MULTIPATH DATA TRANSPORT IN NAMED DATA NETWORKING

by

Klaus Michael Schneider

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

2019

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by: **Klaus Schneider,** titled: **Multipath Data Transport in Named Data Networking** and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Beichuan Zhang Date: 13. November 2019

Chris Gniady Date: 13. November 2019

John Hartman Date: 13. November 2019

Alon Efrat Date: 13. November 2019

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.

Beichuan Zhang Date: 13. November 2019
Dissertation Committee Chair
Department of Computer Science

ACKNOWLEDGEMENTS

First, I want to thank my faculty advisor, Dr. Beichuan Zhang, for providing me with advice and guidance over the last few years. Next, I want to thank my dissertation committee members, Dr. Chris Gniady, Dr. John Hartman, and Dr. Alon Efrat for their helpful comments in our meetings. I also want to thank my previous advisor at the University of Bamberg, Prof. Dr. Udo Krieger, and his students Philipp Eittenberger and Marcel Großman for their support and helpful advice. In addition I want to thank Prof. Dr. Guido Wirtz and Prof. Dr. Ute Schmid for writing the recommendation letters that allowed me to join the University of Arizona.

I would to like to thank my current and previous colleagues from the University of Arizona, Teng Liang, Eric Newberry, Junxiao Shi, Mathias Gibbens, Ashiq Rahman, Chavoosh Ghasemi, Zhuzhi Cai, Weiwei Liu, and Ju Pan. They were invaluable in our many personal and group discussions and for helping to implement my work to make it accessible to a larger audience.

Moreover, I am grateful to my colleagues and collaborators in the NDN project team, including Lixia Zhang, Lan Wang, Alexander Afanasyev, Jeff Burke, Peter Gusev, Davide Pesavento, Ashlesh Gawande, Spyros Mastorakis, Susmit Shannigrahi, Vince Lehman, John Dehart, Nick Gordon, Jay Misra, Yumin Xia, Siham Khoussi, Van Sy Mai, and Lotfi Benmohamed. I also want to thank my former colleagues at Intel Labs, Kathyayani Srikanteswara, Maruti Gupta, Gabriel Arrobo, and Ravikumar Balakrishnan.

Last but not least, I want to thank my friends and family for supporting me during my studies. None of my accomplishments would have been possible without them.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The idea of multipath transport is almost as old as the Internet itself. By using multiple paths, one can build a reliable system out of many unreliable components. Moreover, by distributing the load among multiple paths, one can significantly improve the performance of this system.

In this dissertation, we investigate the use of multipath data transport in *Named Data Networking* (NDN), a novel Internet architecture. NDN's core features (stateful forwarding, pervasive caching, multi-destination routing) make it inherently more capable of exploiting multiple paths than the current TCP/IP Internet. For example, in NDN, the same data can be served by multiple repositories and caches, resulting in many paths being used.

Specifically, this dissertation works out the details to improve NDN's performance in three different aspects: First, we design a *congestion control scheme* that improves performance when data is retrieved over multiple paths and from multiple repositories. Second, we design a *multipath routing protocol* that maximizes the path choice at the NDN forwarding plane, while minimizing path length and avoiding loops. Third, we design a novel *traffic engineering* scheme that distributes traffic *hop-by-hop*, rather than tunneling traffic between multiple endpoints at the network edge.

CHAPTER 1

Introduction

The Internet is of vast size and comprised of many unreliable components. Access points, wires, cables, switches, and routers are all prone to fail, yet if they do, connectivity between endpoints must be maintained at all cost. Hence, it is no surprise that the Internet uses multiple paths between endpoints, going back as far as 1964 [Bar64]. Moreover, it uses multipathing at many layers of the network stack: 1) Switches often use redundant wiring (called link "bonding"), with fall-over if one of the cables fails. 2) Data-centers use redundant topologies (called "fat tree" or "Clos" topologies), which allow parts of the network to fail and still maintain connectivity. 3) Internet Service Providers (ISPs) use the BGP protocol, which can find multiple routes towards any given destination ISP. 4) User's mobile end-devices like smart phones use both WiFi and 4G/5G networks, and switch over if one of them loses connectivity. In all these cases, multipath transport is crucial for maintaining the *reliability* of a system when faced with *unreliable* parts.

Moreover, multipath transport is also used to improve the *performance* of a system. Internet Service Providers (ISPs) use *Traffic Engineering* to make sure their existing networks can handle the maximum amount of traffic and to optimize end-user experience. Specifically, they try to avoid congestion in some parts of the network while free bandwidth is available in other parts. In this work, we focus on both the performance and the reliability aspect of multipath transport.

The current (TCP/IP) Internet architecture was created over 40 years ago, and has enabled growth in many sectors of everyday life, such as e-commerce, social media, and smart phone applications. However, the usage of the Internet has changed dramatically since then: While earlier Internet applications like email and

FTP worked similar to a phone call (a point-to-point *conversation* between two endpoints), modern applications are inherently focused on *content*: users want to access a file, picture, or video, regardless of its exact location. However, the Internet was not designed for this content-centric access, causing numerous problems in content-retrieval efficiency, security, and mobility support [ZEB+10].

Our project, *Named Data Networking* (NDN), aims to solve these issues by making data names, not IP addresses, the main way of accessing content in the network. Since, in NDN, content names are known to the network, routers can cache (temporarily store) content objects and serve them again on later request. This avoids redundant network communication, increases network efficiency, and improves user experience for all Internet applications.

Dissertation Overview This dissertation looks at ways to enable multipath data transport in Named Data Networking. NDN has two characteristics that make multipath communication inherently more frequent than current IP networks: 1) NDN supports ubiquitous and transparent caching, meaning an Interest packet (a packet requesting a chunk of data) does not know beforehand whether it will hit a cache. Every time an Interest is answered by a cache, the resulting path will be shorter (thus different) than when it reaches all the way to its original destination. 2) NDN supports the use of multiple replicated repositories for the same content. Both of these features (caching and multi-repository) cause normal data requests to be sent over a multitude of paths with different characteristics.

More specifically, we investigate how multipath data transport affects two existing mechanisms of computer networks (see Figure 1.1 and Section 2.1). The first one is the **Endpoint Congestion Control,** which determines how fast each endpoint will send or request data, with the goal to avoid network congestion. A single endpoint can adjust its sending rate on very fast timescale, in the order of one round-trip time (typically 10s to 100s of milliseconds). However, when multiple endpoints are

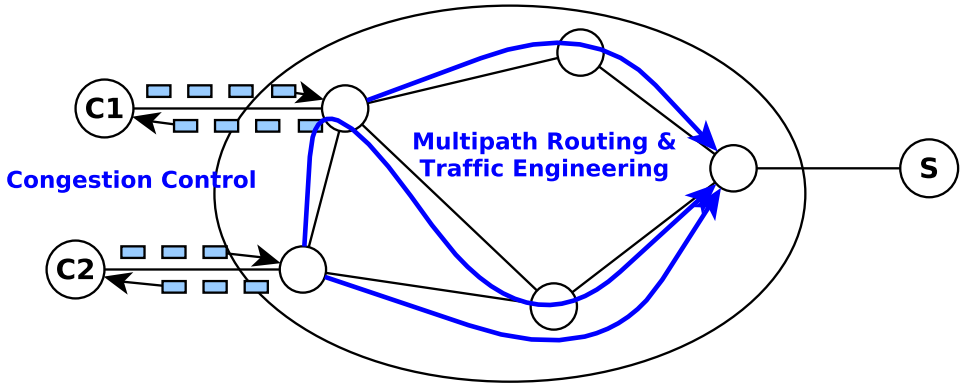


Figure 1.1: Dissertation Overview

involved, it can take longer for them to converge to a stable sending rate, the goal of which is to give every endpoint a fair share of the network bandwidth.

The second mechanism is **Routing and Traffic Engineering**, which determines which paths each packet will take through the network. Here the goal is to choose short paths, but more importantly to choose paths that are not congested/overloaded. The adaptation of routing paths usually happens on a much slower timescale than endpoint congestion control, i.e., many minutes to hours. The main reason for this difference in timescales is that traffic engineering has to consider the aggregate traffic of many (often thousands of) endpoints and that changing routes too quickly can be worse than adapting too slowly (see Section 2.1.3).

As we will see, current congestion control and traffic engineering are implemented separately and do not provide direct feedback to each other. In this work, we describe a joint design that uses explicit congestion feedback to both adjust the sending rate at the endpoints and to steer traffic inside the network.

Dissertation Details As stated above, multipath transfer has implications for the *congestion control* protocol that can be used on end-devices (see Chapter 3). Traditional congestion control mechanisms (see a summary in Section 2.1.1) are designed for regulating the sending rate of one fixed source-destination pair; not

for multipath transport. Hence, we had to design a new, more *practical congestion control* (PCON) protocol for NDN. To make it practical for deployment, our design must not assume known link bandwidths or Data packet sizes, as these assumptions may not hold for overlay links, wireless links, or applications with varying Data packet sizes. PCON detects congestion based on *Active Queue Management* (see a discussion in Section 2.1.2), then signals it towards consumers by *explicitly marking* certain packets, so that downstream routers can divert traffic to alternative paths and consumers can reduce their Interest sending rates. Our simulations show that PCON's forwarding adaptation reaches a higher total throughput than existing work while maintaining similar fairness among endpoints. Moreover, PCON can adapt to the changing capacity of IP tunnels and wireless links, conditions that other hop-by-hop schemes do not consider.

NDN has another characteristic that enables multipath forwarding: its hop-by-hop stateful forwarding plane. Every router maintains a set of nexthops (rather than just one nexthop in IP networks), and makes independent forwarding decisions towards the destination. We use this forwarding plane to design a new *Hop-by-Hop Traffic Engineering* scheme. Our scheme can be, with some adjustments, applied to IP networks, so we keep the description general. This HBH Traffic Engineering works in two steps:

First, we decide which nexthops can potentially be used at each hop (see Chapter 4). The trade-off here is maximizing the number of nexthops, while keeping paths short, and avoiding loops. Most related work (see Section 3.5) chooses the viable nexthops by applying the "Downward Criterion", i.e., only adding nexthops that lead closer to the destination; or more generally by creating a Directed Acyclic Graph (DAG) for each destination. We show that a DAG's nexthop options are necessarily limited, and that, by *using certain links in both directions* (per destination), we can add further nexthops while still avoiding loops. Our solution LFID (Loop-Free Inport-Dependent) routing, though having a slightly higher time complexity, leads to

both a *higher number* of and *shorter* potential paths than related work. LFID thus protects against a higher percentage of single and multiple failures (or congestions) and comes close to the performance of arbitrary source routing.

Second, we design the full traffic engineering scheme that uses these loop-free nexthops and dynamically distributes traffic among them (see Chapter 5). Our work is quite different from state-of-the-art Internet traffic engineering (see a summary in Section 2.1.3), which uses source-based explicit routing via Multi-Protocol Label Switchign (MPLS) or Segment Routing. Though widely adopted in practice, source routing can face certain inefficiencies and operational issues, caused by its use of bandwidth reservations. In this work, we make the case for Hop-by-Hop (HBH) Traffic Engineering: splitting traffic among nexthops at every router, rather than splitting traffic among paths only at edge routers. We show that HBH traffic engineering can achieve the original goals of MPLS (i.e., efficient use of network resources), with a much simpler design that does not need bandwidth reservations or predictions of traffic demand. We implement a prototype in the ns-3 network simulator, to investigate the cost imposed by 1) the restricted path choice of loop-free HBH multipath routing, and 2) the distributed decisions of each router, based on its local network view. We show that the former is more important than the latter, but that, other than a few outliers, our design shows a performance (= aggregate user utility) close to the theoretical optimum.

CHAPTER 2

Background

2.1 The Current Internet

2.1.1 TCP Congestion Control

The Internet protocol suite is comprised of layers that perform different functions. In the middle of the stack (Figure 2.1) lies the *network layer* with the *Internet Protocol* (IP) that establishes global connectivity and routes packets between end hosts. Right above the network layer lies the *transport layer*, which provides common functions shared by most or all applications, such as multiplexing (allowing multiple applications to share the same IP address), reliable transport, and congestion control. The two most popular transport layer protocols are TCP and UDP.

UDP provides only minimal functionality such as *port numbers* to support multiple applications and *checksums* for data integrity. It is *unreliable* (meaning packets might be lost during transport) and *connectionless* (data can be sent before establishing an end-to-end connection). Due to its low overhead, UDP is used for many applications where speed is more important than reliability, such as the Domain Name Service (DNS).

TCP, in contrast, is perfectly reliable and establishes an end-to-end connection before any data is sent. Moreover, TCP uses a stream-abstraction, where it ensures that data is received in the exact *order* in which it is sent. Lastly, and most relevant to this dissertation, TCP performs *congestion control*, meaning it regulates each application's sending rate to avoid overflowing the network. [1]

[1] TCP's functionality has grown immensely over the last four decades, and this section can only show a small slice of the whole picture. For a more complete discussion see [KR12, TW13].

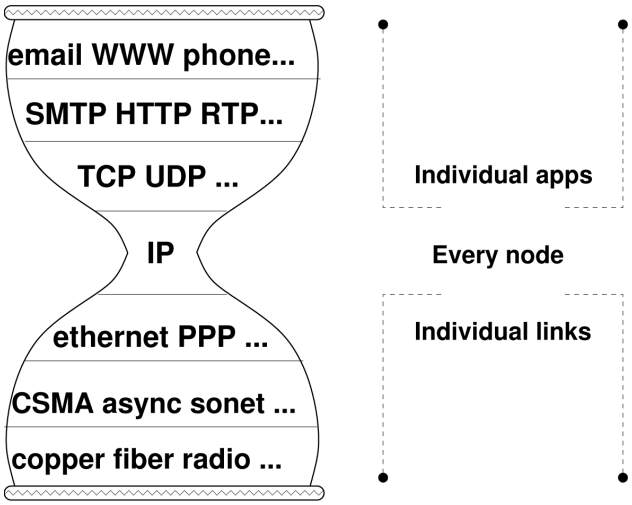


Figure 2.1: IP Hourglass Model [JST+09]

In order to provide reliability and congestion control, TCP uses *sequence numbers* and *acknowledgment* (ACK) packets. Each packet contains a sequence number, which will be answered by a packet with a matching ACK number (for efficiency ACKs can be piggybacked on data packets, and one packet can contain multiple ACK numbers). Whenever an acknowledgment is missing, TCP knows to retransmit this packet, and, since lost packets indicate network congestion, to decrease the sending rate.

The first question is how long to wait for each acknowledgment, i.e., how to set the response *timeout* (RTO). Setting the RTO too high leads to long wait times before retransmission. Setting the RTO too low leads to many unnecessary retransmissions. Both can negatively impact throughput and transmissions delays. The common solution is set the RTO based on the measured round-trip-time (RTT), i.e., the time between sending out a packet and receiving the acknowledgment. More specifically, the RTO is a function of the mean and the variance of this round-trip time:

$RTO = meanRTT + 4 * varRTT$

TCP uses another mechanism to detect lost packets even faster: noticing gaps in the ACK sequence numbers. Whenever TCP receives ACKs for sequence numbers

significantly higher (e.g. +3) than the lowest unacknowledged sequence, it will assume this packet was lost and resend it. This mechanism is known as *fast retransmit.* The assumption here is that packets will always arrive in correct sending order, and *out-of-order* packets are a sign of packet loss and congestion.

In Section 3.2, we will show that these two mechanisms (RTO setting and fast retransmit) do not work well when data is received over multiple paths and caches.

2.1.2 Bufferbloat & Active Queue Management

In the current Internet, packets often cluster together to form bursts of high activity followed by times of idleness or low activity. In order to absorb these bursts, routers in the network keep *buffers* that queue up packets when the incoming rate is temporarily faster than the outgoing rate. As long as these buffer queues are *temporary* (disappear after a few RTTs) they are useful and necessary to allow TCP to achieve the maximal possible sending rate.

However, as noted by Nichols, Jacobson, and others [NJ12a], buffers will often show a *permanent* queue that does not dissipate even after many round-trip times. These permanent queues do not increase TCP's throughput, but only increase end-to-end latency. This has been called the *Bufferbloat* problem.

In order to solve bufferbloat, a large literature has developed on *Active Queue Management* (AQM), which goal is to avoid permanent buffer queues and to reduce packet latency. The general solution is to drop some packets early, before the buffer fills up completely, hence signaling TCP to slow down and let the queue dissipate.

An addition to AQM is, instead of dropping packets, to explicitly mark them. This is known as *Explicit Congestion Notification* (ECN). The goal is to avoid the retransmissions caused by packet drops, and thus be more efficient than a dropping AQM. We use active queue management together with ECN packet marking in the design of our congestion control scheme in Section 3.3.

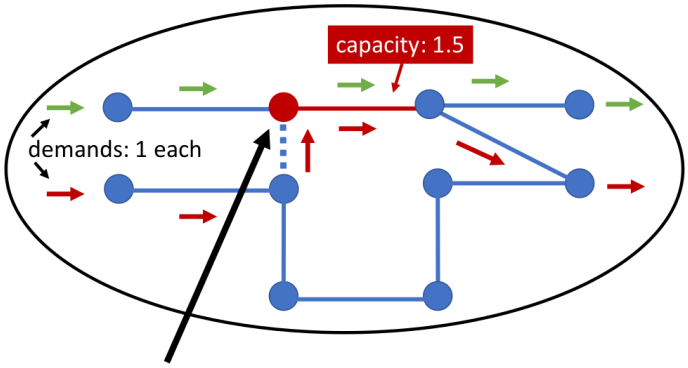


Figure 2.2: Congestion on shortest path [GVKH18a]

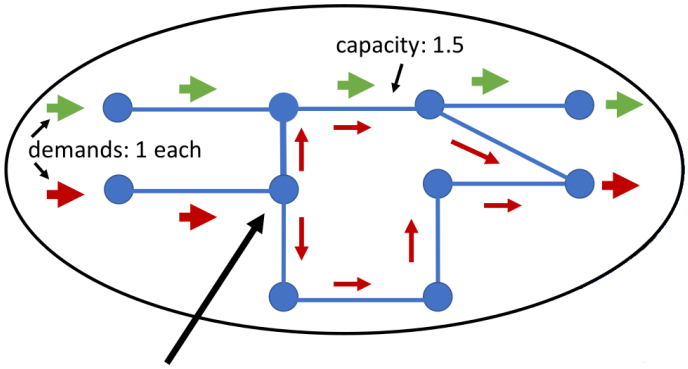


Figure 2.3: Reduced congestion via traffic engineering [GVKH18a]

2.1.3 Routing & Traffic Engineering

As discussed above, the IP network layer is responsible for routing packets to their destination. The common way to do this is to send packets on the *shortest path*, which can be computed efficiently via Dijkstra's algorithm [Dij59]. However, using shortest path routing alone can sometimes leave some part of the network congested, while other parts contain free capacity (see Figure 2.2).

The solution for this is to send some traffic on a longer, non-shortest path (like in Figure 2.3). This is called *Traffic Engineering*. There are many ways to perform traffic engineering in modern computer networks. A first approach, tried out in the early ARPANET, is to dynamically adjust the link weights to the current network

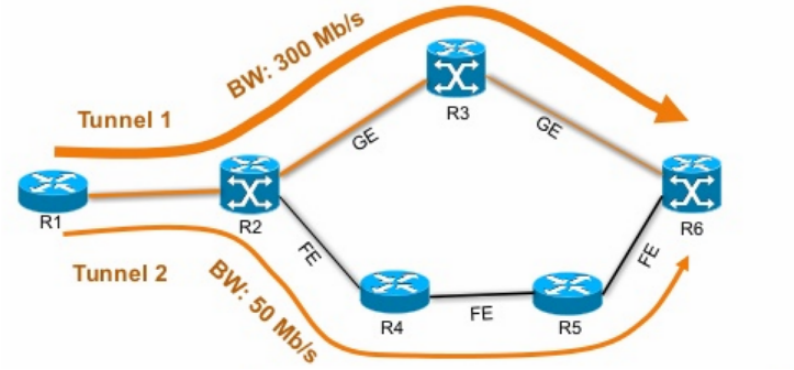


Figure 2.4: Example of MPLS Traffic Engineering

load, more specifically to the measured end-to-end latency. Now, if certain links get congested, routers will detect the additional delay via periodic measurements and increase the *weight* or *metric* of the link. For the next shortest path calculation, the link with a higher weight will have less paths using it, thus the load will decrease.

While this idea sounds good in theory, when tested in the real ARPANET, the results were quite bad [KZ89a]. The dynamic adjustment of link weights caused routing instabilities and wasted network bandwidth and processing power. The algorithm was later revised [KZ89a] to limit the maximum link metric (to three times the minimum metric) and also limit the amount change in each iteration, both of which resulted in a much more stable network.

The general idea of the ARPANET approach remains used in many networks today: tuning the link weights in order to avoid congestion and balance the load. It remains popular in large parts because it is easy to deploy in existing routing protocols (all of which employ a version of Dijkstra's algorithm). However, link-weight tuning is a coarse mechanism and it is hard to make fine-grained decisions about how much and where the traffic should be shifted.

Network operators looked for a more powerful mechanism, which they found in Multiprotocol Label Switching (MPLS). In MPLS, each edge-router creates multiple tunnels to each other edge router, called Label-switched paths (LSPs). These LSPs

are assigned a fixed path and a bandwidth reservation, for example, 300 Mbit/s (see Figure 2.4). New classes of network packets (called *flows*) are assigned to one specific LSP. This gives operators fine-grained control over where to send the traffic in the network, even based on the type of traffic. We will revisit MPLS traffic engineering in the comparison against our traffic engineering design in Section 5.2.2.

