tables, the forwarding plane in an NDN network is split into a two-step process: consumers first send out Interest packets, then Data packets flow back

along the same path in the reverse direction. Routers keep state of pending Interests to guide Data packets back to requesting consumers. The pending Interest state, together with the two-way Interest and Data exchange, enables NDN routers' *forwarding* process to measure performance of different paths, quickly detect failures and retry alternative paths. This unique feature of NDN is called *adaptive forwarding*.

NDN is the first network architecture that provides an adaptive forwarding plane. With adaptive forwarding, NDN routers can detect and recover from network problems quickly without waiting for global routing convergence. For example, NDN routers can estimate the round-trip time (RTT) for each name prefix by recording pending Interests and observing Data packets coming back. Network problems can be detected by setting up a timer based on the RTT estimate: if Data is not received before the timer expires, there may be potential forwarding problems (e.g., link failures) in the network. Once problems are detected, NDN routers can start exploring alternative paths immediately using local state information. Since Interest forwarding is loop-free (see Section 2.2.2), routers can effectively utilize multiple paths to recover from the problems.

The idea of adaptive forwarding appears promising. However, there has not been any systematic design of NDN's forwarding plane that capitalizes on this powerful feature. The seminal paper by Jacobson et al. [36] only sketched out a blueprint of the overall NDN architecture; the operations of its forwarding plane were not fully explained and the design specifics remained to be filled in. CCNx[1] [5] is an open source prototype implementation of NDN. However, it only serves as a proof of concept. Its forwarding algorithm is not well documented and has several limitations (see Section 2.3). Additionally, the performance of adaptive forwarding has not been extensively evaluated. In this dissertation, we address the limitations of the CCNx implementation by providing a new forwarding plane design for NDN that fully exploits adaptive forwarding. We also comprehensively evaluate NDN's packet forwarding performance under adverse conditions, and compare it against IP

[1]Version 0.8.1 as of when this dissertation was written.

and IP-based solutions.

The rest of this dissertation is organized as follows. Chapter 2 gives a brief overview of the NDN architecture. Chapter 3 summarizes the approach of this dissertation. Chapter 4 presents our own design of NDN's forwarding plane. Chapter 5 studies failure handling and role of routing in NDN. Chapter 6 presents a novel congestion control mechanism for NDN. Chapter 7 discusses the potential directions for future research. Chapter 8 summarizes related work and Chapter 9 concludes this dissertation.

CHAPTER 2

BACKGROUND

In this chapter, we first briefly review the IP architecture. Then we introduce the overall NDN architecture as well as the benefits it provides compared to IP.

2.1 IP Architecture

Today's IP Internet has a layered hourglass architecture (see Figure 1.1). Each layer in the architecture provides a different functionality; lower-layer protocols define interfaces to be used by upper-layer protocols. IP is the primary network-layer protocol at the narrow waist of the hourglass. On top of IP are transport-layer protocols (e.g., TCP), application-layer protocols (e.g., HTTP) and actual applications (e.g., WWW), while link-layer protocols (e.g., Ethernet), physical-layer protocols (e.g., CSMA) and actual transportation medium (e.g., optical fiber) reside below IP. This architecture allows protocols at both upper and lower layers of IP to innovate independently, leading to the continued success of the Internet.

As a network-layer protocol, IP is responsible for delivering packets across the network. Since IP was designed for point-to-point communications, it names end points by assigning them globally unique IP addresses. IP communication is sender-driven: the sender constructs an IP packet, puts both the source and destination addresses into the packet header (see Figure 2.1[1]) and sends it to the network. IP routers forward the packet based on the destination address following the FIB, which is constructed by IP routing protocols. Routers announce IP prefixes (e.g.,

[1]The source and destination addresses in IPv4 packets are only 32 bits long, providing a maximum of 2^{32} addresses. As a result, IPv4 faces the address exhaustion problem where the pool of unallocated IPv4 address is depleted. IPv6 was introduced to address this problem, but it is out of the scope of this dissertation and thus not described here.

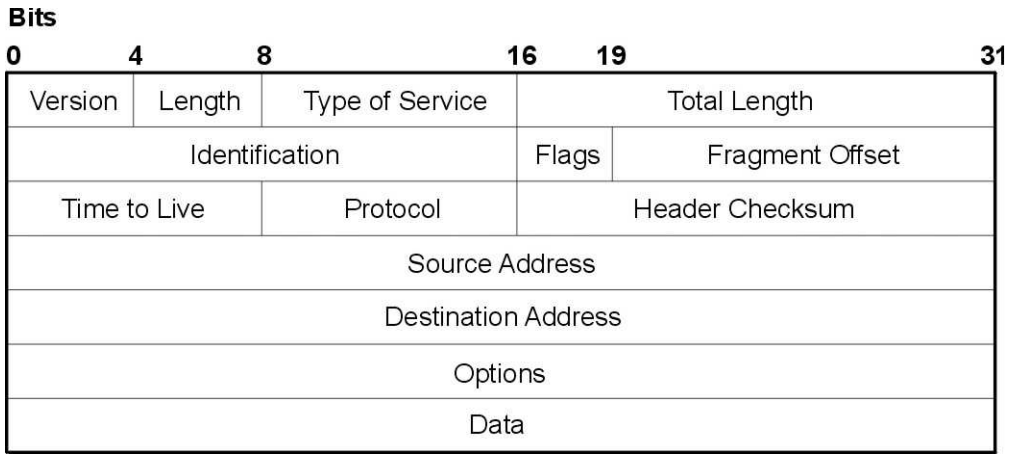


Figure 2.1: IPv4 Packet

150.135.68.0/24) they serve to the routing protocol, which then distributes the prefix as well as connectivity information to the rest of the network. When a router receives routing information from its neighbors, it may re-compute the routing table and update the FIB accordingly. IP FIB provides mapping from IP prefixes to the best next hops (e.g., 150.135.68.0/24 $\rightarrow$ intf1). When an application receives an IP packet, it can simply swap the source and destination addresses of the packet and send responses back to the network.

With the emergence of new applications such as e-commerce, digital media, social networking and smartphones, the Internet communication is becoming increasingly content-centric. The Internet has many limitations and grows exceedingly complex in the attempt to address the content distribution problem with a point-to-point communication protocol. First, a Domain Name Service (DNS) system is required to translate data names into IP addresses. Second, IP multicast is not widely deployed due to its complexity, leading to tremendous waste of resources when multiple users are accessing the same data. Third, in-network caching is not supported by IP. Large content providers have to use Content Distribution Networks (CDN) to provide temporary caching of content. Fourth, IP is not secure against malicious attacks. Although many efforts have been devoted to securing the communication channels, security breaches still keep increasing. As a result, Jacobson et al. proposed Named

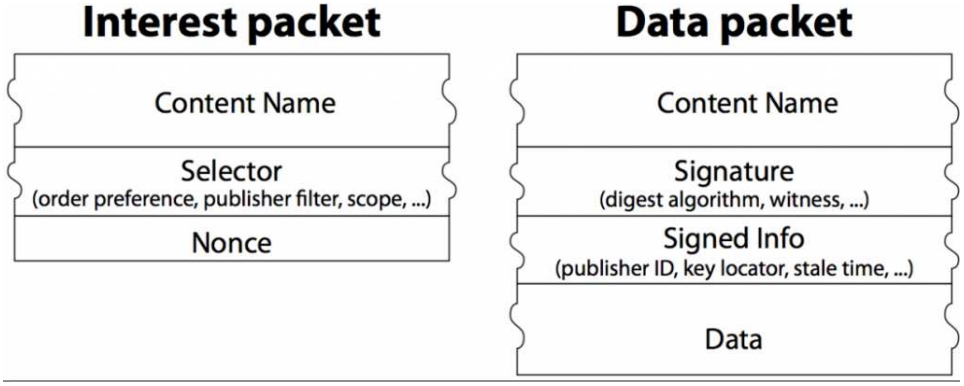


Figure 2.2: NDN Packets from [36]

Data Networking to address the limitations and better accommodate the emerging communication patterns.

2.2 NDN Architecture

NDN is a receiver-driven, network-layer communication protocol. This section gives a concise overview of the NDN architecture. A more detailed description can be found elsewhere [36, 73].

2.2.1 Packets and Names

NDN retains IP's hourglass architecture, but at the thin waist it names content instead of communication endpoints (see Figure 1.1[2]). This fundamental change leads to the shift of communication paradigm from *location-centric* to *data-centric*. All communication in NDN is performed using two distinct types of packets: *Interest* and *Data* (see Figure 2.2). Both types of packets carry a *name*, which uniquely identifies a piece of data that can be carried in one Data packet. To retrieve Data, a consumer puts the name of desired data into an Interest packet and sends it

[2]With NDN as the new network-layer protocol, old network-layer and transport-layer protocols such as IP and UDP are pushed down as link-layer protocols in the new architecture. Applications also need to be re-designed to work with the new network layer.

to the network. Routers use this name to forward the Interest towards the data producer(s), and the Data packet whose name matches the name in the Interest is returned to the consumer.

Each Interest packet also carries a *selector* field which provides more specific descriptions of the desired Data, and a *nonce* field which is a random number generated by the consumer. A more detailed explanation of nonce can be found in Section 2.2.2. A Data packet carries the actual data, descriptions about the data, as well as a cryptographic signature that binds the data to the name. Data packets are signed by the producers when they are created. The signature allows consumers to verify the integrity of Data packets; thus trust in a Data packet is decoupled from how it is obtained or where it is from. NDN's built-in data-centric security is useful in many aspects. For example, signatures on network routing and control messages provide a solid basis for securing routing protocols against attacks such as spoofing and tampering.

NDN names are opaque to the network, i.e., routers do not understand the meaning of the names. This enables applications to define their own naming schemes independent from the network. NDN names are hierarchically structured. For example, this dissertation file can be named `/edu/arizona/cs/chengyi/dissertation.pdf` in NDN[3], where '/' is the delimiter between name components. This hierarchical structure has two advantages. First, applications can put the context and relationship among data elements into the names. For example, segment 1 of version 2 of this dissertation can be named `/edu/arizona/cs/chengyi/dissertation.pdf/2/1`. Second, it allows name aggregation which is essential in scaling the routing system. For example, let University of Arizona be an autonomous system in NDN, it could distribute the name prefix `/edu/arizona/` through routing protocols in a way similar to distributing IP prefixes in today's Internet.

Similar to IP packet delivery, an NDN network performs best effort data retrieval. An Interest or Data packet can be lost, and it is the end consumer's responsibility

[3]Similarly, the file can be named `http://www.cs.arizona.edu/chengyi/dissertation.pdf` in today's Internet.

to retransmit the Interest if it does not receive the desired Data after expected RTT and it still wants the Data[4]. However, unlike IP's location-centric approach to data delivery, NDN packets carry data names instead of addresses. This basic difference in design leads to two profound differences in data delivery operations. First, although the name in an Interest packet is used to guide its forwarding, in a way similar to how a destination address is used to guide the forwarding of an IP packet, the Interest may cross a copy of the requested Data at an intermediate router or data repository and bring the Data back, while an IP packet is always delivered to the destination (if not dropped along the way). Second, an Interest packet carries neither address nor name to identify the requesting consumer that can be used to return the requested Data packet. Instead NDN routers keep track of incoming interfaces for each forwarded Interest (a pending Interest) and use this information to bring matched Data packets back to consumers.

2.2.2 Forwarding Process

To implement the Interest and Data forwarding functions, each NDN router maintains three major data structures: a *Forwarding Information Base* (FIB), a *Pending Interest Table* (PIT), and a *Content Store* (CS) (see Figure 2.3). An NDN router's FIB is roughly similar to the FIB in an IP router except that it contains name prefixes instead of IP address prefixes, and it may show multiple interfaces for a given name prefix (see Section 4.2). By its name, the PIT stores all Interest packets that have been forwarded but not yet satisfied. Each PIT entry records the name and incoming interface(s) of the Interest(s), as well as the outgoing interface(s) to which the Interest(s) has been forwarded. The CS provides temporary in-network storage of the received Data packets. Each NDN router is also equipped with a *Forwarding Strategy* module, which determines whether, when and where to forward an Interest packet based on the information stored in these data structures (see Section 4.3).

[4]In this dissertation the term **retransmit** is used exclusively for end consumers re-expressing Interests; another term, **retry**, is used when intermediate routers explore alternative paths after network problems are detected.

Content Store (CS)

Name	Data
/foo/bar/0/0	…

Pending Interest Table (PIT)

Name	Incoming Interfaces	Outgoing Interfaces	Nonces
/foo/bar/0/1	1,4	3	374152, 214950328

Forwarding Information Base (FIB)

Prefix	Interfaces
/foo	3,2,4,1

Figure 2.3: NDN Major Data Structures

Figure 2.4 illustrates the forwarding process in NDN. When a router receives an Interest packet, it first checks whether there is a matching Data in its Content Store. If a match is found, the Data is immediately sent back to the interface from which the Interest is received. Otherwise the Interest is checked against the entries in the PIT using its name. If the name already exists in the PIT, it can be either a duplicate Interest that should be dropped, an Interest retransmitted by the consumer that may need to be forwarded using a different outgoing interface, or an Interest from another consumer asking for the same Data which requires the incoming interface of this Interest to be added to the existing PIT entry. This effectively constructs a

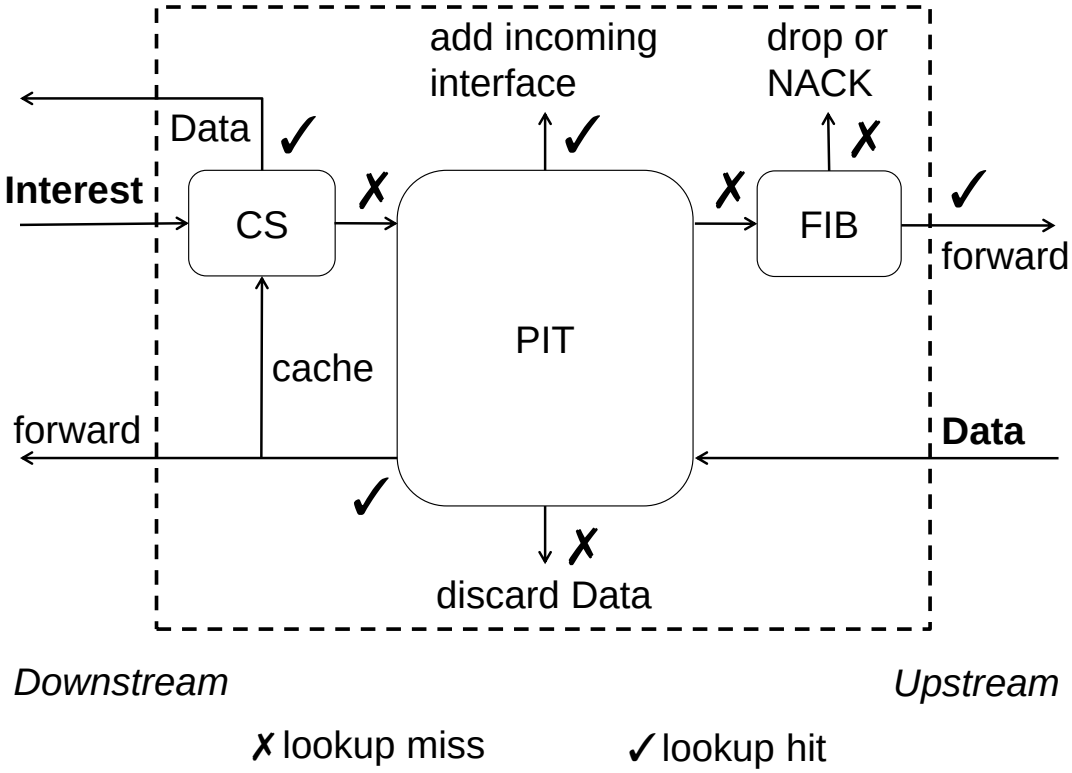


Figure 2.4: Interest and Data processing in NDN

multicast tree for consumers requesting for the same content at the same time. If the name does not exist in the PIT, the Interest is added into the PIT and further forwarded to the interface chosen by the forwarding strategy module. When a Data packet is received, its name is used to look up the PIT. If a matching PIT entry is found, the router sends the Data packet to the interface(s) from which the Interest was received and removes the PIT entry. As a result, Data packets always take the reverse paths of Interests. If no match is found, the Data packet is unsolicited and discarded. Each Interest also has an associated lifetime set by the consumer; a PIT entry is removed if the Interest not satisfied before its lifetime expires.

A router may choose to cache a Data packet it receives in the CS depending on its caching policy. Since a Data packet carries a name and a signature, it is meaningful independent of its source and destination. Therefore Data packets cached in the CS can be used to satisfy future Interests. CS is analogous to buffer memory in

IP routers, but IP routers cannot reuse a packet after it is forwarded. In-network caching is useful in many circumstances. For example, a Data packet may get dropped by the network due to congestion after being cached by router A. When the consumer retransmits the Interest, it can retrieve the Data directly from router A without being further forwarded to the actual producer.

Forwarding in NDN is loop-free. Each Interest carries a *nonce* field which, together with the Interest name, uniquely identifies the Interest. A router records the nonce of each Interest they receive in the PIT entry; hence it can tell whether a newly arrived Interest carrying the same name as an existing pending Interest is a previously forwarded Interest that looped back (in which case the Interest is dropped). Therefore Interest packets cannot loop. Because Data packets follow the reverse paths of the corresponding Interests, they do not loop either. This enables routers to freely retry multiple alternative paths in Interest forwarding. Notice that retry should be limited in scope and duration because (1) routers are not ultimately responsible for getting the Data, and (2) if all routers along the path perform retry, it may potentially lead to Interest explosion and significant overhead.

In the absence of packet loss, one Interest packet retrieves exactly one Data packet on each interface, providing a *flow balance*. When transferring a large content that does not fit into one Data packet, this flow balance can provide a fine-grained feedback loop to every node on the forwarding path. This is essentially similar to ACKs in TCP, except that ACKs only provide feedback to end hosts.

2.2.3 Datagram State

An NDN router maintains an entry in its PIT for *every* pending Interest packet, thus we say the router contains "datagram state." This state leads to a closed-loop, two-way symmetric packet flow: over each link, every Interest packet pulls back exactly one Data packet, maintaining one-on-one flow balance, except in (rare) cases where packets get lost or matching data does not exist.

It is worth noting that NDN's datagram state differs in fundamental ways from the virtual circuit state for ATM or MPLS. First, a virtual circuit sets up a single

path between an ingress-egress router pair; when it breaks, the state has to be reestablished for the entire path. Second, a virtual circuit pins down the path to be used for packet forwarding; if any of the links along the path gets overloaded due to traffic dynamics, packets on the same virtual circuit cannot be diverted to adapt to the load changes. In contrast, NDN's datagram state is per-Interest, per-hop. At each hop, the router makes its own decision on where to forward an Interest. When a router crashes or a link fails, the failure only affects the Interests at that specific location; the previous hop routers can quickly detect the failure and get around the problematic area.

2.2.4 Routing

NDN packets are routed and forwarded by names. Therefore, NDN does not have the address-related issues as in the IP architecture. First, there is no address exhaustion problem as in IP since the namespace in NDN is unbounded. Second, there is no NAT traversal problem since neither public nor private addresses are needed in the routing system. Finally, address assignment and management is no longer required in local area networks.

Conventional routing algorithms such as link-state or path-vector can be used in NDN. Instead of IP prefixes, NDN routers announce name prefixes that they are willing to serve to the routing protocols, which then propagate these announcements across the network. Other routers construct their own FIBs according to the routing announcements. Multiple routers can announce the same name prefix, leading to natural anycast routing. Traditional IP routing protocols such as OSPF can be easily adapted to route on name prefixes [65]. By treating names as sequences of opaque components, routers can perform component-wise longest prefix match of Interest names against the FIB.

Pseudo-code 1 CCNx Forwarding Strategy

```
1: function FORWARD(Interest)
2:     for each Intf in Sort(AvailableInterfaces) do
3:         Transmit(Interest, Intf)
4:         Wait(t)
5:         if Satisfied(Interest) then
6:             t ← t × (1 - 1/128)
7:             Return
8:         end if
9:         t ← t × (1 + 1/8)
10:    end for
11: end function
```

2.3 CCNx Overview

CCNx is an open-source implementation of the NDN protocol. It is currently deployed on the NDN Testbed [8], where NDN applications such as lighting control [19], chatting [74] and video production [40] are running. The successful deployment of NDN Testbed proves that the NDN architecture works well in real-world communications. In this section, we present a high-level overview of CCNx's forwarding plane design and discuss its limitations.

2.3.1 CCNx Forwarding Algorithm

The forwarding plane design of CCNx has not been well documented. A high-level abstraction of CCNx's Interest forwarding algorithm[5] is summarized in Pseudo-code 1. The algorithm iterates through all interfaces except the one(s) where the Interest is received. The interfaces are tried one-by-one in a sorted order: the best interface (the one with the shortest RTT) is tried first, followed by the second best interface; the rest of the interfaces are tried in the order of when they are added

[5]This pseudo-code is based on CCNx version 0.8.1. The actual implementation is much more complex and scattered all over the code. Interested readers can refer to the code for more details.

to the FIB entry. The router will start a timer t after forwarding the Interest to an interface. The algorithm returns if Data is received before the timer expires; otherwise the router will continue to forward the Interest to the next interface. In the worst case, the router will try all available interfaces before it gives up.

The setting of timer t is essential to the performance of the forwarding algorithm. A large t may lead to long fault recovery time while a small t will cause extra overhead. In CCNx, t is set to be the RTO (Retransmission Timeout) for the best interface, and small random numbers in milliseconds for the rest of the interfaces. The value of the RTO is computed as follows. It has an initial value of a few random milliseconds. If Data is not received before the RTO expires, it will be increased by 1/8; otherwise it will be decreased by 1/128. Therefore the RTO value will slowly increase until it exceeds the real RTT, after which it keeps oscillating around the RTT. Suppose the initial RTO value is 10 ms and the real RTT is 100 ms, the RTO value for each Interest is illustrated in Figure 2.5. An Interest will be forwarded to two or more interfaces when the RTO value is smaller than the real RTT, which occurs around once every 16 Interests in this scenario. As a result, CCNx routers are able to recover from network faults within the RTT time scale, at the cost of frequently sending extra probing Interests to the network. Consider that probing is performed by every router for every name prefix, the actual number of extra Interests in the network is not negligible.

2.3.2 Limitations of CCNx

The CCNx forwarding algorithm is simple and proven to work on the NDN Testbed. However, it has many limitations as summarized below.

- The interface ranking method is restricted. Interface ranking determines the order of how interfaces will be tried. Since a CCNx router only records the two fastest interfaces for each name prefix, there is no way to rank interfaces using metrics other than RTT, e.g., hop count. In addition, there is no way to enforce routing policy. Routing protocols should be able to provide better and

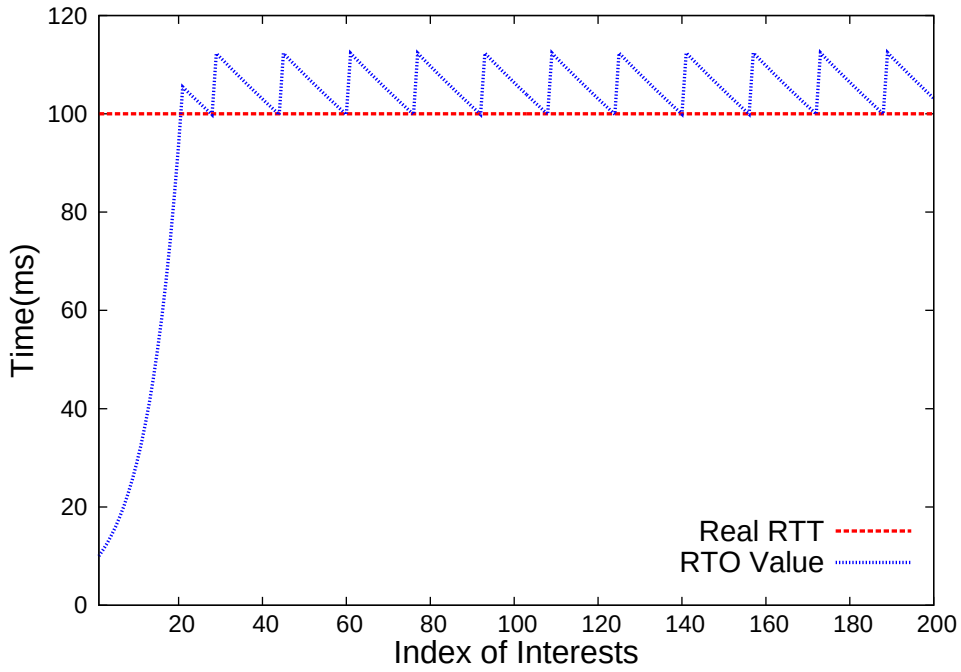


Figure 2.5: CCNx Retransmission Timeout

more flexible interface ranking, but CCNx is not designed to be able to take advantage of routing. An improved interface ranking mechanism is presented in Section 4.2.3.

- Periodical probing incurs extra overhead. As explained in Section 2.3.1, each CCNx router will periodically send extra probing Interests to the network for each name prefix. Since CCNx is designed to work without routing protocols, probing is the only way for routers to learn about changes in the network. However, it also leads to waste of network resources when there is no change in the network. We present our own forwarding strategy design in Section 4.3 to address this issue.

- Fault detection is slow. A CCNx router can detect potential network problems only after the RTO expires. In fact, if an upstream router cannot satisfy an Interest for any reason, it should be able to send a notification to its downstream router. Then the downstream router can start retrying other

interfaces immediately without waiting for the timer to expire. An Interest NACK mechanism is proposed in Section 5.1.1 for this matter.

- CCNx does not exploit the full potential of adaptive forwarding. For example, adaptive forwarding enables routers to perform effective hop-by-hop congestion control. However, no congestion control mechanism is implemented on CCNx routers. CCNx still relies on consumers to perform end-to-end congestion control. We present an effective Interest limiting mechanism in Section 6.2 for hop-by-hop multipath congestion control.

It is worth mentioning that these limitations only affect performance, not correctness. In all adverse conditions except congestion, CCNx should be able to achieve the same level of resilience as our design with higher overhead and/or longer reaction time.

CHAPTER 3

OVERVIEW OF APPROACH

The previous chapter reviewed the overall NDN architecture as well as the CCNx implementation, the existing prototype implementation of NDN. In this chapter, we summarize the approach and contributions of this dissertation.

3.1 Forwarding Plane Design

We propose a concrete design of NDN's forwarding plane which is able to perform effective multipath forwarding. One of the design objectives is to provide specific mechanisms for routers to track data delivery performance, which is vital for routers in making forwarding decisions. We propose a color-code mechanism which indicates the working status of each interface for each name prefix. We also present algorithms and data structures for routers to measure and store the RTT for each name prefix. In addition, we provide an efficient forwarding strategy named *BestRoute*, which determines when and how routers forward Interests. There is a spectrum of strategies between trying a single interface each time and flooding to all interfaces, with different trade-off between the overhead and delay to retrieve data. It is essential to provide a strategy that ensures efficient packet delivery without incurring excessive overhead, even under adverse conditions. We present an interface ranking mechanism which ranks interfaces of a router taking both routing information and forwarding performance into consideration. In the proposed forwarding strategy, Interests are always forwarded to the highest ranked interface that has not been tried before. In the rest of this dissertation, we use *NDN-BestRoute* to refer to our own design and implementation of NDN's forwarding plane. Unless otherwise specified, all experiments for NDN are conducted using the ndnSIM [12] simulator with the NDN-BestRoute implementation.

We use prefix hijack as an example to demonstrate how NDN-BestRoute is able to handle network problems effectively with adaptive forwarding. Prefix hijack occurs when routers falsely announce prefixes that do not belong to them to the routing protocols, either intentionally or by mistake. Prefix hijack is not rare in today's Internet. One famous example is the hijack of YouTube's prefixes by Pakistan Telecom in 2008. However, IP routers have no idea when prefix hijack occurs since there is no way for them to distinguish hijackers' routing announcements from normal ones. End hosts may detect the problem since packets will stop getting delivered, but there is no way for them to inform routers about the problem. Therefore prefix hijack remains a significant issue in today's Internet.

On the other hand, NDN-BestRoute is able to handle prefix hijack at the forwarding plane. The forwarding strategy is designed to prefer current working paths. If the routing protocol provides a new best path, a router will not switch to it immediately. Instead, it will send a probing Interest to the new best path and only switches if probing is successful, i.e., Data is received. Our simulation on the Sprint PoP-level topology shows that for over 94% of the cases routers will not be affected by prefix hijack. Routers will only be affected when the hijacker lies on the current working path, in which case consumers will stop receiving Data. However, consumers can retransmit the Interests so that routers will retry alternative paths in order to resume packet delivery.

3.2 Failure Handling

Link failures are very common in computer networks. In an ideal packet-switched network, each packet should be delivered to its destination in the presence of unexpected failures, as long as the destination is still reachable in the remaining topology. This system robustness in face of component failures is referred to as "perfect switching" in Baran's seminal work [16]. Today's Internet is far from achieving perfect switching, as IP relies on the routing plane to handle failures; packet delivery cannot be guaranteed after failures since routing protocols need time to converge. Research

has been conducted on reducing routing convergence time [13, 29] in order to reduce packet loss after failures. However, these methods have negative impact on the scalability and stability of routing protocols. As a result, failure handling has drawn significant attention from the networking research community.

3.2.1 Failure Handling with Adaptive Forwarding

NDN routers are able to handle link failures at the forwarding plane without routing convergence. When a link failure is detected by either keep-alive messages of routing protocols or link-level mechanisms, the attached routers will mark the interface as unusable; all subsequent Interests will be forwarded to the next highest ranked interface instead. Packets that have already been sent to the failed link may still get lost, in which case consumers can retransmit the Interests to trigger path exploration by routers in a similar fashion as in the prefix hijack scenario.

It may happen that a router does not have any usable interface to forward an Interest due to link failures. In such cases, the Interest will be dropped and consumers need to retransmit the Interest after timeout in order to resume packet delivery, which is slow and inefficient. We introduce an Interest Negative ACKnowledgement (NACK) mechanism to address this issue. If a router has no way to satisfy an Interest, it will return a NACK to the downstream router, which can start retrying alternative paths immediately without waiting for consumer retransmission.

We compare the performance of our forwarding plane design with that of IP and Path Splicing [48], an IP-based multipath forwarding solution in link failure scenarios. Simulation results show that NDN-BestRoute provides the best packet delivery performance after link failures. NDN-BestRoute is able to approach perfect switching while restricting the stretch of alternative paths to less than 1.2 for over 89% of the cases.

3.2.2 Role of Routing in NDN

Having an intelligent adaptive forwarding plane also raises new research questions. Today's IP networks put all intelligence into routing, which disseminates topology and policy information, computes routes, detects and recovers from failures while the forwarding plane merely forwards packets according to the FIB. When the forwarding plane has its own adaptability, are routing protocols still needed? If so, for what purpose and to what extent? If some of routing's tasks can be offloaded to forwarding, would that bring positive impact on routing protocols' design and operation, e.g., making routing more scalable and stable?