2.2 Named Data Networking

2.2.1 The Evolution of the Internet

Since its beginning in the 1960s, the Internet has evolved in incremental steps. As every node supports the IP protocol, new transport and application layer protocols can be easily deployed on top of it. Similarly, the Internet readily supports new link layer technologies, since those only require local, bilateral agreement. For example, if a user wants to migrate his home network from one WiFi standard to another (say from 802.11ac to 802.11ax) he only needs replace his home router and the connecting devices; he does not have to contact his Internet Service Provider (ISP) or update his applications.

The IP protocol itself, however, has to be supported by every network element. Hence, it is called the *narrow waist* of the network stack (see Figure 2.6 on page 27). Changes to this narrow waist are slow since they require global agreement, that is, changing every involved end-device, server, access router, and core router.

One example of the network layer's reluctance to change is the roll-out of IPv6. After being proposed several decades ago, IPv6 is now integrated in most mobile operating systems, desktop operating systems, and home gateways, but still not supported by many ISPs and Internet servers [Leb13]. One reason for people's hesitation is network administration and security. Server administrators tend to learn and adopt new technologies only if their benefits outweigh the transition costs. During this transition, security problems are more likely. For example, IPv6's removal of Network Address Translation (NAT) makes every device in a home network globally reachable and a possible target for hacker attacks. To close this security hole, one needs to deploy firewall policies that drop unwanted incoming connections and hide the internal network from the outside world. However, early IPv6 home gateways lacked this protection and left the user open to outside attacks, privacy breaches, and even legal problems [CJT09].

Thus, all new network layer designs need to justify their deployment overhead through some major benefit. IPv6 has been driven by a strong market pressure, because IPv4 addresses are running out and solutions like NAT create problems for mobility, server applications, and software that uses IP addresses as device identifier.

Moreover, every new network layer design needs to be deployable in *incremental* steps. Since one cannot expect all globally distributed ISPs to switch to a new protocol at one coordinated time point, a change must be possible in smaller steps. This incremental roll-out can be done in one of three ways: 1) By using an IP overlay, 2) by using a dual-stack approach, and 3) by deploying interconnected islands of the new technology. IPv6 routers, for example, work in dual-stack mode that still supports IPv4. Moreover, IPv6 uses tunnels to connect islands of IPv6 networks through IPv4 nodes.

Similar to IPv6, *Named Data Networking* (NDN) proposes to change the network layer, in order to overcome the shortcomings of today's Internet. Anja Feldmann, researcher at Deutsche Telekom in Berlin, argues that today's challenges differ severely from those in the past [Fel07]. The same design principles that ensure the scalability, reliability, and cost-effectiveness of today's Internet also contribute to the current problems; the Internet is not poorly designed, but the problem has changed [Jac06b].

One example is the *end-to-end principle*, the idea to keep the core network simple and to put the intelligence into the end systems. Keeping the network dumb allows for highly efficient stateless routing and best-effort packet delivery, but leads to an inefficient use of the core network which is totally agnostic to the data it transfers. Content-aware routers, on the other hand, could cut down redundant data transfer by pulling the data from in-network caches instead. Hence, NDN could be the next step in the evolution of the Internet.

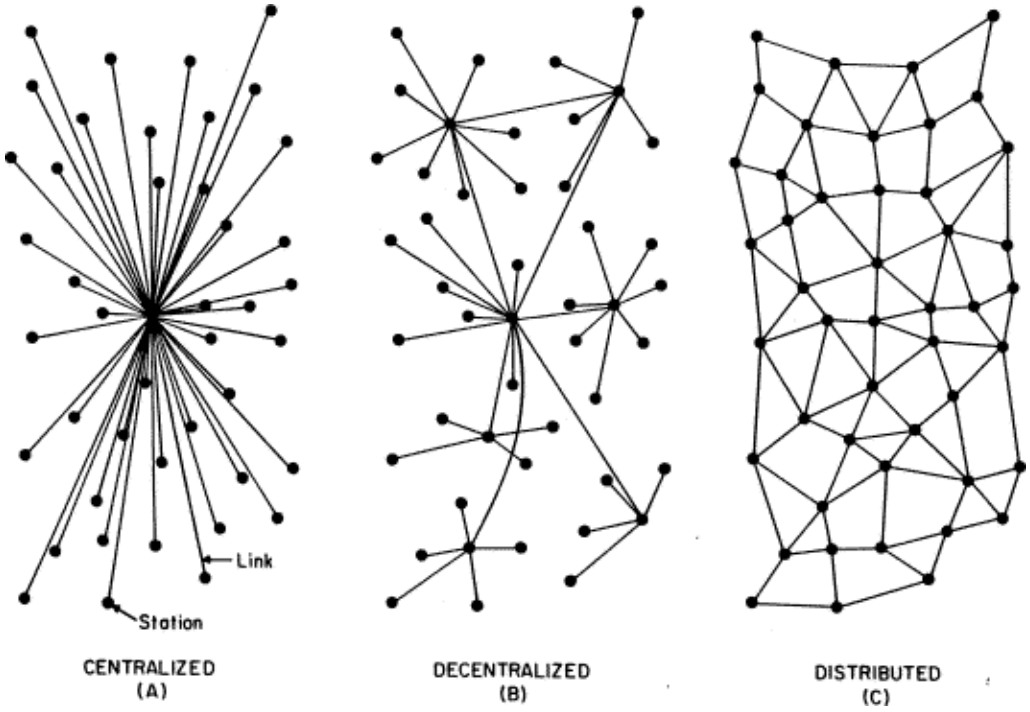


Figure 2.5: Centralized, Decentralized and Distributed Networks [Bar64]

2.2.2 A Brief History of NDN

Back in 2006, in a talk called "A New Way to Look at Networking", Van Jacobson compared the newly proposed NDN architecture with the paradigm shift from circuit switching to packet switching in the 1960s [Jac06a, Jac06b]. A phone user in the 1950s was concerned about end-to-end communication, namely phone calls, but the phone system focused on connecting pairs of wires. To reduce the number of wires (deployment was expensive at the time) the Bell phone system used a centralized or star topology (see Figure 2.5). In a centralized network, however, reliability decreases exponentially with scale: if one component fails the whole path breaks down. The only way to make this system reliable is to use highly reliable *components*, which are expensive.

Moreover, the phone system is based on *circuit switching*, which means that it

has to establish end-to-end connections before communication is possible. The set-up time of a few 100 ms is unproblematic for phone calls, which typically last much longer, but unacceptable for data traffic, which may only consist of a few packets; many protocols today, like DNS, use the connection-less UDP instead of TCP to avoid the overhead of connection set-up.

Paul Baran [Bar64] was the first to solve the problem of connection set-ups for data communication by suggesting to use a distributed topology (Figure 2.5). His idea of *packet switching* was to equip every data packet with a source and destination address, to send it into the direction of the destination, and to let the next hop decide where to send it on. This model avoids a heavy weight connection set-up and performs reliably even when working with unreliable components: if one element on the path breaks down the other routers can choose a different path. When Baran's architecture was first implemented in the ARPANET, it performed so well that it eventually evolved into the Internet that we know today.

However, when Paul Baran published his work earlier in 1964, he was unable to convince the telecommunication engineers at AT&T of its potential. People at the time, Jacobson argued, were stuck in the mindset of circuit switching that solved the past problems of telephony, but was unfit for the new problem of data communication [Jac06b]. He argues that the same is true for today's Internet. The user only cares about content retrieval (like video streaming), but the network still works in terms of host-to-host communication.

In 2009, Jacobson published the seminal paper "Networking Named Content" [JST+09] and the Named Data Networking Project (NDN) [ZEB+10] was founded as part of the NSF Future Internet Architecture (FIA) program [PPJ11]. In the following years, two big prototype implementations were established: CCNx [ccn] from the Palo Alto Research Center (PARC) and the NDN Forwarding Daemon (NFD) [ASZ+14] from the NDN Project community. More recently emerged the NDN-DPDK forwarder [NIS], aimed at improving NFD performance, and the HICN

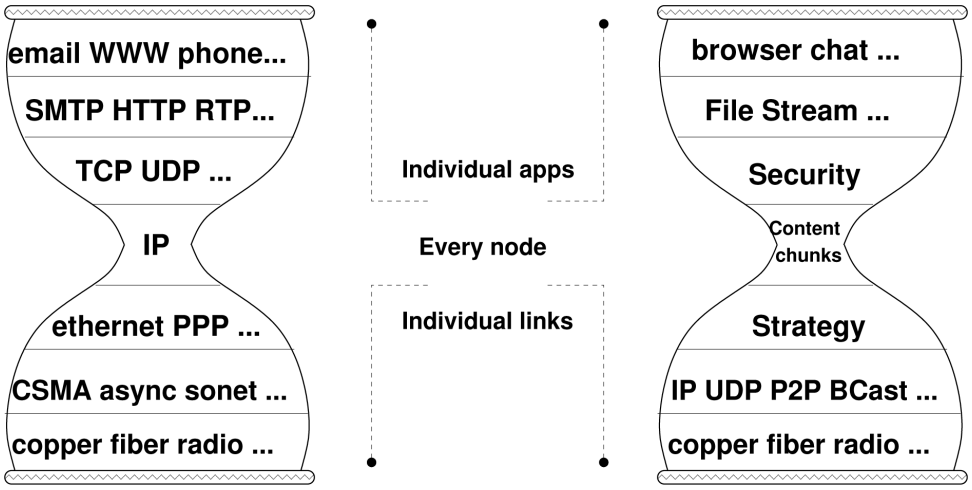


Figure 2.6: IP and NDN Hourglass Model [JST+09]

project [MCAP18], which goal is to build NDN on top of IP(v6).

NDN is now a regular topic on computer networking conferences. After a number of workshops on top-tier conferences, ACM, in 2014, launched the first annual conference of Information centric networking (ACM ICN), which just concluded in its 6th iteration this year (2019). Moreover, the NDN project has received increased industry support from companies like Cisco, Huawei, Intel Labs, and Verisign. In summary, the research field of ICN, and NDN in specific, is growing and advancing to real-world deployments.

2.2.3 The Architecture of NDN

NDN's architecture differs from that of IP in a number of ways. First, it moves the thin waist of the network stack from IP packets to named content chunks. Second, NDN routers use name-based routing and adaptive, stateful forwarding instead of IP-based routing and stateless forwarding. Third, NDN secures content rather than end-to-end connections.

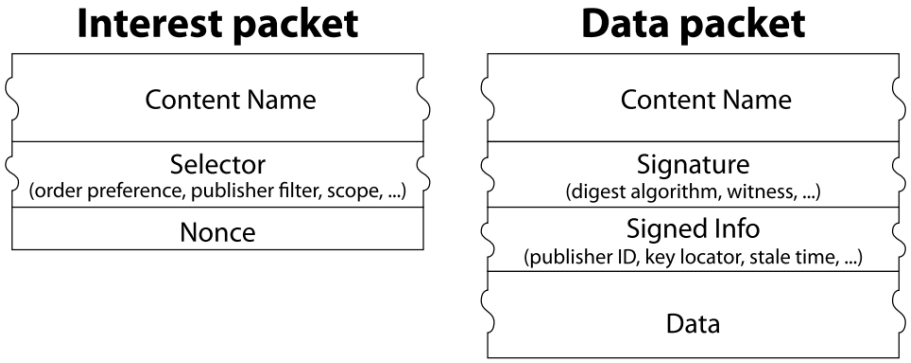


Figure 2.7: NDN Packet Types [JST+09]

NDN Protocol Stack NDN changes the narrow waist of the network stack from IP packets to content chunks (see Figure 2.6). It can be overlayed on almost any link, network, or transport layer protocol to allow for incremental deployment. The main prototype implementations, NFD, runs on top of TCP or UDP for compatibility. But once routers natively support NDN, they can run it directly on the link layer to eliminate the redundant TCP/IP packet headers. In addition to changing the narrow waist, the NDN protocol stack (right side of Figure 2.6) contains two new layers: *Strategy* and *Security*. We discuss NDN's security model in the paragraphs below and take a deeper look at forwarding strategies in Section 2.2.3.

NDN Naming and Packet Layout NDN uses two packet types: *Interest packets* and *Data packets* (Figure 2.7). Contrary to IP packets, these do not contain explicit location identifiers like source and destination addresses. Instead, NDN uses a *pull-based* protocol to retrieve content: A client sends an interest packet which contains the name of the requested content to its nexthop router. The router then forwards the packet towards a known content source or answers it directly from its cache. Interest packets also contain a selector for additional content requirements, such as requesting content from a specific publisher, and a nonce to prevent interest loops.

When the content provider or cache receives an interest packet, it returns a data

packet with the requested information. Data packets contain a publisher signature for data *integrity* check and a signed info field for retrieving publisher information and verifying data *authenticity*.

Data packets follow the path of their corresponding interests like Hansel and Gretel [Gri15] (intended to) follow their *breadcrumbs*. This effectively prevents loops of data packets. Since nonces prevent loops of interest packets, NDN does not need a time-to-live (TTL) field like IP packets for this task. Looping packets are detected and discarded at the first router that receives a duplicate packet or an interest with a duplicate nonce. This sort of loop detection is potentially much faster than relying on hop counts like IP networks.

However, the use of nonces for loop detection has recently been questioned. To detect loops, a router has to store the previously used nonces of a packet in its Pending Interest Table (PIT, see Section 2.2.3). Therefore, the lifetime of a PIT entry must be longer than the potential duration of any loop in the network. The total loop duration is unknown and, on an Internet scale, can be in the order of minutes; storing every PIT entry this long can easily exceed a router's memory capacity. To reduce this memory load, the NFD prototype does not keep the whole PIT entry for this time, but only a list of previously used nonces (the *Dead Nonce List*) [ASZ+14].

Moreover, Garcia-Luna-Aceves and Mirzazad-Barijough [GLA15, GLAMB15] found that interests can loop even when the PIT lifetime is indefinite. To solve the problem, they introduce the *Strategy for Interest Forwarding and Aggregation with Hop-Counts* (SIFAF) that 1) stores the number of hops to named content in the FIB and 2) includes the hop count from the sending router in each interest packet.

Similar to HTTP URL's, NDN content names are composed of hierarchical elements (see Figure 2.8). NDN routers use these elements to aggregate their routing table in the same way that IP routers aggregate IP prefixes; they also use a longest prefix match to look up outgoing interfaces for a content name. In NDN, routing is

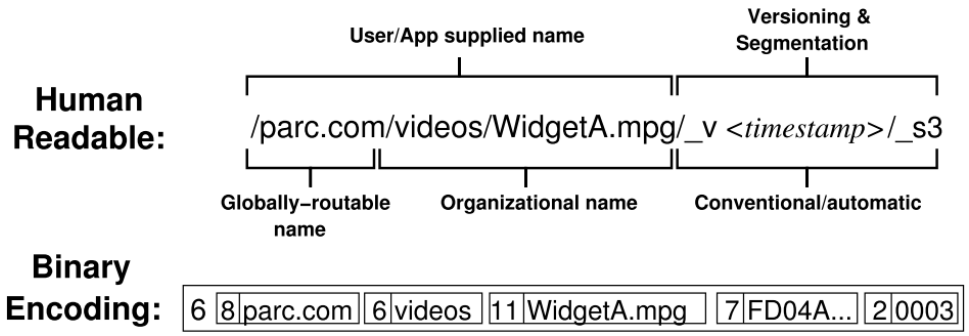


Figure 2.8: NDN Data Names [JST+09]

based on names, but the specific names are meaningless to the routers, just as IP octets are meaningless to IP routers.

NDN names can, however, convey meaning to applications. By convention, names contain a globally routable organization identifier, a version number, and segmenting information. Knowing these conventions, a consumer can traverse the NDN tree with relative expressions (see Figure 2.9). For example, the expression *RightmostChild* of a file name returns its most recent version; the expression *RightSibling* of one specific segment returns its successor, such as the next chunk of a file.

NDN Routing and Forwarding NDN routers are more complex than IP routers (see Figure 2.12). They route on hierarchical names instead of IP addresses and contain three components: The Content Store (CS), the Pending Interest Table (PIT), and the Forwarding Information Base (FIB).

The *Content Store* is a temporary cache. It contains data chunks that the router delivered in the past which may or may not also be permanently stored by that router. Since caching has to be performed at line speed, caches have to use fast and small storage devices like RAM or SSDs [VPL13]. Even when using large hard drives, caches are many orders of magnitude smaller than the total content catalog of the network. When a cache reaches its limit, a *cache replacement policy* decides which packets are replaced by the incoming data. Examples of replacement policies are

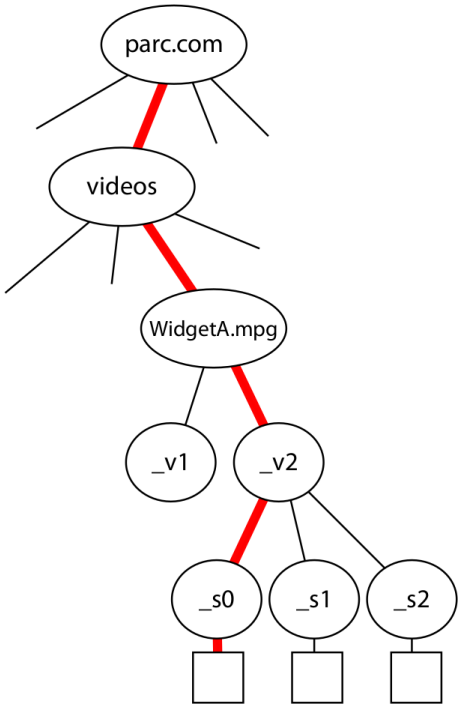


Figure 2.9: NDN Name Tree [JST+09]

Least Recently Used (LRU), Least Frequently Used (LFU), and random replacement. Moreover, a *cache decision policy* decides if a new data packet gets cached on arrival. Typically, the decision policy is Leave a Copy Everywhere (LCE), that is, to cache all incoming data packets; another option is random caching with a fixed probability. Due to the line speed requirement, most existing cache decision and replacement policies may be too complex to use in practice [ANO10]. However, Rossi and Rossini [RR11] have evaluated different caching policies and found that simple randomized policies perform almost as well as more complex ones.

The *Forwarding Information Base* (FIB) contains a set of outgoing interfaces for each known content prefix. IP routing tables, on the contrary, contain exactly one outgoing interface per IP prefix. This difference allows the NDN *forwarding strategy* to decide on which interface of the set to forward an incoming interest packet (see Section 2.2.3 for further details on forwarding strategies). In addition to a set of

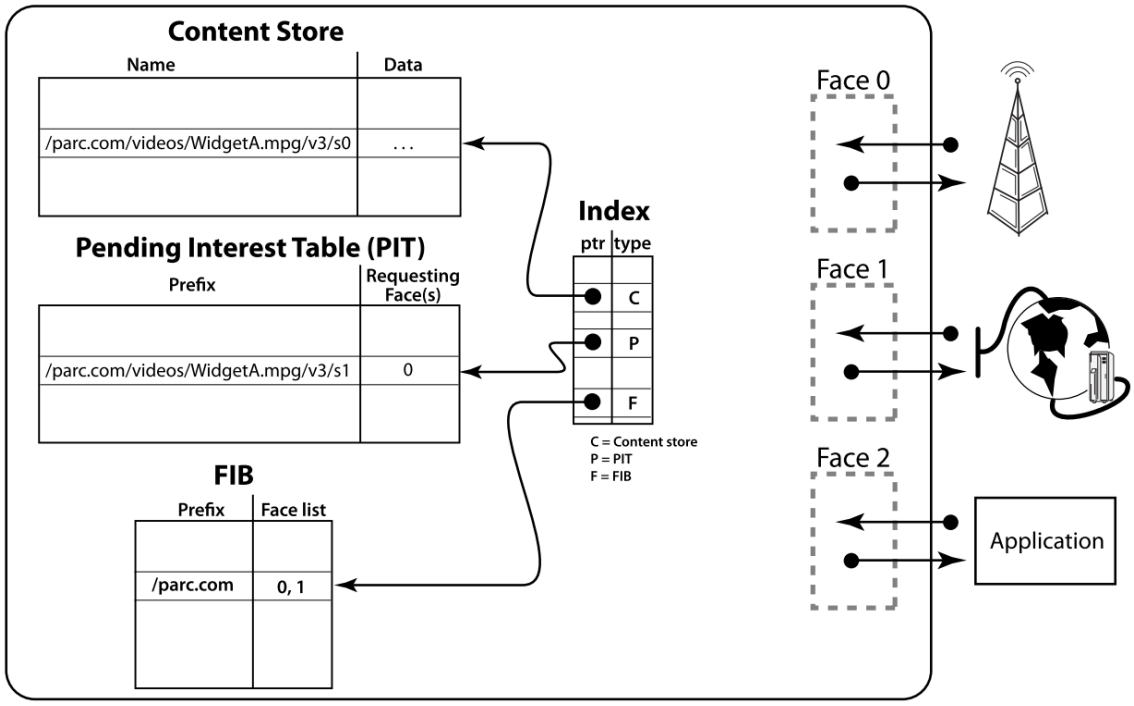


Figure 2.10: NDN Forwarding Engine [JST+09]

outgoing interfaces, FIB entries can contain routing preferences and a stale time, which states how long the entry should be kept after it disappears from routing (see Figure 2.11). The stale time improves content reachability on routing changes: interests of the old routing prefix are still satisfied while a new prefix is established.

The *Pending Interest Table* (PIT) maintains a mapping from content names to requesting interfaces. Each new incoming interest, together with its nonce, is saved in the PIT (see Figure 2.11). When an interest arrives that has the same content name but a different nonce, its incoming interface is added to the PIT entry. After the corresponding data packet arrives, the router forwards it to all interfaces in the PIT record. This interest aggregation allows the PIT to minimize redundant traffic even if the data is not cached in the local content store.

Making NDN's name-based routing efficient requires new algorithms and data structures. Yuan et al. [YSC12] have elicited a number of design principles to

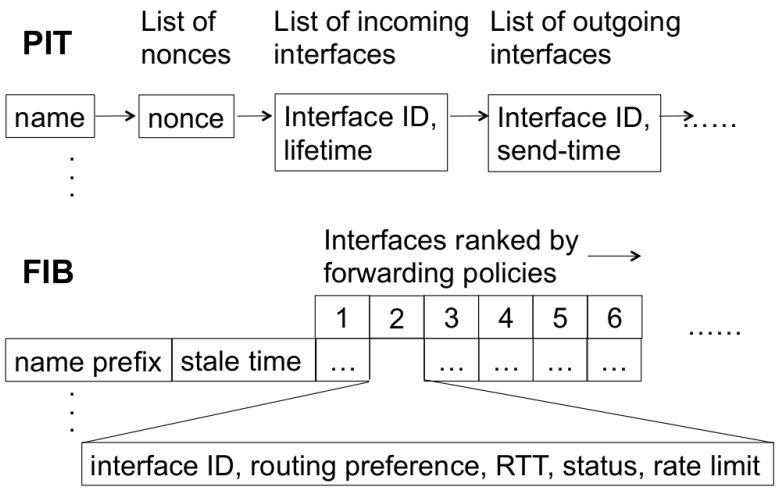


Figure 2.11: Forwarding State in PIT and FIB [YAM+13]

achieve high performance with Internet-scale NDN forwarding, namely 1) fast name lookup, 2) fast longest prefix matches on variable-length names, and 3) large-scale flow maintenance.

Data can be hosted at multiple permanent storage locations called *content repositories*. Interest packets are always routed towards a content repository. Figure 2.12 shows a small topology to exemplify NDN routing. Single circles represent regular IP routers and double circles represent NDN+IP (dual stack) routers. The client, which is connected to E, can retrieve content of the prefix */parc.com/media/art* from repository A, repository B, or any intermediate router. The figure also shows that NDN can be deployed incrementally on top of the IP network: NDN routers can connect to each other by tunneling the traffic through legacy IP routers. However, the more routers support NDN, the higher the performance boost of in-network caching.

NDN's Security Model IP uses a connection-based security model, that is, IP networks secure end-to-end paths and trust content that is received over a trusted path. This host-based security model is appropriate for communication sessions like

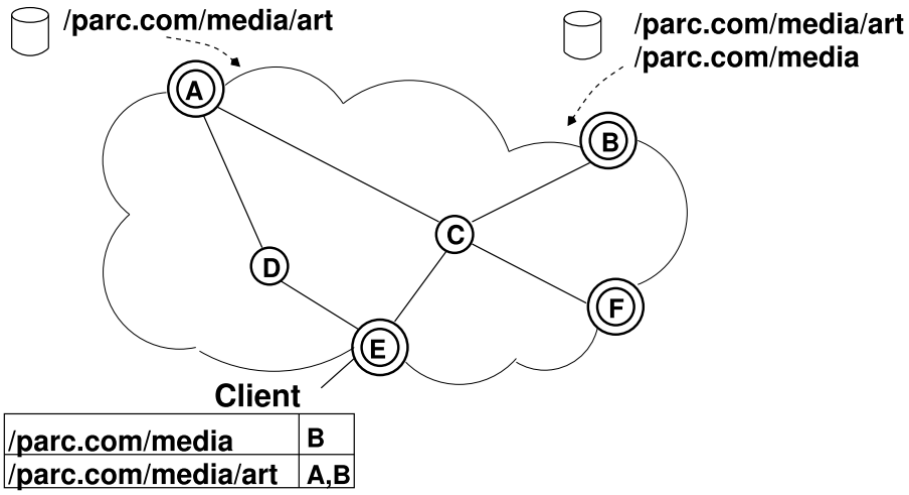


Figure 2.12: NDN Routing Example [JST+09]

bank transfers, but less so for content dissemination. For example, an attacker can alter the data on the server before transmission or on the client after transmission. Although the client receives the content through a secured channel, it has no way to verify the content's validity.

NDN shifts host-based security to content-based security. It is connectionless and works with point-to-point content transfer; thus, it cannot secure end-to-end connections. Since consumers can receive content from any untrusted cache, NDN puts trust information directly into the content: all NDN data packets are authenticated with the publisher's digital signatures and can optionally be encrypted. Every node in the transport chain can check the authenticity and validity of each content chunk. These checks are, however, only mandatory for the consumer; core routers can skip them to improve performance.

In contrast to other ICN architectures, NDN does not use self-certifying names, because they require an indirection infrastructure which itself could become the weakest link. Instead, it authenticates the binding between content and content names and provides all information needed for verification in the data packets [SJ09].

NDN Forwarding Strategies IP routers use *smart routing* and *dumb forwarding*: the routing plane determines a shortest path to the target and the forwarding plane strictly follows the routing decision. This stateless forwarding is efficient, but lacks functionality to explore multiple paths. IP routing is stateful but slow. After changes in the topology, like during link failure, it may take several minutes for all Internet routes to become coherent again. During this phase, IP packets often fail to reach their target, because they end up in loops or black holes. IP's slow routing convergence also prevent IP routing from handling short term network problems like link congestion.

In NDN, on the other hand, both routing and forwarding is stateful and adaptive. NDN forwarding works much faster than routing, in the order of milliseconds instead of minutes, and, thus, can handle problems like link failure, congestion, and prefix hijacking [YAM+13]. However, NDN's datagram state reduces forwarding performance. For each incoming interest packet, a NDN router checks for duplicate nonces, performs a content store lookup, inserts a FIB entry, and runs the forwarding algorithm. In contrast, IP routers only need to look up the outgoing interface in the routing table, a much simpler procedure that can be performed at hundreds of Gbit/s by specialized hardware. This additional complexity of NDN makes the design of both efficient hardware and efficient forwarding algorithms a urgent research topic.

A NDN router knows of multiple paths to satisfy an incoming interest packet and uses a *forwarding strategy* to decide where to send the interest on. In the rest of this section, we review current state-of-the-art strategy designs, both implemented in the two prototype implementations and proposed in the scientific literature.

NDN Forwarding Strategies differ in how strictly they follow routing information. Most of the following strategies only consider the path information in the FIB, that is, information provided by the routing protocol. They send out interest packets towards known content sources, which are likely to return data. The interests may be satisfied by *on-path* cache copies on any of the intermediate routers.

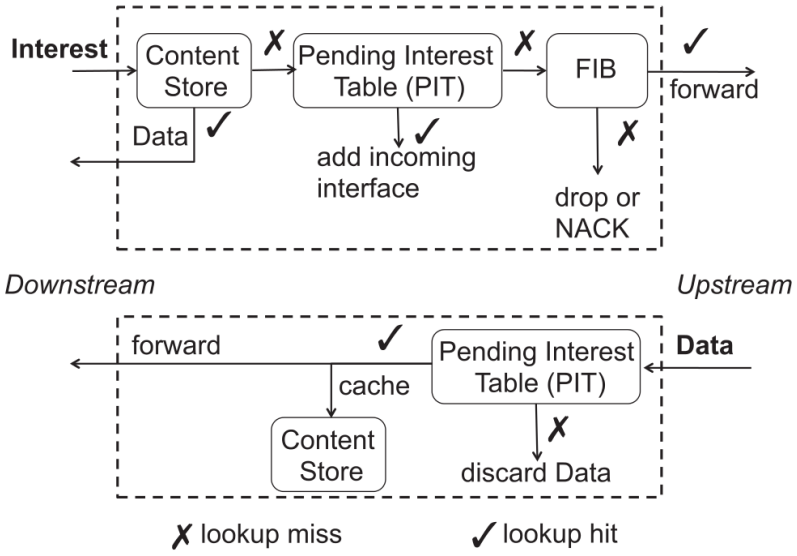


Figure 2.13: Interest and Data Processing in NDN [YAM+13]

Some strategies also use *exploration* [CRR+12], that is, they send out interest packets on paths that are not listed in the FIB. These packets may find *off-path* cache copies that are closer than any on-path copy, but often they do not return any data at all.

A large part of this dissertation is implemented in the NDN forwarding strategy layer. This is especially true for the congestion detection and signaling part of Chapter 3 and the whole traffic engineering design in Chapter 5.

CHAPTER 3

Practical Congestion Control

3.1 Introduction

As discussed earlier, Named Data Networking (NDN) changes the network communication model from IP's host-to-host packet delivery to fetching data by names: consumers pull *Data packets* by sending out *Interest packets* to the network. This new model enables multicast data delivery, opportunistic in-network caching, and multipath forwarding [JST+09, YAM+13]. However, it also complicates network congestion control, as existing solutions cannot be directly applied.

Traditional TCP-like congestion control is based on established connections between two endpoints. The sender detects congestion by measuring round-trip time (RTT) or packet losses, then adjusts its sending rate accordingly. In NDN, however, the concept of end-to-end connections does not apply. Data chunks of the same content may be retrieved from different repositories or different caches along the paths towards these repositories. Since these different content sources result in varying retrieval delay, and the consumer cannot distinguish between them, traditional RTT-based timeouts become unreliable indicators of congestion.

NDN's stateful forwarding plane enables routers to control congestion at each hop, by either dropping Interest packets or diverting them to alternative paths. However, most existing solutions in this direction (HBH Interest Shaping) [CGM12, WRN+13, RF14, LHLX15, ZWL+15, ZZR+14, PJK14, LHX+15] assume known or predictable link bandwidths and Data chunk sizes, which limits their effectiveness in scenarios where these assumptions do not hold true. For example, the largest experimental NDN deployment, the NDN Testbed [ndnb], runs over UDP tunnels

where the underlying bandwidth is unknown and keeps changing; a video-on-demand provider may respond to congestion by adjusting the video quality (hence Data chunk size) dynamically.

This work proposes PCON: a practical NDN congestion control scheme that does not assume known link bandwidth or Data chunk sizes. PCON routers detect congestion on their local links by using an active queue management (AQM) scheme extended from CoDel [NJ12b, NJMI16]. When congestion is detected, a router signals this state to consumers and downstream routers by explicitly marking Data packets. The downstream routers then react by partially diverting subsequent Interests to alternative paths and/or passing the signal further downstream; consumers react by reducing their sending rate. Since in-network routers participate in avoiding congestion, we consider PCON a "hop-by-hop" congestion control scheme. However, it is fundamentally different from "Hop-by-hop Interest Shaping", as PCON routers *do not* reject Interests based on a prediction of how much link capacity the corresponding Data chunks will take up. Instead, they monitor the queue at their outgoing links, which allows to signal congestion early and to consider the available bandwidth implicitly.

We implement PCON in ndnSIM [MAMZ15] and evaluate its performance against a delay-based [CGM+13] and a HBH Interest Shaping scheme [WRN+13] from the literature. We show that PCON prevents congestion in the case of varying RTTs, dynamic Data chunk sizes, IP tunnels, and wireless links. Moreover, PCON's forwarding adaptation (based on explicit congestion marks) achieves a higher throughput than the existing solutions that are based on equalizing pending Interests. PCON is able to adapt to the changing available bandwidth of IP tunnels and wireless links, a case that is not considered by current HBH Interest Shaping proposals. We identify a few remaining issues as our future work, including the definition of "fairness" in data-centric networks and effective handling of unresponsive consumers.

The rest of this chapter is organized as follows. Section 3.2 explains the design

rationale of PCON. Section 3.3 describes PCON's design specifics and Section 3.4 evaluates its performance through simulations. Section 3.5 compares PCON with related work.

3.2 Design Rationale

NDN's new communication model requires us to rethink the traditional congestion control mechanisms which have been shown to work well in TCP/IP. Here we elicit four groups of unique NDN characteristics and show how they lead to our design approach.

1) Multiple Paths and Endpoints As a data-centric architecture, NDN removes the concept of connections between two endpoints and enables *ubiquitous caching*, *native multicast*, and *multipath forwarding.* Since an NDN Interest may be answered by any cache along the way to the content repository, the consumer application cannot tell where the corresponding Data packet has originated from. Even if the knowledge about the origin was carried in the Data packet, the consumer would still not know where the next Data packet in the stream would come from (except using the solutions from [SCP13, BMST13] – see a discussion of them in Section 3.5) This limitation leads to the following three problems:

First, traditional TCP retransmission timers [Jac88, PACS00] perform poorly, as they assume a normal RTT distribution *between two end nodes*. In NDN, however, data may be retrieved from multiple nodes or through multiple paths. Moreover, NDN routers aggregate Interests of the same name into one Pending Interest Table (PIT) entry and deliver the returning Data packet to all aggregated faces. Interests that arrive after the first will see a shorter retrieval delay (since the Data is already on its way). These three cases (multi-source, multipath, and PIT aggregation) can result in periods of short RTT measurements where the RTO setting (meanRTT + 4 * varRTT) falls too short, leading to a decreased sending rate and unnecessary

retransmissions. One can avoid these false positive timeouts by setting a higher minimum RTO, but that also increases the time to detect packet losses, thus the reaction time to network congestion. PCON avoids this dilemma by combining a high minimum RTO with *explicit congestion signaling* for fast reaction.

Second, the optimal congestion window for the consumer (defined as the bandwidth-delay product – BDP) can change due to cache diversity: If one cache runs out of data, the next Interests will be answered by another cache (or the producer) further away. If the path to the new cache has a lower BDP, the high number of in-flight Interests may overflow the queue at the new bottleneck, before the consumer is able to adapt its sending rate. One may try to avoid such *cache switchovers* by using a caching policy that always retains complete content objects. However, this is not feasible when caches never see the full object in the first place (e.g., due to multipath splitting of traffic or consumers requesting only parts of a content object). We mitigate this problem by making buffers large enough to handle temporary traffic bursts, and by detecting and signaling congestion with the CoDel AQM scheme *before* the buffers reach their limit (see Section 3.4.2).

Third, a consumer can no longer use gaps in sequence numbers from received data packets (out-of-order delivery) as a sign of packet loss, hence congestion, as out-of-order arrivals could be caused by receiving data from different caches, or along multiple paths.

2) Pull-based Data Retrieval & HBH Flow Balance NDN consumers send out Interests to retrieve Data packets. Since Data packets always follow the "bread-crumbs" of their corresponding Interests, to control congestion over a link from nodes B to A, A must control the rate of Interests it sends to B. In theory, NDN's *Hop-by-Hop Flow Balance* (over each link, one Interest packet brings back exactly one Data packet [ndna, JST+09]) can be used to control the link load from B to A. In practice, however, that would require a-priori knowledge of the link capacity,

which may not be available for IP tunnels or wireless links. Instead of trying to estimate the link bandwidth or expected load, PCON uses the packet queuing time [NJ12b] and local link loss detection (Section 3.3.5) as more reliable indicators of congestion.

3) Diverse Deployment Scenarios NDN must work in a diverse range of deployment scenarios, including *IP-Overlay* links and *wireless* links. One example of IP-Overlay deployment is the NDN Testbed [ndnb], where NDN can run over either TCP or UDP tunnels. TCP tunnels provide reliable delivery (similar to a wired link) with the trade-off of increasing the total data retrieval delay, which can be unnecessary and even undesirable for loss-tolerant and delay-sensitive applications (like audio/video conferencing). Since NDN needs to work for all applications, the Testbed currently runs UDP tunnels. However, UDP tunnels may drop packets without notification and their capacity is quite unpredictable; it may vary continuously depending on the competing cross-traffic in the IP underlay network.

Another common NDN deployment scenario is wireless links. As with IP overlay links, the achievable bandwidth of a wireless link is unknown and can be highly variable; it depends, among other factors, on wireless interferences and wireless traffic load levels which result in packet collisions.

PCON considers both wireless and IP-Overlay networks by detecting silent drops on these networks and signaling them back with explicit NACKs (see Section 3.3.5).

4) Flow Semantics and Fairness Fairness is an important topic for NDN transport control. However, the existing concept of "per-flow fairness" does not directly apply to NDN. In IP networks, a flow refers to traffic between *two endpoints*, often identified by the 5-tuple of src/dst IP address, src/dst port, and the transport protocol. In NDN, packets may not flow between two fixed endpoints. One can define an NDN *content flow* by requests for a specific *content object* from one or

multiple consumers [ORS12]. To identify such a flow, one needs a global agreement on the definition of a content object. For example, for a layer-encoded video, would the whole video be considered a single content object or, as some consumers may fetch a sub-layer only, each sub-layer a separate one? Even assuming we can agree on the definition of contents, "per-content fairness" remains difficult, as some NDN routers may "see" only a subset of a content flow when other chunks are answered by prior cache hits or sent elsewhere by multipath forwarding.

Furthermore, NDN's multicast data delivery creates a new trade-off between efficiency and fairness. A router does not know whether an Interest is generated by a single consumer, or represents an aggregation of Interests from multiple consumers further downstream. The latter case (retrieving data for multiple consumers) is arguably more important, but a simple "per-content flow fairness" cannot give it a higher preference. One could account for the higher demand in a virtual control plane [YHC+14], which however incurs more complexity and state overhead.

We evaluate the per-consumer fairness of PCON in Section 3.4.4. However, the above consideration suggests that more work is required to define the fairness metric in an NDN network (e.g. whether fairness should be based on contents or on consumers). Thus, we leave the topic of fairness to future work.

3.3 System Design

Following these considerations, we design PCON with the principles of *1) early congestion detection by adopting an AQM*, *2) explicit congestion signaling*, *3) exploiting multipath forwarding*, and *4) special consideration of IP-overlay and wireless links.* As shown in Figure 3.1, PCON consists of five components:

- *Congestion Detection* (3.1): Each node detects congestion locally by monitoring its outgoing queues.
- *Congestion Signaling* (3.2): After detecting congestion, nodes mark Data

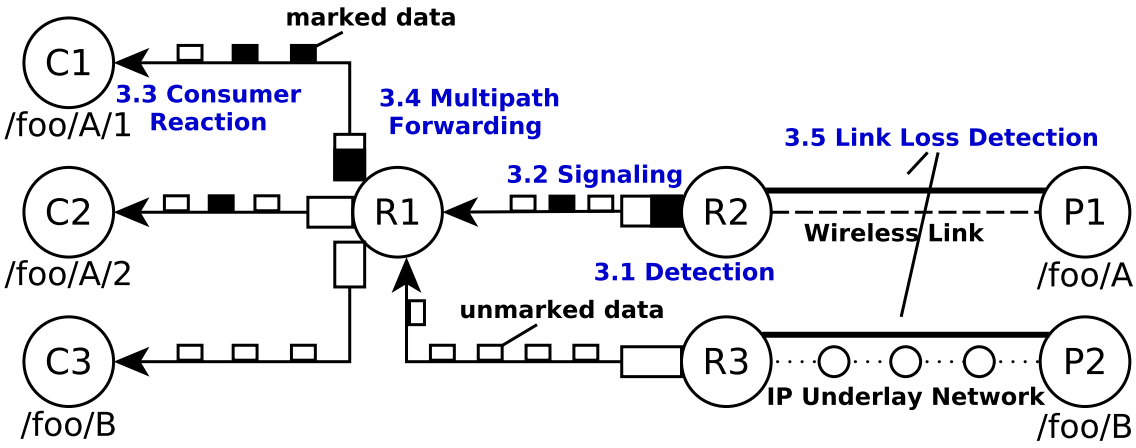


Figure 3.1: System Architecture

packets to inform downstream routers and consumers.

- *Consumer Rate Adjustment* (3.3): End consumers react to congestion signals by adjusting their Interest sending rate.
- *Multipath Forwarding* (3.4): Downstream routers react to congestion signals by adjusting their traffic split ratio.
- *Local Link Loss Detection* (3.5): On wireless and IP overlay links we locally detect packet loss and signal it back using NACKs.

3.3.1 Congestion Detection

As Nichols and Jacobson have pointed out [NJ12b, NJMI16], the most reliable place to detect congestion is at the link where it occurs. Thus, we detect congestion by monitoring the outgoing queues of each router, instead of trying to infer congestion at the consumer by monitoring packet loss or round-trip time (as discussed earlier, these techniques are less reliable in NDN than in IP networks). To detect congestion on router queues, we use a well-studied AQM mechanism, CoDel [NJ12b, NJMI16], as it allows routers with large buffers to both absorb traffic bursts and to keep their

queues small, by detecting congestion *before* the buffer overflows. CoDel measures the queuing delay ("sojourn time") of each packet on its outgoing links. If the minimum sojourn time over a time period (default: 100ms) exceeds a threshold (default: 5ms), it considers this link as congested. CoDel's use of queuing delay instead of queue size fits our design assumption of unknown link capacity: queuing delay implicitly considers the link bandwidth (for a fixed queue size, higher bandwidth leads to lower queuing delay).

PCON monitors congestion in both the downstream (Data) and the upstream (Interest) direction. Interests can cause congestion when they are larger than Data packets, on asymmetric links, or when there is cross-traffic in the upstream direction. After detecting congestion in the Interest direction (like R1–R2 in Figure 3.1), R1 will *mark the PIT entry* of that Interest, and consider the corresponding Data for congestion signaling.

In addition to monitoring outgoing queues, we use two alternative ways of detecting congestion. The first is used in scenarios where the state of the router queues is inaccessible, as in UDP tunnels. Here we infer the congestion level by monitoring packet loss at the local link (see Section. 3.3.5). The second addresses another problem that can be considered "congestion": A PIT that is growing too large and threatens to overflow the router's memory (which is analogous to a TCP sender overflowing the receiver buffer). Just like congestion due to insufficient link bandwidth, the state of overgrowing PIT can be signaled downstream to reduce the number of incoming Interests. However, in this work, we focus on bandwidth-related congestion and leave the detection of overgrowing PIT for future work.

3.3.2 Congestion Signaling

After a router has detected that one of its outgoing links is congested, it signals this state to the consumers and to all routers along the path. Signaling is only necessary in the downstream direction, since only downstream routers can reduce the amount

of traffic that the signaling router will receive. Thus, we signal congestion by marking NDN *Data packets*, but not Interests.