In this dissertation we investigate the role of routing in NDN networks. Through analysis, design, and extensive simulation, we find that routing is important in bootstrapping the forwarding plane for effective data retrieval, as well as for efficiently probing new links or recovered links. However, NDN routing protocols do not need to converge fast after the network changes; such changes can be handled by adaptive forwarding more promptly. This enables one to significantly improve the scalability and stability of the routing system using larger keep-alive timer values that ignore short-term failures. Furthermore, routing algorithms that would not work well in current networks may work fine in NDN due to its reduced role of bootstrapping adaptive forwarding.

3.3 Congestion Control

In NDN, one Interest retrieves at most one Data packet. This one-to-one flow balance coupled with NDN's two-way symmetric traffic pattern enables routers to perform congestion control through Interest limiting. Since Data packets carry the actual content, their sizes should be much larger than Interests most of the time. Therefore, congestion should be more likely caused by Data packets in NDN networks. A downstream router can effectively limit the number of Data packets coming from an upstream router by limiting the number of Interests forwarded to the upstream. Thus by enforcing Interest limit on each interface, NDN routers are

able to perform hop-by-hop multipath congestion control.

3.3.1 Simple Interest Limiting

We first experiment with a simple Interest limiting mechanism which computes Interest limit for each interface using the link bandwidth and average Data packet size. Simulation results shows that it outperforms TCP in 93% of the cases because of NDN's multipath forwarding capability. However, this mechanism has several limitations. First, the sizes of Data packets in NDN networks are unpredictable and may change rapidly. Second, the returning delay of Data packets is also unpredictable due to caching and multipath forwarding. Third, the mechanism does not consider the cases when NDN is deployed as an overlay on top of IP, where the link bandwidth is also unknown. Fourth, routers cannot ensure fairness among different flows. Therefore, this simple Interest limiting mechanism is not suitable for practical usage.

3.3.2 Dynamic Interest Limiting

We propose a Dynamic Interest Limiting (DIL) mechanism to address the limitations of simple Interest limiting. The design objectives of DIL are to provide high throughput, short queuing delay, and fairness among concurrent flows. Keeping queuing delay short is essential for applications that require consistently low latency or jitter-free transmission. We dynamically adjust the Interest limit on each interface using an Additive-Increase-Multiplicative-Decrease (AIMD) algorithm similar to TCP. The Interest limit is increased when Data is received, and decreased when congestion is detected.

We design two novel methods for congestion detection in NDN networks. Random Early NACK (REN) is designed for native NDN networks. The upstream router keeps monitoring the queue length of its output queue, and send NACKs to the downstream router at certain probability. The NACK probability is computed using an algorithm similar to Random Early Detection (RED) [28]. These NACKs

are regarded as signals of congestion by the downstream router, which will reduce its Interest limit accordingly. In an NDN-over-IP setup, however, congestion may happen in the underlying IP routers, which the NDN routers don't have access to. Therefore, REN may not work well in such scenarios. We introduce Link-layer Congestion Detection (LCD) for NDN overlay networks. Each upstream router adds a link-layer header containing a sequence number to each NDN packet it forwards; downstream routers can detect congestion by observing gaps in the sequence numbers they receive. We also propose a Fair Interest Limiting (FIL) algorithm which fairly divides the Interest limit to all active flows. Through extensive simulations we show that DIL is able to achieve all the design goals and outperforms TCP in all the cases.

3.4 Contributions

The contributions of this dissertation are summarized as follows:

- We presented NDN-BestRoute, a concrete forwarding plane design for NDN to fully exploit the adaptive forwarding feature (Chapter 4). An effective forwarding strategy is presented to provide high packet delivery performance and resilience with low overhead. Prefix hijack is used as an example to illustrate the strength of NDN adaptive forwarding.

- We investigated failure handling and role of routing in NDN (Chapter 5). An Interest NACK mechanism is proposed to speed up network fault detection and notification. With the forwarding plane being able to handle link failures effectively, the role of NDN routing is reduced to bootstrapping the forwarding process and handling link recovery.

- We studied hop-by-hop multipath congestion control in NDN (Chapter 6). An effective dynamic Interest limiting mechanism is presented, including two novel methods for NDN congestion detection and a fair Interest limiting algorithm to provide fairness among multiple flows.

CHAPTER 4

FORWARDING PLANE DESIGN

In this chapter we present a concrete forwarding plane design which aims to address the limitations of CCNx and fully exploit NDN's adaptive forwarding feature. We develop a detailed design for two key data structures of NDN: PIT and FIB, and explain how forwarding performance is collected and stored by each router. Then we present an improved interface ranking mechanism taking both routing information and forwarding performance into consideration. Finally, we propose a new forwarding strategy named BestRoute based on the interface ranking. At the end of this chapter, we use prefix hijack as an example to illustrate how our design is able to handle network faults effectively through adaptive forwarding.

4.1 PIT Design

PIT maintains per-datagram forwarding state (Figure 4.1). A PIT entry is created for each requested name. It contains a list of nonces that have been seen for that name, a list of incoming interfaces from which Interests for that name have been received, as well as a list of outgoing interfaces to which the Interest has been forwarded. In a PIT entry, each incoming interface records the longest lifetime of the Interests it received; when the lifetime expires the incoming interface is removed from the PIT entry, and the entire PIT entry is removed when all its incoming interfaces have been removed. Each outgoing interface records the time when the Interest is forwarded via this interface, so that RTT can be computed when Data packet returns. The RTT measurement is then used to update the RTT estimate for the corresponding name prefix stored in the FIB (Section 4.2).

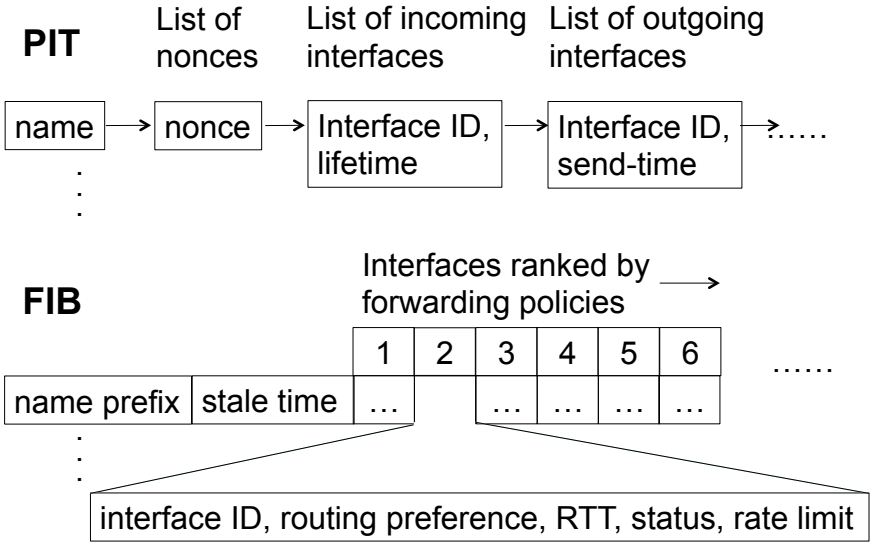


Figure 4.1: Forwarding State in PIT and FIB

4.2 FIB Design

NDN FIB differs from IP FIB in two fundamental ways. First, an IP FIB entry usually contains a single best next-hop, while an NDN FIB entry contains a ranked list of *multiple* interfaces. Second, an IP FIB entry contains nothing but the next-hop information, while an NDN FIB entry records information from both routing and forwarding planes to support adaptive forwarding decisions (see Figure 4.1)[1].

4.2.1 Routing Plane Information

FIB entries are added for all name prefixes announced in routing. When a name prefix disappears from routing, it is not immediately removed from the FIB, but kept for a *stale time* period or longer, if Interests under that prefix continue to be satisfied. This helps reduce unreachability caused by routing convergence, when some reachable prefixes may undergo temporary withdrawals.

For each name prefix, its FIB entry lists all interfaces that are allowed by rout-

[1]Here we assume there is an NDN routing protocol that disseminates routing information and computes the routing table for each router.

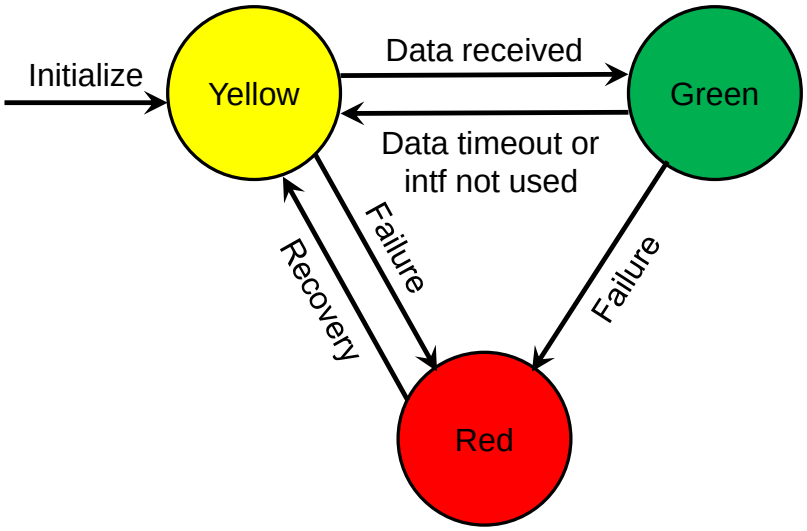


Figure 4.2: Interface Color Transition

ing policy, together with their associated preferences. Routing preference reflects routing policy as well as path cost, typically calculated using static link metrics; it is one of the inputs that we use to rank the interfaces.

4.2.2 Forwarding Performance Information

A FIB entry records the working status of each interface with regard to data retrieval (see Figure 4.1). Interface status is per-name-prefix-per-interface. To search for the best way to represent this status, we start by experimenting with a simple coloring scheme:

- Green: the interface can bring data back.
- Yellow: it is unknown whether the interface may bring data back.
- Red: the interface cannot bring data back.

The transition between different color is demonstrated in Figure 4.2. When a new FIB entry is created or a new interface is added to a FIB entry, the interface's initial status is Yellow. It turns Green when Data flows back from that interface. A

Prefix	Stale Time	Interfaces				
		ID	Routing Preference	RTT	RTO	Status
/foo	120s	3	50	45ms	50ms	Green
		2	70	60ms	80ms	Yellow
		4	100	N/A	N/A	Yellow
		1	110	N/A	N/A	Yellow

Figure 4.3: A Sample FIB Entry

Green interface turns Yellow when a pending Interest times out (i.e., no Data comes back within the expected time)[2], or after Data ceases flowing for a certain amount of time. An interface is marked Red when it goes down. Green interfaces are always preferred over Yellow ones; Red interfaces should never be used to forward Interests.

A FIB entry also maintains a per-interface estimate of the RTT to retrieve data. It is a moving average of RTT samples taken every time a Data packet is received over the corresponding interface. This RTT estimate is used in setting up a *retry-timer*, which serves two purposes: (1) before the timer expires, subsequent Interests carrying a name that already exists in PIT will be suppressed because the router is waiting for Data to be retrieved by the previously forwarded Interest; (2) after the timer expires, the router will retry alternative interfaces upon receiving a consumer retransmission.

Figure 4.3 shows a sample FIB entry for name prefix */foo*. The routing preference for each interface is computed by the routing protocol. There is no RTT information recorded for interfaces 1 and 4 because they haven't been used yet. The RTO is computed based on the RTT history. Because only interface 3 is currently in use, it is marked Green while others are marked Yellow.

[2]When there is a timeout on the interface, it indicates something may be wrong with the current path. The interface is marked Yellow, even though the problem may or may not be due to the local interface. If any other Green interface exists, it will be used. If no Green interface exists, next incoming Interest will be still forwarded to the current interface if it is the best option.

Pseudo-code 2 NDN-BestRoute Interest Processing

```
1: function PROCESS(Interest)
2:     Name ← Interest.Name
3:     if Data ← ContentStore.Find(Name) then
4:         Return(Data)
5:     else if PitEntry ← PIT.Find(Name) then
6:         if Interest.Nonce ∈ PitEntry.NonceList then
7:             Return
8:         end if
9:         Add Interest.Interface to PitEntry.Incoming
10:        if PitEntry.RetryTimer is expired then
11:            Forward(Interest, PitEntry)
12:            Return
13:        end if
14:    else
15:        PitEntry ← PIT.Create(Interest)
16:        PitEntry.Incoming ← Interest.Interface
17:        Forward(Interest, PitEntry)
18:    end if
19: end function
```

4.2.3 Interface Ranking

Interfaces in a FIB entry are ranked in order to help forwarding strategy choose the best interface(s) to use. When a router learns a new name prefix from routing, it ranks the interfaces for this prefix based on routing preference, since no forwarding performance has been observed yet. When information about forwarding performance becomes available, *forwarding policy* adjusts the interface ranking by taking into consideration both types of information.

A wide variety of forwarding policies can be supported in an NDN network. For example, if the policy is simply "follow routing", the interface ranking will be solely

Pseudo-code 3 BestRoute Forwarding Strategy

```
1: function FORWARD(Interest, PitEntry)
2:     if FibEntry ← FIB.Find(Interest.Name) then
3:         for each interface in FibEntry by rank do
4:             if interface ∉ PitEntry.Outgoing and
5:                interface ∉ PitEntry.Incoming then
6:                 Set PitEntry.RetryTimer
7:                 Transmit(interface, Interest)
8:                 Add interface to PitEntry.Outgoing
9:                 Return
10:            end if
11:        end for
12:        GiveUp(Interest)
13:    else
14:        GiveUp(Interest)
15:    end if
16: end function
```

determined by routing preference; if the policy is "the sooner the better", an interface with smaller RTT will be ranked higher. Yet another example is to give higher preference to the current working path, which helps ensuring performance stability experienced by applications. Note that forwarding policies are different from *routing policies* [1]. Routing policies determine which routes to be made available to the forwarding plane. In the case of BGP [55], routing policies are reflected in routing announcements, which propagate from data producers to consumers. *Forwarding policies*, on the other hand, determine which routes actually get used and in which order. They are reflected in Interest forwarding, which go from the consumers towards producers[3].

[3]It is conceivable that forwarding policies could also decide whether the outgoing interfaces for a given name prefix should be, or should not be, limited to those learned from the routing protocols.

Name	Nonces	In	Receive Time	Out	Send Time	Retry Timeout
/foo/bar/0/0	246234	1	05/20/2014 13:21:43.02	3	05/20/2014 13:21:43.02	05/20/2014 13:21:43.07

(a) New Interest

Name	Nonces	In	Receive Time	Out	Send Time	Retry Timeout
/foo/bar/0/0	246234, 1532689	1	05/20/2014 13:21:43.02	3	05/20/2014 13:21:43.02	05/20/2014 13:21:43.07
		4	05/20/2014 13:21:43.04			

(b) Suppressed Subsequent Interest

Name	Nonces	In	Receive Time	Out	Send Time	Retry Timeout
/foo/bar/0/0	246234, 1532689	1	05/20/2014 13:21:43.02	3	05/20/2014 13:21:43.02	05/20/2014 13:21:43.07
		4	05/20/2014 13:21:44.01	2	05/20/2014 13:21:44.01	05/20/2014 13:21:44.09

(c) Forwarded Subsequent Interest

Figure 4.4: Sample PIT Entry for New and Subsequent Interests.

4.3 BestRoute Forwarding Strategy

Given the information stored in PIT and FIB, a router's strategy module determines when and which interface to use to forward an Interest, making forwarding decisions adaptive to network conditions. Our design includes the handling of new and subsequent Interests. The overall Interest processing mechanism is illustrated by Pseudo-code 2; Figure 4.4 shows a sample PIT entry based on the sample FIB entry in Figure 4.3.

New Interest: When a newly arrived Interest does not find a match in either Content Store or PIT, a new PIT entry will be created (see Figure 4.4(a)). The new Interest is then forwarded to the highest-ranked available Green interface if one exists, otherwise the highest-ranked available Yellow interface will be used (see

Pseudo-code 3). When the router forwards the Interest, it starts a retry-timer, which is set to a small value at the time scale of RTT plus variance.

Subsequent Interest: If an Interest matches an existing PIT entry, and its nonce does not exist in the nonce list, this Interest is considered a subsequent Interest. A subsequent Interest can be a retransmission from the same consumer, or originated from a different consumer requesting the same piece of Data. When a subsequent Interest is received before the retry-timer expires (see Figure **??**), it will not be forwarded because the router is still waiting for Data to be brought back by a previously forwarded Interest. Otherwise, this subsequent Interest will trigger the router to retry the Interest and start the retry-timer (see Figure 4.4(c)).

In the above situations, if a router needs to forward an Interest but it has no Green or Yellow interface left that has not been tried, it will give up and delete the PIT entry. Routers perform best effort to get around forwarding problems through local retries, however consumers are ultimately responsible for re-expressing the Interest if they still want the data.

Probing Interest: When there is no fault in the network, a router will keep using the highest ranked interface for all traffic under each name prefix. Thus all other interfaces will be marked Yellow because they are not in use. If a router wishes to switch traffic under certain prefix from a Green interface to a Yellow interface (e.g., due to routing changes), it will not do so immediately since the working status of Yellow interfaces is unknown. Instead, it will send a probing Interest to the Yellow interface to test whether it is able to bring Data back. If the test succeeds, the interface will be marked Green and traffic switch will then take place.

4.4 Prefix Hijack: A Case Study

In this section we use simulations to evaluate how well NDN's adaptive forwarding plane works and whether it achieves robust packet delivery under prefix hijack scenario. We contrast with the behavior of IP to illustrate the difference between NDN's *stateful* forwarding plane and the traditional IP's *stateless* forwarding plane.

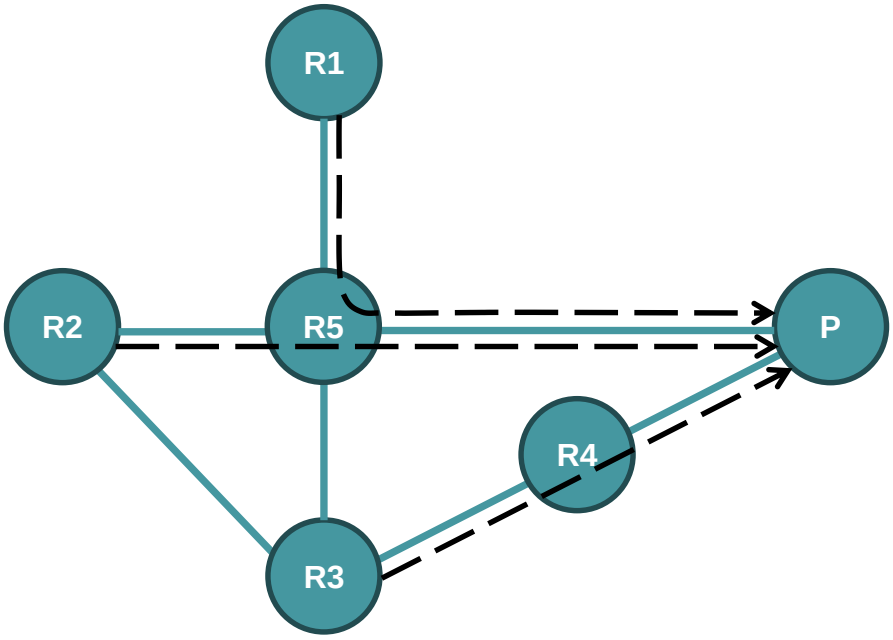


Figure 4.5: Prefix hijack: IP and NDN-BestRoute forwarding before the attack.

We also include the comparison with Path Splicing [48], an adaptive multipath enhancement to IP, to observe the differences in the performance between NDN and Path Splicing and to understand the underlying causes of these differences. We implemented our forwarding plane design in ndnSIM [12], an NS-3 [9] based NDN simulator. We also implemented Path Splicing in NS-3 according to [48]. In the rest of this section, we first present a brief description of Path Splicing, then the simulation setup, followed by our simulation results.

4.4.1 A Simple Example

We use a simple example shown in Figures 4.5, 4.6 and 4.7 to explain how IP and NDN-BestRoute behave in blackhole hijack attacks. In these figures, A is the attacker, P is the producer, and other nodes are good routers. Figure 4.5 shows the forwarding paths before the attack. Since there is no attacker, NDN-BestRoute and IP nodes have the same forwarding paths.

Figure 4.6 shows the forwarding paths for IP during the attack. In this figure,

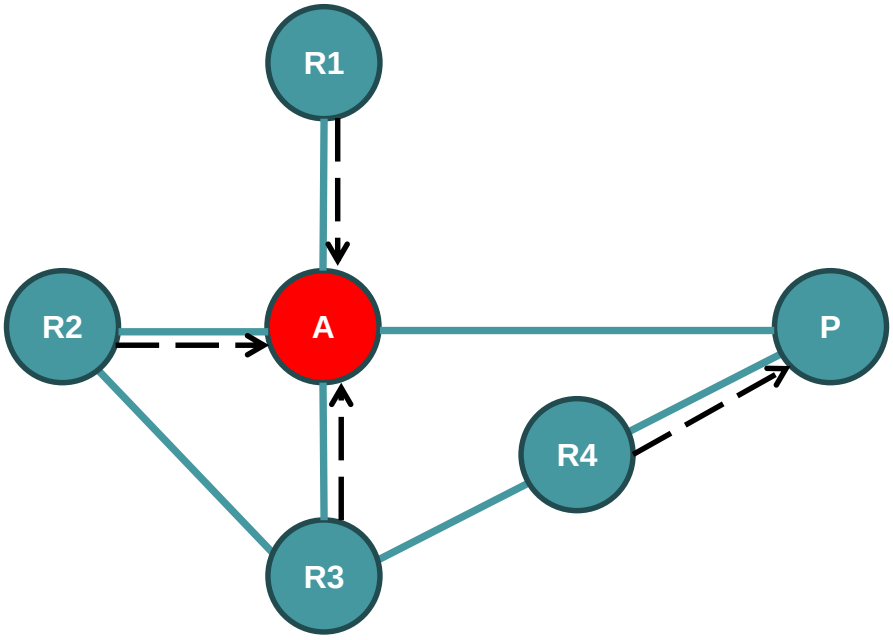


Figure 4.6: Prefix hijack: IP forwarding during the attack.

$R5$ becomes the hijacker and starts announcing P's prefixes through the routing protocol. All $R1$, $R2$ and $R3$ switch their forwarding paths toward A because they are closer to A than P; thus their packets are successfully hijacked. $R4$ is not affected by the hijack because it is closer to P than A.

The forwarding paths for NDN-BestRoute during the attack is shown in Figure 4.7. Similar to the IP case, $R1$'s packets will be hijacked because A is its only neighbor, and $R4$ will not be affected by the attack, so we focus on the other cases. We first consider the routers who do not have the attacker on their original shortest path to reach the producer, e.g., $R3$. When A announces P's name prefix, the routing system would rank the attacker path ($R3$-A) higher than the producer path ($R3$-$R4$-P). With NDN's intelligent forwarding plane, since the existing interface ($R3$-$R4$) to the producer has been bringing data back, it is colored Green. A's false routing announcement makes the interface to the attacker ($R3$-A) ranked higher by routing, but when $R3$ tries its out by sending an Interest to it now and then, it does not return a Data packet, thus it remains Yellow. Unlike IP routers, NDN routers

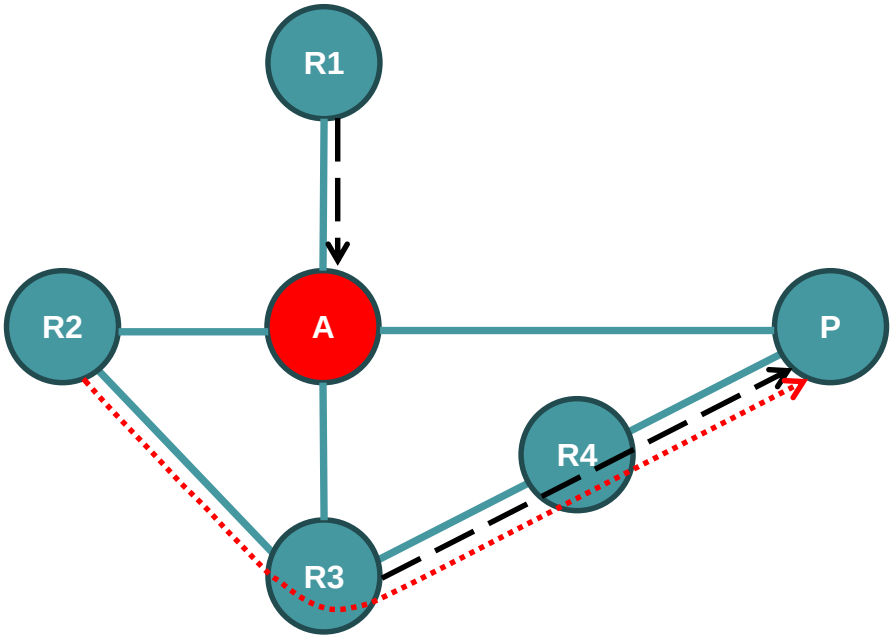


Figure 4.7: Prefix hijack: NDN-BestRoute forwarding during the attack.

do not direct traffic to a higher ranked path until it is observed to perform well.

Let us now consider the routers who have the attacker on their shortest path to the producer, e.g., $R2$. $R2$'s Interest packets will be blackholed by the attacker, thus $R2$'s retry-timer expires without getting data packets back, and interface ($R2$-A) will be marked Yellow; $R2$ does not automatically retry alternative interfaces because there is not any feedback. End consumers will timeout and retransmit the Interests. Upon receiving a retransmitted Interest, if $R2$'s retry-timer has expired, $R2$ will retry a different interface than the previously failed one. When the retransmitted Interest, following a different path, arrives at the producer and brings back the requested Data packet, $R2$ will mark the working interface ($R2$-$R3$) Green and keep using the working interface.

4.4.2 Path Splicing

Path Splicing enables source hosts to utilize multiple paths in IP packet delivery. In Path Splicing, each router maintains multiple routing tables, called *slices*. All the

routers in a network are pre-configured with the same number of slices. A router computes its first routing table (the original slice) by using the standard routing protocol metrics, it then computes the rest of the slices by using the same topology and the same shortest-path algorithm but different sets of link weights, which are generated by randomly perturbing the original set of link weights learned from the routing protocol.

When a host sends a packet, it adds an ordered list of *tags* to the packet header. Each tag is an index to the slice to be used at each hop, and routers forward the packets according to the tags. A tag is removed from the list after it is used; when a packet's tag list becomes empty, routers will use the original slice to forward the packet. End-hosts can choose a *different* path for a packet by tagging it differently, although the hosts do not know the exact path the packet may take.

When a host detects a network fault, e.g. a packet loss, it retransmits the packet by using a different tag list. The recommended operation is for the end-host to examine the current list of tags and change each tag with a probability of 0.5. If a tag is to be changed, the new tag will be randomly chosen from the available slice numbers. For a client/server application to successfully retrieve a data packet, the tags on request and reply packets must both identify a working path. In simulating Path Splicing, unless otherwise specified, we use 10 slices by default and allow up to 20 retransmissions.

On the surface, Path Splicing seems similar to NDN forwarding in that, once hosts detect network delivery problems, they attempt to get around the problems by trying different paths. The additional state, i.e., multiple slices installed at each Path Splicing routers, enables the path adjustments by end hosts. However, there exist two fundamental differences between the two. First, the adaptability in Path Splicing can only be done by end-hosts, as opposed to by any nodes in an NDN network. Second, given the end hosts know neither the network topology nor the problem location, they adjust the path by random selections, as opposed to the informed decision by NDN nodes based on their observed performance and feedbacks. These functional differences lead to significant performance differences

in the simulation results, as we present below.

4.4.3 Simulation Setup

We examine the packet delivery performance of NDN-BestRoute, IP and Path Splicing under the prefix hijack scenario, in which an attacker announces the victim's prefix and silently drops the traffic. Since we focus our evaluation on the effectiveness of stateful adaptive forwarding, we disabled NDN's in-network caching in the simulation. The simulations are done using the Sprint PoP-level topology [61], the same topology used in [48] to show the improvement of Path Splicing over native IP. This topology contains 52 nodes and 84 links, including 19 single-homed nodes.

In all simulations we assume that each IP (NDN) router announces one IP (name) prefix, and we precompute the best paths and install them in each router's FIB. For IP, a single shortest path is installed for each IP prefix at a router. For Path Splicing, a number (default is 10) of slices are installed per IP prefix. For NDN, each router maintains a list of all outgoing interfaces ranked by the routing path length for each name prefix in the FIB. In each simulation run, we create a prefix hijack scenario shortly after the simulation starts and observe the behavior of each scheme. Since NDN clients retrieve data by sending an Interest packet first, for fair comparison we use the same traffic pattern in IP and Path Splicing simulations, i.e., we run client/server applications where a client first sends a request to the server, which then sends a reply data packet back to the client. These request/reply packets have the same sizes as Interest/Data packets in NDN.

4.4.4 Simulation Results

In prefix hijack simulations, an attacker announces the victim's prefix and silently drops all data traffic it receives, creating a "blackhole" of victim's traffic. This is exactly what happened during the well publicized incident of YouTube's prefix hijack [4].

[4]In an NDN network, a hijacker may return bogus Data instead of silently dropping Interest packets. How to handle bogus data attacks is beyond the scope of this dissertation. Interested

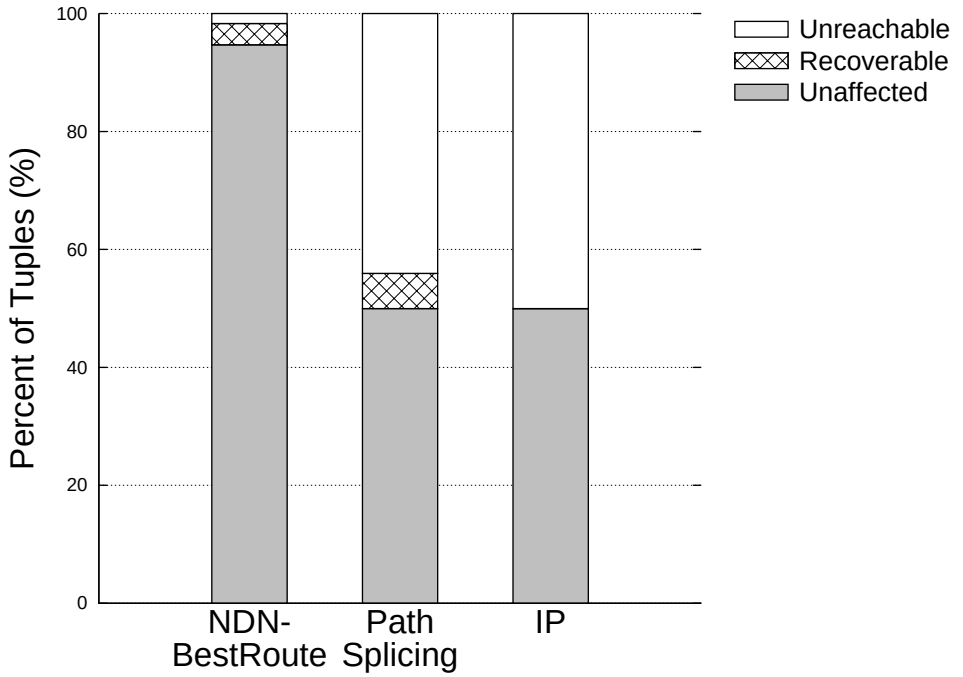


Figure 4.8: Reachability during prefix hijack

End-hosts can detect the problem when they do not receive the content they are requesting. Traffic is said to be *affected* if it is forwarded towards the attacker, or unaffected if it is still forwarded to the true destination. If affected end-hosts have means to try other paths, they may find an alternative path to reach the true destination [69], in which case we say they have *recovered* from the hijack.

To simulate "blackhole" hijacks, in each simulation run we choose one node as the producer, one as the attacker, and the rest of the nodes as data consumers. The attacker announces to the routing system the IP prefix (for IP and Path Splicing) or the name prefix (for NDN) of the producer. We exhaust all combinations of (consumer, producer, attacker) tuples in the topology. We run NDN-BestRoute, IP, and Path Splicing to see whether the traffic will be affected and whether affected traffic can recover. The results are summarized in Figure 4.8.

Let us consider the result for running IP simulation first as it is easier to under-

readers can refer to [30] for more details.

stand. In an IP network, traffic will be affected if (1) the attacker is on the best path from the source to the destination, or (2) the attacker is closer to the source than the true destination. For the given topology, in less than 6% of the cases, the attacker is on the best path from the consumer to the producer and simply drops all requests, thus no consumer gets data back. In another 44% of the cases the attacker is closer to the consumer than the producer, thus traffic is affected as well. Note that in total 50% of all the cases traffic is affected and in the other 50% it is not. This is because the traffic that is affected under one producer/attacker pair will become unaffected when the producer and attacker switch roles. Since IP routers strictly follow the paths given by routing protocols, none of the affected traffic is recoverable. The end-hosts, although being able to detect the problem, cannot change the paths their packets take. For NDN-BestRoute, in the less than 6% of cases where the attacker is on the original best path from the consumer to the producer, all traffic is recoverable except the cases where the consumer or producer is single-homed to the attacker. For the remaining over 94% of the cases, consumer-producer communications are not affected by the hijack.

In simulating Path Splicing under the same hijack attacks, the results show the same 50% of affected traffic as in the IP case. Different from IP, however, when a client sends out a request packet and times out after RTT, it will retransmit the request with a different tag list, thus the packet will be routed along a different path. If the data packet comes back, the client knows the previous tags have worked and will keep using the same tags. Thus, Path Splicing enables end-hosts to influence path selection in an attempt to escape routing hijack. This capability, however, is rather limited because the available options are bounded by the number of pre-computed slices, and hosts perform *random* selections of tags because they have no knowledge about the network internals. Thus the recovered traffic is only 6% out of 50% of affected cases, and the random trials can take substantially long time. Figure 4.9 shows the number of retransmissions needed to find a working path with each router keeping 10 slices. In most cases consumers in Path Splicing need many retransmissions, while NDN consumers only need a few.

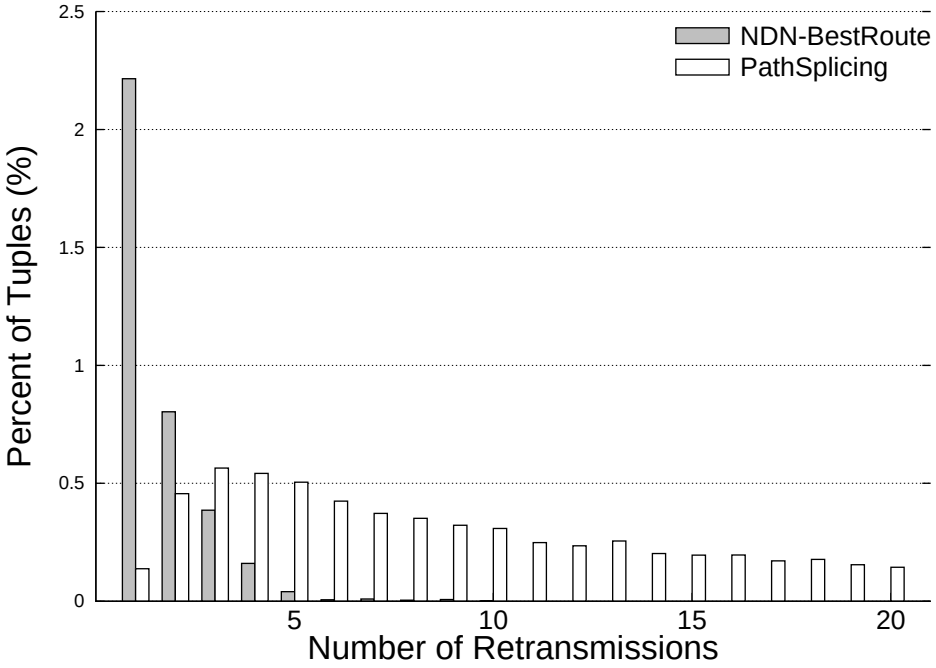


Figure 4.9: Retransmissions needed during prefix hijack

NDN-BestRoute is both faster and more effective in finding working paths because every NDN router is able to make its own informed decisions on which alternative paths to try. These decisions are based on the observation from previously forwarded Interest packets, which is made available by maintaining per-datagram state at each router. In addition, the ability to retry alternative paths by every router whenever a problem is noticed speeds up recovery and does not require any precomputed tables.

CHAPTER 5

FAILURE HANDLING

In this chapter, we first present an Interest NACK mechanism for fast fault detection and notification. Then we evaluate the packet delivery performance of NDN-BestRoute in link failure scenarios, and compare it against IP and Path Splicing. Our results show that NDN-BestRoute can effectively handle link failures at the forwarding plane without global routing convergence. This is a significant advance over today's IP networks, which mainly rely on routing to handle network failures. Hence we also investigate the role of routing in NDN in this chapter.

5.1 Interest NACK

In the original sketch of NDN [36], routers discover failures by timeout only. More specifically, when a router N forwards an Interest, it starts a timer based on the estimated RTT to the data producer. If the corresponding Data packet comes back before the timer expires, the RTT is updated; otherwise N will try alternative path if one exists, or otherwise give up. However, this timer-based fault detection can be relatively slow. Furthermore, when router N has exhausted its options and gives up, the unsatisfied Interest (which we call the *dangling state*) is left on the PIT of those routers between the consumer and N that the Interest has traveled through, until the Interest's lifetime expires. Such dangling state can potentially block other consumers from getting the same data, since the intermediate routers believe that they have already forwarded the Interest and just wait for the Data to return.

5.1.1 NACK Design

We address these issues by introducing *Interest NACK*. When an upstream NDN node N_u can neither satisfy nor further forward an Interest, it sends an Interest

NACK back to the downstream node N_d. If N_d has exhausted all its own forwarding options, it will send a NACK further downstream. Note that Interest packets flow from downstream node to upstream node, Data packets flow from upstream node to downstream node, and Interest NACKs are always sent from upstream to downstream.

An Interest NACK carries the same name and nonce as the original Interest, plus a NACK code explaining why the Interest cannot be satisfied or forwarded so that proper actions can be taken accordingly. Below are the NACK codes in our current design; additional codes may be added as the need arises.

- *Duplicate*: A pending Interest with identical name and nonce has been received earlier by the upstream node. This occurs if the Interest is looped back to the upstream node, or if some node forwarded multiple copies of the same Interest that happen to meet at the upstream node.
- *Give Up*: The upstream node has exhausted its forwarding options and cannot further forward the Interest.
- *No Data*: The upstream node (which can be either a router or the producer) does not have the requested data and has no path to forward the Interest.

In the absence of packet losses, every pending Interest is consumed by either a returned Data packet or a NACK. Returning NACKs brings two benefits to the system: it cleans up the pending Interest state much faster than waiting for the lifetime to expire, and it allows the downstream nodes to learn the specific cause of a NACK, so that they can take informed recovery actions. Note that an Interest NACK is different from an ICMP message; the former goes to the previous hop while the latter is sent to the source host, hence their effects are entirely different.

5.1.2 Updated Forwarding Plane Design

We make necessary changes to the forwarding plane design presented in Chapter 4 to accommodate this new packet type. First, interface status needs be updated

Pseudo-code 4 NDN-BestRoute Interest NACK Processing

```
1: function Process(NACK)
2:     PitEntry ← PIT.Find(NACK.Name)
3:     if PitEntry ≡ ∅ or
4:         PitEntry.RetryTimer expired or
5:         NACK.Nonce ∉ PitEntry.NonceList
6:     then
7:         Return
8:     end if
9:     Forward(NACK.Interest, PitEntry)
10: end function
```

upon receipt of NACKs. When a NACK with code "Duplicate", "Give Up" or "No Data" is received on an interface, the interface should be marked Yellow. Second, the forwarding strategy also needs to be updated. Before Line 7 of Pseudo-code 2, a NACK with code "Duplicate" should be returned before it stops processing. Similarly, NACKs with code "Give Up" and "No Data" should be returned before Lines 12 and 14, respectively of Pseudo-code 3. Third, Interest NACK will trigger path exploration. In Section 4.3, we let routers retry alternative paths only upon consumer retransmission after the retry-timer expires. The recovery process can be accelerated by start retrying immediately upon receipt of Interest NACK.

Pseudo-code 4 illustrates how Interest NACK is processed by NDN routers. When an Interest for data name M is returned in the form of a NACK, if the retry-timer is still running, a router will send an Interest with the same name and nonce to the next highest-ranked available interface. Ideally, we want routers to try a few alternatives but not for too long (the application may have moved on without the Data) nor consume too much network resource. After the retry-timer expires, therefore, a router should stop trying alternative interfaces upon NACKs to limit the overhead caused by local retry. However, the PIT entry for M is kept until the Interest's life time expires, during which time retry can still be triggered by a

subsequent Interest of the same name.

5.2 Failure Handling with Adaptive Forwarding

Robust packet delivery in the presence of link failures is the classic measure used in Baran's early work to evaluate the resilience of packet switched networks [16]. In this section, we run simulations for NDN-BestRoute, IP and Path Splicing using the Sprint PoP-level topology and the same setup as described in Section 4.4.3. We measure packet delivery performance before the routing protocol adapts to the failure, as is done in [48]. Our goal is to measure how well, or poorly, the forwarding plane can perform before routing converges.

We simulate link failures as follows. After the initial simulation warm-up period for NDN routers to obtain forwarding plane performance, we associate each link with a uniform failure probability and fail links randomly according to this probability, producing one failure scenario. All packets sent over a failed link are dropped. If the original best path from A to B contains at least one failed link, the traffic flow between A and B is considered affected; if the network is able to switch packets from A to B to an alternative working path without routing convergence, we say this traffic flow can recover from the failure. We run NDN-BestRoute, IP, and Path Splicing respectively to see how many flows (i.e., consumer-producer pairs) can recover from given failures. We run each experiment 1000 times (i.e. running each of NDN-BestRoute, IP, and Path Splicing over 1000 randomly generated failure scenarios for each link failure probability) and our results are presented below.

Figure 5.1 shows the percentage of host pairs that cannot recover, averaged over 1000 failure scenarios for each link failure probability. The "best possible" curve is the percentage of host pairs that are physically disconnected by the failed links, thus no solution can achieve disconnection ratio below this curve. For Path Splicing, s and r stand for numbers of slices and allowed retransmissions, respectively. The NDN-BestRoute curve overlaps with the best possible one, meaning that a consumer is able to retrieve data from a producer as long as *any* working path exists in between.

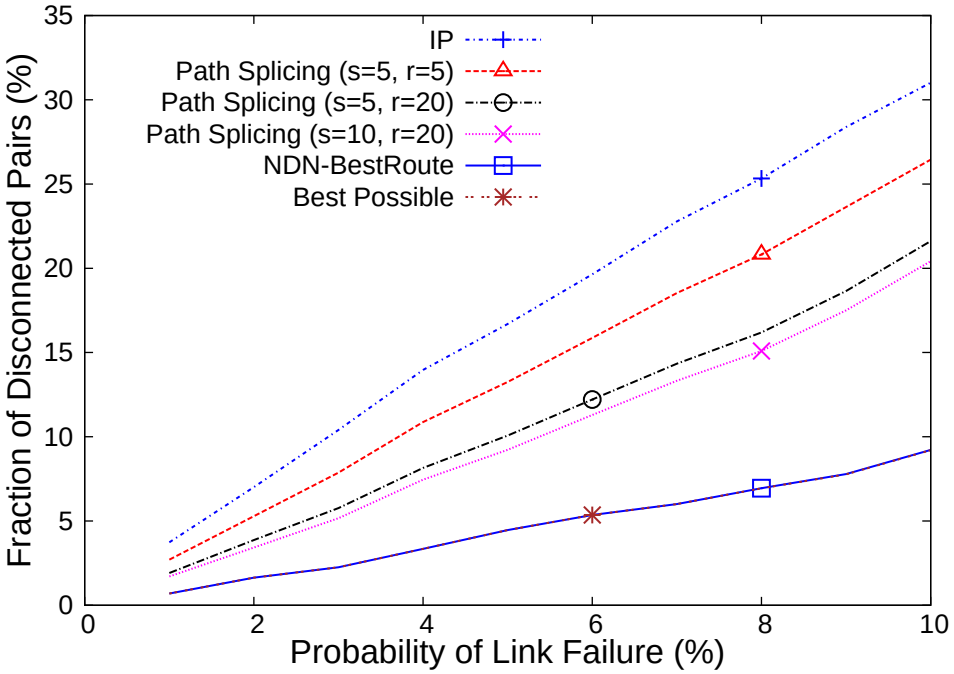
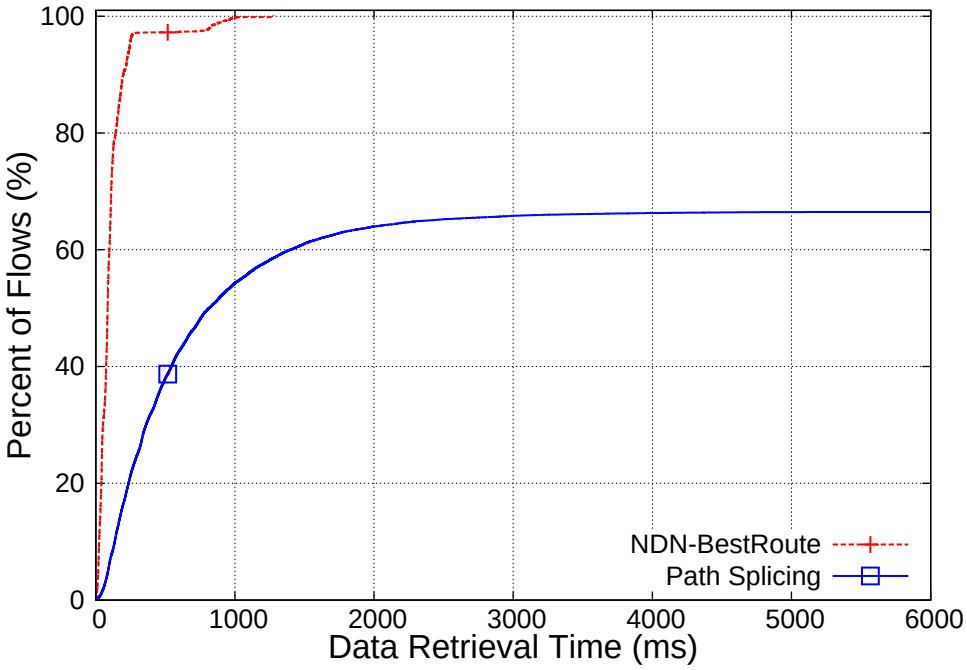


Figure 5.1: Reachability after link failures

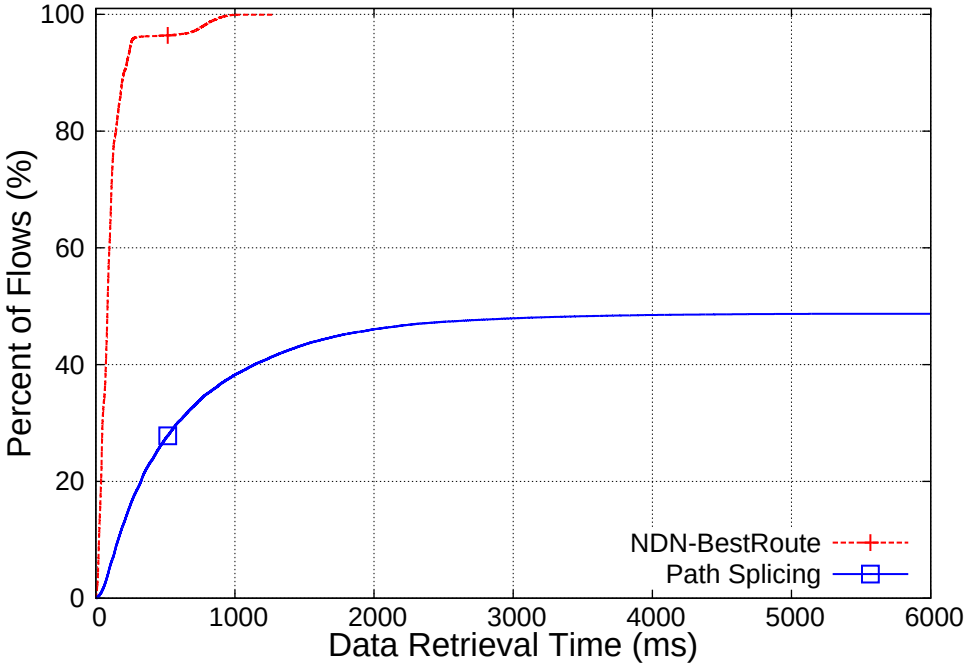
Not only can NDN-BestRoute recover from link failures, it also finds alternative paths quickly.

Figure 5.2 shows the Cumulative Distribution Function (CDF) of data retrieval time from 1000 failure scenarios[1], which is from the first transmission of a request/Interest by the consumer to the arrival of the requested Data. It includes the time of possible retransmissions by the consumer in Path Splicing and NDN-BestRoute, and router retries in NDN-BestRoute. With 1% link failure probability, the median data retrieval time in NDN-BestRoute is 85 ms, and the 90th-percentile 198 ms; when the link failure probability is 10%, the median is not changed while the 90th-percentile increases slightly to 203 ms. The alternative paths that NDN-

[1]In Figure 5.2 and 5.3 we only consider packet exchanges that are affected by failures but recoverable (i.e., the failed link is on the original best path, but at least one alternative working path exists after the failure). These figures do not consider IP because although end hosts in an IP network may also retransmit, the packet will still be forwarded along the failed path, thus all affected pairs fail.



(a) 1% Link Failure Probability



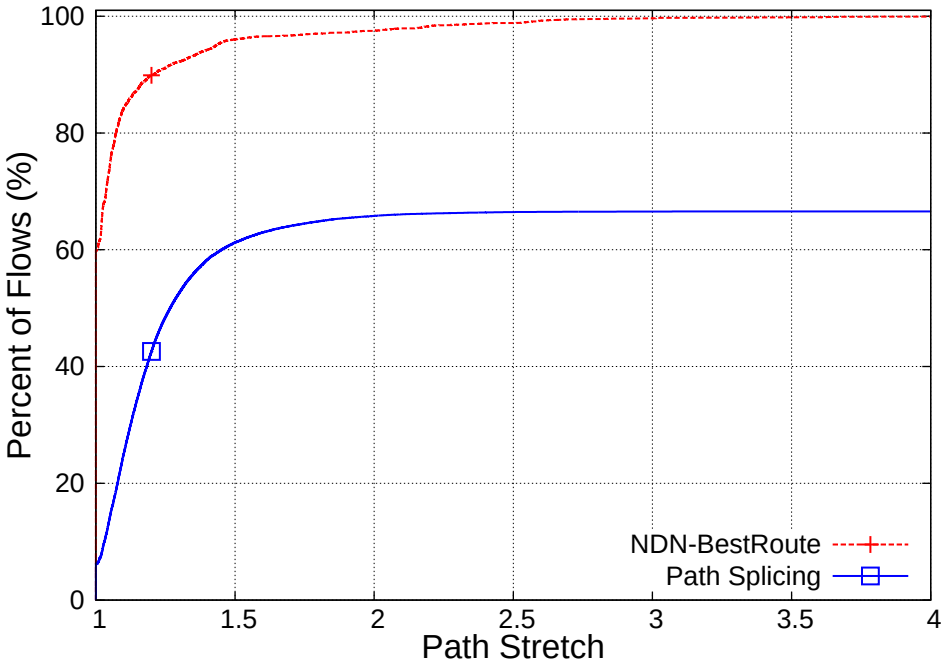
(b) 10% Link Failure Probability

Figure 5.2: Data retrieval time under different link failure probability

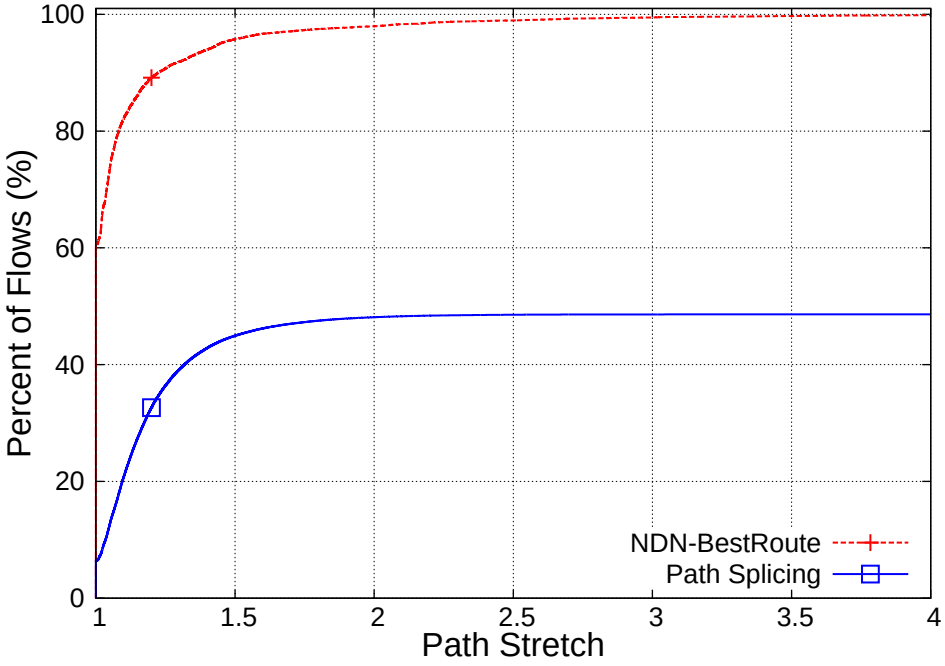
BestRoute finds are also of good quality. Figure 5.3 shows the CDF of path stretch, which is the path length ratio of the selected path over the shortest path after failures. Under either 1% or 10% link failures, about 60% of paths in the NDN network have stretch of 1, which means that the adaptive forwarding plane found the shortest paths; the 90-percentile of path stretch increases marginally from 1.21 to 1.22 when failure probability increases from 1% to 10%.

NDN-BestRoute's resiliency to failures is due to its fast local recovery. When a link fails, the NDN router will mark this interface Red and try other interfaces following its forwarding strategy. If the router has tried and failed all possible interfaces, it returns a NACK to downstream node, which will then explore its own alternatives. When the Interest brings back Data via a working interface, this interface will be labelled Green and used to forward future Interests. Therefore, network retry starts from where the failure happens and pushes back towards the consumer until a working path is found, if one exists. Since a FIB entry's outgoing interfaces are ordered not only based on routing preference but also observed working status, a router tries most promising interfaces first, which leads to finding working paths sooner and finding good working paths.

In contrast, since IP's forwarding plane has no adaptability of its own, its percentage of disconnected pairs in Figure 5.1 reflects the number of host pairs whose shortest paths contain at least one failed link. An IP network relies on routing protocols to handle link failures. Once a policy-compliant working path is found, routing will converge and packets will be delivered along the new path. However the convergence process, which includes failure detection, route propagation, route computation, and FIB update, can take time to complete. Measurements have shown that routing convergence may take tens of seconds to minutes, during which applications suffer from packet losses and long delays. Therefore, many efforts, including Path Splicing, have gone to improving packet delivery during the transient period after failures. The essence of these IP-based solutions is to find a loop-free path, either precomputed or computed on-the-fly, without relying on routing.



(a) 1% Link Failure Probability



(b) 10% Link Failure Probability

Figure 5.3: Path stretch under different link failure probability

Path Splicing improves packet delivery under link failures over IP[2]. When links fail and packets are dropped, the consumer host will time out the request after RTT and retransmit using different tags. Figure 5.2 and 5.3 show data retrieval time and path stretch of Path Splicing under different link failure probability. Among all the affected but recoverable host pairs, 66% and 49% of them succeed in data retrieval when the link failure probability is 1% and 10%, respectively[3].

The performance of Path Splicing depends on the number of slices and the number of retransmissions allowed. The former represents the number of choices for alternative paths, the latter represents how many attempts are allowed to find a working path among the choices. We experimented with different settings to understand the impacts of these two tuning knobs. If the maximum number of retransmissions allowed is increased from 5 to 20 while the number of slices is kept at 5, the number of disconnected pairs of Path Splicing reduces significantly as shown in Figure 5.1. But increasing the number of slices from 5 to 10 only makes a small improvement. This observation suggests that, for the specific topology used in this simulation, a small number of slices can provide adequate path diversity to get around the link failures, however end-hosts may do many random trials before they succeed. Figures 5.2 & 5.3 show that for those flows which succeed in recovery by Path Splicing, they takes much longer time to retrieve data than NDN-BestRoute and in general the found paths are longer. This is because end-hosts randomly pick different tags, without knowing where the failures are to make an informed selection. Furthermore, such recovery attempts are initiated by end hosts after timeout, which necessarily takes

[2]The result in Figure 5.1 appears to be worse than that in Figure 6 of [48], because the result in [48] is for one-way traffic, and our result here is for two-way traffic, which requires working tags for paths in both directions. When we run simulations for one-way traffic only, the result is similar to that in [48].

[3]Figure 5.1 shows that, with link failure probability of 10%, 31% of all host pairs are affected (i.e. the failure percentage of IP traffic), 9.2% are unrecoverable, hence 21.8% of host pairs are affected but recoverable. NDN-BestRoute can recover all the 21.8%, while Path Splicing can only recover 10.6% of them. Hence only about 49% of recoverable pairs succeed in Path Splicing as shown in Figure 5.2(b) and Figure 5.3(b).

much longer time compared to NDN routers performing local recovery.

5.3 Role of Routing

Simulation results in the previous section shows that NDN routers are able to handle network failures locally without relying on global routing convergence. This fundamental change prompts us to rethink the role of routing in NDN networks: does it still need a routing protocol? If so, what impact may an intelligent forwarding plane have on the design and operation of NDN routing protocols? In the remainder of this chapter we seek answers to these questions and investigate the role of routing in NDN.

Since NDN's forwarding model is a strict superset of the IP model, any routing scheme that works well for IP should also work well for NDN [36]. However, today's IP routing protocols suffer from issues such as slow convergence or poor scalability. In addition, NDN has a smart and powerful forwarding plane, which is able to take over part of routing's responsibility in IP. In this subsection, we first review IP routing, and then rethink the role of routing in NDN.

5.3.1 Routing in IP

IP's routing plane is intelligent and adaptive, but its forwarding plane is stateless and strictly follows routing. Therefore the routing plane is also regarded as the control plane. Routing is responsible for building the routing table and maintaining it in face of network changes, including both long-term topology and policy changes as well as short-term churns. When there is a change in the network, routers need to exchange routing updates with each other in order to reach new global consistency. The time period after a change happens and before all routers agree on the new routing state is called the routing convergence period. IP routing protocols need to converge fast in order to reduce packet loss and resume packet delivery after network changes.

However, fast routing convergence is challenging in large operational networks.

The fundamental reason is that it conflicts with other design goals for routing protocols, i.e., routing stability and scalability. Routing stability ensures stable routing paths within the network. It is important for applications that suffer from RTT fluctuation; it also helps routers achieve traffic engineering goals. Routing scalability is essential for supporting a large number of nodes, links and prefixes[4] in the network. For link-state routing, each router knows the entire topology. These protocols can converge fast, but at the cost of poor stability and limited scalability. For distance/path-vector routing, routers do not have a full knowledge of the topology. They are able to achieve better scalability, but the convergence time may be as long as tens of minutes. Below we use link-state routing as an example to explain the issues with today's routing protocols.

The routing convergence period can be divided into four phases: failure detection, update propagation, route computation and FIB update. In link-state routing, routers periodically exchange HELLO messages to maintain connection: if no HELLO message is received within the DEAD interval, the link is considered down. Previous research ([13, 29]) recommended setting the HELLO interval to be on the order of milliseconds in order to detect failures quickly. However, this not only increases overhead but also affects routing stability, since a temporarily congested link may be mistakenly considered fluctuating down and up. After a link failure is detected, attached routers need to generate routing updates and propagate them to the rest of the network; when a router receives a routing update, it needs to recompute the routing table. To achieve fast routing convergence, all these steps should be done as quickly as possible. However, if the network is unstable (e.g., there is a flapping link), generating routing updates and recomputing routing table frequently will increase bandwidth and computation overhead as well as harm routing stability. At the same time, shortest path first (SPF) computation time increases with the size of the network; FIB update time depends on the number of prefixes. To achieve fast convergence, both the network size and the number of prefixes need to be limited,

[4]Supporting large number of prefixes is particularly important in NDN since the number of name prefixes will be orders of magnitude larger than the number of IP prefixes in today's Internet.

leading to poor scalability.

There are mechanisms to improve link-state routing stability and scalability. Dynamic timers improve routing stability by limiting the rate of update generation and SPF computation. However, these timers are increased exponentially each time, potentially increasing convergence time significantly when the network is unstable. Therefore, short initial timers have been suggested [29]. *Area* was introduced to improve routing scalability [49]. However, it leads to sub-optimal paths between areas and increases the complexity of configuration. Although inter-area routing can utilize distance-vector or path-vector routing algorithms that may scale better, they converge much slower.

In summary, it is hard to achieve fast convergence, stability and scalability simultaneously in a routing protocol. If there are other mechanisms to handle failures without global convergence, the requirement on fast convergence can be relaxed, making it possible to achieve routing stability and scalability.

5.3.2 Routing in NDN

In NDN, the forwarding plane is the actual control plane since the forwarding strategy module makes forwarding decisions on its own. This fundamental change prompts us to rethink the role of routing. The first question is whether NDN still needs routing protocols. Conventionally, routing protocols are responsible for disseminating topology and policy information, computing routes and handling short-term network changes. For NDN to work without routing, routers need to be able to do the following things efficiently: 1) retrieve Data when the network is stable; 2) handle link failures; and 3) handle link recovery. Can NDN achieve these solely with the forwarding plane?

Another question that arises is if NDN does need routing protocols, how will they be different from today's existing routing protocols? With the intelligent and adaptive forwarding plane, can some of the routing plane's functionality be offloaded to the forwarding plane, and which? In addition, how will the design and operation of routing protocols benefit from this shift of functionality? In the next subsection

we try to give answers to these questions.

5.4 Routing and Forwarding Coordination

In this subsection, we seek answers to the questions raised above. Section 5.2 shows that NDN routers are able to detect and recover from link failures effectively without routing. In this subsection we focus on the other questions: whether NDN routers can efficiently retrieve Data and handle link recovery without routing. We show that NDN does need routing protocols to help bootstrap the forwarding process and handle link recovery. In addition, we specify how the routing plane coordinates with the forwarding plane, and present a simple method to improve routing stability and scalability in NDN.