We signal congestion similar to CoDel's use of ECN marking: The first packet is marked when entering the *congested* state and later packets are marked in an increasing "marking interval" (similar to CoDel's drop spacing); the marking interval starts at 1.1 * the CoDel interval (110 ms) [NJMI16] and is decreased in inverse proportion to the square root of number of marks, in order to achieve a linear decrease in the consumers sending rate. On congested upstream links, we use the same marking interval for setting the flag in the PIT entry, which then marks the associated Data packets.

Due to NDN's built-in multicast data delivery, one marked Data packet may branch out to reach multiple consumers, which will all slow down their rate. This is exactly the desired behavior, because if any of the consumers fails to slow down, the congested link will not see a reduction in rate at all (it would just see less PIT aggregation).

One example of the signaling mechanism is shown in Figure 3.1. Consumer C3 retrieves data from P2 (under the prefix /foo/B). Since there is no congestion on the path, C3 only receives unmarked Data packets. C2 retrieves data from P1 (/foo/A/2) and receives congestion marks from R2, since the link between R2 and R1 is congested. C1 also retrieves data from P1, but under a different prefix (/foo/A/1). In addition to receiving congestion marks from R2, C1 also receives them from R1, since the link R1–C1 is congested too. This allows C1 to adjust to the bottleneck at R1, while C2 adjusts to its bottleneck at R2.

3.3.3 Consumer Rate Adjustment

Once the congestion marks reach the consumer, the consumer has to decide on how to adapt its rate. We use a congestion window (specifying the maximum number of in-flight Interest packets) which is increased on unmarked Data packets and

decreased on marked ones, NACKs, and timeouts. Data packets act as *selective acknowledgements* of Interest packets, meaning that each Data packet acknowledges one corresponding Interest. Selective ACKs require some thought in our design, since NACKs/timeouts often occur in bursts (on buffer overflows). If we perform a multiplicative decrease on each packet loss in a burst, the congestion window is reduced too drastically (often back to 1) after a single congestion event. We prevent this over-adjustment by applying the "TCP SACK-based Conservative Loss Recovery Algorithm"[BFA03] (see Algorithm 1): The consumer performs at most one window decrease per RTT.

Algorithm 1: NDN Conservative Window Adaptation[1]

```
Function OnData (dataPkt, seq)
    if m_highData ≤ seq then
        m_highData ← seq;
    if dataPkt.isMarked() then
        windowDecrease();
    else
        windowIncrease();
Function OnTimeout (seq)
    // Same for OnNack()
    // Only one window decrease per RTT
    if m_highData > m_recPoint then
        m_recPoint ← m_highInterest;
        windowDecrease();
```

We don't (need to) apply the "Conservative Loss Adaptation Algorithm" to marked packets, since the CoDel marking already avoids bursts of marks. Moreover, CoDel marking adapts to the congestion level (marks become more frequent when congestion persists) while the loss adaptation algorithm does not (it always allows at most one window decrease per RTT); thus disabling it allows the consumer to react faster to highly congested links (see Section 3.4.3). We still use the conservative loss adaptation for timeouts (resulting from buffer overflows) and NACKs resulting from

buffer overflows in the IP underlay network (see Section 3.3.5).

With the conservative loss adaptation and CoDel marking in place, PCON can implement a number of classic loss-based TCP algorithms like Taheo, Reno, New Reno, HTCP, HSTCP, BIC, and CUBIC. The only difference to traditional TCP is that a window decrease is triggered not only by timeouts, but also by packet marks (similar to TCP ECN marks) and NACKs. After experimenting with TCP Reno, we switched to TCP BIC [XHR04], because (similar to TCP CUBIC) it performs better on high BDP networks.

When router buffers are large enough, PCON removes the traditional sources of packet drops: *1) Router buffer overflows* (it prevents them by explicitly signaling congestion), *2) Drops by the AQM scheme* (it marks packets instead), and *3) Drops by the "link"*, like wireless or UDP tunnel (it either recovers the loss by in-network retransmission or sends explicit NACKs – see Section 3.3.5). Therefore, the only sources of packet loss should be malfunctioning (host or link failure) or misconfiguration. This provides a solution for the high RTT variance caused by caching and multipath: We can use higher RTOs to avoid false positive timeouts and still achieve fast retransmissions, as packet loss is either avoided or explicitly signaled.

3.3.4 Multipath Forwarding

The scheme described so far will effectively prevent congestion when used with single-path routing, as it adapts the consumers' sending rate to the available network bandwidth. Below, we describe how PCON routers use the NDN forwarding layer to shift traffic away from congested paths.

[1] m_highInterest denotes the highest sequence number of an Interest that the consumer has sent so far; m_highData denotes the highest sequence number of any received Data. m_highInterest, m_highData, and m_recPoint are all initialized to 0. Note that these sequence numbers are only used at the consumer and are independent from the Interest name; an application may choose an arbitrary naming scheme as long as it locally keeps track of the sequence in which packets are sent and received.

We design our multipath forwarding strategy with the objective to *maximize end-user throughput* and, if there are multiple best options, to minimize the network cost (as measured by path length). We assume that the strategy has access to a ranked list of path lengths, set by the routing protocol. The optimal strategy decision at each hop depends on the demanded rate (determined by the capacity of the downstream links and the consumer) relative to the capacity of the outgoing links. If the demand can be satisfied by the shortest path, traffic should not be split on other paths. If the demand exceeds the capacity of the shortest path, it should be incrementally distributed on the next-shortest path(s). If the demand exceeds the sum of the capacity of all available links, it should be split in a way to fully take up the bandwidth, irrespective of path length.

Instead of adjusting the forwarding ratio to pending Interests or round-trip time like related work (see Section 3.4.4), we design our forwarding adaptation based on *received congestion marks*. For each FIB prefix, PCON maintains a forwarding percentage of each face, which is initialized to 100% for the shortest path (given by the routing protocol) and 0% for all other paths. Then for each marked Data packet, it reduces the forwarding percentage of the incoming face, while at the same time increasing the forwarding percentage of all other faces to an equal amount:

$reduction = fwPerc(F) * \frac{CHANGE\_PERC}{f(Distance)}$

$fwPerc(F) -= reduction$

$fwPerc(\bar{F}) += \frac{reduction}{NUM\_FACES-1}$

fwPerc(F) is the forwarding percentage of the current face and *CHANGE_PERC* is a fixed parameter, which makes a trade-off between how fast it adjusts the forwarding ratio to the optimal value (higher is better) and how much it oscillates around that value (lower is better). Through experiments, we found a value of 1%-3% to work well for a number of different bandwidths. *f(Distance)* is a factor to adjust the forwarding ratio more strongly the closer the adjusting router is to the congested link. We discuss it later in this section.

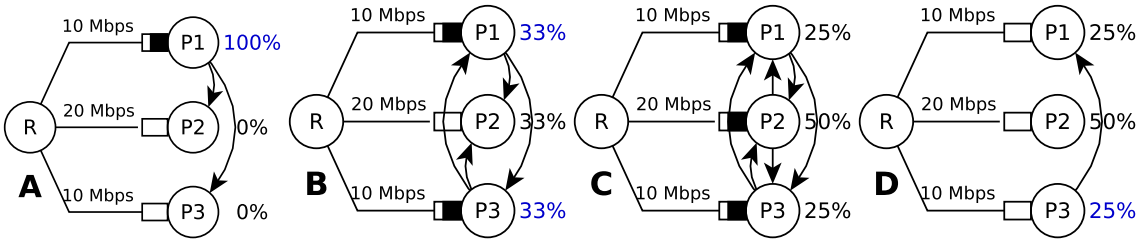


Figure 3.2: Multipath Forwarding Example

For a new FIB prefix, our forwarding scheme achieves the goal of staying on the shortest path until the demand exceeds the capacity of that path (only then will congestion marks arrive and change the forwarding ratio). It also achieves the goal of taking up all available network bandwidth (converging to the optimal split ratio) when the demand is high. However, when the demand falls again later in the lifetime of a FIB entry, we need a mechanism to shift traffic back to the shortest path (which is now underutilized). We detect free link capacity by not receiving any congestion signals for at least a "marking interval" (110 ms) and then shift a fixed percentage of traffic from the longest to the shortest path (e.g. 10% of CHANGE_PERC). After some time, this will move the full traffic back to the shortest path, if its capacity is large enough (if not, the arriving congestion marks will shift the traffic to other paths again).

We show an example of our forwarding adaptation in Figure 3.2. First, R sends traffic on the shortest path towards P1. If the rate stays below 10 Mbps, traffic will stay on that path. If not, R will see congestion marks from P1 and starts shifting traffic to other links (part A). Once the traffic is split equally on all links, R will see marks from both P1 and P3 (part B), and will reduce both their ratios, thus increasing the one of P2. In the final state (part C), R has reached the optimal split ratio and receives marks from all 3 routers. The split ratio will oscillate around this equilibrium and react to changes of the unbound capacity of the links, as seen by the received congestion marks. If the demanded rate falls below 40 Mbps (part D),

none of the links will see congestion marks, and R will shift traffic from the longest path (P3) back to the shortest path (P1).

One remaining question is the dynamics of multiple routers adjusting their forwarding ratio along the path of the marked Data packets: with multipath forwarding, routers that are close to the content repository see less congestion marks than routers that are close to the consumer, as the latter see marks from many more links. To avoid over-adjustment at these routers and to address congestion closer to where it occurs, we adjust the forwarding ratio less at routers far away from the congested link: $f(Distance) \geq 1$. The factor f(Distance) can be based on the number of hops between the router and the congestion point, on the number of routers that already have adjusted their ratio earlier (closer to the congested link), or on the number of alternative links on the path. For our evaluation, we use a simple linear reduction (f(Distance) = HopCount to congested link), and leave the optimal setting of f(Distance) for future work.

A related question is the dynamics between the router forwarding adaptation and the consumer rate adaptation. For PCON, the consumer reacts more strongly to each congestion mark (e.g., by halfing the cwnd) than routers, because the consumer can also react to Data packets as sign of available bandwidth, while routers only react to the much rarer congestion marks. However, this topic may need further study.

Another addition to the forwarding scheme is to consider the trade-offs in using paths with a very low forwarding ratio. There are costs associated with using multipath forwarding instead of single-path forwarding: Adding a new path can lead to higher RTTs, more out-of-order delivery, or even packet drops. These costs are offset by the additional bandwidth that the new path contributes. However, if the split ratio of the path is very low (e.g. less than 5%) the additional bandwidth might not be worth the cost. Thus, the strategy can set a *lower threshold* for each link, below which it will disable the link, but maintain its forwarding percentage and send

probing traffic to adapt the percentage to congestion marks. Should the percentage exceed the threshold, the strategy will enable the path again.

Algorithm 2: Router Signaling and Forwarding Adaptation

```
Function OnData (dataPkt, incomingFace, fibEntry, pitEntry)
    if dataPkt.isMarked() then
        reduceFwPerc(fibEntry, incomingFace, CHANGE_PERC);
    // For each DownstreamFace
    for dsFace IN pitEntry do
        if dataPkt.isMarked() OR dsFace.isCong() OR pitMarked then
            MarkOutgoingData();
Function OnNack (incomingFace, fibEntry)
    // Only one reduction per CoDel Interval
    if time::NOW ≥ lastUpdate + CODEL_INTERVAL then
        reduceFwPerc(fibEntry, incomingFace, CHANGE_PERC);
        lastUpdate = time::NOW;
```

3.3.5 Local Link Loss Detection

The design described so far assumes that the only cause for packet loss in the network is router buffer overflow, which is the case for wired links with low bit error rate, but not for IP tunnels or wireless links. NDN routers connected by UDP tunnels experience packet losses when the IP underlay network has congestion or routing problems. Since we can't access the router queues inside the UDP tunnel, to the NDN layer, a UDP tunnel looks like a link that may drop packets unpredictably, often in bursts. Wireless links behave in a similar way: interference, multipath propagation, or disconnections can all result in packet losses. The wireless link layer may use FEC codes and retransmissions to mitigate those losses, but does not guarantee reliable delivery; after exceeding a preset retransmission limit, it silently drops the packet.

Since silent packet drops are costly in NDN (see Section 3.2), we avoid them by implementing a shim layer (like [VMAZ16]) on top of loss-prone NDN link

abstractions (which should not be confused with the OSI data link layer, as NDN can run on top of different layers of the protocol stack). The only functionality this shim layer needs to implement is to reliably indicate whether a given packet was lost on a certain "link". It may report packets as lost which weren't lost (false positive), but it may never fail to report a packet that was actually lost (false negative), as consumer and PIT timeouts are more costly than unnecessary retransmissions. We call this notion *reliable loss detection*, which is different from reliable packet delivery. We achieve reliable loss detection by using *positive acknowledgments* for each frame (Interest or Data packet) and detecting losses by observing gaps in sequence numbers or timeouts of the ACKs. This gives the sender a definite report of packet loss in only a small multiple of the link propagation delay (when accepting the chance of false negatives). In contrast, with negative link ACKs the receiver is responsible for detecting the loss, and if the NACK is lost, the sender might never know about the original link loss.

The knowledge of packet loss is signaled back to the consumer to allow a faster and more reliable retransmission, compared to relying on timeouts. The signaling works slightly different depending on whether the loss happens on a wireless or IP overlay link: For IP overlays, the loss can be signaled back to the consumer with a *marked NACK*, where the reaction is the same as to a marked Data packet. However, since these NACKs might happen in bursts, the consumer needs to apply the conservative loss adaptation algorithm, that is, only reduce the cwnd once per RTT. Routers in the network reduce their forwarding ratio of these links and need to apply a similar mechanism: they only perform one reduction of the forwarding percentage per CoDel interval (100ms), which is an estimate of the upper threshold of consumer RTTs in the network (see Algorithm 2).

For wireless links, the mechanism is simpler: the loss is signaled with an *unmarked NACK* and only triggers a consumer retransmission, but no change in sending rate or forwarding ratio. The queues at the end of each wireless link are still able to

effectively signal congestion (by marking Data packets), should it occur.

Although this reliable link loss detection is more complex and less effective for controlling congestion levels than measuring queuing delay, we show in Section 3.4.6 that it still allows the consumers to adapt to congestion in the IP underlay network and to recover wireless packet losses.

An addition to the link adaptation layer is to locally retransmit a packet a certain number of times before reporting the loss to the network layer [VMAZ16], which can improve the performance by reducing the number of consumer retransmissions. However, the more important task of signaling the congestion level in the underlay network to both routers and consumers is achieved even without in-network retransmissions.

3.3.6 Highly Congested Links

In some cases, PCON cannot prevent congestion on a certain link: the congestion may be caused by unresponsive consumers, IP cross-traffic, software failure, overflowing PIT entries, or hardware limitations. Even though we cannot avoid congestion in these cases, we can still avoid using these "highly congested" links by disabling them in the forwarding strategy. There are multiple ways to detect the highly congested state; for example, when the queue keeps persisting after many intervals of congestion signaling, or when the router memory approaches its limits. Whenever a router enters the highly congested state, we signal it by marking *all* outgoing Data packets with two additional bits: one that indicates this state ("highCong"), and one that indicates whether a highly congested link is a direct neighbor or not ("highCongLocal"). Marking all packets allows us to know whether the state persists in the adjacent router, which isn't possible with the spread-out CoDel marks.

Whenever a router receives a packet with the high congested state or detects this state on its own upstream links, the router disables this path for Interest forwarding. After disabling the link, a router checks the FIB entry for alternative links, and if it

doesn't find one, it propagates the highCong state further downstream. It also resets the highCongLocal bit, so that the next further downstream router knows that this state only applies to given FIB prefix, but not to the whole link it is connected to. The second downstream router then disables the path only for this FIB entry. Note that, when no other option is available, a router still forwards Interests on disabled paths, and expects prior downstream routers to choose better alternatives.

One could use NACKs to signal the highCong state, but our marking scheme has the benefit that marked packets *still contain the data*, and thus reduce retransmissions. They can be used in the case where the upstream is still able to deliver data, but wants to strongly discourage its use. In Section 3.4.5, we show how the highly congested state improves performance in the case of non-reactive cross-traffic.

3.4 Evaluation

We evaluate PCON using ndnSIM 2.1 [MAMZ15] in various scenarios to demonstrate the effectiveness of its decisions. In two of those scenarios (3.4.4 and 3.4.6), we compare PCON against related work from the literature: one delay-based scheme and one based on HBH Interest shaping.

3.4.1 Consumer Rate Adjustment and Router Queue Management

We first evaluate the effects of using an AQM scheme (CoDel) and explicit congestion signals. We use a linear 3-node topology (Figure 3.4) with a single consumer and no caching. We compare three different cases:

- **FIFO:** A naive implementation of TCP BIC with FIFO queues and the consumer using the conservative loss adaptation.
- **CODEL_Drop:** Routers use CoDel dropping queues instead of drop-tail FIFO; consumers *don't* use conservative loss adaptation.

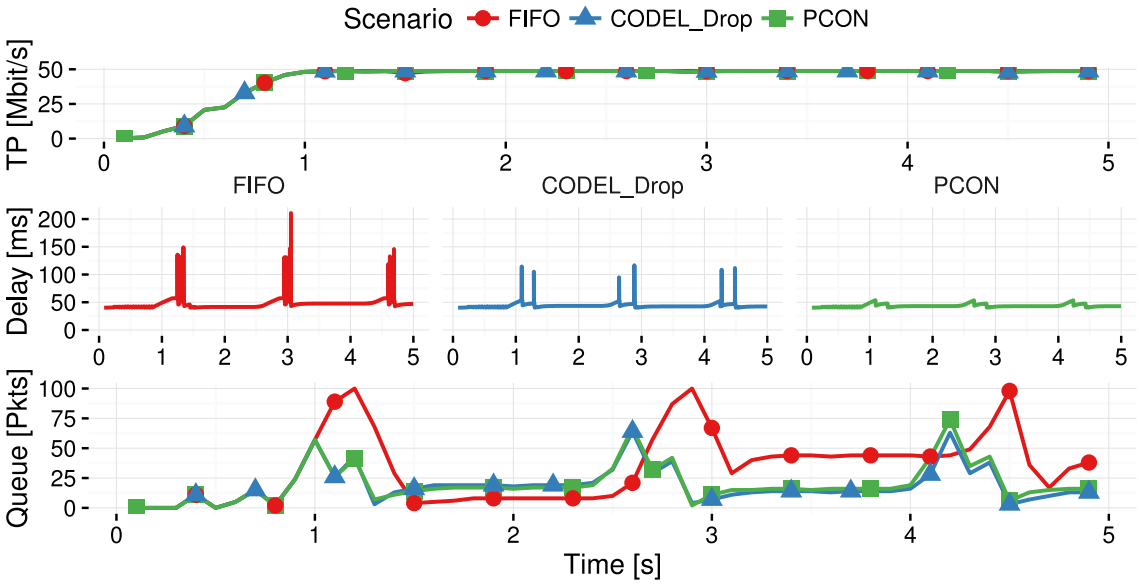


Figure 3.3: Consumer Rate Adjustment and AQM

- **PCON:** As CODEL_Drop plus explicit congestion signaling.

In all scenarios, we use the TCP BIC window adaptation and the traditional TCP RTO estimation (which performs properly, since there is no cache diversity or multipath). BIC uses a "slow start" phase with a pre-configured "maximum increment" that specifies how much the cwnd maximally increases per RTT (here: 16 Interests). After a loss event, it uses a "binary search" to rapidly take up available bandwidth, similar to the initial slow start phase (see [XHR04] for the details). A loss event is triggered either by a timeout (FIFO, CODEL) or by a marked Data packet (PCON).

The results are shown in Figure 3.3. We found that even in this simple scenario, using timeouts from drop-tail queues (FIFO) is problematic: they always need to completely fill the queue, potentially lead to persistent queues, and increase the delay of retransmissions (as discussed in Section 2, we can't use out-of-order delivery like TCP "fast retransmit" in NDN). CoDel and PCON have the benefit that they react before the queue reaches its limit (here 100 packets). Raising the queue size to 1000 packets would have no impact on CoDel/PCON, but heavily increase the

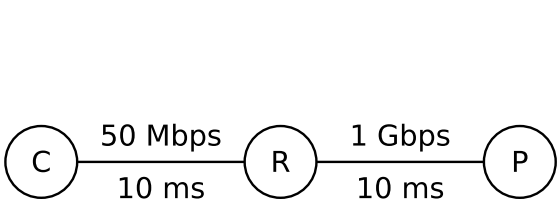


Figure 3.4: Topology 4.1

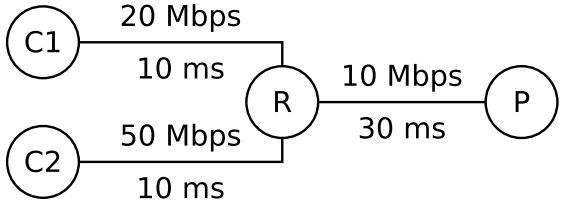


Figure 3.5: Topology 4.2–4.3

queuing delay for FIFO queues. Moreover, PCON's explicit marking scheme achieves the same throughput as the others, but avoids packet drops and thus retransmission delays (spikes in the Delay plot indicate retransmissions).

3.4.2 Caching and Multicast

Next, we evaluate the effect of caching and RTO timer settings. We use two consumer nodes retrieving the same data (see Figure 3.5), with C2 starting at a later time point (5s). Moreover, we increase the delay of the bottleneck link (between R and P) to 30ms, to emulate a situation where consumers have access to a close, high-bandwidth cache and a far away lower-bandwidth repository. We compare PCON against CoDel dropping with traditional TCP timers at the consumer (RTO = meanRTT + 4*varRTT).