5.4.1 Interface Ranking

The forwarding plane design presented in Chapter 4 assumes interfaces are ranked by routing preference. Can NDN routers retrieve Data efficiently without such interface ranking? The answer is, no, they cannot. In the extreme case, we can implement a forwarding strategy that floods every Interest to all available interfaces. This way we can always retrieve Data quickly through the best paths. However, it will also incur substantially large overhead. We can also implement a strategy that randomly explore the interfaces one-by-one in a round-robin fashion. Given enough time, routers should be able to find working paths since all possible paths will be explored. One big issue with this method is that path exploration may take extremely long time as shown in Section 5.5.

Consequently, NDN routers need good interface ranking to help bootstrap the forwarding process. The responsibility of providing interface ranking lies in the routing protocols. Existing routing algorithms such as link-state or distance/path-vector routing can be used to rank the interfaces[5]. The details are explained as

[5]The case of path-vector routing, i.e., BGP is more complex because it also takes routing policy into consideration. How to accommodate routing policy in interface ranking is out of the scope of

follows.

Link-State Routing: Link-state routing protocols store the entire network topology in the link-state database (LSDB), making it possible to compute optimal interface ranking. Suppose a node N has n interfaces I_1 .. I_n. For Data provided by node M, we rank these interfaces using $C^M_{N,k}$, which is the cost of the best path from N to M through interface I_k. One simple method to compute $C_{N,k}$ for all destinations is to remove all interfaces except I_k from N's LSDB, and run *Dijkstra's algorithm* to compute the shortest paths. This may not be the best method since it will end up calling Dijkstra's algorithm once for every interface. It is just used to illustrate how interface ranking can be done in link-state routing. Optimization of the algorithm is possible but out of the scope of this dissertation.

Distance/Path-Vector Routing: In distance-vector or path-vector routing, routers announce the hop count or complete routing path towards each destination to their neighbors. When router N receives a routing announcement for Data provided by M from interface I_k, it simply records the hop count $H^M_{N,k}$[6]. The interfaces are then ranked by the hop count.

Notice that a router may not receive routing announcement from all interfaces, since these routing protocols often incorporate split-horizon route announcement to prevent routing loops. If router N learns a route towards M through interface I_k, it will not advertise its route to M over I_k. Interfaces that do not receive routing announcement are assigned infinite hop count to ensure they stay at the end of the ranked interface list. They will only be used as the last resort if all higher-ranked interfaces fail to retrieve Data.

These interfaces are useful in many situations. For example, in BGP if a provider P uses a customer C as the next hop, it will not make routing announcement to C. If C's best path fails, it will not have an alternative path until routing converges, in which case P will announce its alternative path to C. RBGP [41] is proposed to address this issue by allowing P to announce its alternative path to C even without

this dissertation.

[6]Hop count can be easily extracted from the path in path-vector routing.

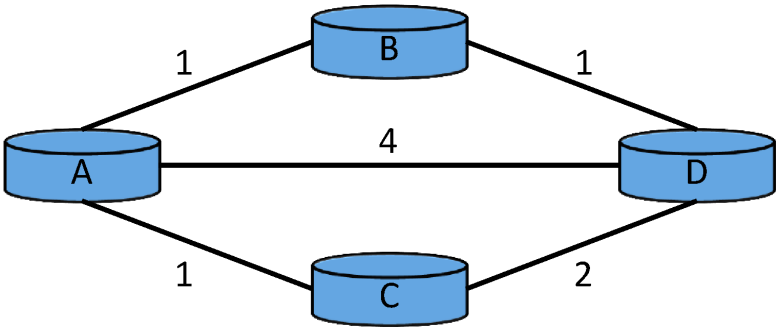


Figure 5.4: A simple network example.

failures. NDN, on the other hand, is able to achieve the same effect without changing the protocol.

5.4.2 Probing

Section 5.2 showed that NDN routers can handle link failures locally at the forwarding plane. In this subsection we answer the question of whether the same applies to link recovery. Routers can detect link failures quickly by observing Interest-Data exchanges or Interest NACK. However, there is no explicit signal for link recovery from the forwarding plane. Let us take Figure 5.4 as an example. The cost of the links are marked in the figure; routers rank the interfaces as in link-state routing. If interface B-D fails, A will use interface A-C as its major next hop for content provided by D. After interface B-D recovers from the failure, interface A-B becomes the new best interface. However, A will continue using interface A-C because the forwarding strategy prefers Green interfaces over Yellow ones. In this case, A needs to probe interface A-B by sending a copy of an Interest to it. If the probing Interest successfully brings Data back, interface A-B will be marked Green and be used to forward subsequent Interests to D.

There is a research question of when to perform probing. An Interest copy is used for probing so that regular Data retrieval will not be affected if probing is unsuccessful. However, this causes extra Interest and Data in the network. There is a trade-off between how fast a link recovery is detected and the amount of overhead

Pseudo-code 5 ProbingDue Algorithm

```
1: function ProbingDue(FibEntry, ForwardedIntf)
2:     if ForwardedIntf ≠ FibEntry.RoutingPreferredIntf then
3:         if FibEntry.LastProbingTime + M ≤ Now() or
4:             FibEntry.PacketsSinceLastProbing ≥ N then
5:             Return True
6:         end if
7:     end if
8:     Return False
9: end function
```

caused by probing. In CCNx [5], routers probe alternative interfaces periodically in order to detect better paths. This enables routers to detect link recovery at the forwarding plane. Fast recovery detection is achievable through aggressive probing. However, it will incur significant overhead as we show in Section 5.5.

In fact, routing is able to help with this dilemma. If there is a routing protocol, it will be able to detect link recovery and converge to it. We can take advantage of routing by only probing a Yellow interface if its ranking is higher than the Green interface(s). This way we can keep the probing overhead low, and switch back to the optimal paths as soon as routing converges. Routing convergence time is not critical because the alternative paths found by the forwarding plane are of good quality (Section 5.2). Notice that probing is also useful in failure handling if the alternative paths found by the forwarding plane are not optimal.

We propose a probing algorithm as presented in Pseudo-code 5 and 6. After forwarding each Interest (before Line 9 of Pseudo-code 3), the strategy module calls **ProbingDue** to check whether probing is needed. Two thresholds are introduced to further limit the probing overhead. For each FIB entry, M is the minimum time interval, and N is the minimum number of packets forwarded between two consecutive probings. The algorithm returns true only if at least M time has elapsed or at least N packets have been forwarded since last probing. Actual numbers of M

Pseudo-code 6 Probing Algorithm

```
1: function PROBE(Interest, FibEntry, PitEntry)
2:     interface ← FibEntry.RoutingPreferredIntf
3:     if interface ∉ PitEntry.Outgoing and
4:        interface ∉ PitEntry.Incoming then
5:         Interest.Nonce ← GenerateNonce()
6:         Transmit(interface, Interest)
7:         Add interface to PitEntry.Outgoing
8:         FibEntry.LastProbingTime ← Now()
9:         FibEntry.PacketsSinceLastProbing ← 0
10:    end if
11: end function
```

and N depends on the traffic load as well as the probing overhead network operators are willing tolerate. Pseudo-code 6 describes the probing algorithm. It sends a copy of the Interest to the routing preferred interface using a different nonce. The nonce is changed so that routers can distinguish between probing Interests and Interests that looped back.

5.4.3 Improving routing stability and scalability

Link-state routing protocols exhibit poor stability and scalability in IP due to the fast routing convergence requirement. However, there is a simple method to address these issues in NDN. Since NDN routers can handle network failures at the forwarding plane, we can actually mask the short-lived failures from the routing protocols. Research shows that the duration of network failures follows a long-tail distribution, and over 50% of failures last less than one minute ([47, 64]). Therefore, the number of routing events can be significantly reduced if routing protocols do not need to react to the short-lived failures. As a result, the bandwidth and CPU cycles consumed by routing updates can be reduced, and there will be less routing fluctuation. In addition, since there is no fast routing convergence requirement, larger networks and

Table 5.1: Topologies used in the simulations.

Topology	Before Processing		After Processing	
	Node #	Link #	Node #	Link #
Abilene	12	30	11	28
AS1239-PoP	52	168	32	128
AS701-PoP	83	438	47	366
AS1239-Router	284	1882	N/A	N/A

more name prefixes become affordable. Therefore, routing overhead can be greatly reduced, both routing stability and scalability can be significantly improved.

For link-state routing, we can implement the idea by increasing the HELLO and DEAD interval. For example, if we set the DEAD interval to be one minute, over 50% of the link failures will not be detected by the routing protocol. Alternatively, we can increase the routing update generation and SPF computation timers to achieve the same effect. Although this idea looks simple, it can be applied to any existing IP routing protocol to improve its stability and scalability. We will evaluate the effectiveness of this method in the next subsection.

5.5 Benefit for Routing

In this subsection we use extensive simulations to evaluate how NDN's routing and forwarding plane benefit from each other. Results show that by masking short-lived failures from routing, its stability and scalability can be significantly improved.

5.5.1 Simulation Setup

We simulate both NDN-BestRoute and IP in the QualNet simulator [2][7], which provides complete implementations of OSPF and RIP. We implemented basic NDN operations and the BestRoute forwarding strategy presented in Chapter 4. We also make necessary changes to the routing protocols as described in 5.4.1 to support NDN-BestRoute. Different from Section 5.2, all experiments in this subsection are run with dynamic routing protocols, i.e., routing will converge after failures.

[7]We did not use ndnSIM since NS-3 does not have dynamic routing protocols.

We use the Abilene topology [3] and selected Rocketfuel topologies [62] in the evaluation. A summary of the topologies is presented in Table 5.1. We process the first three topologies to remove all single-homed nodes. This is because if links of single-homed nodes fail, these nodes will be disconnected from the network and therefore cannot provide any useful result. For OSPF, we use propagation delay as the cost of the links. Unless otherwise specified, we report results from the AS1239-PoP topology. Results for other topologies are similar and lead to the same conclusions. The AS1239-Router topology is only used to show the improvement of routing scalability.

For each topology, we generate random link failures as follows. We use a shifted Pareto distribution to generate time-to-failure and time-to-recover distributions for each link independently [44]. We use 120 seconds as the mean-time-to-recover, and 1000 seconds as the mean-time-to-fail. We also tune the parameters so that 50% of the failures last less than one minute [47]. When a link fails, both directions of the link stop working. With this model, multiple network events (failures and recovery) can happen concurrently.

5.5.2 NDN without Routing

In this experiment we show how NDN forwarding works without routing. Since routers have no idea how to rank the interfaces without input from routing, we implement a forwarding strategy that prefers Green interfaces over Yellow ones, but randomly picks a Yellow interface if no Green interface exists. If Data is brought back from an interface, the interface will be marked Green and used to forward subsequent Interests.

In each run of the experiment we pick one node as the consumer and another as the content provider. Assuming the consumer keeps retransmitting Interests until Data is received, we measure how long it takes. We enumerate all combinations of consumers and providers and draw the CDF in Figure 5.5. In 89% of the cases, the consumer retrieves Data within one second. However, it can take up to 40 seconds to find a working path in some rare cases. The situation can get worse as the

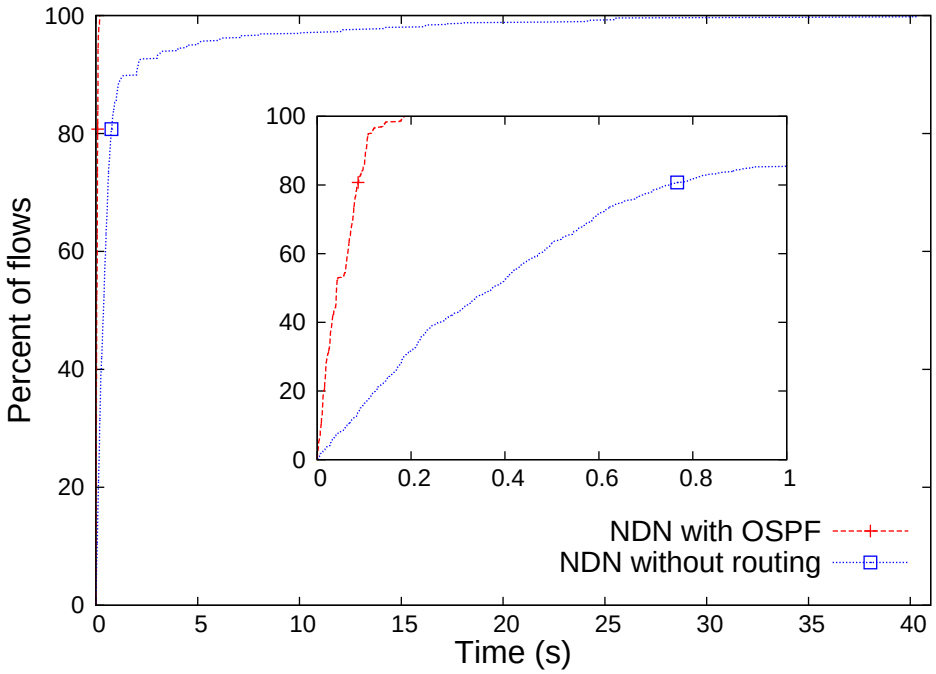


Figure 5.5: CDF of time to find working paths with and without routing.

network becomes larger. In contrast, Data retrieval always follows the best paths when routing protocol can provide interface ranking. Therefore, although NDN's forwarding plane can find working paths on its own, it benefits from a routing protocol to provide interface ranking to make the local search more effective.

CCNx is another example that works without routing. Different from the random strategy, CCNx floods the first few Interests in order to get proper RTT estimate. After that, it will keep using the interface with the shortest RTT while probing other interfaces periodically. We run CCNx version 0.8.1 in the simple topology shown in Figure 5.4 using Mini-CCNx [7] network emulator. Node A sends 10 ccnping [4] Interests towards node D every second. The number of Interests sent by node A at different time is shown in Figure 5.6. In the first second, node A sends 30 Interests as it floods all Interests to all its interfaces. After 3 seconds, A only sends 10-12 Interests every second. The extra Interests are due to periodic probing. In contrast, NDN always sends 10 Interests per second when the network is stable.

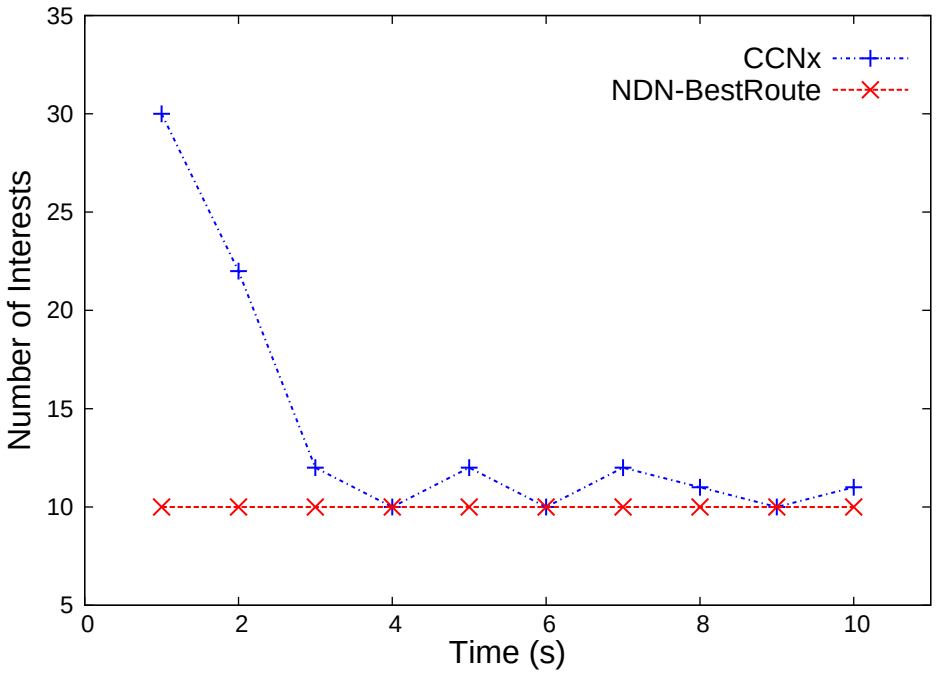


Figure 5.6: Number of Interests sent at different time.

5.5.3 Impact of Routing Convergence Time

In this experiment, we study the impact of routing convergence time on packet delivery performance in NDN-BestRoute and IP. For OSPF, we vary the HELLO and DEAD intervals to create different routing convergence time. We inject random link failures into the network as described in simulation setup. In order to measure packet delivery performance in NDN-BestRoute and IP, we run simple applications among all pairs of nodes in the network. For NDN, each node announces a distinct name prefix and provides content under this prefix. Each node also acts as a consumer requesting data from all other nodes. A consumer sends one Interest towards each name prefix every second. If Data is not received, a consumer will retransmit the Interest every second up to twice. Different consumers request different pieces of Data from the same name prefix so that they do not affect each other. Caching is also disabled so that we can focus on routing and forwarding behaviors[8]. For IP,

[8]If consumers request the same content and caching is enabled, NDN would perform even better.

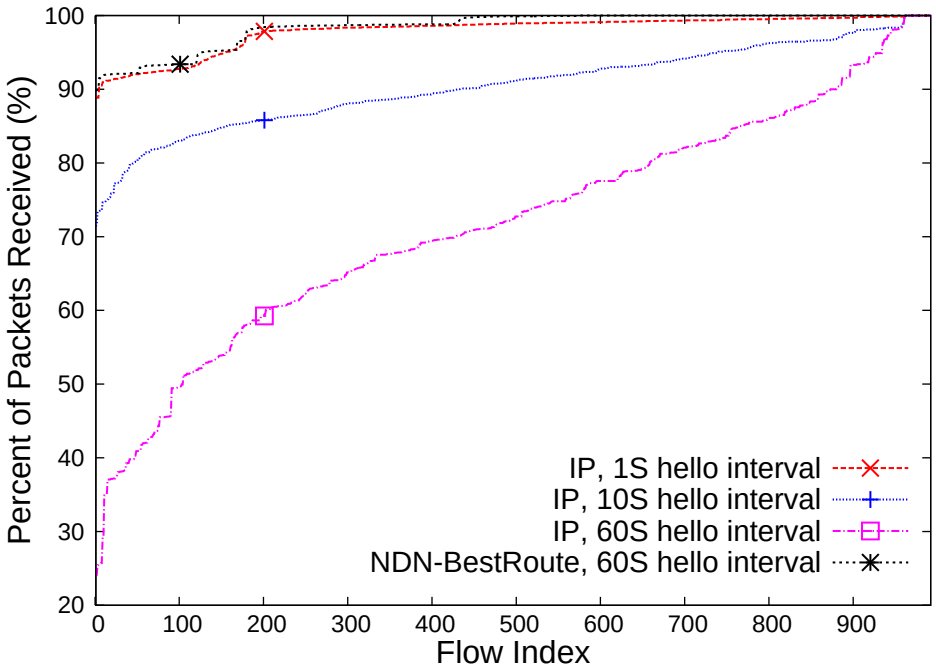


Figure 5.7: Packet delivery performance in IP.

each node acts as both client and server. Each client sends one UDP request to each server every second[9]. The server responds with UDP packet carrying the content. Similar to NDN consumers, these clients also retransmit requests if replies are not received. The sizes of the UDP packets are the same as those in NDN.

Figure 5.7 and 5.8 present the packet delivery rate for each node pair in IP and NDN-BestRoute under different HELLO interval settings. Figure 5.7 shows that HELLO interval has a huge impact on the packet delivery performance in IP. The shorter HELLO interval, the faster packet delivery can be resumed. The median packet delivery rate of IP is 99%, 91% and 72% when the HELLO interval is 1S, 10S and 60S respectively. Figure 5.7 also shows that NDN-BestRoute with 60S HELLO interval even works slightly better than IP with 1S HELLO interval.

Figure 5.8 shows the impact of HELLO interval on the packet delivery rate of

[9]The packet rate is much lower than real Internet traffic due to performance limitation of the simulator. In fact, the IP packet delivery performance will be worse if the packet rate is higher.

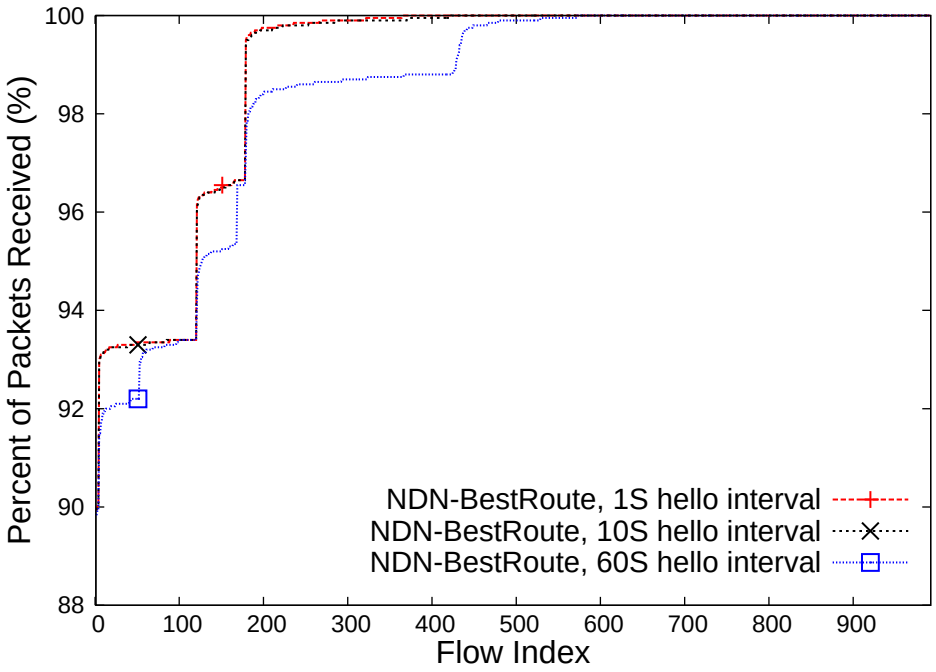


Figure 5.8: Packet delivery performance in NDN-BestRoute.

NDN-BestRoute. When the HELLO interval increases from 1S to 10S, the performance degradation is negligible. When the HELLO interval increases from 10S to 60S, the packet delivery rate decreases slightly. This is because only two consumer retransmissions are allowed. The packet delivery performance can be further improved by allowing more consumer retransmissions. Overall, the HELLO interval has little impact on the packet delivery performance of NDN-BestRoute.

We also evaluate the packet delivery performance under different routing protocols. Figure 5.9 shows the CDF of packet loss rate of NDN-BestRoute and IP when OSPF and RIP are used. Although RIP is generally considered to have poor routing convergence properties, it performs quite well with NDN-BestRoute. NDN-BestRoute with RIP performs much better than IP with OSPF or RIP. The performance difference between OSPF and RIP in NDN-BestRoute is due to the difference in interface ranking. Recall that RIP may not provide hop count for all interfaces, thus OSPF is able to provide better interface ranking.

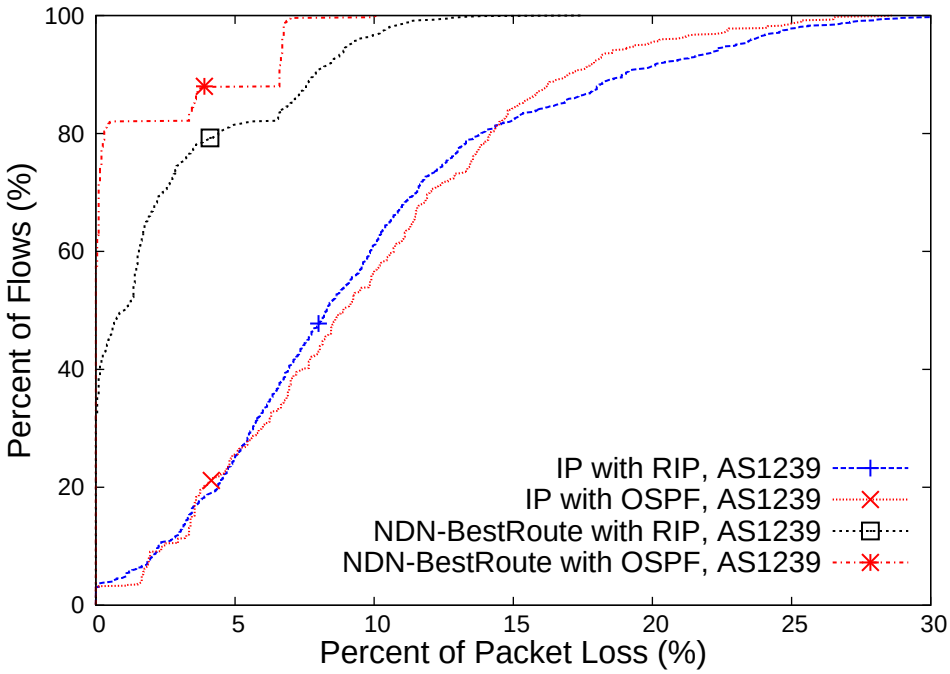
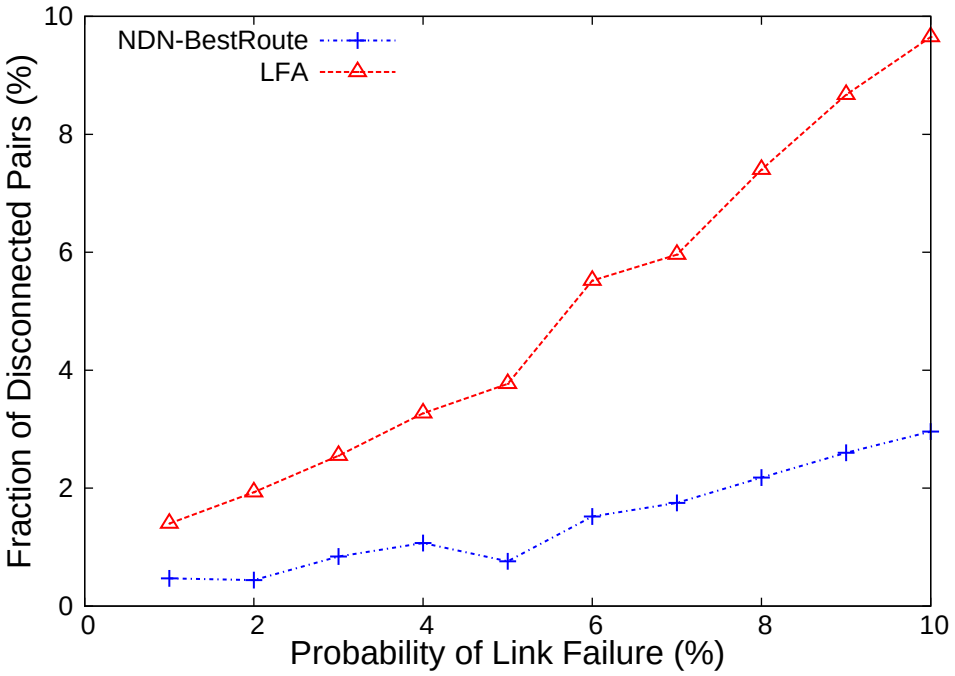


Figure 5.9: CDF of packet loss rate under different routing protocols.

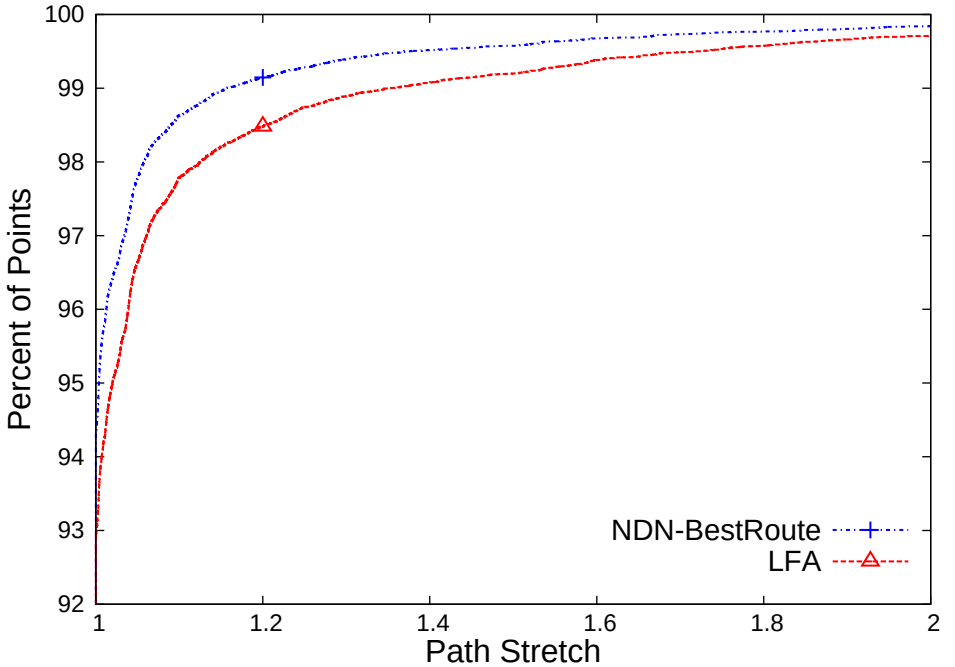
5.5.4 Comparison with IPFRR

In the previous experiment we evaluate the packet deliver performance of plain IP, which totally relies on routing to handle network failures. However, today's ISP networks often adopt solutions that handle network failures without routing convergence, e.g., IPFRR. In this experiment, we compare NDN-BestRoute against Loop-Free Alternate (LFA) [15], the only commercially available IPFRR solution. We implement LFA in a custom simulator, and repeat the link failure experiment in Section 5.2. In each run of the experiment, we associate each link with a probability of failure, and randomly generate link failures. We run each experiment 1000 times and report the average result.

Figure 5.10(a) shows the fraction of disconnected pairs under different failure probability. It shows that NDN-BestRoute is always able to recover much more failure scenarios than LFA. Figure 5.10(b) shows the CDF of stretch of alternative paths found by NDN-BestRoute and LFA. The 98-percentile of path stretch



(a) Fraction of disconnected pairs.



(b) CDF of path stretch.

Figure 5.10: Comparison between NDN-BestRoute and IPFRR.

for NDN-BestRoute and LFA is 1.06 and 1.13 respectively. In conclusion, NDN-BestRoute is able to cover more failure scenarios and find better alternative paths than LFA.

5.5.5 Prefix Unreachable

Previous experiments show that NDN-BestRoute performs well in handling link failures. When a node fails, however, the name prefix served by the node may become unreachable. In such cases, path exploration may lead to extra Interests all over the network. In this experiment we evaluate NDN-BestRoute's exploration overhead when a name prefix becomes unreachable. In each run of the experiment we fail one node and let all other nodes request content from this node before routing convergence[10]. Both NDN-BestRoute and IP applications will retransmit the same request twice. For each flow, we count the number of hops that each packet traverses in both NDN-BestRoute and IP, and compute the ratio of hop count of NDN-BestRoute over IP. We run the experiment for every node failure scenario and present the CDF of the ratio in Figure 5.11.