The results are shown in Figure 3.6. First, C1 is the only consumer and adapts its sending rate to the bottleneck bandwidth of link R–P, namely 10 Mbps. At 5s, C2 starts and initially gets all the data from the cache at R (with a higher bandwidth of 50 Mbps). At 7s, C2 exhausts the cache and joins C1 in retrieving data from P. This is where the problems of TCP timers begin to show: Both consumers retrieve the same content, thus both occasionally get data from the cache (R) with an RTT of around 20 ms. This smaller RTT reduces the RTO, often to a value smaller than the RTT of the following packets (see "RTO - RTT"), leading to *unnecessary retransmissions*. PCON avoids this problem by setting a fixed higher RTO ("RTO - RTT" is always positive) and using explicit marks for the window adaptation: It

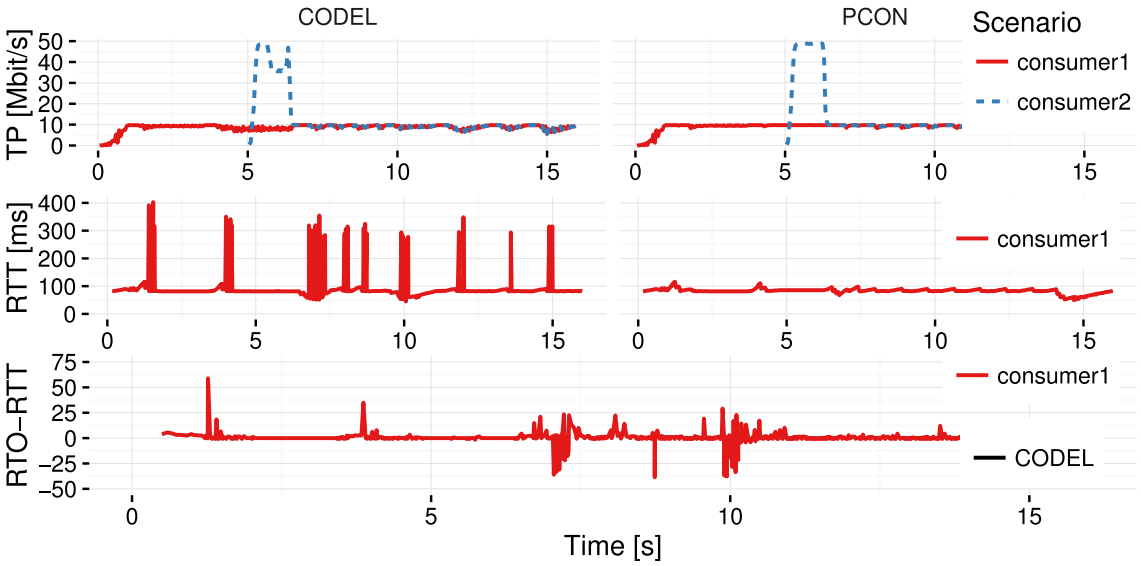


Figure 3.6: Caching and Multicast

achieves a faster congestion reaction (marked packets reach the consumer faster than the RTO) and doesn't cause any packet drops or retransmissions.

3.4.3 Changing Data Packet Size and Cross-Traffic

Next, we evaluate the effect of sudden changes in Data packet size and available link bandwidth (caused by unresponsive cross-traffic). We use the same topology as in the last measurement (Figure 3.5), but only C1 retrieves regular Data packets; C2 is used only for cross-traffic. We look at 4 different scenarios:

- **Payload:** At 5s, the Data payload size is suddenly increased to 50x its original value (from 100 Bytes to 5KB). At 15s, it is reduced back to the original value.
- **Cross_Same:** C2 requests an unresponsive, fixed rate of cross-traffic with 98% of the available bottleneck link bandwidth (reducing the available bandwidth to 1/50 of the original). At 15s, the cross-traffic stops.
- **Cross_Opp_1:** The same amount of cross-traffic as in Cross_Same, but now

in the opposite direction (C2 sends data towards P), resulting in congestion on the Interest path. First, we ignore this congestion and only consider congestion on the Data path (marked packets).

- **Cross_Opp_2:** Same as Cross_Opp_1, but now the routers react to congested Interest packets by marking the PIT entry and then marking the returning Data packet.

The results (Figure 3.7) show that when the Data chunk size suddenly increases (at 5s), the router queues will fill up and the consumer will experience a temporary higher delay, proportional to the change in Data size (here: 50x the RTT of 80 ms, that is, 4 seconds). PCON then quickly reduces the cwnd, so that the queue starts draining and comes back to the normal value at about 12s. At 15s, after the Data size returns to the original 100 Bytes, the consumer rate drops, but the consumer quickly adapts the cwnd to again take up the full bandwidth (at about 18s).

The cross-traffic in the same direction has a similar effect, but creates less delay (300 ms; note the exponential scale). In both scenarios, the consumer quickly reacts to packet marks and reduces its sending rate. If the Interest lifetime (and RTO) is long enough, this works without packet loss; otherwise some packets may be lost (and retransmitted). Nonetheless the adaptation works the same way.

The scenario of cross-traffic in the opposite direction shows why it is necessary to consider the congestion status of Interest packets: If we don't (Cross_Opp_1), the queuing delay at R1 (and thus the consumer RTT) rises indefinitely and the first sign of congestion will be a packet loss due to router buffer overflow. If we consider congestion on the Interest path (Cross_Opp_2), we avoid the queue buildup, at least when the cross-traffic stays below the available bottleneck link bandwidth. For a solution when the cross-traffic exceeds the available bandwidth, see Section 3.4.5.

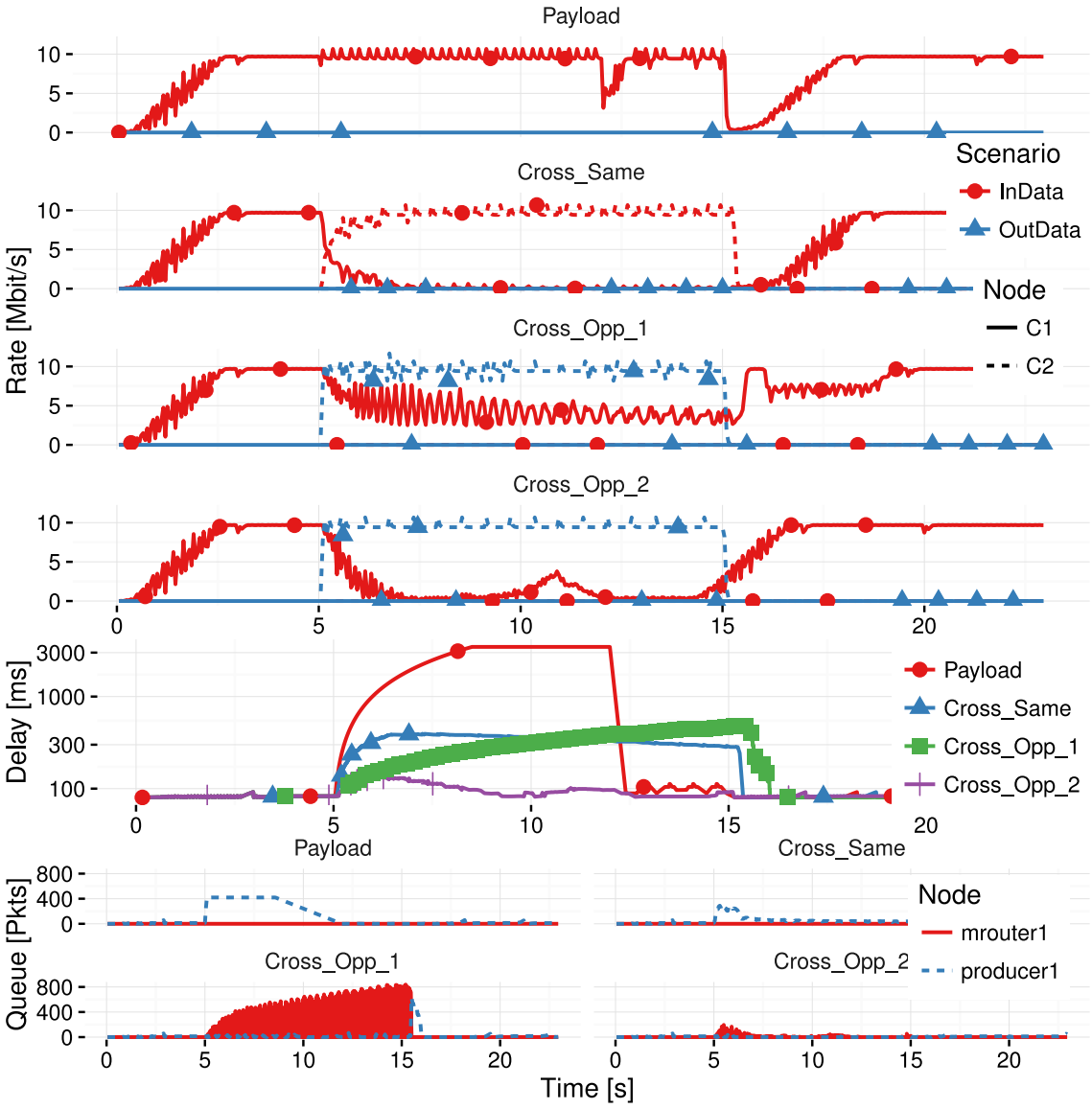


Figure 3.7: Changing Data Packet Size and Cross-Traffic

3.4.4 Multipath Forwarding

Next, we compare PCON's multipath forwarding adaptation (based on congestion marks) against alternatives from the literature. Starting from Detti et al.'s review [DPBM15], we compare PCON against pending Interest adaptation, as all of the alternatives make stronger assumptions: "Fast Pipeline Filling" [DPBM15] assumes knowledge of the pipeline capacity and "RTT Equalization" has the problem of potentially mixing RTT measurements from different "flows" or different cache hits, leading to a suboptimal forwarding ratio.

PI-based schemes don't make these assumptions and the information about pending Interests is readily available at the forwarding layer. Thus, we consider two of these schemes: *PI*, which chooses the face with the lowest number of pending Interests, and *CF* [CGM+13], which uses a weighted round-robin distribution: $weight(face, prefix) \leftarrow 1/avgPI(face, prefix)$, where "avgPI" is the exponential moving average of the number of pending Interests.

In each measurement, we waited until the forwarding split ratio has stabilized and then take its average. We use three scenarios in a topology composed of the nodes R2, P1, P2, and P3 in Figure 3.9 (where R2 acts as the consumer):

- **Equal:** 3 paths with an equal bandwidth of 10Mbit/s and equal RTT of 20ms.
- **Diff_Delay:** 3 paths with an equal bandwidth of 10Mbit/s, but a different RTT of 100ms (face 257), 50ms (face 258), and 10 ms (face 259).
- **Diff_BW:** 3 paths with an equal delay of 20ms, but different bandwidth of 22.5Mbps, 6 Mbps, and 1.5Mbps. Here we expect a different split ratio of 75%–20%–5%.

The results (Figure 3.8) show that PI-based distribution (CF and PI) works well when the bandwidth and delay of all available paths is the same: it achieves the optimal split of 33%–33%–33%. However, when delays differ, both of them prefer

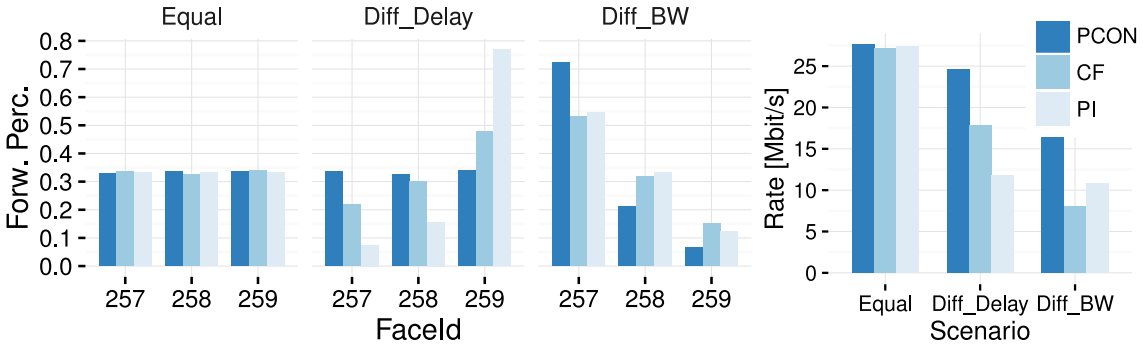


Figure 3.8: Multipath Forwarding: PCON vs. PI vs. CF [CGM+13]

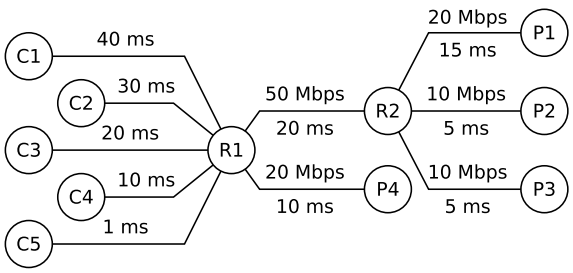


Figure 3.9: Topology 4.4

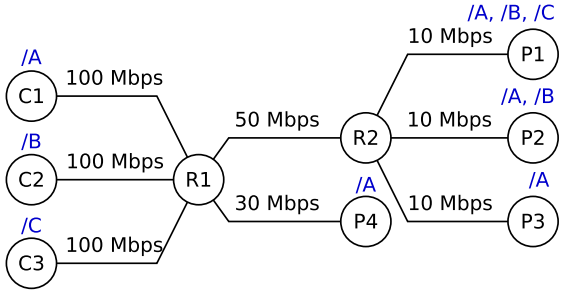


Figure 3.10: Topology 4.5

the lower-delay path at a cost of total throughput (CF achieves a slightly better split ratio and higher bandwidth than PI). When the bandwidth differs, both CF and PI bias against high-bandwidth paths (instead of the optimal ratio of 75%, they only achieves about 54%), again reducing the total throughput. These findings confirm Nguyen et al.'s analysis [NFST15] that PI-based forwarding biases against paths with longer RTT or higher bandwidth. Because PCON responds directly to the congestion state of the involved routers, it achieves the expected split ratio in all three cases, which results in a significantly higher overall throughput in the cases of different delay and different bandwidth.

Next, we compare the full PCON scheme against an alternative from the literature. Namely, Carofiglio et al.'s work of delay-based congestion control, together with route-labeling, Remote Adaptive Active Queue Management (RAAQM), and their multipath forwarding strategy (CF) [CGM+13].

We use a larger topology (Figure 3.9) with 5 consumer apps, all having a high-speed access link (100 Mbps), but different delays (1ms to 40ms). They start at different times (from second 0 to 4) and retrieve different files under the same prefix. All consumers start with the path C–R1–R2–P1 until the forwarding ratio is adapted (based on congestion marks for PCON; based on pending Interests for ICP). After a discussion with the authors, we choose the following ICP parameters, which work well in this scenario: pMin=0.00001, dropFactor=0.005, beta=0.9. We report 3 main findings (Figure 3.11):

First, ICP is faster in adjusting the forwarding percentage to the final value. This is because the number of pending Interests is available almost immediately, but it takes some time (here: about 2.5 seconds) for the congestion marks to arrive.

Second, as shown earlier, PCON arrives at a split ratio that results in a rate that is much closer to the full network bandwidth (60 Mbps) than ICP. ICP, on the other hand, tends to choose paths with a lower link delay. This leads to a different load at the router queues: ICP only creates noticable queuing delay on the bottleneck of the shortest path (P4); PCON distributes the queuing delay evenly among all bottlenecks (R1 and R2 aren't shown in Figure 3.11, because they don't experience any queuing delay). Morever, ICP results in a shorter end-to-end delay at the consumers (see Table 3.1). The average per-packet RTT of all consumers is very close (78ms vs. 79ms), since PCON sends a higher percentage of traffic to the lower-delay consumers.

Third, we see that both ICP and PCON result in a consumer rate that is indirectly proportional to the difference in RTT. Both schemes are close to *linear RTT fairness*, meaning that the throughput ratio is roughly linear to the RTT ratio. ICP achieves a moderately fairer split: its throughput ratio of C5 to C1 is 2.72 instead of 3.71 for PCON (with a RTT ratio of 2.66 and 2.52). If the RTTs of all consumers are equal, both schemes achieve the same fair (meaning equal) split of the available bandwidth.

A minor difference is the way in which the congestion window is adapted: While PCON uses TCP BIC with an initial slow start phase, the current implementation

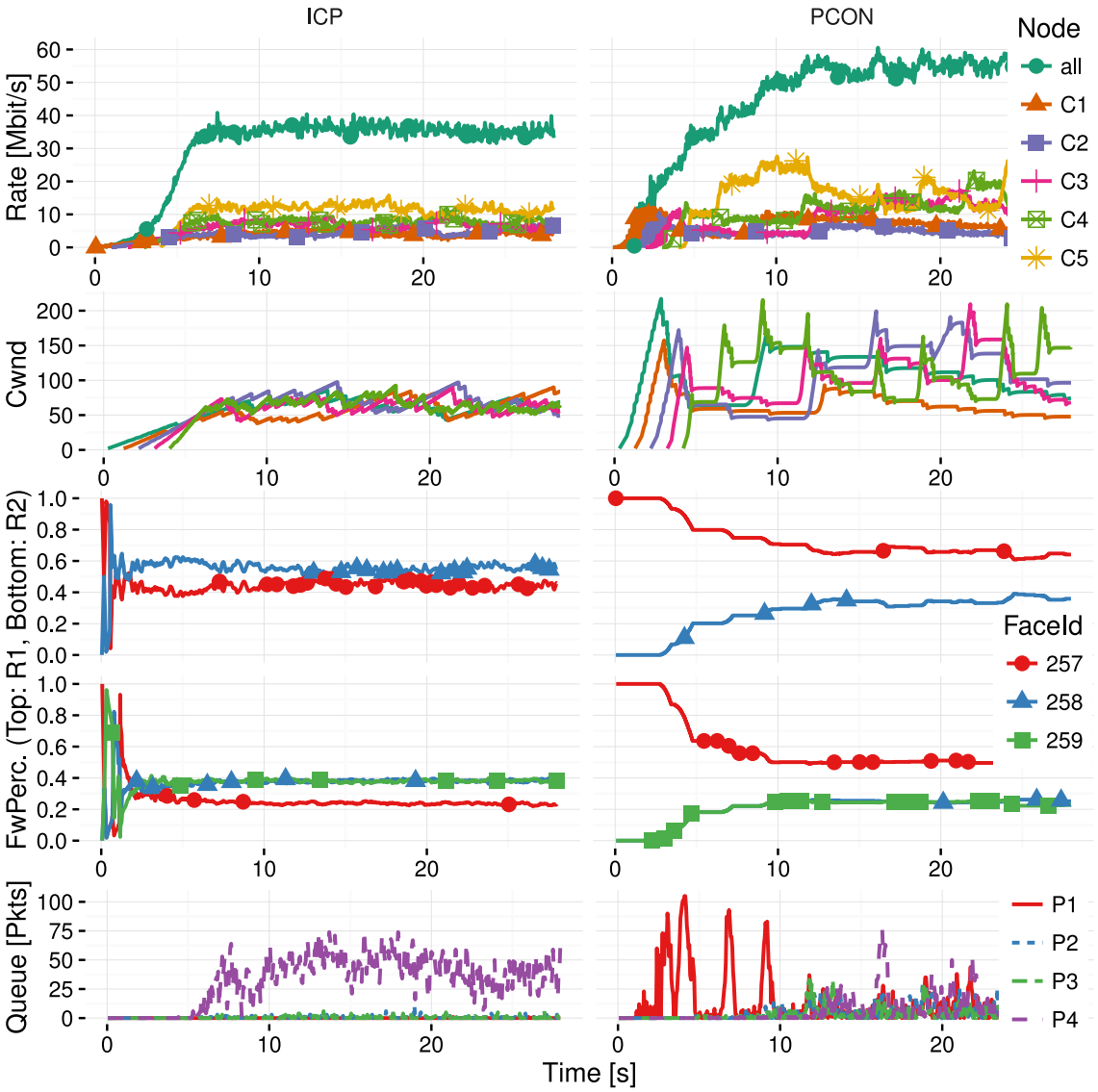


Figure 3.11: Multipath Forwarding: ICP (left) vs. PCON (right)

	ICP		PCON	
	RTT [ms]	Rate [Mbps]	RTT [ms]	Rate [Mbps]
C1	128.31	4.48	132.08	5.53
C2	107.83	5.16	112.37	6.77
C3	88.28	6.34	92.26	9.32
C4	68.01	7.86	72.19	12.69
C5	48.21	12.20	52.46	20.52
All	78.08	7.21	79.23	10.96

Table 3.1: Mean RTT and Rate per Consumer

of ICP uses regular AIMD (like TCP Reno) without a slow start. This difference in the window adaptation mechanism does not affect the other results.

3.4.5 Highly Congested Links

The next scenario investigates PCON's reaction to fully congested links and confirms the rationale behind signaling the "highly congested state" (**PCON**) over only using 1-Bit congestion marks (**CODEL**). We use the topology in Figure 3.10, where each link has a one-way delay of 5ms. There are three consumers with different start times: At the beginning, C1 requests traffic from prefix /A (served by all producers). At 5 seconds, C3 requests traffic from prefix /C (served only by P1) with a high fixed rate and doesn't adapt to congestion marks. Thus, C3 quickly overflows the outgoing queue of P1 (see "Q" in Figure 3.12) and makes the link practically useless for any other traffic. At 15 seconds, C2 starts, requests traffic under prefix /C, and like C1, reacts to congestion marks. At 30 seconds, C3 stops and the link to P1 becomes usable again.