In IP, retransmitted requests will be sent to the same paths, whereas in NDN-BestRoute, retransmitted Interests may trigger path exploration, leading to large overhead. Surprisingly, NDN-BestRoute incurs less overhead than IP in 26% of the cases. This is because retransmitted Interests do not always trigger path exploration in NDN-BestRoute. If a node has already explored all its interfaces, a further retransmission will only get a NACK back to the application without being further forwarded. In contrast, IP routers will always forward the packets before routing convergence. The ratio is smaller than 5 in 93% of the cases. Only in some rare cases does NDN-BestRoute cause excessively high exploration overhead.

The exploration overhead becomes significant when popular content becomes unreachable, as many consumers will be requesting the content and their Interests will trigger many attempts by routers to find working paths. But on the other hand, popular content is usually hosted and served by multiple servers placed at

[10] After routing converges, routers will learn about the failure and stop forwarding the requests.

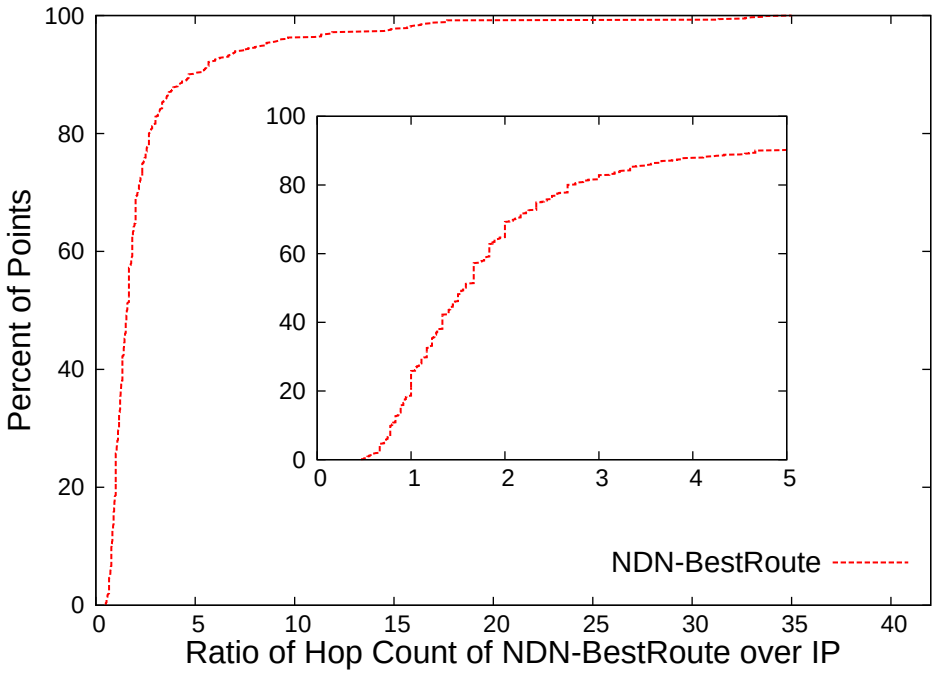


Figure 5.11: CDF of ratio of hop count of NDN-BestRoute over IP.

different locations. In addition, popular content is more likely to be cached by routers. Thus its chance of becoming unreachable is slim. The overall impact in large scale networks needs further investigation.

5.5.6 Probing Overhead

We evaluate probing overhead in this experiment. In each run of the experiment, we fail one link and run applications to let routers find working paths. Then we bring the link back up again, and run applications after routing convergence to measure the number of hops that probing Interests and Data traverse. Interest NACKs are counted as probing Interests. Applications are only run between node pairs whose traffic is affected by the failure. We run the experiment on all link failure scenarios and report the CDF in Figure 5.12. In 36% of the cases, probing Interests and Data only traverse 2 hops; they traverse no more than 6 hops in 94% of the cases. Probing Interests traverse more hops than Data in some rare cases, because a probing Interest

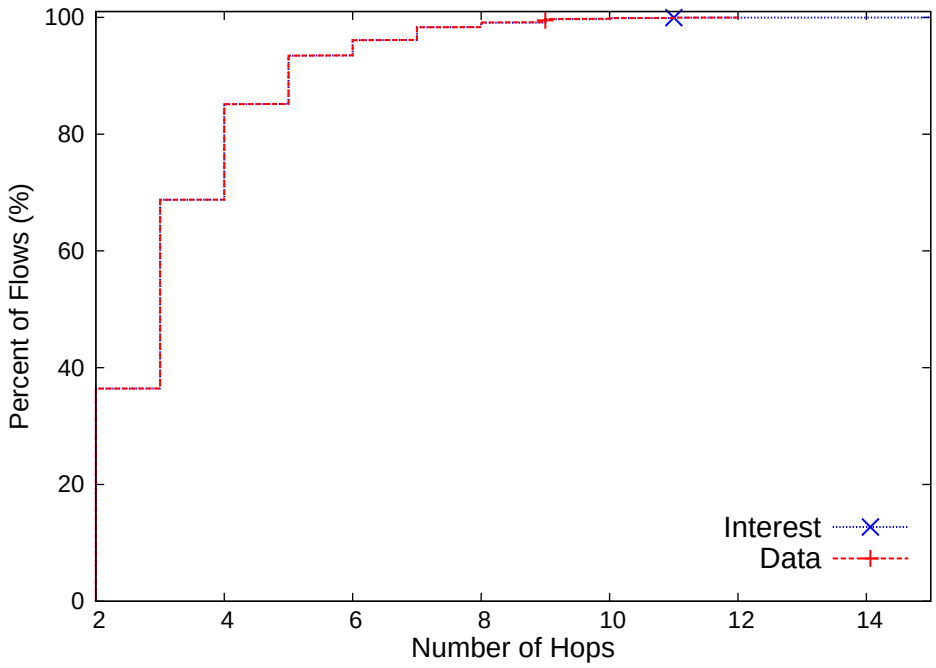


Figure 5.12: CDF of number of hops for probing Interests and Data.

does not necessarily bring Data back, and some of them may loop back to previously visited nodes and trigger NACKs. This experiment shows that by taking advantage of routing, probing only incurs very small overhead.

CCNx does not rely on routing to handle link recovery. Instead, it periodically sends probing Interests to alternative interfaces. We run CCNx and NDN-BestRoute in Abilene topology to show the probing overhead introduced by CCNx. We run applications between each pair of nodes for 20 minutes; the Interest rate is one per second. The first 2 minutes are not counted to exclude the initial flooding Interests in CCNx. The total number of Interests forwarded by each node is presented in Figure 5.13. On average, every node sends 58% more Interests in CCNx than in NDN-BestRoute. This is because each node in CCNx periodically performs probing for every name prefix. The situation will only get worse as the topology grows larger or number of name prefixes increases.

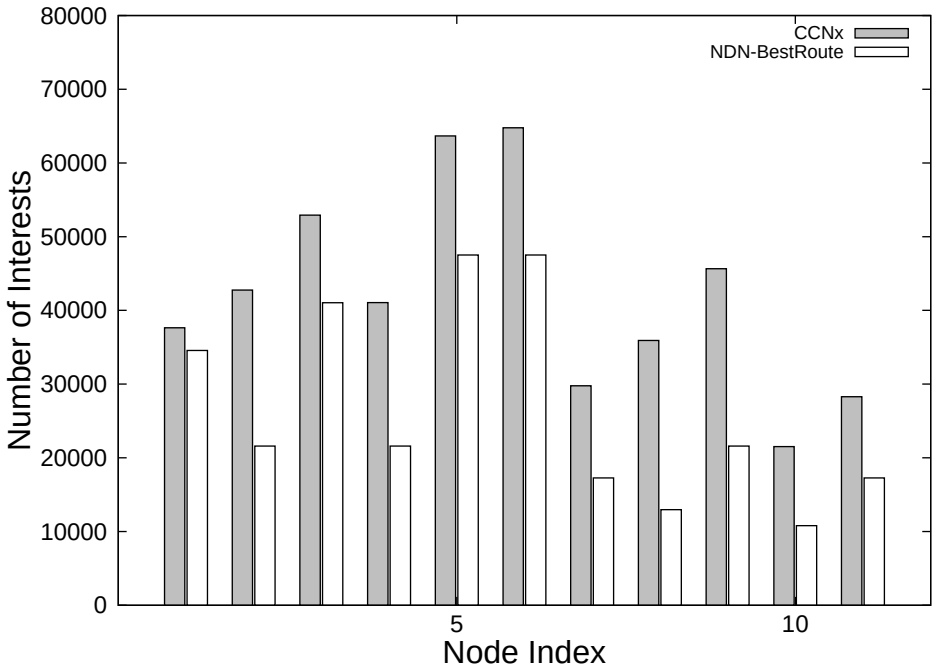
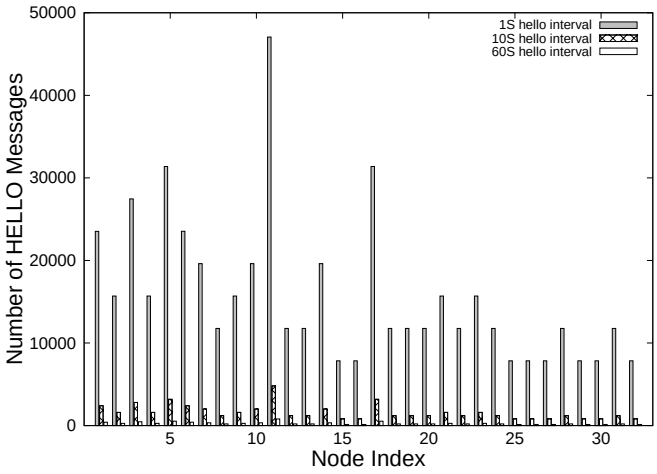


Figure 5.13: Number of Interests forwarded by each node.

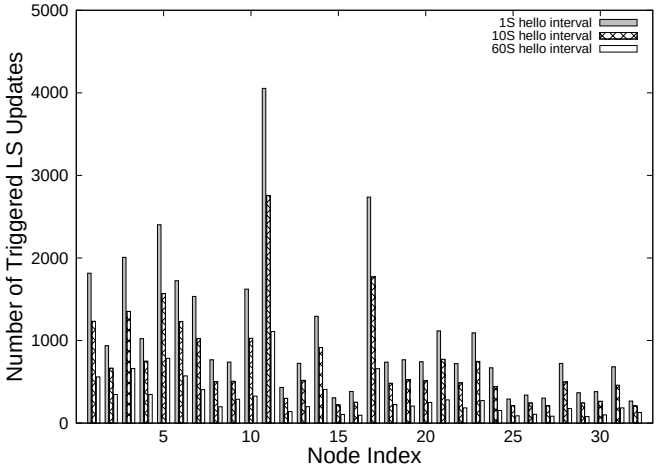
5.5.7 Routing Overhead

In this experiment, we evaluate the routing overhead of OSPF under different HELLO and DEAD interval settings. Specifically, we measure the number of HELLO messages, link-state (LS) updates and SPF computations for each node. HELLO and LS update messages constitute the majority of routing messages triggered by failures and recovery. We set the HELLO interval to be 1S, 10S and 60S; the DEAD interval is always four times the HELLO interval. Random link failures are injected into the network as described in simulation setup, and each experiment is run for 3000 seconds. Only LS updates and SPF computations triggered by failures and recovery are counted[11]. The numbers obtained in this experiment are the same for both NDN-BestRoute and IP.

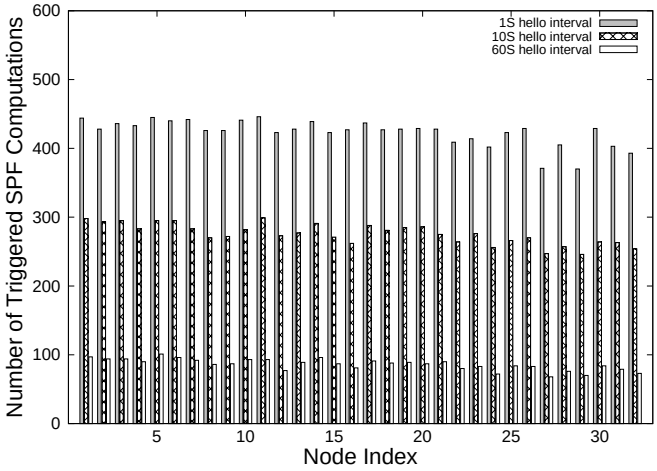
[11]Notice that OSPF also floods refresh link-state announcements periodically even in the absence of network event. These refresh updates are not counted since they are not affected by routing convergence behaviors.



(a) Number of HELLO messages.



(b) Number of triggered LS updates.

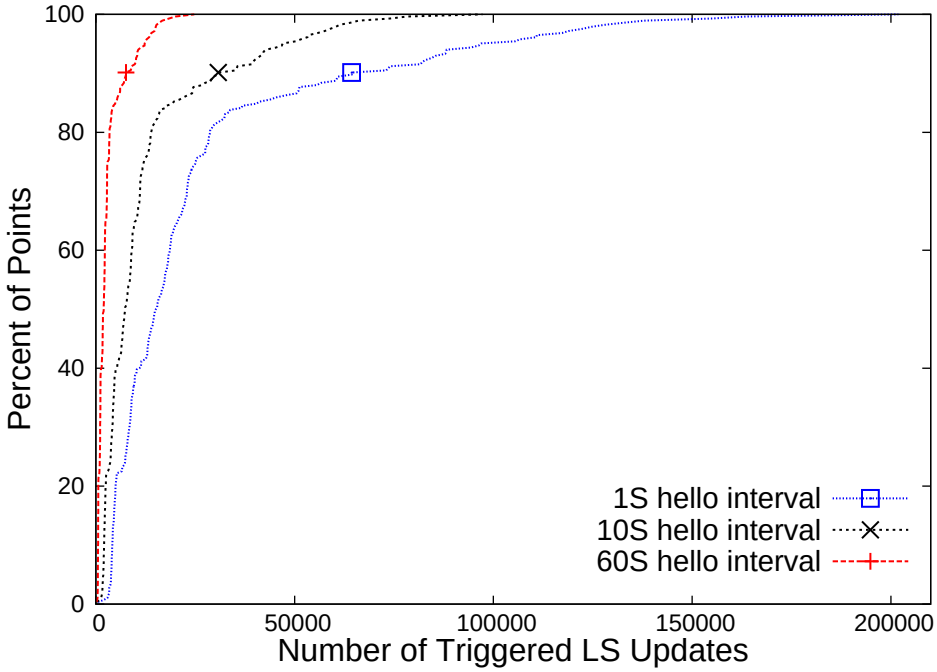


(c) Number of triggered SPF Calculations.

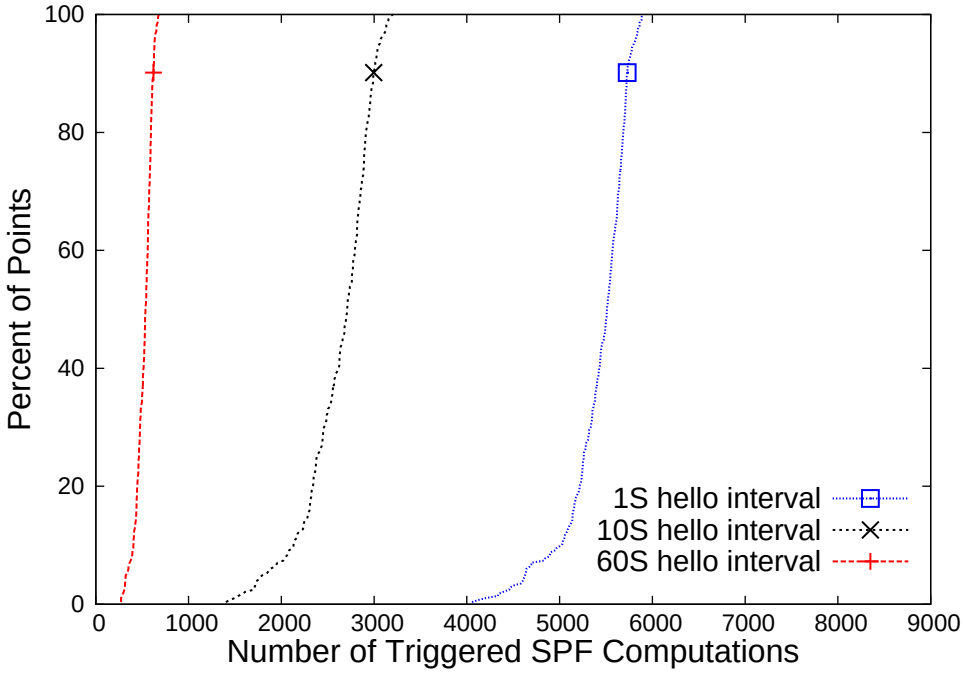
Figure 5.14: Routing overhead in AS1239 PoP-level topology.

Figure 5.14(a) shows the number of HELLO messages sent by each node under different HELLO interval settings in AS1239-PoP topology. As the HELLO interval increases from 1 second to 60 seconds, the number of HELLO messages sent by each node is decreased by 98% as one would expect. Figure 5.14(b) and 5.14(c) present the number of triggered LS updates and SPF computations for each node. As the HELLO intervals increase, less failure events will be detected by OSPF. No routing update will be generated and propagated for the undetected failures, and thus no SPF computation will be performed. If we increase the HELLO interval from 1 second to 60 seconds, the number of LS updates is decreased by 52% to 80%, and the number of SPF computations is decreased by 77% to 82%. Therefore, we can effectively reduce the overhead caused by HELLO messages, LS updates and SPF computation by increasing the HELLO interval.

We ran the same experiments in AS1239 router-level topology to illustrate how the method works in large ISP networks. The CDF of number of triggered LS updates and SPF computations are presented in Figure 5.15. The median numbers of LS updates and SPF computations are decreased by 87% and 90% when HELLO interval increases from 1 second to 60 seconds. In conclusion, routing overhead can be significantly reduced by masking short-lived failures from the routing protocol. Since less LS updates are generated and propagated and less SPF computations are performed, routing becomes more stable and scalable.



(a) CDF of number of triggered LS updates.



(b) CDF of number of triggered SPF computations.

Figure 5.15: Routing overhead in AS1239 router-level topology.

CHAPTER 6

CONGESTION CONTROL

This chapter studies congestion control in NDN with adaptive forwarding. The one-to-one flow balance and symmetric forwarding paths between Interest and Data packets give NDN an effective way to prevent congestion inside the network. By pacing Interests sent to the upstream direction (towards the producer) of a link, one can effectively prevent congestion (caused by Data) on its downstream direction. Specifically, we propose to enforce an *Interest Limit* on each interface which specifies the maximum number of pending Interests allowed on this interface. We first present a simple Interest limiting (SIL) mechanism which, combined with adaptive forwarding, is able to achieve hop-by-hop multipath congestion control. Then we design a more practical Dynamic Interest Limiting (DIL) mechanism and extensively evaluate its performance under different congestion scenarios.

6.1 A Simple Interest Limiting Mechanism

We set a limit on how fast Interest packets can be forwarded over an interface and experiment with a simple calculation of the Interest limit: $L_i = \alpha \times C_i/\bar{S}_i$, where L_i is the Interest limit of interface i, C_i is the upstream link capacity of i, $\bar{S}_i$ is an estimate of the size of the Data packets that have been received over i, and α is a configurable parameter. The ratio $C_i/\bar{S}_i$ is the maximum Data rate that is allowed from upstream measured in packets per second (pps), which should be the same as the maximum Interest rate going upstream[1]. The coefficient α is used to compensate for errors in the calculations (e.g., imprecise Data size estimate, link and network layer overheads). When L_i is not reached, interface i is said to be

[1] A slightly more complicated formula can be obtained if we take the sizes of both Interest and Data packets into consideration, as is done in [66].

available for forwarding Interests, otherwise *unavailable*.

Let us assume there are three nodes, N_1–N_2–N_3, and Interests flow from N_1 to N_3. N_1 computes L_i and sends Interests to N_2 no faster than L_i, which prevents the link between the two nodes from being congested. N_2 will also respect a similar Interest limit when it forwards Interests to N_3. We introduce a new NACK code "Congestion" to indicate congestion in the network. An interface is not marked Yellow upon "Congestion" NACK since congestion is considered temporary. If link N_2-N_3 has less capacity than link N_1-N_2, it is possible that N_1 sends more Interests than N_2 can forward. In this case, N_2 will send extra Interests to alternative paths, or send NACKs with code "Congestion" back to N_1 if none exists. When N_1 receives the NACK, it will try its own alternative paths or return the NACK further downstream. Pseudo-code 3 can be easily adjusted to reflect this change.

6.1.1 Evaluation

Today's Internet routing does not react to congestion due to concerns of routing oscillation and frequent routing updates. When a link is congested, the routing plane at each of the two incident routers either does not see the problem at all if routing protocols have their keep-alive messages pass through, or considers the link failed if enough keep-alive messages are lost. The responsibility of congestion control is solely on end-hosts, which run TCP to detect congestion and adjust sending rate reactively. In NDN, on the other hand, the forwarding state enables routers in the network to prevent, detect, and react to congestion by utilizing multiple paths when needed, resulting in effective and efficient congestion control. In this subsection we experiment with SIL using ndnSIM [12].

Let us first use a simple 6-node topology to shed the light on the basic differences between NDN and TCP NewReno in their reactions to congestion (Figure 6.1). The server and client each has a 10 $Mbps$ link connecting to a router. Each router has buffer size of 20 packets and all the links between routers have 1 $Mbps$ bandwidth. The lower path has an RTT of 130 ms, while the upper path's is 134 ms. Data packet size in both NDN and TCP is 1040 bytes, and both Interest size in NDN and

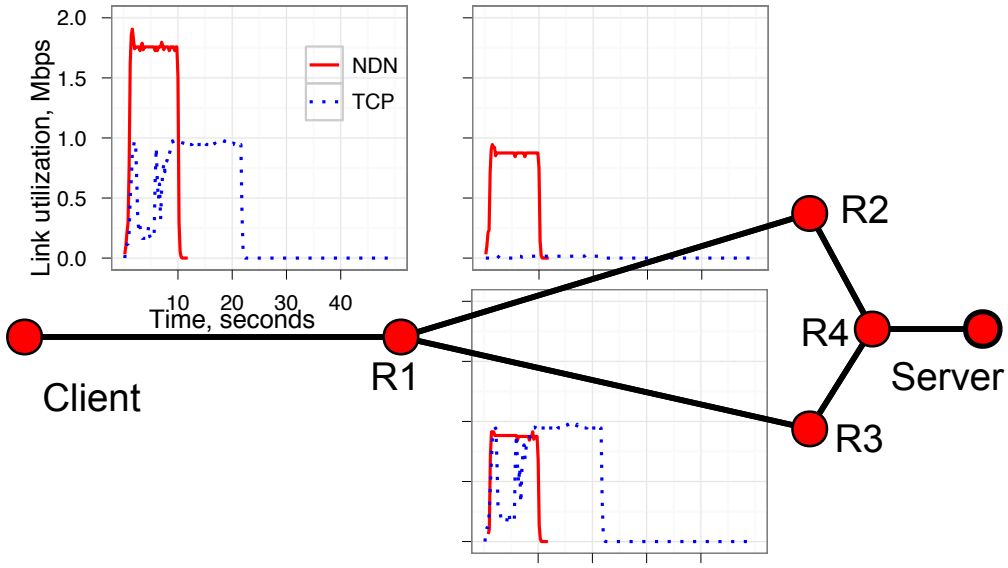


Figure 6.1: Link utilization under congestion

TCP ACK size is 40 bytes. For NDN, the client adjusts its sending window using an AIMD mechanism similar to TCP. The client downloads content from the server and the figures show the link utilization achieved by NDN and TCP respectively. We can make two observations from the results. First, while TCP/IP uses the shorter path only and saturates the bottleneck link, NDN is able to use both paths. In NDN, R1 first uses only the lower path because it is the most preferred, but when the rate-limit of the lower path is reached, R1 starts using the upper path too. Second, over each path, NDN is able to grab available bandwidth more quickly than TCP, which takes longer time to settle at a stable rate. Consequently TCP takes more than twice as long to download the same amount of data.

NDN has a number of means to prevent, control, and alleviate congestion. First, a downstream node controls the rate of Interest forwarding based on its estimate of the bandwidth needed to carry the returning Data traffic. This prevents excessive Data from being pulled into the network, and is enabled by the symmetric two-way flow of Interest/Data packets. In TCP/IP, on the other hand, because data is *pushed* from the sender to the receiver, when a data packet arrives at a link where it cannot be forwarded further, the router simply drops it, after the the packet has

already consumed considerable bandwidth along the way from the sender to the congested link. While TCP congestion control also aims to achieve flow balance as an NDN network does, it sends data packets to probe the network's available bandwidth and takes much longer time to detect congestion (end-to-end vs hop-by-hop); meanwhile additional excessive packets may have been pumped into the network, which eventually get dropped.

Second, Interest NACKs allow NDN routers to adapt to congestion hop-by-hop. A "Congestion" NACK is generated if the Interest cannot be forwarded upstream due to congestion. The downstream node will try its other interfaces for this Interest. This hop-by-hop retry inside the network reacts much faster than the end-to-end solutions for stateless IP networks, leading to quick local workaround as we have seen in the case of link failure recovery. When the network cannot satisfy the demand, Interest NACKs will eventually be pushed back to inform the consumer to adjust its Interest sending rate properly. This is in contrast to TCP, which can only guess whether congestion occurred in the network, and can only use AIMD window adjustment to tune towards the right sending rate.

Third, NDN can use multiple paths simultaneously to retrieve data whenever needed. As illustrated in the cases of hijack and link failure, NDN can find loop-free alternative paths quickly. When traffic is below the rate limit of a single upstream link, all will be forwarded along the best path. When traffic is over a single path's capacity, NDN can divert excess Interests to one or more alternative working paths. This capability of on-demand multipath forwarding enables efficient use of all available network resources.

Fourth, even though we did not simulate caching in this study, in a real NDN network, caching can further help speed up recovery from faults including congestion. When a Data packet arrives at a congested or failed link, it cannot be forwarded further but can be cached along the way. When downstream routers send another Interest, in response to either a NACK or end-host retransmission, via a different interface, this subsequent Interest will bring the requested data back as soon as it hits a cached copy of the data. With caching, recovery from packet losses can

be much faster and more efficient in network resource usage than the end-to-end retransmission in IP-based solutions.

We ran a larger-scale simulation using the Sprint topology as described in Section 4.4.3 and generate a number of flows that lead to cross traffic at multiple locations in the network. Each link has a 20-packet queue, and all links are assigned 1 *Mbps* bandwidth but different propagation delay according to the topology file. In each run, 20 client/server pairs are randomly selected and each client downloads the same amount of data from its server [2]. The clients start in a random order with 1 second apart. Packet size is the same as in the previous simulation. Figure 6.2 shows the results from 100 runs, where each dot represents the finish time of the flow that finishes last. As the figure shows, NDN finishes sooner than TCP in all but 7 runs (including one run in which they finish almost the same time), demonstrating that NDN can utilize network resources more efficiently and handle congestion better.

We can explain the six cases where NDN took slightly longer time than TCP to finish as follows. In NDN, because all consumers try to retrieve data as fast as possible, and all routers explore multiple paths to satisfy consumers demand, consequently those pairs of nodes that have multiple parallel paths in between can capture more bandwidth and finish fast. However a number of flows in the simulation have only one single path between client C_i and server S_i, i.e. they must go through at least one specific link L_B to reach each other. If L_B is not shared with other traffic, C_i can finish data retrieval from S_i as soon as possible. But if L_B is shared by other traffic flows, which is more likely to be the case in NDN than in TCP/IP, C_i will take longer to finish.

The above observation suggests that multipath forwarding deployment should be accompanied by support for fair share of network resources. This fair share support can be added into the decision process when a node needs to return "Congestion" NACKs. The node has the discretion on which Interest to send a "Congestion" NACK back. Through the decision criteria one can achieve fair share goals, enforce

[2]We place clients/servers randomly and run the experiment multiple times in order to evaluate NDN and TCP in general situations.

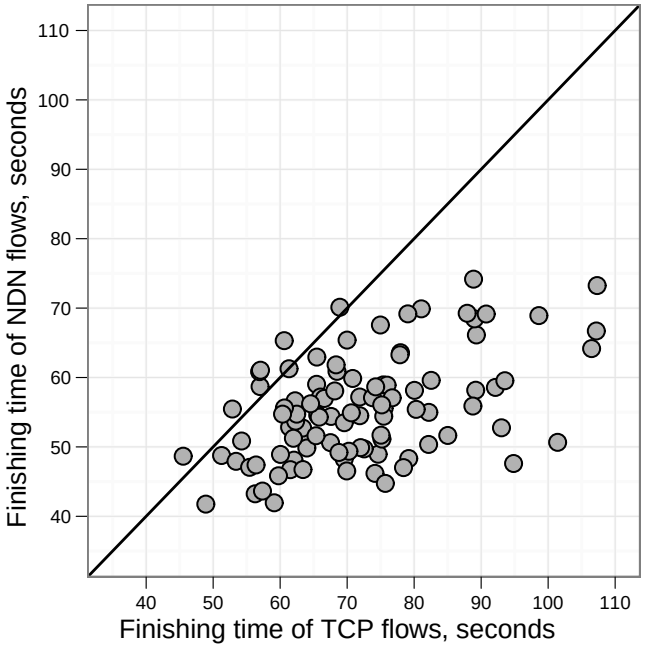


Figure 6.2: Flow finish time under congestion

bandwidth limit to downstream, maintain QoS targets, and even push back excessive Interests in the case of DDoS.

6.2 Dynamic Interest Limiting

The above simple Interest limiting mechanism shows the strength of hop-by-hop multipath congestion control. However, it also has many limitations. First, it cannot properly handle the dynamics in returning Data traffic. After forwarding an Interest, a router has no idea when the corresponding Data will be returned since it can be returned from either content providers, intermediate repositories or router caches. Nor does a router know how big the Data will be. Therefore Data traffic can be bursty and cause congestion even if strict Interest limit is enforced. Second, it cannot effectively address the bufferbloat issue. One option is to apply AQM mechanisms on the upstream router of the bottleneck link, but then Data packets will be dropped silently without notification. While consumers can still retransmit the corresponding Interests in order to retrieve the Data, it does not fully utilize the

Table 6.1: Summary of notation used for DIL

L_i	Total Interest limit on Interface i.
$L_{i,n}$	Interest limit for prefix n on i.
P_i	Pending Interests number on i.
$P_{i,n}$	Pending Interests number for prefix n on i.
α	Interest limit increasing factor.
β	Interest limit decreasing factor.
Min_L	Min Interest limit for the interfaces.
Max_L	Max Interest limit for the interfaces.
Min_{Th}	Min threshold for REN.
Max_{Th}	Max threshold for REN.

power of adaptive forwarding. Third, it does not provide fairness among concurrent flows. Therefore, it cannot fairly allocate network resources in face of ill-behaved consumers.