The results (Figure 3.12) show that 1-bit marking is unable to quickly disable and completely avoid the congested link. With the spaced-out packet marks, R2 has no explicit knowledge about whether the queue at R3 is overflowing or not. Thus, R2 sends a small proportion of the traffic (under prefix /A and /B) to P1. These

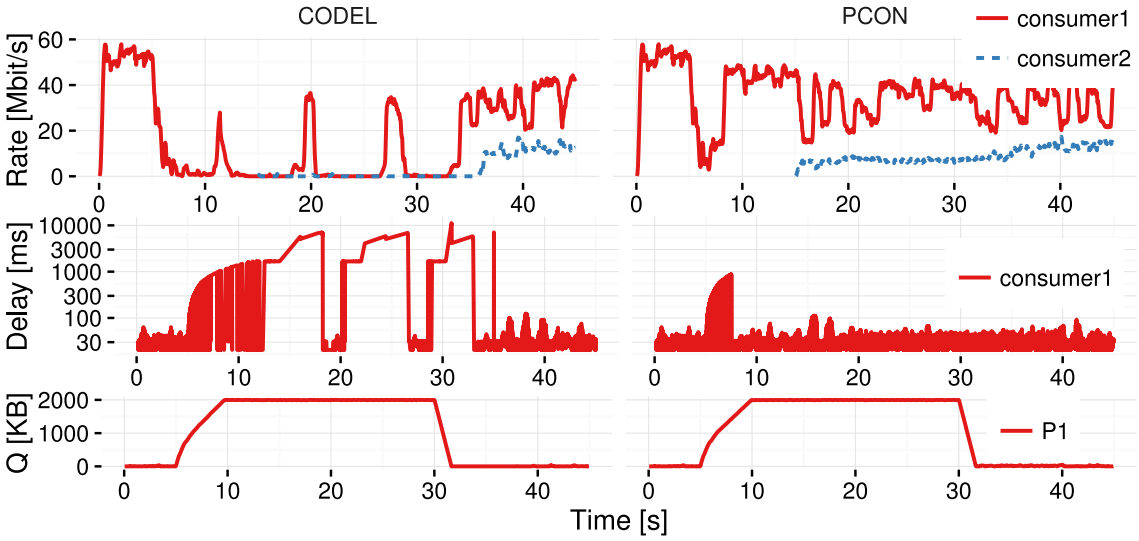


Figure 3.12: Highly Congested Links

packets result in a burst of timeouts and quickly reduce the congestion window of both consumers (C2 can't even go much higher than 1). Moreover, even after R2 has reduced forwarding percentage towards P1 (face 257) to 0%, every time another path receives a congestion mark it will increase again (here to 0.5%) and lead to more timeouts.

PCON's signaling of the highly congested state avoids that problem: At about 7 seconds, it completely disables the use of the interface towards P1 and only re-enables it at about 32 seconds, when the queue has drained. Moreover, since this state can be saved *per-face* at R2 (and per-prefix at R1), the traffic of R2 immediately avoids using P1, even though it is under a previously unseen prefix. This wouldn't be possible without the third bit (highCongLocal), which indicates that the congested link is directly connected. After the cross-traffic disappears at 30s, both consumers quickly take up and share the newly available bandwidth.

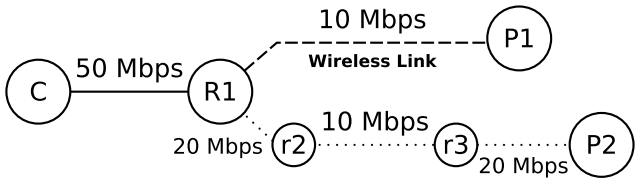


Figure 3.13: Topology 4.6

3.4.6 IP Overlay and Wireless Links

Lastly, we evaluate the performance of PCON against the concept of Hop-by-hop Interest Shaping. In our topology (Figure 3.13), router R1 is connected to P1 over a wireless link and to P2 over a UDP tunnel. We simulate the wireless link as one that randomly drops packets with a probability of 1%. The routers inside the UDP tunnel (r2 and r3) use a drop-tail FIFO queue with a maximum size of 250 KB (about 250 Data packets). We show the queue size of r3, but don't control any behavior of the underlay routers.

For comparison, we implement a simplified version of HBH Interest Shaping based on Wang et al.'s work [WRN+13]. The shaper reads out the fixed link capacity from the network interface and uses a moving average estimation of the Data chunk size to compute the *Interest shaping rate*. Whenever the incoming Interest rate exceeds the shaping rate, the excessive Interests are put into an internal queue (not shown in the Figure) which uses the CoDel logic to signal back congestion via NACKs. We look at 3 different scenarios:

- **Shaper_Ideal:** The shaper somehow knows the available link capacity in the underlay network (10 Mbps) and uses it as parameter.
- **Shaper_Overlay** The shaper at R1 uses its local link bandwidth (20 Mbps) as shaping parameter, which is higher than the available (but unknown) capacity in the IP-underlay network (10 Mbps).
- **PCON:** Our design as described earlier. We detect losses with the link

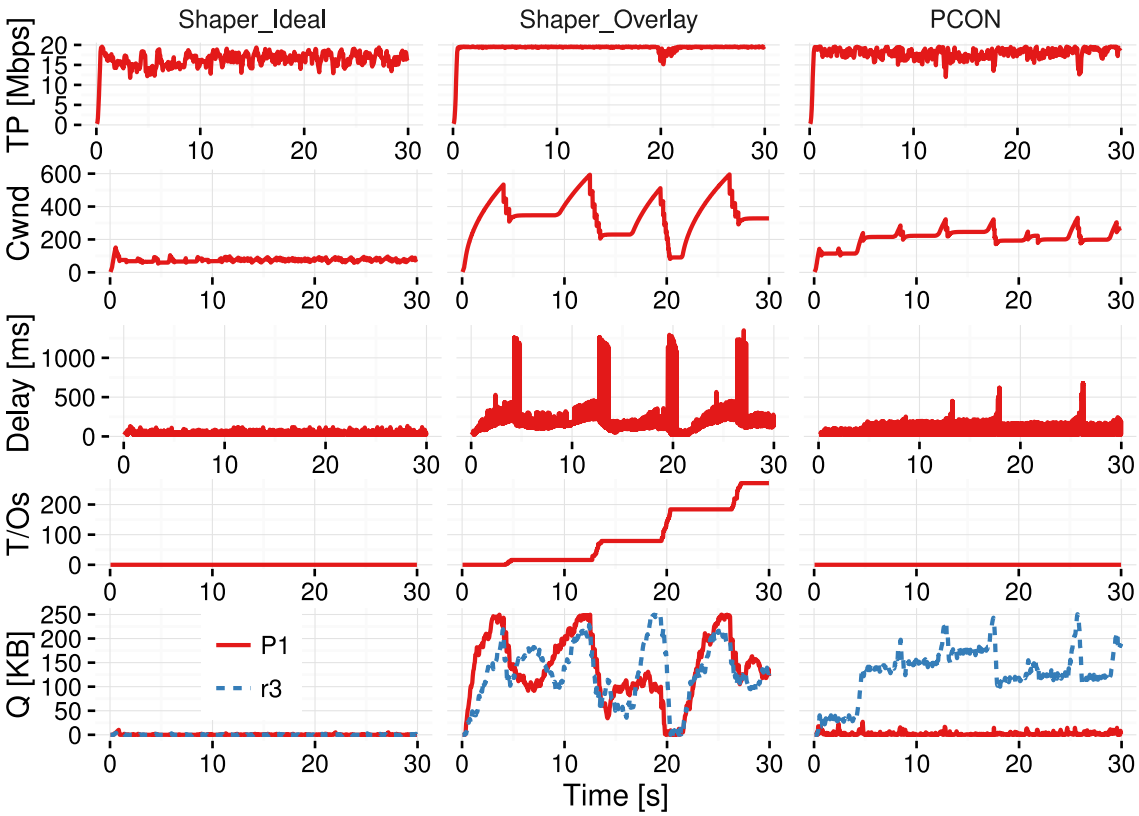


Figure 3.14: IP Overlay & Wireless Links

adaptation layer and signal them back via NACKs. NACKs created by the overlay tunnel contain congestion marks (and thus reduce the cwnd), while NACKs from the WiFi link only trigger a retransmission, but no window decrease. The consumer considers a burst of NACKs during one RTT as *one* loss event. Moreover, the router's forwarding ratio adapts to NACKs at most once per CoDel Interval (100 ms).

All three scenarios start out with the optimal split ratio of 0.5. PCON adjusts it based on the received congestion marks and NACKs (which in this case is unnecessary), while the Interest shaper keeps the optimal value.

The results are shown in Figure 3.14. First, we see that in the ideal case (known link capacity and fixed Data chunk size), Interest shaping achieves its goal of keeping

router queues small, reducing Data retrieval delay, and avoiding timeouts. However, when the shaper assumes a bandwidth that is too high (second column), we see high queuing delay and periodic bursts of timeouts ("T/Os" shows the cumulative timeouts over time). When the shaper bandwidth is set too low (not depicted), we see a predictable lower throughput (when the shaper assumes a link capacity of 5 Mbps, the maximum consumer throughput will be 5 Mbps).

PCON results in a higher queuing and consumer delay than the ideal shaping scenario, because it detects congestion based on losses in the underlay network and isn't able to fully drain the underlay router's queues. However, it is able to control the consumer rate and completely avoid timeouts, even without knowing the bottleneck link bandwidth. Moreover, PCON's forwarding strategy converges to the optimal split ratio, even in the case where congestion signals result from a mix of bursty NACKs and non-bursty CoDel markings.

Our implementation of HBH Interest shaping is arguably simplistic, as it doesn't even try to predict the real bottleneck bandwidth and instead uses the incorrect value of the local link capacity. However, the related work we know of ([CGM12, WRN+13, RF14, LHLX15, ZWL+15, ZZR+14, PJK14, LHX+15]) doesn't consider the possibility of unknown link bandwidth, and leaves the behavior in this case unspecified. Moreover, the task of explicitly estimating the bandwidth of an IP tunnel or wireless link is non-trivial. We would like to see more work in this area, but as for now, we consider our implementation of HBH Interest shaping as representative of the state of the art.

3.5 Related Work

Over the last couple years there have been a number of proposals for congestion control in NDN. Van Jacobson's original paper states that "flow balance is maintained at each hop" [JST+09], but leaves the details of congestion control to future work. Other researchers soon recognized that NDN's caching and multipath forwarding makes

traditional congestion detection by RTT measurement or packet losses inadequate, and proposed to use explicit congestion notification in the form of a truncated Data packet [ORS12] or a NACK [YAM+13].

Carofiglio et al. [CGM+13] proposed a delay-based congestion control scheme, which triggers an Interest window decrease based on a sample of past RTT measurements. To overcome the problem of mixing RTT measurements from different packet sources, they introduced the concept of *route-labels*, tags in each Data packet that indicate which node the packet has originated from and which path it took. However, the practicality and scalability of route-labels has been questioned [NFST15], as the number of potential routes increases exponentially with the number of nodes in the network.

Moreover, even with route-labels, a consumer still does not know from where the next Data packet would come, making it difficult to set a correct timeout value. As an alternative, [SCP13, BMST13] have proposed mechanisms to predict the sources of the next Data packet based on markings of previously returned Data packets. However, the reliability of these schemes in real networks (where the content of each cache might change rapidly) remains an open question.

Another approach to NDN congestion control is *Hop-by-Hop (HBH) Interest Shaping* [CGM12, WRN+13, RF14, LHLX15, ZWL+15, ZZR+14, PJK14, LHX+15], based on the assumption that a router can control the amount of returning data on its links by controlling the Interest sending rate in the opposite direction. As shown in Section 3.4.6, these schemes require explicit knowledge of the available link bandwidth, which limits their effectiveness for wireless networks and IP tunnels.

CHAPTER 4

Multipath Route Computation

4.1 Introduction

While traditional routing protocols (like OSPF [Moy98] or IS-IS [isi90]) send packets on the *shortest path* from source to destination, there are now many recognized benefits of using multiple *non-shortest paths* [HR08], most importantly 1) *failure protection:* routing around link/node failures, and 2) *traffic engineering:* distributing the traffic load to avoid congestion.

A common approach to failure protection is *IP Fast Rerouting* (IPFRR) [SB10], which provides alternative nexthops for quick local packet detours on the data plane. Compared to shortest path routing, IPFRR has clear benefits: it can reroute packets almost instantly instead of waiting for routes to converge (often hundreds of milliseconds), which avoids loops and packet drops during the convergence phase [SB10].

A common approach to traffic engineering is to use end-to-end (E2E) tunnels where the source (ingress) node determines the entire path towards the destination, as done by MPLS [VRC01] or segment routing [FPG+18]. Though frequently used in practice, E2E traffic engineering does have certain downsides: 1) Scalability: endpoints need to select a small number of actual paths from an exponential number of potential ones, trading off path diversity and path length. 2) Data plane overhead: packets need to carry additional headers to steer them through the network.

In this work, we consider an approach to combine both failure protection and traffic engineering: *Hop-by-Hop* (HBH) Multipath Routing, which shares the benefits of IP Fast Rerouting (instant failure protection without relying on routing convergence)

Dest.	NH	Cost	Split
LA	SV	2	100%
DV	DV	1	100%
	SV	2	0%
NYC	DV	5	60%
	SV	6	40%
...			

(a) FIB for **Seattle**

(b) Abilene Topology

Figure 4.1: Ex. of HBH Multipath Routing. Depending on the destination, Seattle uses either 1 or 2 nexthops. The cost denotes the shortest path (in hops) of using a certain nexthop. The split ratio can be adapted to react to link failures or congestion.

and also provides traffic engineering without per-packet overhead by distributing the path/nexthop decision throughout the network. In HBH Multipath Routing, each router's FIB is equipped not with a single nexthop per destination, but with a set of nexthops. Every router on the path (not just edge routers) can then steer traffic by changing the *split ratio* for each of its *nexthops* (not for the entire path) – see an example in Figure 4.1. These nexthops can be used both for failure protection (instantly reset the split ratio to 0%) and for reducing congestion (gradually change the split ratio).

HBH Multipath Routing comprises two fundamental steps: First, one needs to decide *which* nexthops to put in the FIB and what cost these nexthops should be assigned. This step determines the *potential* paths that packets can take, based on the network topology and link cost/propagation delay. Second, one needs to determine the *split ratio* for each nexthop. This step selects the *actual* paths from the vast number of potential ones, considering the current network state, such as link failures and link utilization. In this work, we focus solely on the *first step:* choosing the right nexthop set.

To choose the right nexthop set, we first establish a list of requirements. These requirements overlap with the ones for pure failure protection (see IPFRR in Section 4.5), but with two crucial differences: First, *path lengths are more important:* For failure protection, backup paths are used only temporarily until either the failed link is restored or the routing protocol finds a better path – maintaining connectivity is more important than optimality. But for HBH traffic engineering, multiple paths may be used for a much longer duration, making it crucial to maintain a short length. Second, *failed links are bidirectional, congestion is unidirectional.* Thus, if router A detects that link (A→B) is down, it is safe to assume that link (B→A) is also down, which extends A's rerouting options. As a corollary of this, one can use the incoming port of packets to infer which other links have failed, and thus to alter forwarding decisions [Atl06, VLK13, YLS+14, CNM+16]. However, when splitting traffic to avoid congestion, other nodes' forwarding options are naturally less restricted (due to the absence of failures), thus nexthops must be chosen *more* restrictively:

1. **Avoid loops for arbitrary NH choice:** Packets must not return to a node they have previously visited, even if any number of routers can choose independently (without communication) and arbitrarily from their nexthop set. An example violation of this requirement is shown for *Loop-free Alternates* in Section 4.5.

2. **No per-packet state:** No packet-specific state is manipulated in either packet headers or the FIB. Thus, forwarding decisions are based only on a packet's destination, incoming port, and currently failed links.

3. **High Number of NHs:** A high number of nexthops (and thus potential paths) allows routers to circumvent more cases of link/node failure and congestion.

4. **Short Paths:** The potential paths resulting from the nexthop set should be kept as short as possible. This helps to use the available link capacity more

efficiently, and also to reduce the end-to-end latency for delay-sensitive traffic, like audio/video conferencing.

5. **Preserve accurate path length in the FIB:** The data plane should be given an accurate estimate of the path length caused by using a certain nexthop, to increase traffic engineering efficiency. We show the downsides of violating this requirement when discussing ECMP link-weight tuning in Section 4.2.

Existing HBH Multipath Routing schemes [NST99, Vil99, XCR07, XCR11, GRJ08, PM09, MT15, YAM$^+$13]) have, often implicitly, answered these requirements with a natural solution called the "Downward Criterion": Routers only add nexthops that lead closer to the destination. More generally, the idea is to turn the network into a *directed acyclic graph* (DAG) per destination, where arcs in the graph represent viable nexthops (see Figure 4.4).

We find that downward paths and DAGs satisfy requirements 1), 2) and 5), but often fall short in the number of potential paths and sometimes lead to longer paths than necessary (see Section 4.2 and 4.4). In our work, LFID, we extend the concept of DAGs to use certain links in *both directions* (per destination), but we exhaustively prune nexthops so that the remaining paths are guaranteed to be loop-free (= acyclic) when excluding the incoming port at each step.

Our contribution, LFID, can be interpreted in two ways: First, it is a *local failure rerouting* scheme that, without using per-packet state, gives close to optimal protection against an arbitrary number of uncorrelated link or node failures.

Second, it is a *first step towards HBH traffic engineering.* Here we are using a small amount of state in routing tables (the *split ratio*) in order reduce the load on congested links. Here LFID only specifies the first step: Which nexthop set to choose, to create good *potential* paths. It leaves the rest to future work, e.g., how to detect congestion, what granularity to split (per-flow/per-prefix), or how exactly to determine the split ratio. However, irrespective of these specifics, we show that LFID

comes close to the optimal, in number of provided paths and with an average path stretch of only +1% (Section 4.4.5). Compared to DAG-based work, LFID protects against a higher percentage of single and multiple failures/congestions. Moreover, LFID often provides better protection at the node *adjacent* to the failure than related work can by *backtracking* to earlier nodes or all the way to the source (see Section 4.4.4). Lastly, LFID does have a slightly higher time and space complexity than related work (Section 4.3.5), but we show it to be scalable for networks with at least multiple hundreds of routers (Section 4.4.2).

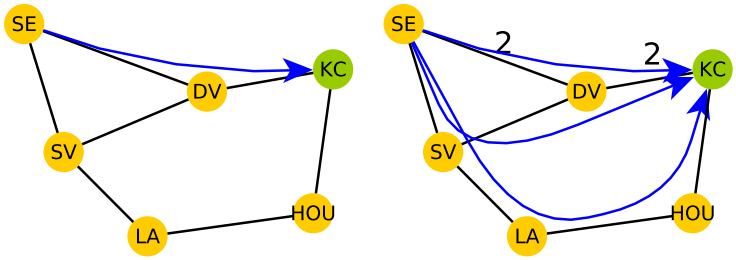
LFID can be implemented as an extension of current link-state routing; all required topology information is already signaled in a link-state protocol like OSPF. The only changes made are to the route calculation part.

4.2 Increasing the Nexthop Choice

We first discuss related work that meets at least the first two requirements: being loop-free when routers arbitrarily choose nexthops, without using per-packet state. Later in Section 4.5, we discuss work in the area of IP Fast Rerouting (IPFRR) which either requires per-packet state, or restricts nexthop choice, for example, to only use backup nexthops once all primary nexthops are down.

The earliest and simplest of the related work is *Equal Cost Multi-Path* (ECMP) routing [ecm00], in which a router uses all nexthops that share the exact shortest path cost towards the destination[1]. ECMP paths are always loop-free and as short as possible. However, the constraint of equal cost matching creates a dilemma: the more *fine-grained* the link cost metric, the fewer nexthops will be available. For example, using the inferred link weights of the Rocketfuel topologies (ranging from 1 to 22.5, in steps of 0.5), only 9.7% to 16.8% of nodes can protect against the failure of an adjacent link (see Figure 4.9 in Section 4.4). If the link metric is set to the

[1]We use the terms "path cost" and "path length" interchangeably. Similarly, we use the terms "link cost", "link weight", and "link metric" interchangeably.



(a) (b)

Node	NH	Cost	Split
SE	DV	4	33%
	SV	4	66%
SV	DV	3	50%
	LA	3	50%

(c)

Figure 4.2: Example of ECMP + Link weight tuning. a) Base topology that allows only one equal-cost path. b) Topology with changed link weights to create three equal-cost paths. c) FIB for changed topology with destination KC.

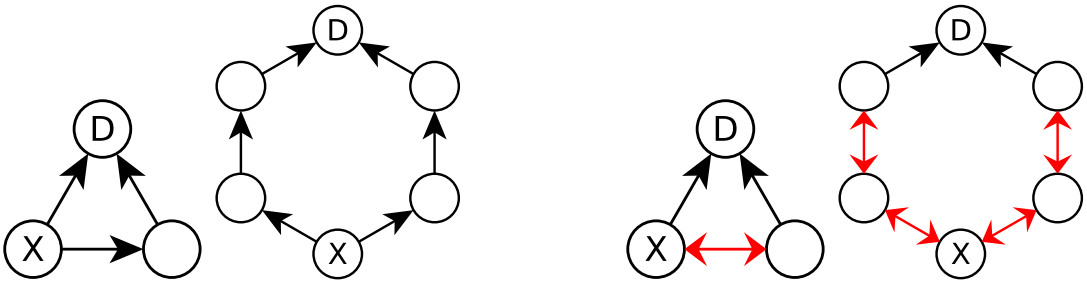
hop count these numbers rise to 29.4% to 59.4%, but this ignores the real cost of paths (e.g., determined by physical distance), and still provides less protection than the work discussed below. Lastly, some work [FT00, FRT02, ICBD04] suggests to further increase the number of equal-cost paths by carefully tuning the link weights to that goal. This does increase the nexthop choice, but also violates requirement 5 (preserving the real path cost), with undesirable results: Now, the forwarding plane treats some paths of different length as equal, which leads to inefficient use of network resources and higher end-to-end delay. Consider the example in Figure 4.2 for traffic from Seattle to Kansas City. If the link metric is set to the hop count (i.e., every link has a weight of 1), ECMP will only provide one path: SE→DV→KC (Figure 4.2a). Now, one can adjust the link weights to add a path SE→SV→DV→KC, or even SE→SV→LA→HOU→KC, as done in Figure 4.2b. However, the only way ECMP can use these paths is to split traffic on them *equally*, i.e., every path receives about 33% of traffic (Figure 4.2c). As long as link utilization is below capacity, this is wasteful: some traffic that could have used the shortest path (SE→DV→KC) is needlessly sent over a longer path. A better approach would be to keep traffic on the shortest path until demand exceeds its capacity, and only then switch to the longer path. However, ECMP + link weight tuning cannot achieve this, as it doesn't preserve the real cost of the path.