We present Dynamic Interest Limiting (DIL) to address these limitations. DIL dynamically adjusts the Interest limit on each interface based on the usage of the corresponding link. The Interest limit is increased when valid Data is received on the interface, and decreased when congestion is detected. We present two congestion detection methods which do not rely on the RTT estimate. Random Early NACK (REN) is proposed for native NDN networks, where the upstream router monitors the queue length and proactively sends NACKs to the downstream router when the queue keeps growing. Link-layer Congestion Detection (LCD) is introduced for NDN-over-IP scenarios, in which every NDN router adds a link-layer header containing a sequence number to every NDN packet it forwards, and the router on the other end of the link detects packet losses (i.e., congestion) by observing gaps in the sequence numbers it received. The above design ensures that link bandwidth is efficiently utilized by the aggregate traffic without considering the usage of individual flows. For bandwidth sharing among multiple flows, we propose a Fair Interest Limiting (FIL) mechanism which fairly divides the total Interest limit on one interface among all active flows. This way we are able to decouple utilization control from fairness control in DIL as advocated in XCP [38].

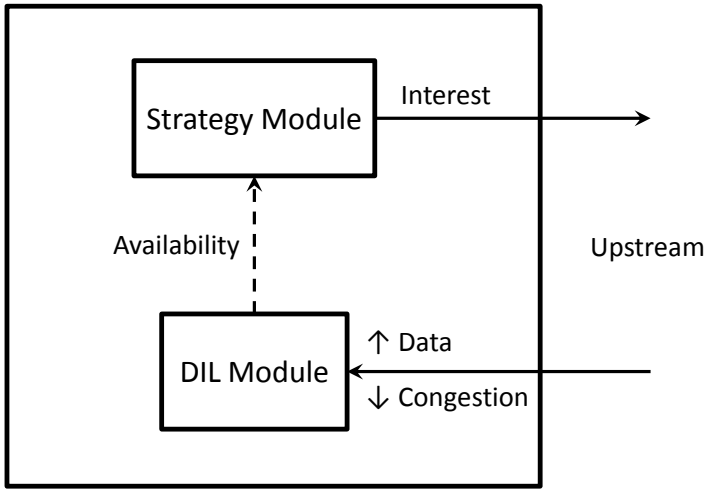


Figure 6.3: Router Model for Dynamic Interest Limiting.

We implement DIL in the ndnSIM [12] simulator and extensively evaluate its performance under different congestion scenarios. Results show that DIL is able to effectively utilize the network bandwidth while keeping application delay and jitter low in both native NDN and overlay scenarios. We also show that DIL provides fairness among multiple flows. DIL combined with adaptive forwarding is able to utilize the network resources more efficiently than SIL proposed in 6.1.

6.3 DIL Design

This subsection explains in detail the design of every component of DIL and how they work as a whole. We assume the forwarding plane design of NDN-BestRoute described in Chapter 4 is deployed on all routers. The notation used in this section is summarized in Table 6.1.

6.3.1 Dynamic Interest Limit Adjustment

Each interface i is assigned a total Interest limit L_i. The initial value of L_i can be calculated using the formula provided in Section 6.1; afterwards it will be dynamically adjusted based on the load of the link. Figure 6.3 illustrates the router

model for DIL. The forwarding strategy module consults with the DIL module on the availability of interfaces when making forwarding decisions. This can be easily added to the forwarding strategy after Line 5 of Pseudo-code 3. We use an AIMD algorithm to dynamically adjust the Interest limit similar to TCP. When a valid Data packet is received, L_i is updated as follows.

$$L_i = min(Max_L,\ L_i + \alpha/L_i)$$

We enforce an upper bound Max_L on L_i to prevent it from increasing unlimitedly when there is no congestion in the network. Similarly, a lower bound Min_L is also placed on L_i. When congestion is detected from the upstream on interface i, L_i is updated as follows.

$$L_i = max(Min_L,\ L_i - \beta)$$

With DIL, routers can adapt their Interest limit on each interface without prior knowledge of returning delay or sizes of the Data packets. Although DIL uses AIMD to adjust the Interest limit, it is essentially different from TCP in that it works in a hop-by-hop, receiver-driven fashion whereas TCP is end-to-end and sender-driven. Below we introduce two novel methods for congestion detection on local links.

6.3.2 Random Early NACK

For DIL to work properly, a router needs to decrease its Interest rate towards an upstream router if the link in-between is congested in the ingress direction. However, there has not been good ways to detect congestion from the upstream in NDN. The traditional method of setting up timers based on RTT estimate is not only slow but also inaccurate in NDN.

We propose a new method called Random Early NACK (REN) for congestion detection in native NDN networks. The router model for REN is shown in Figure 6.4. When a router receives an Interest, it first consults the REN module on whether to accept it or not. The REN module makes the acceptance decision based on the current average length of the output queue of the interface from which the Interest

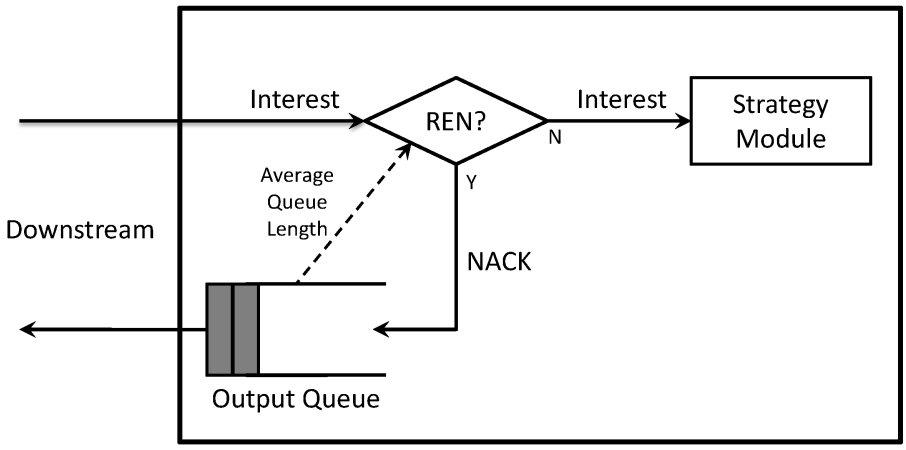


Figure 6.4: Router Model for Random Early NACK.

was received. If the REN module decides to accept the Interest, it will be handed to the forwarding strategy module for further processing; otherwise the router will return a NACK with code "Congestion" to the downstream router. The rationale behind this design is that if the output queue is piling up, it means the router has received more Interests than the link could handle. Therefore the router will return "Congestion" NACKs to its downstream as congestion signals. Upon receiving the NACKs, the downstream router will slow down its Interest rate accordingly. By always keeping the output queue of the upstream router short, we not only prevent congestion but also address the bufferbloat issue.

The idea of REN is derived from RED [28]. The REN module rejects incoming Interests at certain probability computed based on the average queue length. If the average queue length is less than Min_{Th}, no Interest will be rejected; if it is larger than Max_{Th}, all Interests will be rejected; otherwise the probability of rejecting an incoming Interest is computed using the same algorithm as presented in RED [28]. However, REN is different from RED in two fundamental ways. First, RED randomly drops packets coming from the upstream, whereas REN does not drop any packet. REN only rejects Interests from the downstream by sending back NACKs. Second, unlike RED, REN does not change the behavior of the queues. A drop-tail queue with the capability of collecting average queue length will suffice.

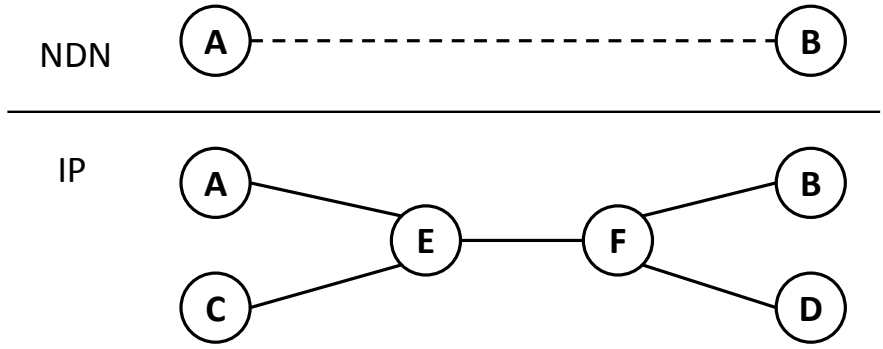


Figure 6.5: An NDN overlay example.

It is worth mentioning that there is another type of NACK that the upstream may send to the downstream due to congestion. If none of the interfaces of the upstream router is available for an Interest, i.e., the upstream router has reached the Interest limits on all its interfaces, it will also return a NACK to the downstream router. This type of NACK is already covered by the BestRoute forwarding strategy described in Section 5.1.

6.3.3 Link-layer Congestion Detection

REN works well on native NDN networks. However, the situation becomes more complex when NDN is deployed as an overlay network on top of IP, as in the NDN Testbed [8]. Take the NDN overlay scenario shown in Figure 6.5 as an example, the NDN link A–B is actually comprised of three IP links, A–E, E–F and F–B, all of which are shared by the underlying IP traffic. The overlay link A–B will become congested if any of the three underlying link is congested. Therefore, the output queues of A and B cannot be used to determine the congestion condition of the overlay link. If the underlying link E–F is congested due to traffic from C to D, A and B will not be able to detect the congestion by monitoring their output queues. Thus, REN will not work effectively in overlay scenarios.

We propose a simple link-layer protocol for NDN to detect congestion without monitoring the queue (Figure 6.6). The link-layer protocol adds a sequence number

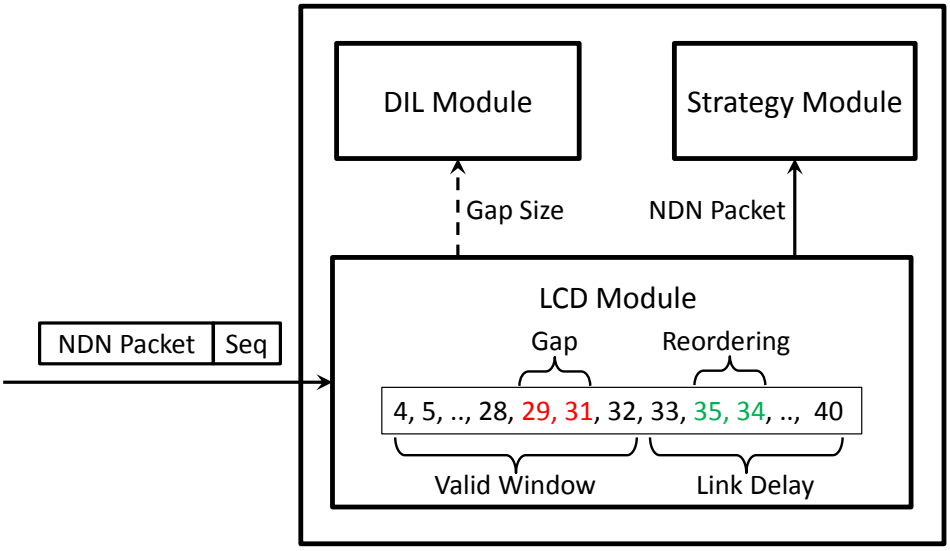


Figure 6.6: Link-layer Congestion Detection.

to each NDN packet forwarded on one link. The router on the other end of the link maintains a window of sequence numbers it recently received. Gaps in the sequence numbers indicate potential packet losses, which are regarded as signals of congestion on the underlying path. We pick the end point of the window carefully such that packet reordering will not be treated as packet losses. Specifically, sequence numbers received during the past link delay are not considered when counting the gaps.

6.3.4 Fair Interest Limiting

In NDN, there is no concept of end-to-end connection as in IP, therefore NDN cannot inherit the definition of fairness from IP. Since NDN routers have no idea of the sources and destinations of the packets, fairness can only be defined based on the names carried by the packets. However, consensus has not been reached on the granularity of fairness. One extreme is per-FIB-entry fairness[3], where the bandwidth is fairly shared by all FIB entries that have active traffic. The other extreme is per-file fairness[4], where the bandwidth is fairly shared by all file transfers. The proper

[3]Assuming FIB entries do not overlap.

[4]Assuming file names all follow the naming convention of /FileName/SegNo.

Pseudo-code 7 Availability of interface i for prefix n

```
1: function InterfaceAvailable(i, n)
2:     if P_i < L_i then
3:         Increase(L_{i,n})
4:     else
5:         m ← LargestPrefix(i)
6:         if P_i = L_i then
7:             if L_{i,n} < L_{i,m} - 1 then
8:                 Decrease(L_{i,m})
9:                 Increase(L_{i,n})
10:            end if
11:        else
12:            Decrease(L_{i,m})
13:            if L_{i,n} < L_{i,m} then
14:                Increase(L_{i,n})
15:            end if
16:        end if
17:    end if
18:    AdjustPrefixList(i, n, m)
19:    Return (P_{i,n} < L_{i,n})
20: end function
```

granularity of fairness for NDN is still subject to further research and investigation. In this dissertation we adopt per-FIB-entry fairness just to show the effectiveness of FIL. It can be easily adjusted to work with other fairness granularity.

Pseudo-code 7 is used to determine whether interface i is available for forwarding an Interest under name prefix n. The prefix Interest limit $L_{i,n}$ for all active prefixes are stored in $i.PrefixList$, a data structure with two indices: a hash index on name prefixes and a doubly linked list sorted by $L_{i,n}$. If the number of pending Interests P_i is less than L_i, it means the total Interest limit has not been reached yet; thus we

can safely increase $L_{i,n}$ and forward the Interest to i. If P_i is equal to L_i, it means the total Interest limit on i has been used up. In this case we need to decrease the Interest limit of the largest prefix [5] and give it to n. If P_i is larger than L_i, it is because the total Interest limit is reduced due to upstream congestion. In this case, we should always reduce the Interest limit of the largest prefix. The Interest will only be forwarded to i if n is not currently the largest prefix and will not become the largest prefix after the Interest is forwarded.

P_i and $P_{i,n}$ are increased by 1 when an Interest under name prefix n is forwarded to i, and decreased by 1 when an Interest is satisfied or given up. It may happen that P_i or $P_{i,n}$ be temporarily larger than L_i or $L_{i,n}$ respectively due to adjustment of the total Interest limit. This will only last until the next one or few Interests are satisfied or given up, before which further Interests under name prefix n will be rejected. $i.PrefixList$ needs to be adjusted by calling $AdjustPrefixList$ after $L_{i,n}$ and/or $L_{i,m}$ are changed to keep the list in sorted order. Adjusting $i.PrefixList$ is very efficient since it is a sorted list, and $L_{i,n}$ or $L_{i.m}$ will only be increased or decreased by 1 each time. In most situations, we only need to compare $L_{i,n}$ with its previous or next neighbor in the list and swap them if necessary; therefore the time complexity is $O(1)$. In the worst case where all name prefixes have the same Interest limit, the time complexity for $AdjustPrefixList$ is $O(N)$ where N is the number of active prefixes in $i.PrefixList$.

Pseudo-code 7 achieves max-min fairness by definition since it always tries to make small flows as large as possible. FIL is, in essence, similar to fair queuing. The fundamental difference is that FIL maintains fairness in the number of pending Interests instead of the number of packets queued for each flow because of the receiver-driven nature of NDN. With FIL, flows with shorter RTT from the bottleneck link may get advantage in throughput. This is similar to today's TCP and can be justified since flows with shorter RTT will consume less network resources [27].

[5] By largest prefix we mean the prefix with the largest Interest limit.

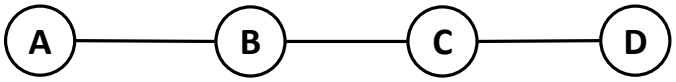


Figure 6.7: A 4-Node Linear Topology.

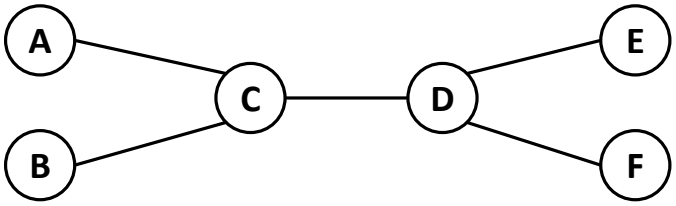


Figure 6.8: A 6-Node Dumbbell Topology.

6.4 Evaluation

In this subsection, we conduct comprehensive simulations to evaluate the performance of DIL under different congestion scenarios. The main metrics used in the evaluation are throughput, application delay and fairness. There is a trade-off between throughput and application delay. For routers to maintain high and stable throughput, there should always be packets in the queue waiting to be forwarded. However, application delay will suffer if the queue becomes too long. Therefore, it is essential to keep the queue non-empty but short. For throughput we are interested in the application finishing time as well as stable bandwidth utilization. We also evaluate the application delay and jitter for each scheme. Additionally, we study whether and how fast each scheme can achieve fairness. Our results show that DIL is able to achieve efficient and fair bandwidth utilization while maintaining low application delay when the network is congested. Results also show that DIL combined with adaptive forwarding provides more effective multipath congestion control than SIL presented in Section 6.1.

6.4.1 Simulation Setup

We implement DIL in the ndnSIM [12] simulator. The BestRoute forwarding strategy presented in Chapter 4 is used in all NDN simulations. Caching is disabled unless otherwise specified. Three different topologies are used in the simulations. A 4-node linear topology as shown in Figure 6.7 is used to show the efficiency of DIL; a 6-node dumbbell topology as shown in Figure 6.8 is used to show how DIL provides fairness among multiple flows; the Sprint PoP-level topology from Rocketfuel [62] is used to demonstrate the performance of DIL in multipath congestion control.

Four different congestion control schemes are considered in the evaluation. 1) DIL with Constant Interest Rate (CIR) consumers which keep expressing Interests at constant rates; 2) AIMD consumers[6] with no hop-by-hop congestion control; 3) Hop-by-hop Interest Shaper (HIS) as proposed in [66] with AIMD consumers; 4) TCP NewReno. The schemes are referred to as DIL, AIMD, HIS and TCP respectively in the rest of this section. The first three are for NDN while the last one is for IP. HIS is similar to SIL presented in Section 6.1. The major differences are that HIS takes two-way traffic into consideration when computing the Interest limit; it also introduces an Interest queue to hold extra Interests when the limit is reached instead of returning NACKs immediately. We use CIR consumers together with DIL since DIL provides effective queue management as well as fairness control[7]; CIR consumers do not work well with HIS or without hop-by-hop congestion control. Therefore we use AIMD consumers together with HIS as is done in [66].

For DIL, we set the initial values of L_i to be 50; Min_L and Max_L are set to be 30 and 70 respectively. We set α and β to be 0.4 and 0.8. For REN, Min_{Th} and Max_{Th} are set to be 1 and 4. Unless otherwise specified, we set the Interest queue size to be 20 packets for HIS[8]; drop-tail queues with size 100 packets are used for

[6]The consumers increase their Interest rate on Data and decrease the rate on NACK or timeout similar to TCP.

[7]We are not suggesting that CIR is a good option for practical usage, it is only used in the simulations to show the strength of DIL.

[8]We experimented with different values for the Interest queue size and chose the one that works the best in the given scenarios.

all links. In the cases where RED queues are used, Min_{Th} and Max_{Th} are set to be 5 and 15 respectively.

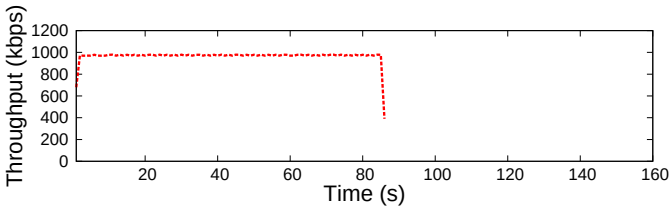
6.4.2 Efficiency of DIL in Native NDN Networks

In this set of experiments, a linear topology (see Figure 6.7) is used to show the efficiency of DIL under different congestion scenarios. All links are native NDN links. Unless otherwise specified, all links have delay of 50 ms; link bandwidth is 1 $Mbps$ for B-C and 10 $Mbps$ for the others. Thus link B-C is the bottleneck. For DIL, the Interest rate of the consumers is set to be 200 per second. We evaluate the performance of different schemes under different scenarios, including varying Data packet size, varying RTT as well as two-way traffic.

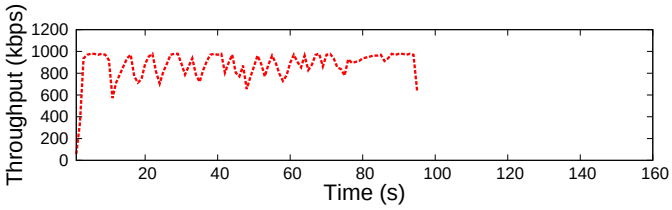
Base Case:

In this experiment, A is the consumer and D is the provider for NDN. A requests 10000 pieces of Data from D, each of which is 1040 bytes. Each Interest name contains a sequence number, and Interests are expressed in the order of the sequence number. An Interest will be retransmitted when NACK is received or timeout is triggered. For TCP, a client and a server are installed on D and A respectively; D sends the same amount of data towards A.

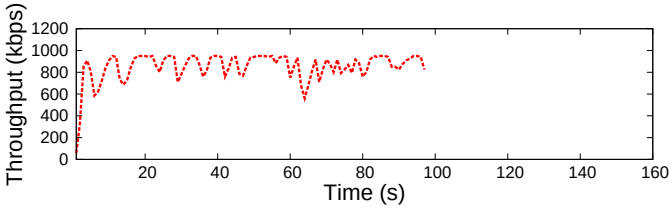
Figure 6.9 presents the throughput and finishing time for different schemes. Among all schemes, DIL (Figure 6.9(a)) achieves the most stable throughput and shortest finishing time. All schemes except DIL experience a slow-start period during which the throughput increases slowly from 0. DIL does not need slow start since the initial Interest limit is a pre-computed value instead of 0. The throughput for AIMD (Figure 6.9(b)) keeps fluctuating, leading to a longer finishing time. This is because when congestion is detected, the consumer reduces its sending window aggressively. Since there is no hop-by-hop congestion control, timeout is the only signal for congestion for AIMD. Even with per-hop Interest limiting, HIS (Figure 6.9(c)) achieves similar throughput as AIMD. This is because router B will send



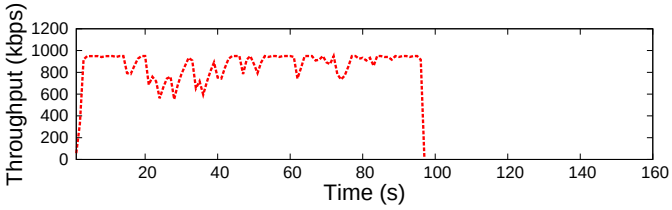
(a) DIL



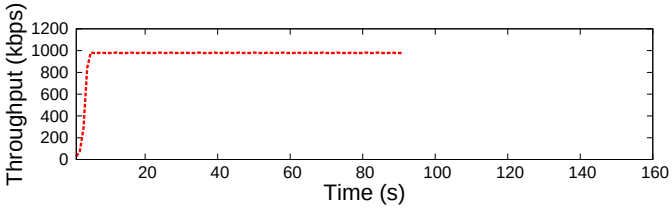
(b) AIMD



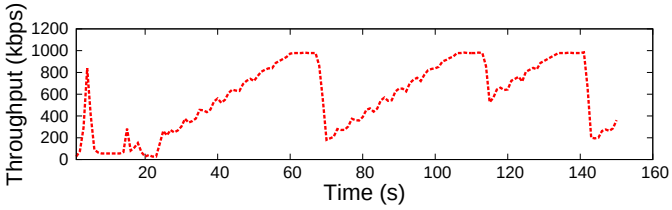
(c) HIS



(d) HIS with big Interest queue



(e) TCP



(f) TCP with RED queue

Figure 6.9: Throughput and finishing time for linear topology.

"Congestion" NACKs back to A when the Interest limit is reached and the Interest queue is full, which will cause the consumer to decrease its sending window just as in AIMD. Increasing the Interest queue size from 20 to 100 for HIS has little impact on the finishing time as shown in Figure 6.9(d). Surprisingly, TCP (Figure 6.9(e)) provides good throughput in this scenario because of the long drop-tail queue on link B-C. The throughput drops sharply and exhibits the typical fluctuation when RED queue is used instead (Figure 6.9(f)).

Figure 6.10 shows the CDF of application delay for NDN. Over 98% of the Interests have RTT of less than 340 ms for DIL, while the 98th percentile of RTT for HIS is 453 ms. The application delay gets much worse when the Interest queue size is increased from 20 to 100 for HIS. AIMD also has long application delay due to the long drop-tail queue on the bottleneck link. It is hard to measure application delay for TCP due to packet fragmentation, thus we measure the average queue length instead. Figure 6.11 shows the average queue length of the bottleneck link for DIL and TCP. The queue length for TCP is around 55 almost all the time. It becomes smaller when RED queue in installed, but at the cost of throughput loss as we previously show. In contrast, the average queue length for DIL is stabilized to around 2 packets soon after the initial stage. In summary, this experiment shows that DIL is able to effectively utilize the bandwidth of the bottleneck link while keeping application delay short.

Varying Data Size:

For this scenario, the same topology setup as the base case is used. The only difference is that the sizes of the Data packets are randomly distributed between 600 and 1400 bytes. The throughput and finishing time for different NDN schemes are shown in Figure 6.13. Compared to the base case where Data size is constant, this experiment introduces more dynamics in the returning Data traffic. AIMD and HIS rely on the output queue of the bottleneck link to absorb such dynamics. Consequently, the queue of the bottleneck link becomes more occupied and the throughput is actually improved compared to the base case. As a trade-off, however,

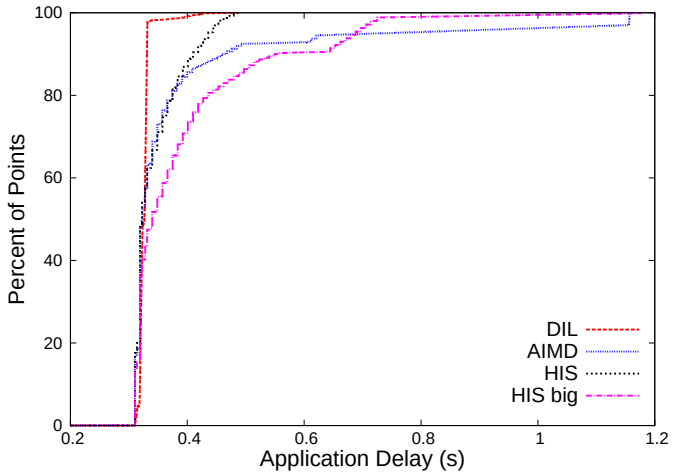


Figure 6.10: CDF of application delay for linear topology.

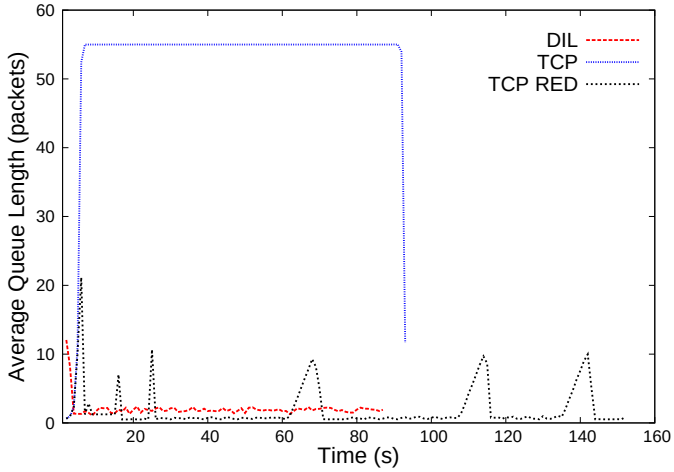


Figure 6.11: Average queue length for linear topology.

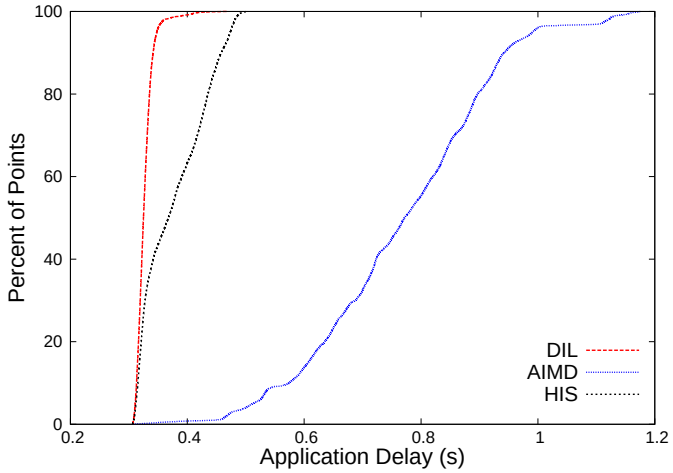
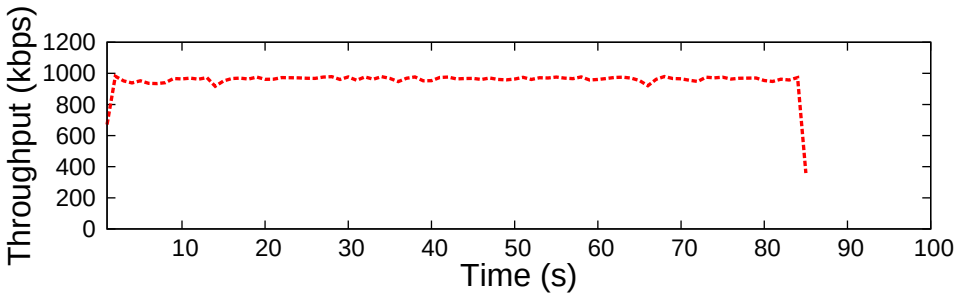
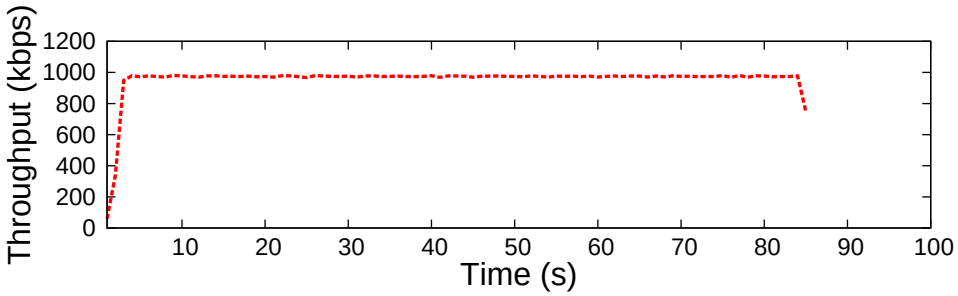


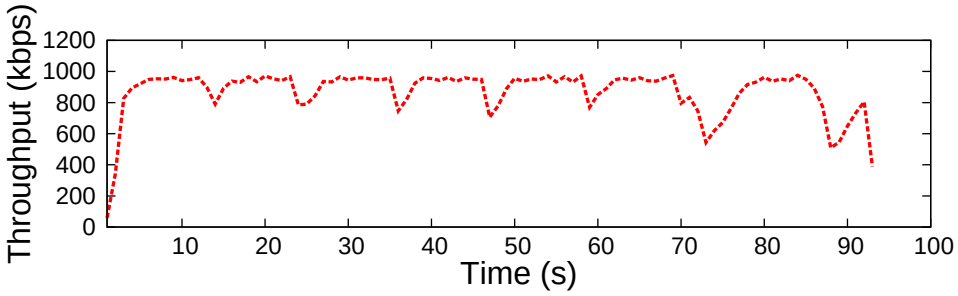
Figure 6.12: CDF of application delay with random Data size.