A higher path choice and a more accurate cost representation can be reached by *non-equal cost* multipath algorithms, the most prominent of which is the *Downward Path Criterion* (DW) [HR08]. Downward paths relax the equal cost constraint by including the shortest path nexthop plus any nexthop n_i that is closer to the destination (has a lower cost) than the current node x: $cost(n_i) < cost(x)$. Downward nexthops are simple to compute, requiring only one shortest path computation for each neighboring node, and achieve a higher path choice than ECMP. Thus, they are used frequently in the literature, known under the names of Loop-Free Invariant (LFI) [VGLA99], "viable" nexthops [NST99], Rule 1 (One Hop Down) Deflection Set [YW06], and Relaxed Best Path Criterion [Vil99].

One extension of Downward Paths (which we'll call downward+equal – DWE) is to also consider nexthops with the same cost: $cost(n_i) \leq cost(x)$. However, to prevent packets from forming loops, one needs to add a *tiebreaker* which assures that traffic only crosses one direction of the equal-cost link. This tiebreaker could be based on the node degree [KDM09] or simply the node id: $cost(n_i) < cost(x) \vee (cost(n_i) = cost(x) \wedge id(n_i) < id(x))$. This approach is still guaranteed to be loop-free and, compared to downward paths, provides at least as many nexthops, and often more.

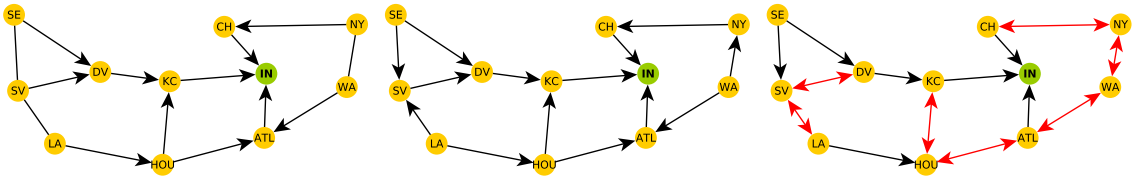
A further improvement of nexthop choice is achieved by the three algorithms from the work "Maximum Alternative Routing Algorithm" (MARA) [OIVM09], which create a "Maximum Adjacency Ordering", that is equivalent to turning the network into Directed Acyclic Graphs (DAGs). Most relevant for our purposes are the variants MARA-MC, which "maximizes the minimum node connectivity", and MARA-SPE which does the same but with the constraint to always include the shortest path tree (see Section 4.4).

The algorithms above all turn the network into a DAG, with both DWE and MARA achieving the highest nexthop choice theoretically possible in a DAG, since both use one direction for every link for every destination (in contrast, in ECMP and DW, some links are not used for some destinations – see Figure 4.4). However, DAGs



(a) DAG (b) LFID

Figure 4.3: Triangle and Ring topologies



(a) Downward Paths (b) DAG (c) LFID

Figure 4.4: Routing entries in the Abilene topology for destination Indianapolis (IN). Comparing strictly Downward Paths, a Directed Acyclic Graph (DAG) using all edges, and Loop-Free Inport-Dependent (LFID) routing.

face a theoretical limit in the failure protection they can offer: for every topology and every destination there is at least one link that is not protected against failure (or congestion). This is called the "last-hop problem" [IBTD03]. Consider the simple triangle and ring topologies in Figure 4.3a. In a DAG, only node X will be protected against failure of its primary nexthop.

4.3 Moving Beyond DAGs

To increase nexthop choice beyond the limits of a DAG, we (per destination) use certain links in both directions. This leads to many more paths, which can be used to handle both failure and congestion: Parts of the topology that form a ring can be traversed in both directions, leaving every node with two paths towards the destination (see Figure 4.3b). And resilience to single link failure in the Abilene topology (Figure 4.4) increases from around 30% to almost 100% (see Figure 4.9 in Section 4.4). However, when using links in both directions, one needs to be careful to avoid loops. We do so with two mechanisms:

First, we exclude 1-hop loops at the data plane. Every router will always exclude the incoming port of a packet from the viable nexthop set – hence the name *Loop-Free Inport-Dependent* routing. For example, if node Atlanta (ATL) in Figure 4.4 receives a packet from Houston (HOU), it will only consider Indianapolis (IN) and Washington (WA) as nexthops, but never send the packet back to Houston.

Second, one needs to avoid loops longer than one hop. Preferably, while also maximizing link & node protection and minimizing path stretch. Unfortunately, there is no simple rule like the downward criterion to do this.

Thus, we approach the problem as follows. For each destination: First, we add all nexthops, distinguishing between ones going closer to the destination (downward) and ones moving further away (upward) – see Section 4.3.1. Second, we iterate through all upward nexthops and remove the ones that would cause a loop (Section 4.3.2). Here, the ordering in which nexthops are checked is crucial. We discuss the

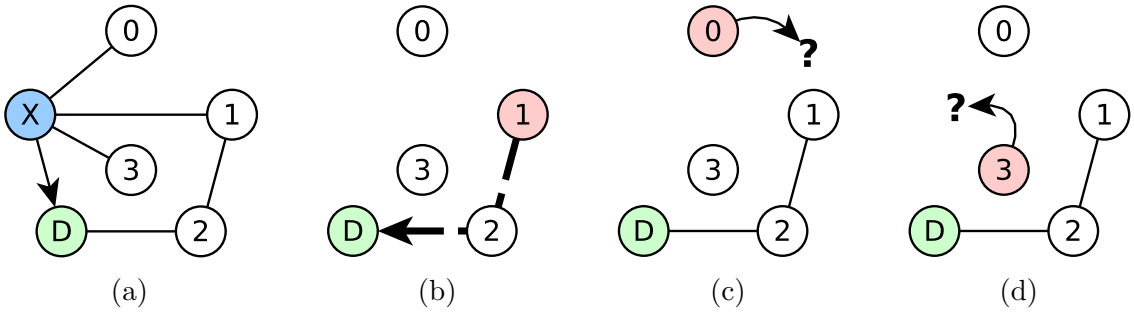


Figure 4.5: Avoiding obvious loops. a) base topology; b-d) candidate nexthops 1, 0, and 3.

one that produces the best results in Section 4.3.3. Lastly, this loop removal process can leave certain nodes as a dead end, where incoming packets can only return to the previous node. We prune these dead ends in the final step (Section 4.3.4).

4.3.1 Adding Nexthops & Deciding Their Cost

We assume that each link is given a cost/weight, for example, based on its propagation delay. Given this link cost, what cost metric should be assigned to each nexthop in the FIB? Intuitively, it should be the cost of the shortest path a packet can take by using this nexthop. This cost can be computed efficiently by adding the link weight between current node x and nexthop n_i to the shortest path cost *from* the nexthop to the destination: cost(x to dst via n_i) = w(x, n_i) + sp(n_i, dst).

However, this approach has one drawback: Sometimes a nexthop can only reach the destination by going back through the current node, causing a loop. Consider Figure 4.5, where current node X sends packets to destination D. All three nexthops (0, 1, and 3) have the same shortest path cost of 2 hops, but only nexthop 1 can be used without looping back to X. This flaw can be fixed by calculating the shortest path cost of neighbors in a graph where the current node X (or all of its links) was removed. Now, nexthops that would loop back through X will receive a cost of infinity, thus will not be added to the FIB (see Figures 4.5c and 4.5d).

Algorithm 3: Filling the FIB for all nodes

```
Function fillFib (Graph g)
    AllNodeFib fib;
    forall x in allNodes do
        map⟨DstId, Cost⟩ shortestPathCosts ← runDijkstra(g,x);
        map⟨nbId, map⟨DstId, Cost⟩⟩ neighborCosts;
        Remove node x from graph g;
        forall n_i in neighbors do
            neighborCost[n_i] ← runDijkstra(g,n_i);
        Add node x back to graph g;
        forall dstId in destinations do
            spCost ← shortestPathCosts.at(dstId);
            forall n_i in neighborCosts.at(dstId) do
                totalNCost ← n_i.cost + linkWeight(nodeId, n_i.id);
                if neighborCost < spCost then
                    fib[x][dstId].add(n_i.id, totalNCost, DW);
                else if neighborCost < ∞ then
                    fib[x][dstId].add(n_i.id, totalNCost, UW);
    return fib;
```

The pseudocode for adding neighbors, while avoiding obvious loops, is shown in Algorithm 3. For every node in the network, we compute the shortest path (towards all destinations) once for the node x itself and once for all of its neighbors n_i with node x removed from the graph. We then add the nexthops to the FIB, distinguishing between *downward* (n_i is closer to the destination than x), *upward* (n_i is further away), and *disabled* (n_i can only reach the destination through x, and this is omitted from the FIB).

The complexity of the first half of this algorithm is as follows (m = number of links; n = number of nodes; k = number of neighbors per node): n (for all nodes) * k (for all neighbors) * $m + n \log n$ (for Dijkstra's algorithm) = $O(kmn + kn^2 \log n)$. The complexity of the second half is n (for all nodes) * n (for all destinations) * k (for all neighbors) = $O(kn^2)$. Thus, the total complexity remains $O(kmn + kn^2 \log n)$.

4.3.2 Removing Loops

After determining the cost and type of each nexthop, we check for each one whether it will cause a loop, and remove the ones that do. We only need to check *upward nexthops*, i.e., ones that lead further away from the destination., since each loop contains at least one upward step. Thus, after removing all loop-causing upward nexthops, the network is loop-free.

Algorithm 4: Removing Loops

```
Structure NodePrio
    nodeId, remainingNh, set⟨upwardNh⟩;
Function removeLoops (allNodeFib)
    forall dstId in destinations do
        DiGraph dg ← getDigraphFromFib(allNodeFib.at(dstId));
        // Queue ordered by max. remainingNh, then cost
        priorityQueue⟨NodePrio⟩ pq;
        pq.push(allNodeFib.getAllUwNexthops(dstId));
        while !pq.empty() do
            node ← pq.pop();
            nh ← node.getHighestCostUwNh();
            // Remove opposite of upward NH from graph:
            dg.erase(nh.id, node.id);
            // Check if Node is still reachable from uwNh:
            bool willLoop ← dg.isConnected(nh.id, node.id);
            if willLoop then
                node.remainingNh−−;
                allNodeFib[nodeId][dstId].remove(nh);
                dg.erase(node.id, nh.id);
            dg.add(nh.id(), node.id()); // Add opposite link back
            if node.hasRemainingUwNHs() then
                pq.push(node);
```

As shown in Algorithm 4, we simulate the network graph for each destination (both downward and upward nexthops are arcs in the graph). We iterate through all upward nexthops, ordered by using a priority queue (see next subsection), and perform the loop-check as follows: For each of the upward nexthops ($x \to n_i$), we temporarily remove the opposite of the upward link ($n_i \to x$), and then check whether

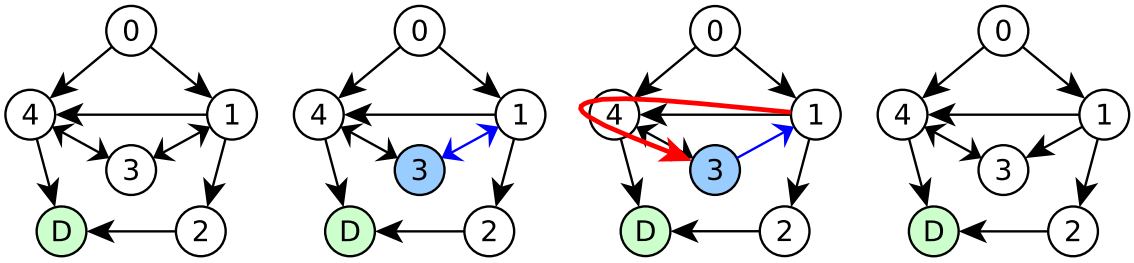


Figure 4.6: Avoiding non-obvious loops. a) base topology; b) candidate upward nexthop (3→1); c) BFS loop check; d) topology afterwards.

there is a path from n_i to x. If a path exists, it means that the upward nexthop may cause a loop and thus we remove the nexthop $(x \rightarrow n_i)$ from the graph and from the FIB. If there is no path, it means the nexthop cannot cause a loop, so we move on to the next one in the list.

We give an example in Figure 4.6. We select upward nexthop (3→1) for the check (b). After removing the reverse nexthop, (1→3), there still exists a path from 1 to 3 (c), thus we remove upward nexthop (3→1) from the graph (d).

Since 1) all remaining upward nexthops will not cause loops longer than 1 hop, 2) a sequence of downward nexthops cannot loop, and 3) 1-hop loops are avoided by excluding the incoming port at each step, it follows that every possible path in the network is guaranteed to be loop-free.

4.3.3 Ordering of Nexthops

For the loop-removal step, the order of nexthops is crucial: the resulting network is always loop-free, but some orders require more nexthops to be removed and/or lead to longer paths. We obtained the best results with the following order:

1. Sort all nodes by their number of remaining total nexthops (upward + downard), starting with the node with the most remaining nexthops.

2. If multiple nodes have the same number of remaining nexthops, sort those by

the cost of their most costly upward nexthop (starting with the highest).

Both of these steps can be implemented efficiently by using a priority queue filled with an ordered data structure, which we call *NodePrio* (see Alg. 4). The algorithm will pick the first node from the priority queue (which has the most remaining nexthops, and on a tie the most costly upward nexthop), and then pick its most costly upward nexthop for the loop check. If the algorithm removes this nexthop, it will reduce the number of remaining total nexthops for the current node, thus changing its position in the queue. Lastly, each upward nexthop is only checked once, so nodes without any remaining upward (not just total) nexthops will drop out of the queue.

Nexthops that are checked for loops earlier in this process have a higher chance of being removed, since the graph contains more upward nexthops that can contribute to a loop. Thus, ordering nodes by their number of remaining nexthops helps to equalize the number of remaining nexthops after the loop removal is complete. For example, a node with 4 nexthops (3DW, 1UW) will be checked earlier than another one with only 2 nexthops (1DW, 1UW), leaving the latter one a higher chance of keeping its upward nexthop. Moreover, checking higher cost upward nexthops before lower cost ones helps to prune longer paths rather than shorter paths, leaving the remaining paths shorter than otherwise.

The complexity of the loop removal is n (for all destinations) * m (upper bound for all upward nexthops) * $m + n$ (for the connectivity check) = $O(mn * (m + n)) = O(m^2 n)$, since $m > n$. This is the most time-consuming step in our algorithm.

However, the actual runtime (Section 4.4.2) is much faster than this (worst-case) complexity implies. The connectivity check is most efficiently implemented with a bidirectional (BFS) search, which on average runs much faster than m+n. For example, in the Sprint topology, only an average of 32 (out of 315) nodes and 46 (out of 972) links need to be visited.

Algorithm 5: Removing Dead Ends

```
Function removeDeadEnds (allNodeFIB)
    forall dstId in destinations do
        set⟨pair⟨NodeId, FibNextHop⟩⟩ uwNhSet;
        uwNhSet ← allNodeFIB.getUwNhs();
        while !uwNhSet.empty() do
            pair⟨nodeId, nh⟩ ← uwNhSet.pop();
            // Get reverse FIB entries:
            reverseEntries ← allNodeFIB.at(nh.id).countNh(dstId);
            // If there is just one reverse entry: found dead end!
            if reverseEntries == 1 then
                allNodeFIB.at(nodeId).erase(dstId, nh);
                // Push into Set: All NhEntries that lead to nodeId!
                uwNhSet.push(allNodeFIB.at(dstId).getNhsTo(nodeId));
```

4.3.4 Removing Dead Ends

This exhaustive search through all upward nexthops avoids all forwarding loops, but can lead to a small number of *dead ends*, cases where a router receives a packet (via an upward nexthop) but its only forwarding option is to directly return the packet to the previous node. Fortunately, compared to loops, dead ends are easy to detect and remove (see Algorithm 5): We iterate through all remaining upward nexthop entries and check which of them lead to a node with a FIB size of 1 (meaning that the only nexthop of the neighbor is the downward nexthop leading back to the original node). We then remove these upward nexthops, effectively eliminating all dead ends.

An example of a dead end is shown in Figure 4.6d. Nexthop (4→3) leads to a node (3) which can only directly return the packet (FIB size=1), thus this nexthop should be removed.

The time complexity of this step ($O(mn)$) is smaller than the one of adding nexthops or removing loops.

Table 4.1: Worst Case Time Complexity

Algorithm	At each node	Network Total
ECMP	$O(m + n \log n)$	$O(mn + n^2 \log n)$
DW, DWE	$O(km + kn \log n)$	$O(mn + n^2 \log n)$
MARA-MC	$O(mn + n^2)$	$O(mn + n^2)$
MARA-SPE	$O(mn + n^2 \log n)$	$O(mn + n^2 \log n)$
LFID	$O(m^2n + kn^2 \log n)$	$O(m^2n + kn^2 \log n)$

4.3.5 Complexity Analysis

Combining the time complexity (m=links; n=nodes; k=neighbors) of the earlier steps, we get $O(kmn + kn^2 \log n)$ for adding FIB nexthops + $O(m^2n)$ for the loop removal + $O(mn)$ for the deadend removal = $O(m^2n + kn^2 \log n)$.

A comparison with the related work is shown in Table 4.1. Similar to MARA [OIVM09], LFID has the same complexity whether it's run for a single node or for the whole network. Thus, it is possible (but not necessary) to compute the routing table once and push it to all routers. The worst-case time complexity of LFID is higher than MARA by up to a factor of m, the number of links in the network. However, as discussed in Sections 4.3.3 and 4.4.2, the average-case runtime is much closer.

A quick note on *space complexity.* In LFID the router computing its nexthops needs to store the neighbor cost and type (DW, UW) of all nodes, not just its own. Thus, the space complexity increases from $O(nk)$ for computing downward paths to $O(n^2k)$. If the node id is stored as 2 Bytes (allowing up to 65536 nodes), the cost is stored as 4 Bytes (up to 4.3 billion), and type is stored as 1 Bit, our largest tested topology (n=315, k=6.17) needs around 3.8 Megabytes of memory. This should be feasible, given that current routers have memory in the order of tens of Gigabytes.

Table 4.2: Evaluation Topologies

Name	N	L	Deg
Abilene	11	14	2.55
Geant	27	38	2.82
Exodus	79	147	3.72
Ebone	87	161	3.70

Name	N	L	Deg
Telstra	108	153	2.83
Abovenet	141	374	5.31
Tiscali	161	328	4.07
Sprint	315	972	6.17

4.4 Evaluation

In this section, we compare LFID against related work. To emulate the multipath forwarding behavior, we implement a custom C++ simulator, using the Boost graph library for Dijkstra's algorithm and BFS.

We compare 8 topologies (Table 4.2) of different size, node degree (Deg), and link metrics: 1) The Abilene and GEANT topology with link weights set to the geographical distance in miles, rounded to 10 miles (values range from 11 to 224) and 2) the six measured ISP topologies from the Rocketfuel [SMW02] dataset with their inferred link weights (ranging from 1 to 22.5, in steps of 0.5) [MSWA02].

4.4.1 Algorithms & Scenarios

We compare LFID with the multipath routing algorithms discussed in Section 4.2, all of which differ in how they choose the set of nexthops at each router:

- **ECMP:** Equal Cost Multi-Path [ecm00] uses the nexthop of the shortest path n_{sp}, plus any nexthop n_i with the same cost: $cost(n_i, dst) = cost(n_{sp}, dst)$.
- **DW:** Downward paths [HR08] include the shortest path nexthop plus any nexthop that is closer to the destination than the current node x: $cost(n_i, dst) < cost(x, dst)$.
- **DWE:** Downward + Equal Cost Nexthops include all nexthops from DW and, in addition, all nexthops with both an equal cost to the destination and a lower

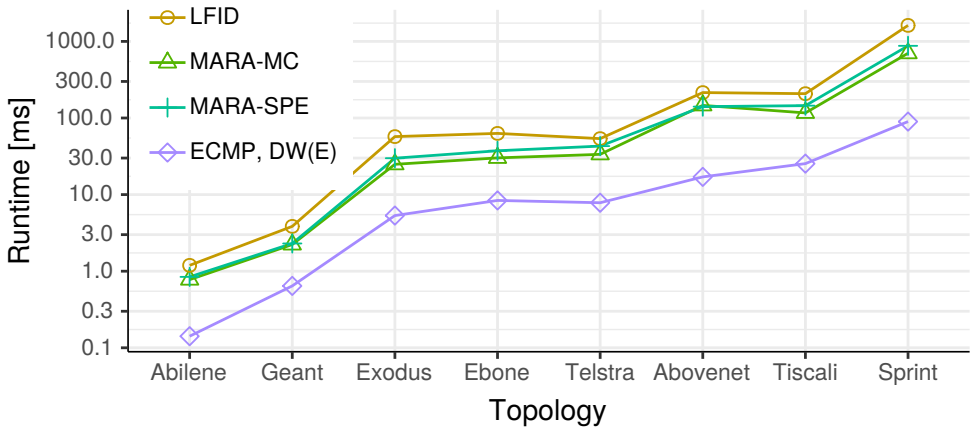


Figure 4.7: Runtime for computing the FIB for all nodes towards all destinations.

node id.

- **MARA-MC** [OIVM09] creates a Directed Acyclic Graph (DAG) with the specific goal of maximizing the minimum connectivity among all nodes.
- **MARA-SPE** [OIVM09] has a similar goal, except that it always includes the shortest path tree in the graph.
- **LFID:** Our algorithm, as described in this work.
- **OPT:** The optimal result based on the network topology constraint. This optimum is usually not achievable via loop-free hop-by-hop routing, and serves to show the theoretical limit of the other schemes.