(a) DIL



(b) AIMD



(c) HIS

Figure 6.13: Throughput and finishing time with random Data size.

the application delay gets longer for AIMD and HIS than in the base case as shown in Figure 6.12. In contrast, REN dynamically adjusts the Interest limit on node B according to the queue length on node C. As a result, DIL is able to achieve high and stable throughput with short application delay similar as in the base case. The 98th percentile of application delay for DIL is 361 ms, slightly higher than 333 ms in the base case.

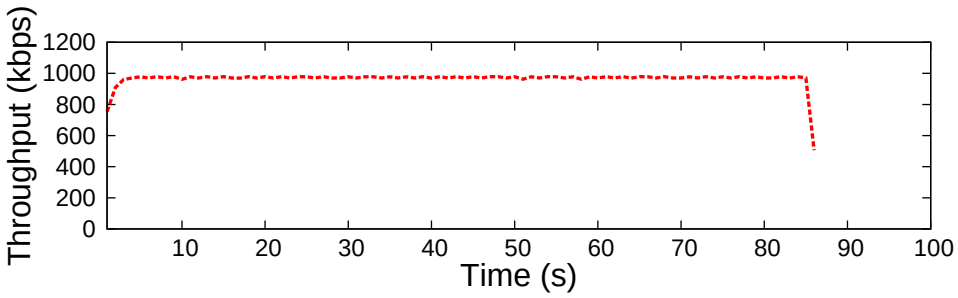
Effect of Caching:

This experiment demonstrates the performance of different NDN schemes under varying RTT caused by caching. We use the same setup as in the base case experiment. However, all Data with odd sequence numbers are cached by node C from the beginning of the simulations. As a result, propagation delay will be 100 ms for half of the Interests, and 150 ms for the other half. Similar to the previous experiment, AIMD and HIS rely on the output queue of the bottleneck link to handle the dynamics in returning Data traffic, while DIL is able to dynamically adjusts the Interest limit. The throughput and CDF of application delay are presented in Figure 6.14 and Figure 6.15, respectively. Again DIL achieves the shortest finishing time among the three schemes. There is a clear difference of application delay for Interests with odd and even sequence numbers for DIL. This is because DIL introduces little queuing delay. The phenomenon is not observed in AIMD and HIS due to the longer queuing delay.

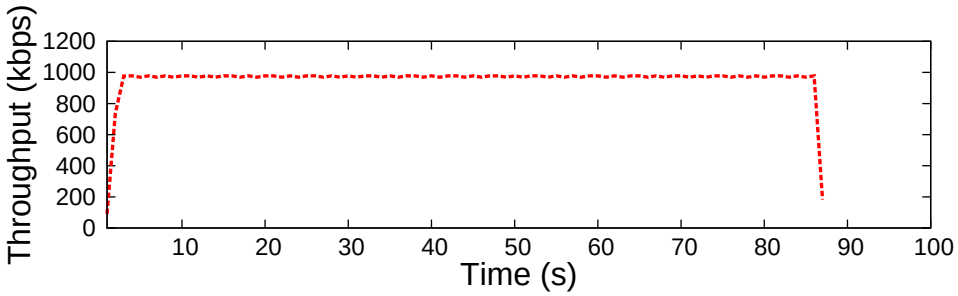
Two-way Traffic:

One of the improvements of HIS over SIL proposed in Section 6.1 is that it takes two-way traffic into consideration when computing the Interest limit. In this experiment we create two-way traffic scenarios by making nodes A and D both consumers and producers, and evaluate the performance of different NDN schemes. The setup is the same as in the base case, except that nodes A and D each sends 10000 Interests towards each other.

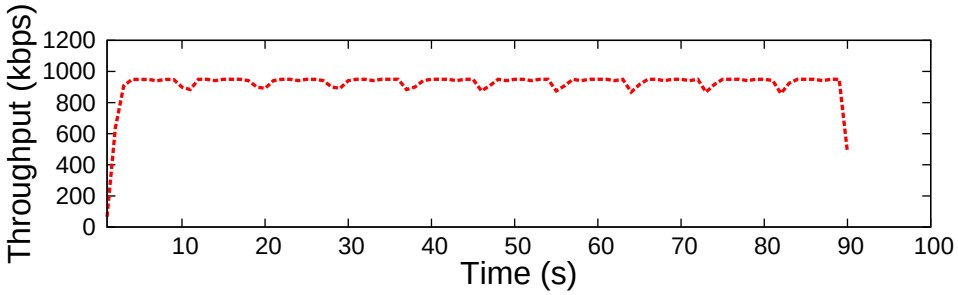
Figure 6.18 shows the throughput and finishing time of different flows in different schemes. The two curves in each figure represent the two flows. For DIL, both flows finish at 91 seconds and the throughput for both flows remains stable. For AIMD, however, the two flows only finish after 120 seconds. The flows cannot effectively utilize the bandwidth due to traffic in both directions, as congestion can also be caused by Interests. By enforcing Interest limit at the bottleneck link, HIS achieves better throughput than AIMD. The flows are able to finish after 100



(a) DIL



(b) AIMD



(c) HIS

Figure 6.14: Throughput and finishing time with caching.

seconds. However, the throughput still fluctuates a lot due to the behavior of the consumers. Figure 6.16 shows the CDF of application delay for each scheme. Only one flow for each scheme is shown since the two flows almost overlap with each other. Although HIS achieves slightly better application delay than DIL in around 62% of the cases, DIL provides better throughput and smaller jitter in application delay.

6.4.3 Efficiency of DIL in NDN Overlay Networks

In this set of experiments we examine the performance of different schemes in NDN-over-IP scenarios. We still use the linear topology shown in Figure 6.7. A is still the

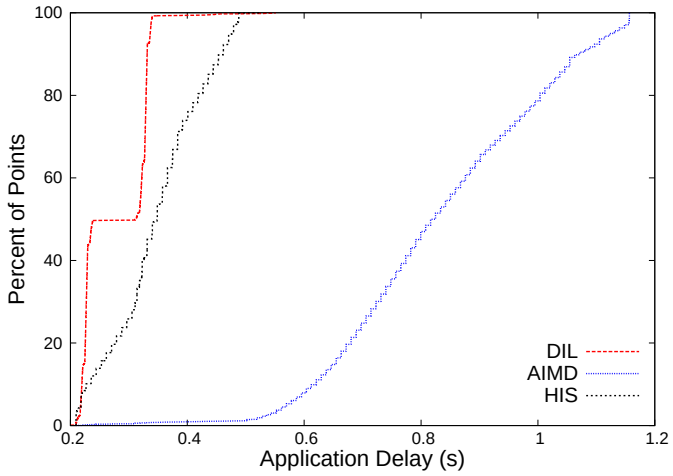


Figure 6.15: CDF of application delay with caching.

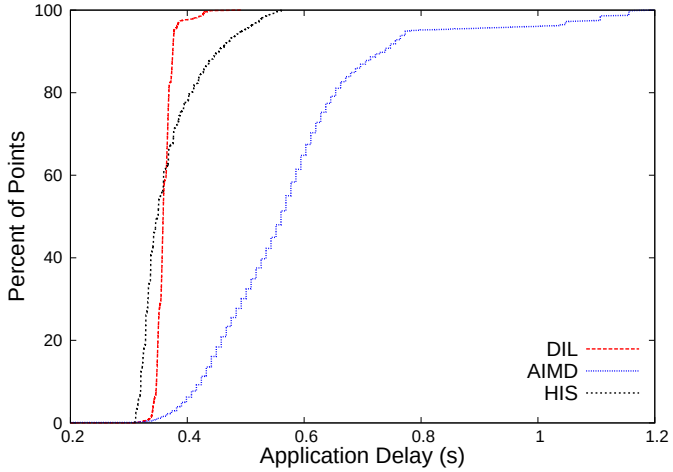


Figure 6.16: CDF of application delay under 2-way traffic.

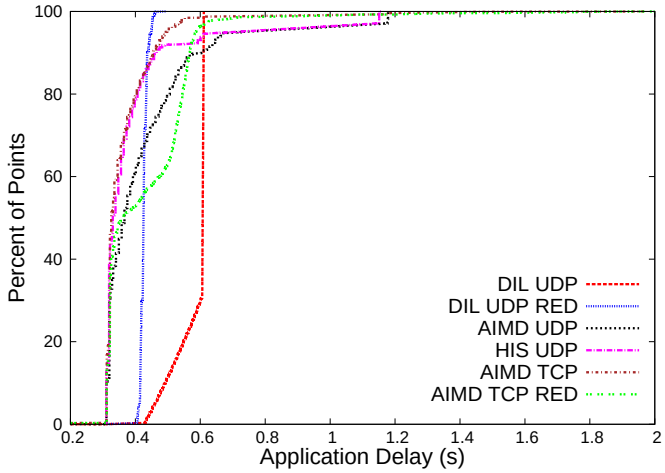
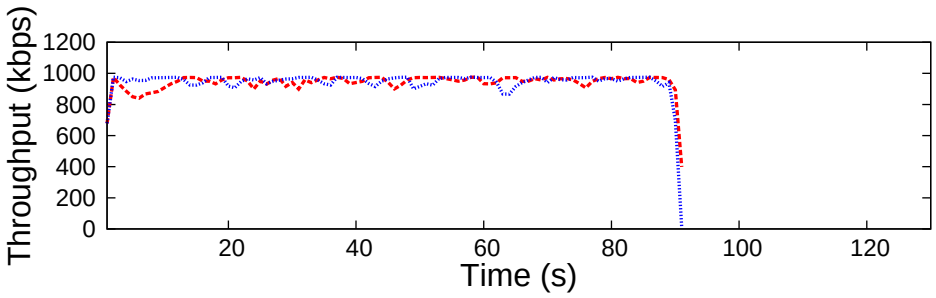
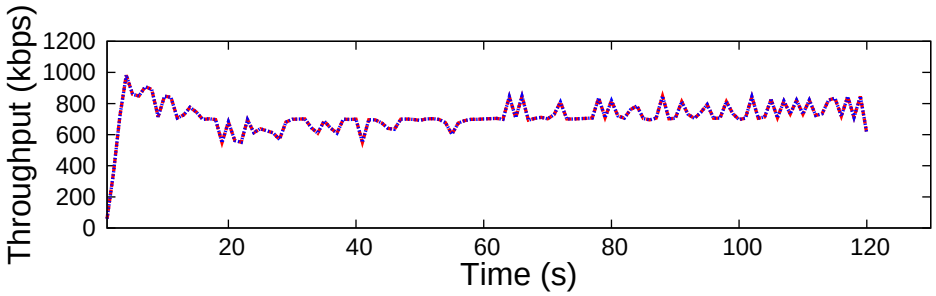


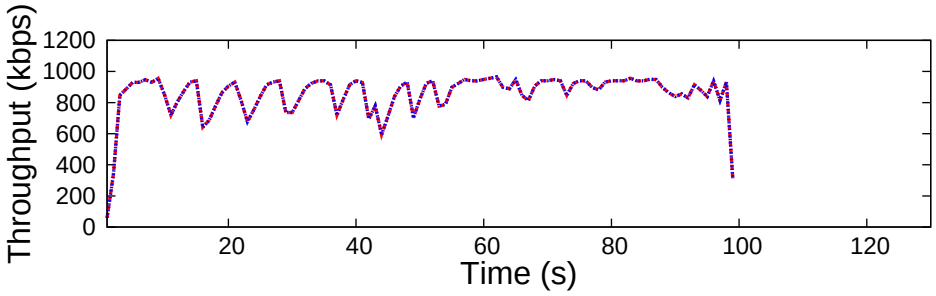
Figure 6.17: CDF of application delay for overlay scenarios.



(a) DIL



(b) AIMD

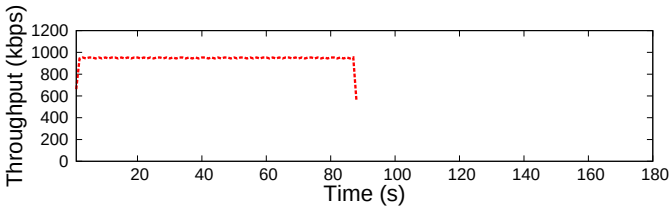


(c) HIS

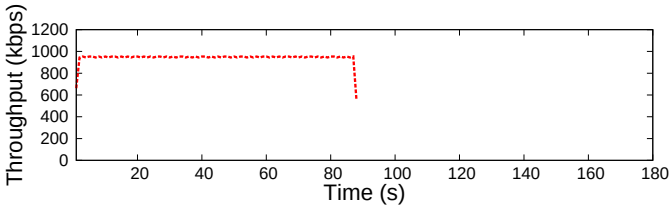
Figure 6.18: Throughput and finishing time under 2-way traffic.

consumer and D is still the producer, but B and C are pure IP nodes that don't understand NDN. Therefore the NDN link A-D is actually an IP path A-B-C-D. All IP links have delay of 50 ms; bandwidth is 1 $Mbps$ for B-C and 10 $Mbps$ for A-B and C-D. In this setup, NDN nodes A and D have no idea where the bottleneck link is in the underlying IP path. Nor do they know the bandwidth of the bottleneck link. REN will not work as effectively as in native NDN networks, because the average queue length observed by D may not accurately reflect the congestion status of the underlying IP path.

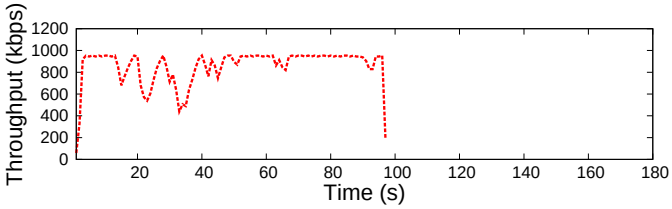
HIS is not designed to work on NDN overlay networks; we place the Interest



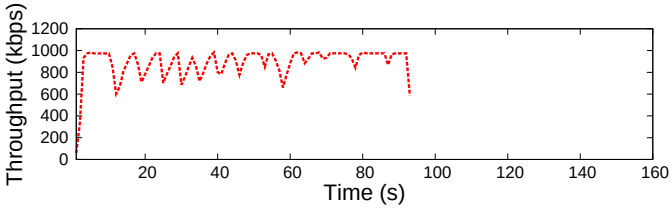
(a) DIL, UDP



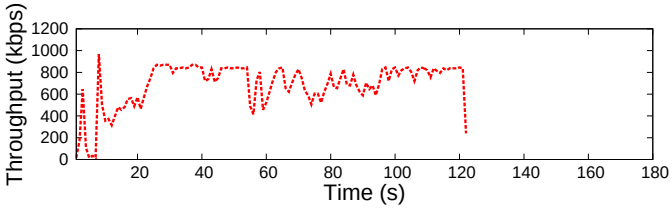
(b) DIL, UDP, RED



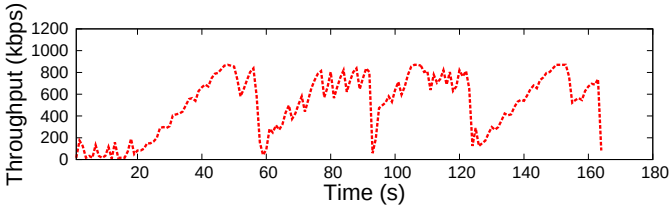
(c) AIMD, UDP



(d) HIS, UDP



(e) AIMD, TCP



(f) AIMD, TCP, RED

Figure 6.19: Throughput and finishing time for overlay scenarios.

shaper at node A and use 10 $Mbps$ as the link bandwidth when computing the Interest limit. This way we can emulate HIS over UDP, where the Interest shaper does not know the exact bandwidth of the bottleneck link (1 $Mbps$ in this case). In each experiment A will send 10000 Interests towards D; the size of each Data packet is 1040 bytes. Figure 6.19 shows the throughput and finishing time for different schemes and configurations. The overlay link A-D is a UDP path in the underlying network for Figures 6.19(a) - 6.19(d), and a TCP path for Figures 6.19(e) - 6.19(f). For Figures 6.19(b) and 6.19(f) we replace the drop-tail queue on B-C with a RED queue. The results show that DIL achieves high and stable throughput no matter what queue type is used on B-C. AIMD and HIS do not work as good as DIL due to the back-off behavior of the consumer. The throughput of AIMD degrades significantly when TCP is used as the transport protocol for the overlay link, and becomes even worse when RED queue is used on B-C. This is because there are two window adjustment algorithms working on their own without any cooperation.

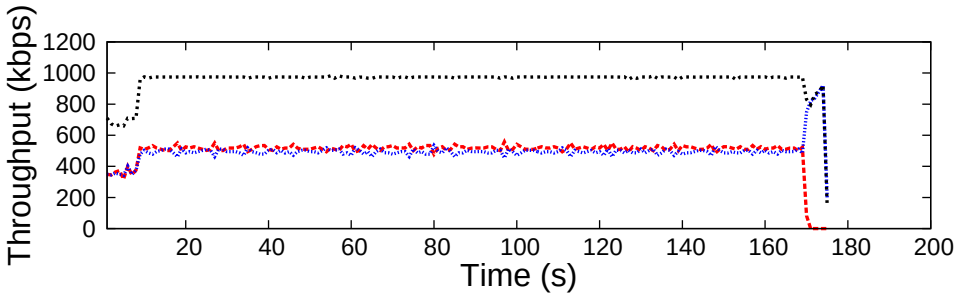
Figure 6.17 shows the CDF of application delay for different configurations. For DIL, LCD can detect congestion upon packet loss, and then DIL will reduce the Interest limit accordingly to avoid further packet loss. If drop-tail queue is used on B-C, packet loss will only be detected after the queue gets full, and DIL will adjust the Interest limit to keep the queue in a close-to-full state. If RED queue is used on B-C, however, DIL will try to keep the queue short to prevent packets from being dropped. Therefore the application delay is significantly improved for DIL when RED queue is used. The jitter of application delay for DIL is small as well compared to other schemes.

6.4.4 Fairness of DIL

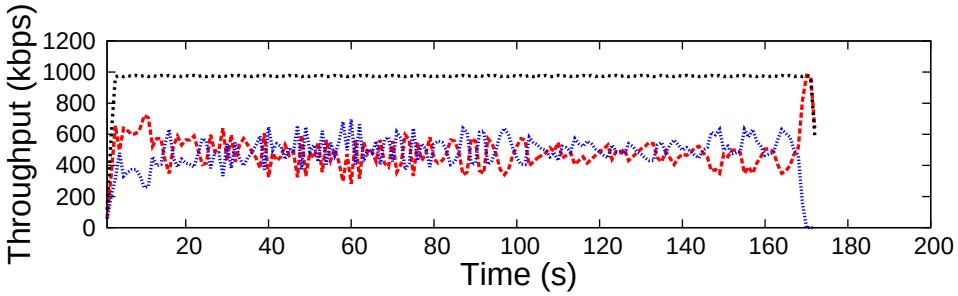
We run simulations on the 6-node dumbbell topology shown in Figure 6.8 to examine the fairness exhibited by each scheme. In this set of experiments, node A and B are consumers, which request 10000 pieces of Data from node E and F respectively. The two flows use different name prefixes. The size of Data packets is 1040 bytes. All links have 50 ms delay unless otherwise specified. Bandwidth is 1 $Mbps$ for C-D

and 10 *Mbps* for other links.

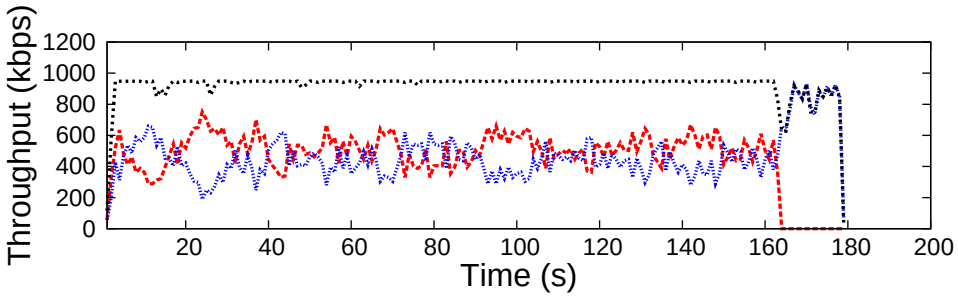
Base Case:



(a) DIL



(b) AIMD



(c) HIS

Figure 6.20: Throughput and finishing time for dumbbell topology.

In this experiment, the two flows start at the same time. Both consumers send 200 Interests per second for DIL. Figure 6.20 shows the throughput of each individual flow as well as the overall throughput for different schemes. All three schemes are able to achieve high overall throughput. However, the throughput for individual flows in DIL is much more stable than in AIMD and HIS. This is because AIMD and HIS rely on the consumers to provide fairness. The consumers back off multiplica-

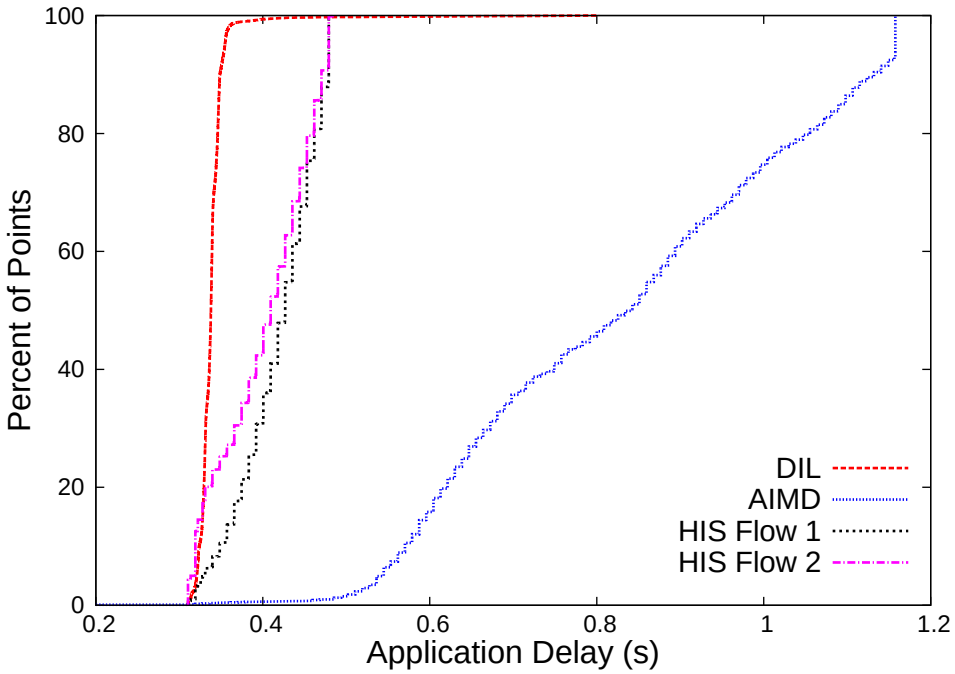


Figure 6.21: Delay for dumbbell topology.

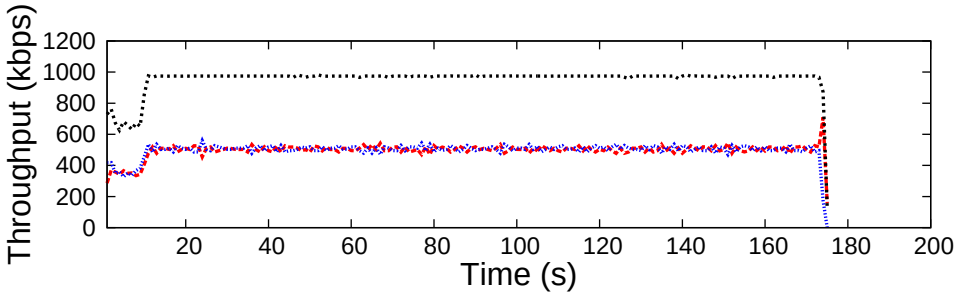


Figure 6.22: Throughput and finishing time with different Interest rate.

tively when encountering signs of congestion. When one flow backs off, the other one will quickly occupy the freed bandwidth. Thus, the throughput of individual flows exhibits obvious fluctuation. An additional observation is that HIS does not provide good fairness as one flow finishes 16 seconds earlier than the other.

Figure 6.21 presents the CDF of application delay for different schemes. Only one curve is shown for DIL and AIMD since the two curves almost overlap with each other. Similar to the linear topology scenarios, DIL is able to achieve shortest

application delay and smallest jitter. The extremely long application delay for AIMD is caused by the long drop-tail queue on the bottleneck link.

Varying Interest Rate:

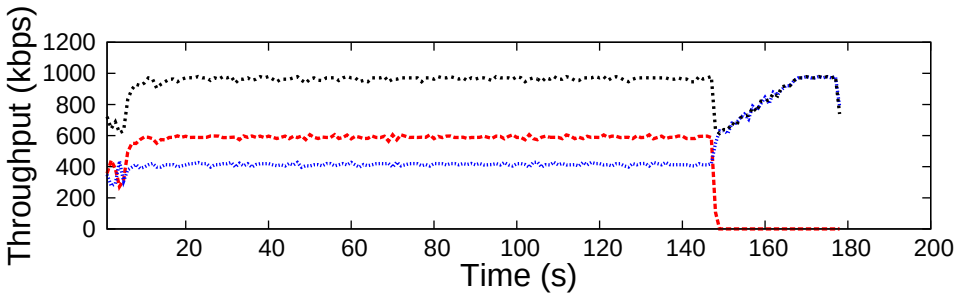
In this experiment, we vary the Interest rate of the two consumers for DIL and see how its performance is affected. Specifically, we increase the Interest rate of one flow from 200 to 300 per second and draw the throughput of the two flows in Figure 6.22. The result shows that the two flows still share the bandwidth equally even though one is sending Interests more aggressively than the other. Therefore the fairness of DIL will not be affected by ill-behaved consumer applications.

Varying RTT:

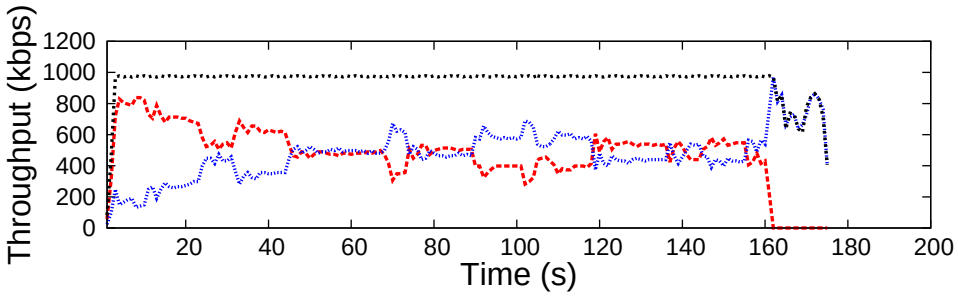
In this scenario, the delay of link *D-F* is set to be 100 ms so that the two flows have different RTT. Other settings remain the same as in the base case. The throughput and finishing time for different schemes are presented in Figure 6.23. As in the base case, DIL still achieves stable throughput for individual flows. The flow with shorter RTT gets higher throughput in all schemes. HIS has the smallest difference of finishing time between two flows. However, the flow with longer RTT still finishes earlier in DIL than in HIS.

Varying Flow Start Time:

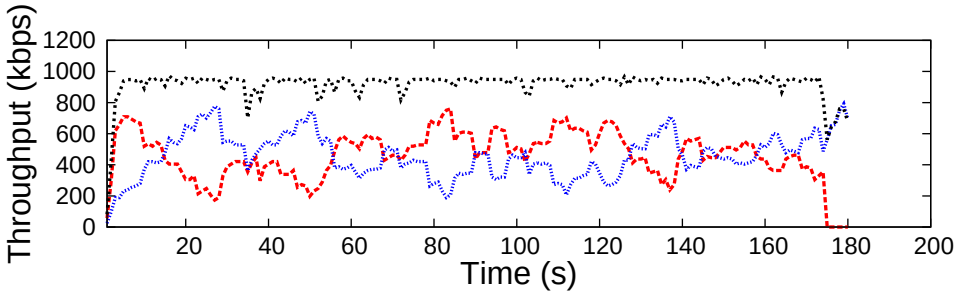
In this experiment, we vary the starting time of the two consumers to study how fast each scheme is able to converge to fairness. We let one consumer start 50 seconds after the other. Figure 6.24 shows that DIL converges to fairness quickly and provides stable throughput for the two flows. On the other hand, the flow that starts later gets lower throughput most of the time in both AIMD and HIS.



(a) DIL



(b) AIMD

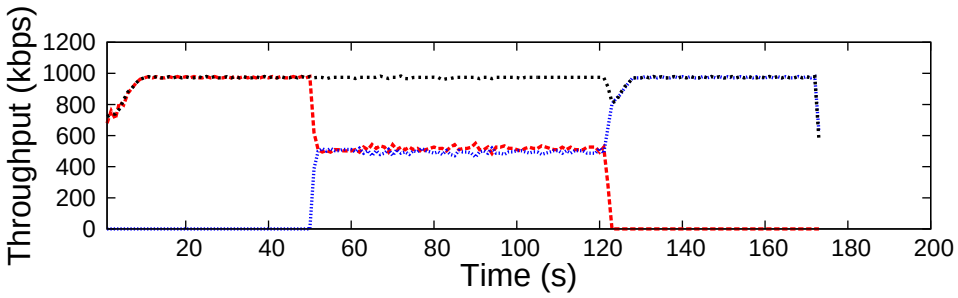


(c) HIS

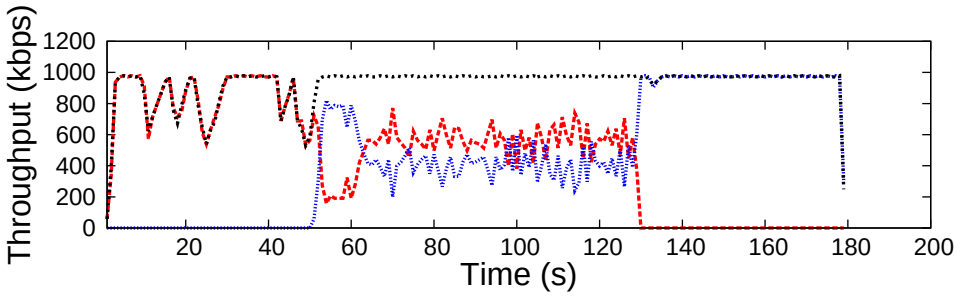
Figure 6.23: Throughput and finishing time with different RTT.

6.4.5 Multipath Congestion Control with DIL

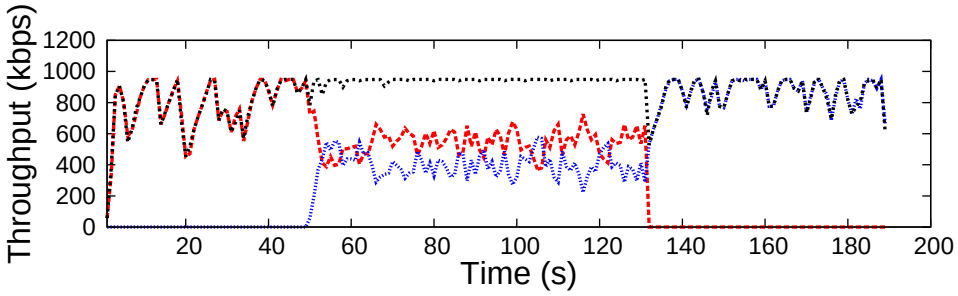
We repeat the large-scale experiment in Section 6.1 on the Sprint PoP-level topology to examine how DIL works in multipath congestion control. The delay and cost of the links are provided by Rocketfuel [62]; bandwidth of all links are set to be 1 *Mbps*. Drop-tail queues with size 20 packets are installed on all links. In each run of the experiment, we randomly generate 20 pairs of nodes and run one flow between each pair. For each flow, the provider sends 2000 pieces of data to the consumer, the size of which is 1040 bytes. The consumer Interest rate is 100 per second for DIL. Each flow starts 1 second after the previous one. We run each experiment for



(a) DIL



(b) AIMD



(c) HIS

Figure 6.24: Throughput and finishing time with different starting time.

100 times and draw the final finishing time of all flows.

Figures 6.25 presents the finishing time of DIL and AIMD. DIL is able to finish faster than AIMD in all except three cases. Figure 6.26 shows the finishing time of DIL and TCP. The figure shows that DIL finishes faster than TCP in all runs, whereas in Section 6.1 TCP finishes faster in 6 of the cases. Therefore, we conclude that DIL is compatible with the BestRoute forwarding strategy presented in Chapter 4 and is able to achieve more effective multipath congestion control than SIL.

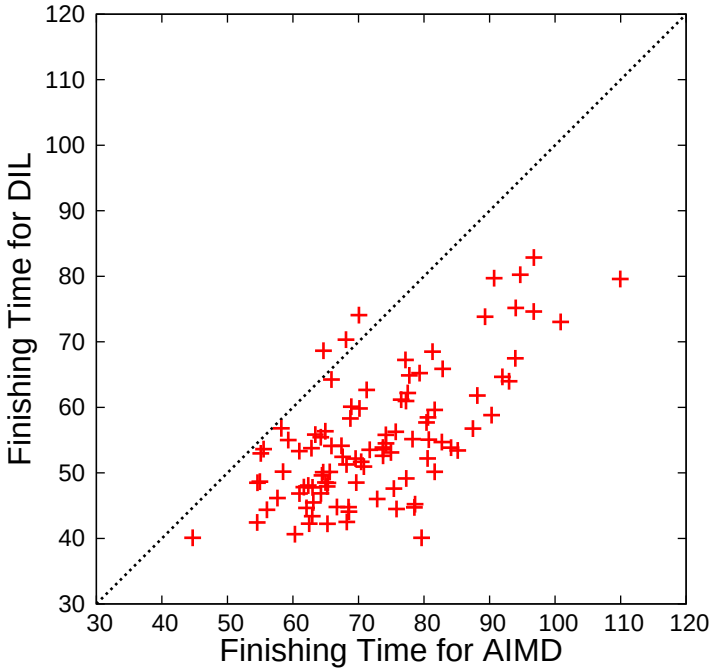


Figure 6.25: Finishing time for DIL and AIMD.

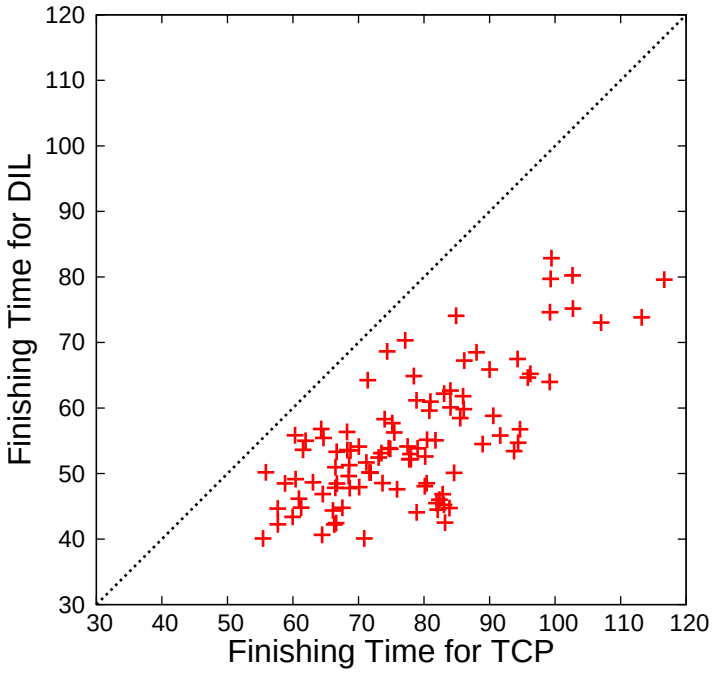


Figure 6.26: Finishing time for DIL and TCP.

CHAPTER 7

DISCUSSION AND FUTURE WORK

In this chapter we discuss the issues that are closely related but not addressed in this dissertation as well as potential directions for future research.

7.1 Forwarding State Overhead

Compared to CCNx, NDN-BestRoute is more efficient at the cost of higher complexity. It works best in an Internet-like environment. It may not suit the scenarios where complexity is the major concern. For example, in wireless sensor networks where memory and power is limited, an alternative design may be required.

The presence of NDN's datagram state in the PIT brings significant cost in both router storage and packet processing. Since an Interest stays in the PIT until the corresponding Data packet returns, the number of PIT entries associated with each outgoing interface is roughly of the order of $Bandwidth \times \overline{RTT}/\overline{P}$, where $\overline{RTT}$ and $\overline{P}$ are the average round-trip time and Data packet size. For an interface with 10 $Gbps$ bandwidth, we will have $100K$ PIT entries assuming $\overline{RTT} = 100\ ms$ and $\overline{P} = 1250$ bytes. If a router has 10 such interfaces, its PIT needs to hold 1 million entries. Although today's core routers can handle more than $1M$ entries in IP routing tables, a PIT entry is larger in size than an IP routing entry due to variable-length names and additional forwarding information. In addition, the PIT table will grow proportionally as routers get more interfaces and bandwidth grows larger over time. Therefore, providing the storage needed by NDN routers is a challenging task.

Operating on the PIT represents an even bigger challenge. IP routers only need to perform lookup over the FIB, but NDN routers need to not only lookup the PIT but also write into it. For example, when a new Interest arrives, a PIT entry

needs to be inserted; when the same Interest arrives from multiple interfaces, a PIT entry needs to be updated; when a matching Data returns, a PIT entry needs to be removed. These are more expensive operations than lookup that require fast and scalable solutions.

In summary, a scalable forwarding engine needs both novel data structures to store the PIT efficiently and novel algorithms to operate on the PIT at wire speed. Designing a scalable forwarding engine is not the purpose of this dissertation and is by itself an important research problem. There have already been considerable research efforts underway to tackle these issues (e.g., [67], [72], [60], [68]). Wang et al. [68] developed a GPU-based name lookup engine that is able to perform $63.52M$ searches per second. So et al. [60] implemented a forwarding engine on a Cisco ASR 9000 router that is able to forward NDN traffic at 20 *Gbps* or higher.

7.2 New Routing Schemes

Routing is a necessary subsystem for any large scale network. Like IP, NDN itself does not dictate what kinds of routing algorithms or protocols to use. However one can take advantage of NDN's adaptive forwarding plane to improve the stability and scalability of existing routing protocols, as well as enable routing protocols that deem difficult to adopt in IP networks.

Traditional Routing Protocols: As we have discussed in this dissertation, traditional routing protocols such as OSPF, RIP, and BGP can benefit greatly from NDN's adaptive forwarding plane. Since fast routing convergence is no longer a requirement, these routing protocols can be tuned for synchronizing among routers long-term topology and policy information without handling short-term churns. Routing assumes a supporting role to forwarding. It provides a reasonable starting point for forwarding which can then effectively explore different choices. Its job becomes more of disseminating topology and policy information than distributed computation of best paths. This new division of labor between routing and forwarding makes routing protocols simpler and more scalable.

Centralized Routing: Routing protocols have been designed to operate in a distributed manner to avoid single point of failure [16]. However with the increasing complexity in network management, Software-Defined Networking (SDN) has emerged to enable centralized management and control of networks, including logically centralized routing scheme. It is much easier to change the routing configurations on a central controller than on all participating routers, and to implement sophisticated traffic engineering schemes at the controller than individual routers. Routing overhead can also be greatly reduced, since routing updates only need to be sent to the controller instead of being flooded to the entire network, and only the controller needs to perform SPF computations.

However, a centralized routing scheme is also associated with several disadvantages, e.g., single point of failure and potentially longer convergence delay. One can mitigate single point of failure by physical replication of the central controller, which adds both the cost and complexity. A biggest concern is potentially prolonged convergence delay, which includes failure detection at local router, report to the controller, route recompilation at the controller, and dissemination of new routes to individual routers. NDN's adaptive forwarding removes the demands on convergence delay. As we have shown, NDN routers can adapt to network changes without waiting for routing to converge, making centralized routing feasible.

Coordinate-based Routing: In coordinate-based routing, instead of disseminate the network topology to routers, the coordinates of nodes are disseminated. The main characteristics of the network topology are embedded in the coordinates. Routers do greedy routing based on coordinates, i.e., forward packets to the neighbor whose distance (computed using coordinates) to the destination is the shortest among all neighbors. One example of such routing scheme is hyperbolic routing [53]. The advantages of this routing scheme include smaller routing tables (i.e., only need to know the destination's coordinates and neighbor routers' coordinates) and minimal routing updates (i.e., link failures and recovery do not affect a node's coordinates). However, in IP networks, this routing scheme is not guaranteed to be able to deliver packets. It is possible that the forwarding process runs into a local minimal,

where all neighbors are farther to the destination than the current router. Path stretch may also get large. NDN's adaptive forwarding can fix these problems and make this routing scheme a possibility.

7.3 Congestion Control

In Chapter 6 we use CIR consumer applications to show the effectiveness of DIL. However, CIR is not a good choice for practical usage. Bandwidth cannot be fully utilized if the Interest rate is too low. But if the Interest rate is too high, there will be too many NACKs between the consumer application and the NDN forwarder, leading to waste of computing resources. On the other hand, AIMD is not a good choice either when routers can provide effective hop-by-hop congestion control, since it reduces the sending window too aggressively upon congestion. It is interesting to explore new consumer strategies to work with DIL, e.g., AIAD (additive increase, additive decrease).

With FIL, flows with bigger Data packet sizes or short RTT will take advantage. The algorithm can be improved to provide strict fairness among all flows. We need to record the average packet sizes and RTT of each flow, and compute the weight for each flow using such information. Then we can implement a Weighted Fair Interest Limiting algorithm taking such information into consideration in computing Interest limit for each flow.

HIS [66] implements an Interest queue, which keeps Interests waiting at the interface when the Interest limit is reached. Therefore consumers may suffer from extra queuing delay introduced by the Interest queue. In DIL, on the other hand, a node will try alternative interfaces immediately if the Interest limit is reached on the current interface. Applications may also suffer from extra delay if the alternative paths are longer than the original paths. We plan to conduct more experiments to study whether Interest queue is needed.

When NDN is deployed as an overlay on top of IP, it shares the bandwidth with underlying IP traffic. It is interesting to learn how DIL works together with

underlying TCP flows. Although DIL also performs AIMD window adjustment, it is not clear whether it is TCP-friendly. We will conduct more experiments for future work to study the TCP-friendliness of DIL.

CHAPTER 8

RELATED WORK

This dissertation studies NDN adaptive forwarding. We first proposed NDN-BestRoute, a new forwarding plane design for NDN; then we examined how NDN-BestRoute handles network failures and studied the role of routing in NDN; finally we explored hop-by-hop multipath congestion control in NDN. In this chapter we present a number of existing work related to this dissertation. Selected forwarding plane design for other network architectures is discussed in Section 8.1. Section 8.2 describes different IP mechanisms for fast failure recovery. In Section 8.3 we summarize popular congestion control mechanisms in both NDN and IP.

8.1 Forwarding Plane Design

The IP architecture takes the "smart routing, dummy forwarding" approach. Due to its stateless nature, the forwarding plane strictly follows the routing state. Recent research efforts have recognized that introducing adaptability to the forwarding plane is a promising approach. Wendlandt et al. [69] and Caesar et al. [20] argue that networks should provide end-hosts with multiple path choices, and end-hosts should be responsible for choosing different paths based on their observed forwarding plane performance. Works such as Pathlet Routing [31], Routing Deflections [71] and Path Splicing [48] are specific designs along this direction. The main differences among them are in the specifics of how alternative paths are obtained and used.

Although NDN is considered as one of the Information-Centric Networking (ICN) architectures, NDN's forwarding plane design is drastically different from that of other ICN architecture proposals. For example, PURSUIT [10] is a recently proposed publish/subscribe network architecture. PURSUIT employs source routing for packet forwarding. Its forwarding path is encoded in a bloom filter carried in

the packet header; routers forward packets according to the path information carried in the bloom filter [37]. In case of failures, PURSUIT relies on preset backup paths to get packets delivered, therefore the network's adaptability is limited. Since each packet carries its own path in a PURSUIT network, PURSUIT routers do not maintain datagram state as NDN routers do. At the same time, individual PURSUIT routers also cannot measure the forwarding plane performance nor collect feedback to adapt to failures as NDN routers do.

8.2 Fast Failure Recovery

Francois et al. show that sub-second link-state routing convergence in large intra-domain networks is achievable by tuning various timers [29]. But their method incurs extra routing overhead and may cause routing instability. Fast reroute (FRR) mechanisms handle link failures using pre-computed alternative paths. MPLS FRR mechanisms ([52, 50]) are proposed to provide backup paths in MPLS-enabled networks in order to handle failures of specific links. Similarly, IPFRR mechanisms ([15, 14, 58, 57, 44]) provide temporary alternative paths before routing convergence in pure IP networks. However, it is hard for the FRR mechanisms to cover all possible failure scenarios. In addition, they cannot efficiently handle multiple concurrent failures. Therefore, the FRR mechanisms still require fast routing convergence.

End-to-end multipath solutions ([31, 71, 48]) provide fast failure recovery controlled by end hosts. Take Path Splicing (PS) [48] for example, each router provides multiple routing tables and let end hosts specify which one to use at each router. PS may take long time to find alternative paths, and sometimes may not be able to find them even if they exist. Therefore fast routing convergence is still required. Multiple routing configuration (MRC) [42] also uses multiple routing configurations to handle network failures. Different from PS, MRC lets routers switch configuration when failures are detected. However, MRC may not handle multiple concurrent failures well. Since each router needs to maintain multiple routing tables in PS and

MRC, the computation overhead during routing convergence increases significantly with the number of routing tables.

There are also solutions that carry routing or forwarding information in packet headers. Failure carrying packets (FCP) [43] put failure information into the packet headers, and let routers recompute the routing tables on-the-fly upon receipt of FCP. However, the method increases computation overhead, and the sizes of FCP headers may become arbitrarily large. Packet Re-cycling (PR) [46] reroutes packets along pre-computed backup paths in case of failures. In addition to the ordinary routing table, each router also creates a cycle following table, the generation of which is an NP-hard problem. When failures are detected, PR bits are set in packet headers to guide packet forwarding. Liu et al. propose Data-Driven Connectivity (DDC) [45] to ensure forwarding connectivity at the data plane. DDC organizes the network as a destination-oriented directed acyclic graph (DAG) to avoid loops, and uses two bits in the packet header to notify link reversal. DDC has its own control plane algorithm, therefore cannot make use of existing routing protocols.

8.3 Congestion Control

Congestion control is a critical issue in any existing network architecture. IP congestion control has been studied extensively over the last three decades. Although NDN has only existed for a few years, its congestion control has also received considerable attention. We summarize the related work in IP and NDN congestion control in Section 8.3.1 and 8.3.2 respectively.

8.3.1 Congestion Control in IP

The TCP congestion avoidance algorithm proposed by Jacobson et al. in 1988 [35] provides the primary basis for congestion control in today's Internet. Since then, many variations of TCP have been proposed to improve the congestion control performance, e.g., TCP Vegas [17], TCP CUBIC [32] and TCP NewReno [34]. However, only TCP's end-to-end congestion avoidance is not enough. When short drop-tail

queues are used on the bottleneck links, multiple TCP flows may experience packet loss concurrently, leading to TCP global synchronization [33]. On the other hand, long drop-tail queues may cause the bufferbloat problem [51], in which packets are held in the queues for excessively long time, causing high latency and jitter for applications as well as reduced overall network throughput. As a result, Active Queue Management (AQM) mechanisms such as Random Early Detection (RED) [28] and Controlled Delay (CoDel) [51] have been introduced to address these issues. In addition, Explicit Congestion Notification (ECN) [54] as an extension to IP and TCP was proposed to provide end-to-end notification of congestion without actually dropping packets.

Fairness is another important aspect of congestion control. TCP alone cannot ensure fairness in the network since there are applications that do not use TCP and nor are they TCP-friendly. Traditional scheduling algorithms like Fair Queuing (FQ) [25] require routers to maintain per-flow state, making them unsuitable for practical usage. Stoica et al. propose Core-Stateless Fair Queuing (CSFQ) [63] to achieve approximately fair bandwidth allocation at routers with small implementation complexity. XCP [38] is the first work that decouples fairness control from utilization control. In XCP, routers provide explicit feedback for applications which can then adjust their sending rate more quickly and accurately. Dukkipati et al. suggest to use flow completion time as the metric for congestion control and propose RCP [26] to help short flows gain their fair share of the bandwidth more quickly.

Multipath TCP (MPTCP) [70] is a recently proposed multipath congestion control mechanism for IP networks. It utilizes multiple interfaces (IP addresses) of multi-homed hosts to set up multiple sub-TCP connections between the two ends. Assuming that each source-destination IP address pair represents a separate physical path, one can split traffic over the multiple end-to-end paths.

8.3.2 Congestion Control in NDN

Considerable research has been conducted on congestion control in NDN. Many of the previous work follow the end-to-end approach of TCP by adjusting consumer Interest window size using the AIMD algorithm [18, 21, 24, 59]. However, the heterogeneous RTT caused by caching, multiple providers and multipath forwarding makes it hard to set Retransmission Timeout (RTO) as in TCP to detect congestion. Consequently, some solutions seek to predict RTT more accurately by piggybacking location information of future content on the Data packets [18, 24, 59].

Hop-by-hop interest shaping mechanisms have also been studied [23, 56, 66, 22]. However, HR-ICP [23], HIS [66] and HoBHIS [56] cannot effectively handle the dynamics in returning Data traffic since their Interest shapers require prior knowledge of either returning delay or sizes of the Data packets. Moreover, HIS and HoBHIS do not consider fairness in their Interest shapers. [22] achieves optimal multipath congestion control, but its objective is to minimize the number of pending Interests on the most loaded interface. Therefore Interests may not be forwarded along the best routing paths even if they are not congested.

CHAPTER 9

CONCLUSIONS

In this dissertation, we present a concrete design of NDN's forwarding plane to make the network *resilient* and *efficient*. NDN's unique communication model of retrieving data by name leads to a forwarding plane that maintains datagram state at every router. Because datagram is the most basic communication unit in packet-switched networks, this per-hop datagram state potentially provides the flexibility to solve a host of existing network problems. In this dissertation we present NDN-BestRoute, a concrete forwarding plane design for NDN that utilizes this datagram state to provide high performance and resilience in NDN networks. Specifically, we design data structures and methods to collect and store forwarding performance information for NDN routers. We also propose a novel Interest NACK mechanism for fast fault detection and an efficient forwarding strategy for multipath fault recovery. We quantitatively evaluate the packet delivery performance of NDN-BestRoute under adverse conditions. Our results show that NDN's adaptive forwarding mechanism can provide excellent performance in handling prefix hijacks and link failures.

Additionally, this dissertation studies the role of routing in NDN. NDN's adaptive forwarding plane leads to a new division of labor between routing and forwarding plane. While the latter can detect and recover from link failures quickly independent from the former, the former helps bootstrap adaptive forwarding and handle link recovery. We specify how NDN routing protocols coordinate with the forwarding plane through interface ranking and probing mechanisms. Our analysis and extensive simulations show that NDN-BestRoute's packet delivery performance after link failures does not rely on fast routing convergence. As a result, NDN routing protocols can benefit greatly from the adaptive forwarding plane. By masking short-lived failures from routing protocols, NDN's routing stability and scalability can be significantly improved. Moreover, adaptive forwarding also enables new and existing

routing schemes that may not work well in IP to be used in NDN.

Furthermore, we propose a Dynamic Interest Limiting mechanism for hop-by-hop multipath congestion control in NDN. The two-way symmetric Interest-Data exchange and one-on-one flow balance enables NDN routers to avoid congestion on the link from an upstream router by limiting the number of Interests sent to it. DIL dynamically adjusts the Interest limit on each interface based on the current traffic condition. Since accurate RTT estimate is hard in NDN, we propose two new methods for congestion detection in native NDN networks and NDN overlay networks which do not rely on RTT estimate. We also present a Fair Interest Limiting mechanism to ensure fairness among multiple flows. Evaluation results show that DIL is able to achieve efficient and fair bandwidth utilization under different congestion scenarios. When combined with adaptive forwarding, DIL can utilize network resources more effectively than today's TCP/IP through multipath congestion control.

REFERENCES

[1] http://www.cisco.com/en/US/products/ps6599/products_white_paper09186a00800a4409.shtml. Accessed on June 10, 2014.

[2] http://www.scalable-networks.com. Accessed on June 10, 2014.

[3] Abilene topology data. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/. Accessed on June 10, 2014.

[4] CCNPing. http://github.com/NDN-Routing/ccnping. Accessed on June 10, 2014.

[5] CCNx. http://www.ccnx.org/. Accessed on June 10, 2014.

[6] COntent Mediator architecture for content-aware nETworks (COMET). http://www.comet-project.org/. Accessed on June 10, 2014.

[7] Mini-CCNx. http://github.com/carlosmscabral/mn-ccnx. Accessed on June 10, 2014.

[8] NDN Testbed. http://named-data.net/ndn-testbed. Accessed on June 10, 2014.

[9] NS-3 Network Simulator. http://www.nsnam.org/. Accessed on June 10, 2014.

[10] PURSUIT Internet Technology. http://www.fp7-pursuit.eu/PursuitWeb/. Accessed on June 10, 2014.

[11] The FP7 4WARD Project. http://www.4ward-project.eu/. Accessed on June 10, 2014.

[12] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN Project, July 2012.

[13] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards Milli-Second IGP Convergence. Internet Draft draft-alaettinoglu-isis-convergence-00.txt, November 2000.

[14] A. Atlas. U-turn Alternates for IP/LDP Fast-Reroute. draft-atlas-ip-local-protect-uturn-03, Feburary 2006.

[15] A. Atlas and A. Zinin. RFC 5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates. RFC 5286, 2008.

[16] Paul Baran. On Distributed Communications Networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, March 1964.

[17] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE J.Sel. A. Commun.*, 13(8):1465–1480, September 2006.

[18] S. Braun, M. Monti, M. Sifalakis, and C. Tschudin. An Empirical Study of Receiver-Based AIMD Flow-Control Strategies for CCN. In *Proceedings of ICCCN*, 2013.

[19] Jeff Burke, Alex Horn, , and Alessandro Marianantoni. Authenticated Lighting Control Using Named Data Networking. Technical Report NDN-0011, NDN Project, October 2012.

[20] Matthew Caesar, Martin Casado, Teemu Koponen, Jennifer Rexford, and Scott Shenker. Dynamic Route Recomputation Considered Harmful. *ACM SIGCOMM Computer Communication Review (CCR)*, 40(2):66–71, April 2010.

[21] G. Carofiglio, M. Gallo, and L. Muscariello. ICP: Design and evaluation of an Interest control protocol for content-centric networking. In *Proceedings of IEEE INFOCOMM NOMEN Workshop*, 2012.

[22] G. Carofiglio, M. Gallo, L. Muscariello, M. Papalini, and Sen Wang. Optimal multipath congestion control and request forwarding in Information-Centric Networks. In *Proceedings of IEEE ICNP*, 2013.

[23] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. Joint Hop-by-hop and Receiver-driven Interest Control Protocol for Content-centric Networks. In *Proceedings of ACM SIGCOMM ICN Workshop*, 2012.

[24] Giovanna Carofiglio, Massimo Gallo, Luca Muscariello, and Michele Papalini. Multipath Congestion Control in Content-Centric Networks. In *Proceedings of IEEE INFOCOMM NOMEN Workshop*, 2013.

[25] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of SIGCOMM*, 1989.

[26] Nandita Dukkipati and Nick McKeown. Why Flow-completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006.

[27] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166, 2008.

[28] Sally Floyd and Van Jacobson. Random Early Detection gateways for congestion avoidance. *IEEE ACM Transactions on Networking*, 1(4):397–413, 1993.

[29] Pierre Francois, Clarence Filsfils, John Evans, and Olivier Bonaventure. Achieving Sub-Second IGP Convergence in Large IP Networks. *ACM SIGCOMM Computer Communication Review (CCR)*, 35(3):35–44, July 2005.

[30] C. Ghali, G. Tsudik, and E. Uzun. Needle in a Haystack: Mitigating Content Poisoning in Named-Data Networking. In *Proceedings of NDSS SENT Workshop*, 2014.

[31] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In *Proceedings of ACM SIGCOMM*, 2009.

[32] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[33] H. Han, C.V. Hollot, D. Towsley, and Y. Chait. Synchronization of TCP Flows in Networks with Small DropTail Buffers. In *Proceedings of IEEE CDC-ECC*, 2005.

[34] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582, 2012.

[35] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of SIGCOMM*, 1988.

[36] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking Named Content. In *Proceedings of ACM CoNEXT*, 2009.

[37] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. LIPSIN: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM conference on data communication*, 2009.

[38] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proc. of SIGCOMM*, 2002.

[39] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-Oriented (and Beyond) Network Architecture. In *Proceedings of ACM SIGCOMM*, 2007.

[40] Derek Kulinski and Jeff Burke. NDN Video: Live and Prerecorded Streaming over NDN. Technical Report NDN-0007, NDN Project, September 2012.

[41] N. Kushman, S. Kandula, D. Katabi, and B. Maggs. R-BGP: Staying connected in a connected world. In *Proceedings of NSDI*, 2007.

[42] A. Kvalbein, A.F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP Network Recovery Using Multiple Routing Configurations. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006.

[43] Karthik Lakshminarayanan, Matthew Caesar, Murali Rangan, Tom Anderson, Scott Shenker, and Ion Stoica. Achieving Convergence-Free Routing using Failure-Carrying Packets. In *Proceedings of ACM SIGCOMM*, 2007.

[44] Sanghwan Lee, Yinzhe Yu, Srihari Nelakuditi, Zhi li Zhang, and Chen nee Chuah. Proactive vs reactive approaches to failure resilient routing. In *Proceedings of IEEE Infocom*, 2004.

[45] Junda Liu, Baohua Yang, Scott Shenker, and Michael Schapira. Data-driven network connectivity. In *Proceedings of ACM HotNets Workshop*, 2011.

[46] Suksant Sae Lor, Raul Landa, and Miguel Rio. Packet Re-cycling: Eliminating Packet Losses Due to Network Failures. In *Proceedings of HotNets*, 2010.

[47] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Transactions on Networking (TON)*, 16(4):749–762, August 2008.

[48] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. In *Proceedings of ACM SIGCOMM*, 2008.

[49] J. Moy. OSPF Version 2. RFC 2328, 1998.

[50] T. Nadeau, K. Koushik, and R. Cetin. Multiprotocol Label Switching (MPLS) Traffic Engineering Management Information Base for Fast Reroute. RFC 6445, 2011.

[51] Kathleen Nichols and Van Jacobson. Controlling Queue Delay. *Queue*, 10(5):20:20–20:34, May 2012.

[52] P. Pan, G. Swallow, and A. Atlas. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090, 2005.

[53] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *INFOCOM, 2010 Proceedings IEEE*, 2010.

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

[55] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[56] N. Rozhnova and S. Fdida. An effective hop-by-hop interest shaping mechanism for ccn communications. In *Proceedings of IEEE INFOCOMM NOMEN Workshop*, 2012.

[57] S. Previdi S. Bryant and M. Shand. A Framework for IP and MPLS Fast Reroute Using Not-via Addresses. draft-ietf-rtgwg-ipfrr-notvia-addresses-11, May 2013.

[58] S. Previdi S. Bryant, C. Filsfils and M. Shand. IP Fast Reroute using tunnels. draft-bryant-ipfrr-tunnels-02, April 2005.

[59] L. Saino, C. Cocora, and G. Pavlou. CCTCP: A scalable receiver-driven congestion control protocol for content centric networking. In *Proceedings of IEEE ICC*, 2013.

[60] Won So, Ashok Narayanan, and David Oran. Named Data Networking on a Router: Fast and Dos-resistant Forwarding with Hash Tables. In *Proceedings of ANCS*, 2013.

[61] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

[62] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

[63] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-speed Networks. *IEEE/ACM Trans. Netw.*, 11(1):33–46, February 2003.

[64] Daniel Turner, Kirill Levchenko, Stefan Savage, and Alex C. Snoeren. A Comparison of Syslog and IS-IS for Network Failure Analysis. In *Proceedings of IMC*, 2013.

[65] Lan Wang, A K M Mahmudul Hoque, Cheng Yi, Adam Alyyan, and Beichuan Zhang. OSPF-N: OSPF for NDN Routing. Technical Report NDN-0003, NDN Project, July 2012.

[66] Yaogong Wang, Natalya Rozhnova, Ashok Narayanan, David Oran, and Injong Rhee. An Improved Hop-by-hop Interest Shaper for Congestion Control in Named Data Networking. In *Proceedings of ACM SIGCOMM ICN Workshop*, 2013.

[67] Yi Wang, Keqiang He, Huichen Dai, Wei Meng, Junchen Jiang, Bin Liu, and Yan Chen. Scalable Name Lookup in NDN Using Effective Name Component Encoding. In *Proceedings of IEEE ICDCS*, 2012.

[68] Yi Wang, Yuan Zu, Ting Zhang, Kunyang Peng, Qunfeng Dong, Bin Liu, Wei Meng, Huichen Dai, Xin Tian, Zhonghu Xu, Hao Wu, and Di Yang. Wire Speed Name Lookup: A GPU-based Approach. In *Proceedings of USENIX NSDI*, 2013.

[69] Dan Wendlandt, Ioannis Avramopoulos, David G. Andersen, and Jennifer Rexford. Don't secure routing protocols, secure data delivery. In *Proceedings of ACM HotNets Workshop*, 2006.

[70] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. of Usenix NSDI*, 2010.

[71] Xiaowei Yang and David Wetherall. Source Selectable Path Diversity via Routing Deflections. In *Proceedings of ACM SIGCOMM*, 2006.

[72] Haowei Yuan, Tian Song, and Patrick Crowley. Scalable NDN forwarding: Concepts, issues, and principles. In *Proc. of IEEE ICCCN*, 2012.

[73] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D. Thornton, Diana K. Smetters, Beichuan Zhang, Gene Tsudik, kc claffy, Dmitri Krioukov, Dan Massey, Christos Papadopoulos, Tarek Abdelzaher, Lan Wang, Patrick Crowley, and Edmund Yeh. Named data networking (NDN) project. Technical Report NDN-0001, NDN Project, October 2010.

[74] Zhenkai Zhu, Chaoyi Bian, Alexander Afanasyev, Van Jacobson, and Lixia Zhang. Chronos: Serverless Multi-User Chat Over NDN. Technical Report NDN-0008, NDN Project, October 2012.