Below, we evaluate the presented routing algorithms in 4 different scenarios: *measured runtime* (4.4.2), resulting *path length & number of paths* (4.4.3), resilience to *single link & node failures* (4.4.4), and lastly resilience to *multiple simultaneous link failures and congestion events* (4.4.5).

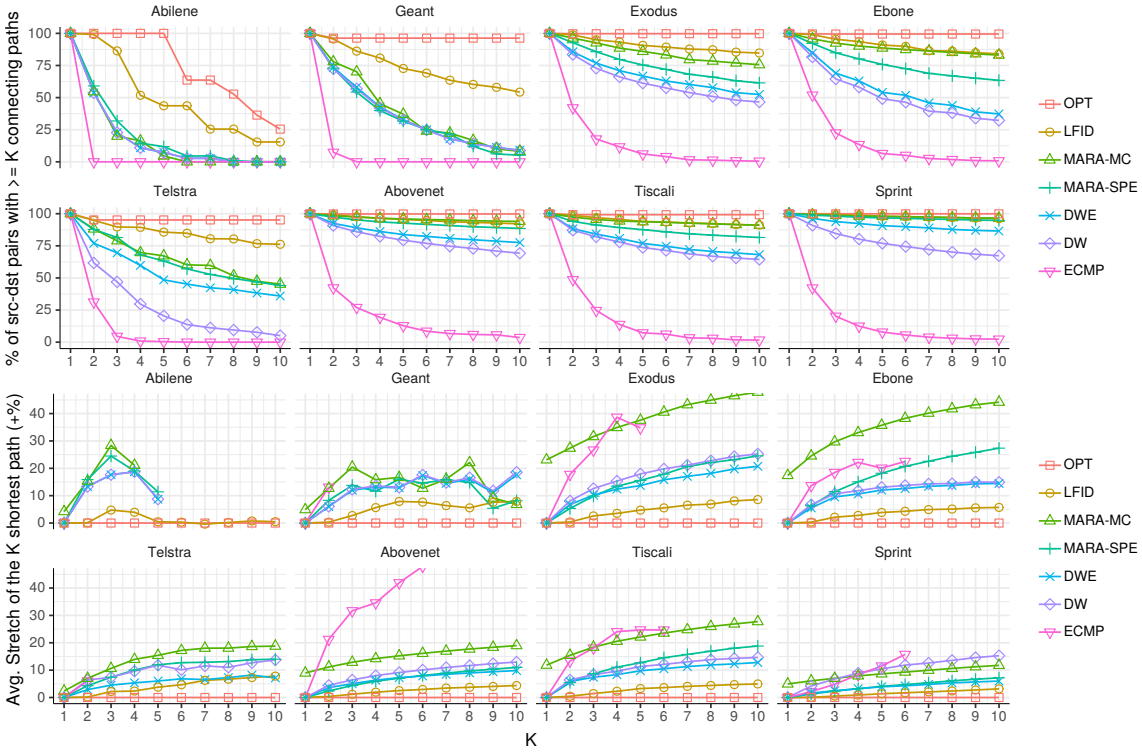


Figure 4.8: Top: Perc. of src-dst pairs that have at least K paths between them; Bottom: Avg. stretch of the K-shortest path.

4.4.2 Measured Runtime

In addition to the complexity analysis in Section 4.3.5, we measure the actual runtime of the presented algorithms. We ran these measurements on consumer-grade hardware (Intel i7-6600U CPU), single-threaded, calculating the FIB for all nodes towards all destinations. As seen in Figure 4.7, even though the worst-case complexity implies a much larger difference (roughly 917x higher for the Sprint topology), LFID is only 25% to 91% slower than MARA-SPE. This is mainly because the loop-removal step (the performance bottleneck) has a much better runtime in the average case than in the worst case (see Section 4.3.2). For larger topologies, there are further options to reduce runtime:

First, in contrast to MARA or ECMP, LFID can easily be parallelized. The

time-critical *removeLoops()* function (Alg. 4) is run once per destination. Since the outcome for each destination does not depend on another, it can be run in parallel, speeding up runtime by a factor of available CPU cores.

Second, the higher nexthop choice allows instant re-routing through an alternative path, which avoids the need for fast route recomputation during most link failures. For example, in 88.9% to 98.2% of single link failures LFID can still reach the destination by rerouting at an adjacent node (Section 4.4.4). Thus, route computation in LFID is only time-criticial in 1.8% to 11.1% of link failures, i.e., 9x–55x less frequent.

4.4.3 Number & Length of Resulting Paths

Next, we look at the number of possible paths, and their length (= sum of link costs) provided between each source-destination pair. We do this by running Yen's K-shortest (simple) path algorithm (from [Qi]) for K=10 paths. For the optimal result (OPT), we run Yen's algorithm on the *undirected graph* of the base topology. For every other scheme, we run Yen's algorithm once for each destination on a *directed graph* that represents the possible nexthops in the FIB.

Figure 4.8 shows the percentage of src-dst pairs that have at least K paths connecting them (top) and their *path stretch* (bottom), i.e., the average ratio of the K-shortest path in the directed vs. undirected topology. For K=1 it shows the path stretch of the shortest path, for K=2 the stretch of the second shortest path (if it exists), and so on. We removed any path stretch values (bottom plot) where the percentage of paths (top plot) was less than 5%, since those are likely to be outliers caused by a small sample size.

Of all the hop-by-hop routing schemes, ECMP, predictably performs the worst. In all tested topologies, more than half of source-destination pairs are connected only through a single path. Moreover, ECMP is missing many short paths, thus the average length (stretch) of the K-shortest paths is high. Downward paths (DW, DWE) are always better than ECMP, and MARA (MC and SPE) is mostly better

than downward paths. Thus, the ranking according to number of paths is roughly: OPT>LFID>MARA-MC>MARA-SPE>DWE>DW>ECMP.

In almost all topologies, LFID has a higher path choice than all related work. The exception is the Abovenet topology, where MARA-MC has a higher path choice. However, MARA-MC buys this higher number of paths with a much higher average path length (Figure 4.8 bottom). Ignoring some outliers of ECMP and DW, MARA-MC's K-shortest paths are the longest of all tested schemes – up to 50% longer on average than optimal. The main reason is that, in contrast to all other schemes, MARA-MC does not always use the shortest path (K=1). In comparison, MARA-SPE shows a lower path stretch (up to +21%), but also a significantly lower number of paths.

LFID shows that one does not have to make the trade-off between high path choice and short potential paths. In most topologies, it has the highest number of paths and also the lowest path stretch (less than +9%).

However, looking at the K-shortest paths only gives an indirect idea of how many of these paths can be used to circumvent failures and congested links. Hence, next we look more directly at the resilience towards single failures/congestions (Section 4.4.4), and at the stretch of the actual paths used by HBH multipath routing (Section 4.4.5).

4.4.4 Resilience to Single Link/Node Failures

Next, we investigate how this path choice can be used to circumvent a single link failure, link congestion, or node failure. Note that for the remaining two experiments, we can treat link failure and congestion interchangeably: The question is whether there exists another path that *avoids* a certain link (or multiple links). The result is the same for failure and congestion, thus we use the term link/node "problem" to denote both cases; we use the term link/node "protection" for the ability to circumvent both types of problem.

We consider two different ways of link/node protection: 1) Rerouting adjacent to the problem and 2) rerouting either adjacent or via backtracking to earlier nodes.

First, we check for an alternative path at the router adjacent to the failure (or congestion). This adjacent recovery is easy to implement in practice: After the primary nexthop goes down, a router simply chooses another nexthop from its set. For all tested algorithms, the resulting path is guaranteed to be loop-free and, in case of a single link failure/congestion, is guaranteed to reach the destination. It doesn't require signaling between routers, nor backtracking of packets.

Next, we consider the possibility of backtracking on a failure: If the node adjacent to the link/node problem does not have an alternative path/nexthop towards the destination, the packet can be returned to the previous node, which then checks for an alternative path. If none is found, the packet will be further backtracked, if necessary, all the way to the source. For now, we only investigate the potential link or node protection of such backtracking, and put aside its implementation complexities, such as path stretch, search cost to find a working path, or additional router state.

More specifically, for the experiment, we look at each link/node on the shortest path between each src-dst pair. We remove this link/node and record the percentage of available alternative paths either 1) from the node directly adjacent to the failure, or 2) from the source node (backtracking). Sometimes, the topology does not contain any possible paths, i.e., when source and destination are disconnected by removing a link or node. Thus, we plot the results relative to the maximum protection possible in each topology (OPT), defined as 100%.

Figure 4.9 shows the result for link (top) and node (bottom) protection, where solid lines show adjacent protection and dotted lines protection via backtracking. For both link & node failure, and in all tested topologies, LFID outperforms the other tested algorithms[2]. If it seems like other schemes perform better, this is because the

[2]With a single exception (1 out of 96 data points): For node protection, in the Abovenet topology, and without backtracking, MARA-MC performs slightly better (92.8%) than LFID (91.2%).

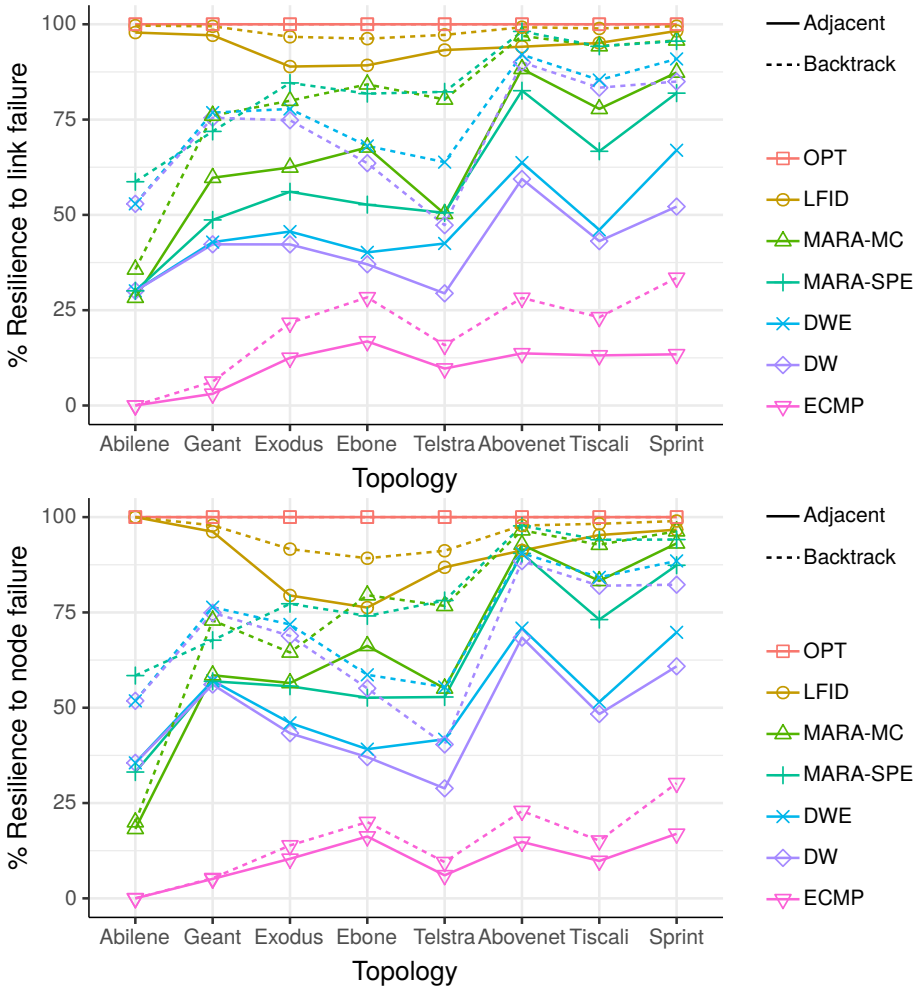


Figure 4.9: Percentage of link/node failures between each src-dst pair that can be recovered at the adjacent node (solid line) or via backtracking (dotted line).

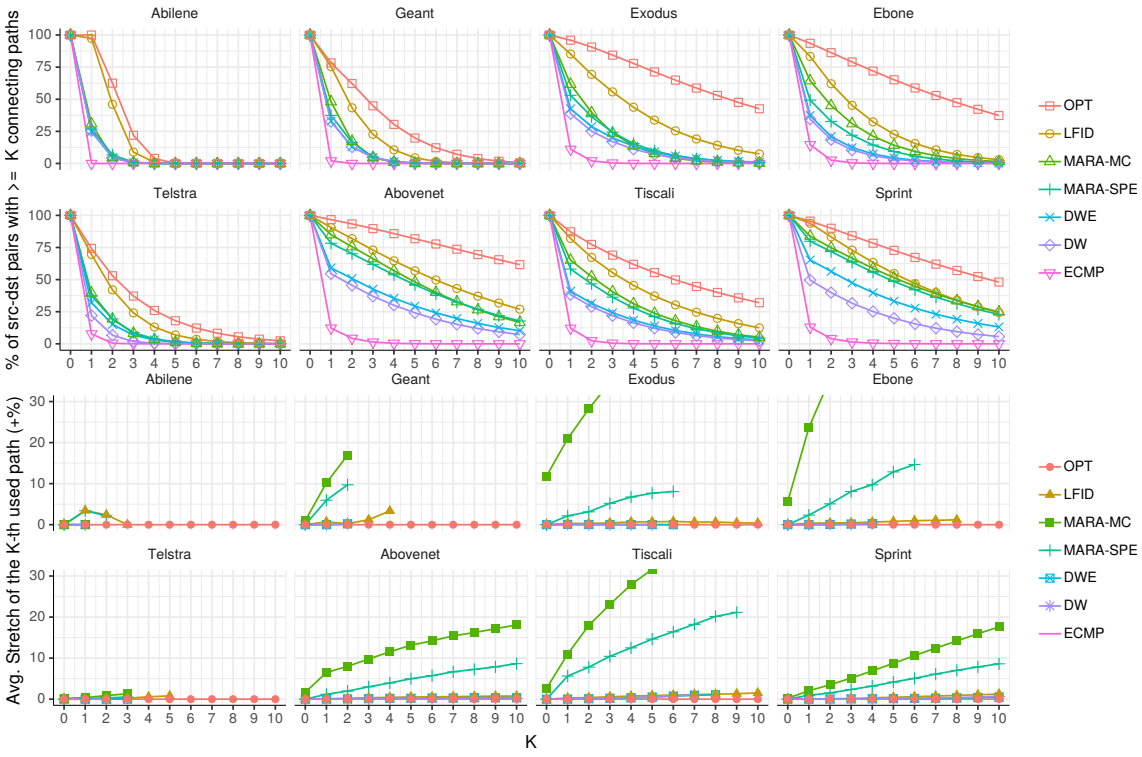


Figure 4.10: Top: Percentage of src-dst pairs that can handle K link problems (failures or congestions) by rerouting at the adjacent node. Bottom: Average Stretch of the k-th path used after rerouting.

plot shows both the adjacent and backtracking case.

LFID can handle 88.9% to 98.2% of all recoverable link failures by rerouting at the adjacent node. Moreover, most of the time[3], LFID provides better link protection *without* backtracking than the other schemes to *with* backtracking. Given the complexities of implementing a backtracking scheme (path stretch, search/probing and state overhead), this is an important result.

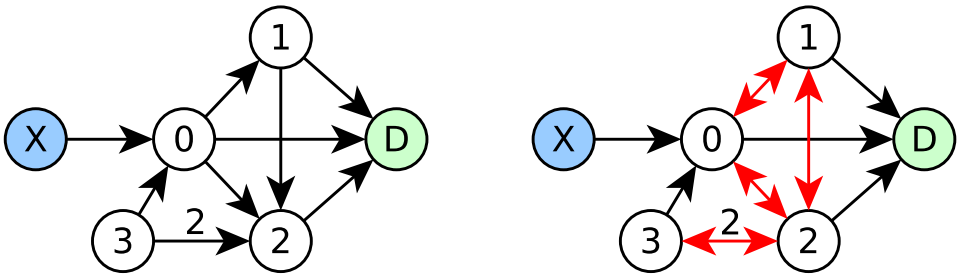
4.4.5 Resilience to Multiple Link Failures & Congestions

Lastly, we evaluate the ability to handle an arbitrary number of link problems (failures or congestions). Here we focus on *adjacent* protection of failed/congested links. We look at the shortest path between each src-dst pair. We remove a *randomly selected link* from the shortest path, then check if there exists another path towards the destination from node adjacent to the removed link (K=1). Then, on this second path, we remove another randomly selected link, and check if there exists a third path that avoids both removed links from the new adjacent node to dst (K=2). And so on, for K = number of avoided links, and $K+1$ = number of simultaneously used paths if all avoided links are due to congestion. Note that removed links are specifically chosen to affect a single src-dst pair. Removing 10 random links from a topology will have a much smaller effect on connectivity than K=10, since most of them will not be on the path between a given src-dst pair.

Moreover, we measure the average stretch of the resulting path, relative to the optimal shortest path that avoids the K removed links. We plot the average over 100 runs.

As shown in Figure 4.10, again LFID is closer to the optimum resilience than other work (except in the Sprint topology, where it is tied with MARA) providing, compared to the optimal, 69.7% to 92.5% resilience against 2 simultaneous failures,

[3]Except 5 out of 48 data points: Both MARA schemes in the Abovenet topology for both link & node protection, and MARA-MC in the Ebone topology for node protection.



(a) DAG (b) LFA

Figure 4.11: Loop-Free Alternate Rules

and 40.2% to 86.7% resilience against 3 simultaneous failures.

Regarding the path stretch, LFID (and also DW/DWE) performs much better than MARA (MC and SPE): The paths created by adjacent rerouting are on average only 1% longer than optimal!

4.5 Related Work

In addition to the multipath routing schemes discussed in Section 4.2, another class of related work is *IP Fast Rerouting* (IPFRR) [SB10], which provides alternative nexthops for local failure protection on the data plane. The main difference between IPFRR and the work discussed earlier (ECMP, DW, etc.) is that IPFRR nexthops cannot be used arbitrarily by multiple routers on a path, risking loops when doing so.

A common restriction is that after taking a backup nexthop, packets *must* follow the shortest path. Consider the example of Loop-Free Alternates (LFA) [AZ08, RTEC11] in Figure 4.11b. The alternate nexthops (marked in red) provide protection against any possible link failure, which a DAG cannot [IBTD03] achieve (here: link 2 → D is unprotected). However, routers cannot freely use LFA nexthops. If they did, packets could form a loop, e.g. 0→1→2→0 or 2→3→0→ 2, and those loops cannot be avoided by excluding the incoming port at each step. The restrictions

for packets to stay on the shortest path (more generally: use only primary nexthops) severely limits the number of possible paths and thus LFA's ability to deal with multiple simultaneous failures or congestions. The same reasoning applies to U-turn alternates [Atl06] and IPFRR Tunnels [BPS13], since those include an even larger set of alternate nexthops.

Another example of IPFRR is permutation routing with joker links [VLK13], which shares following ideas with LFID: It extends DAGs by using certain links in both directions (there called "joker links"), and also excludes the incoming link at each hop. The main difference is, again, that 1) joker links can only be used when all primary nexthops are down and 2) packets from a joker link can only be sent to a primary nexthop, otherwise risking loops. In constrast, in LFID all routers can use all nexthops simultaneously, even if no links have failed.

Other IPFRR schemes require more sophisticated changes to the IP protocol, such as per-packet state [ABK15, EGR16, RN12, LPS+13, LCR+07] or multi-topology routing [MEFV08, KGGZ11, CNM+16]. In one example of multi-topology routing, Chiesa et al. [CNM+16] decompose the routing graph into k arc-disjoint spanning trees. When incurring a link failure on its current path, a packet can switch to a different tree, which substantially increases the resilience towards failures. However, tree switching incurs a path stretch which, while acceptable for circumventing link failures, is likely unacceptable when splitting up traffic for load balancing, as this would involve many more tree switches. In contrast, LFID ensures that packets rerouted at multiple points stay close to the shortest possible path (see Section 4.4.5).

Some IPFRR schemes do, however, provide an important benefit: they maintain optimal connectivity for an arbitrary number of failures [LPS+13, YAM+13]. LFID often comes close to the optimal (see Figure 4.10), but does not reach it. We leave for future work the question, whether LFID can be combined with such a data plane mechanism (i.e., dynamic manipulation of FIB state) to ensure optimal connectivity.

CHAPTER 5

Traffic Engineering

5.1 Introduction

While conventional routing protocols (like OSPF [Moy98] or IS-IS [isi90]) send packets on the shortest path from source to destination, there are many benefits of using multiple *non-shortest paths* [HR08], most importantly *failure protection:* routing around link/node failures, and *traffic engineering:* distributing the traffic load to avoid congestion and improve user satisfaction.

A first approach to traffic engineering was to *tune the link weights* of traditional routing protocols. The weights of highly loaded links are increased, so they are less likely to be used, and their load decreases. While link-weight tuning is simple to implement, it can lead to abrupt shifts in traffic and hard-to-predict network-wide side-effects. A famous example is the early ARPANET, where dynamic changes to link weights have lead to instability and routing oscillations [KZ89b].

A way to solve these issues, is to use *source routing*, e.g., MPLS [VRC01] or Segment Routing (SR) [FPG+18]: letting the source (ingress) node determine the entire path (or tunnel) towards the destination. If a link becomes overloaded, traffic is shifted to a tunnel that avoids this link, leading to much more predictable outcomes than link-weight tuning. Though frequently used in practice, source routing does have certain downsides (see Section 5.2.2): 1) It is unclear which tunnels should be chosen to achieve a good trade-off between shorter paths and failure resilience. 2) Tunnels often have large bandwidth reservations and can only be moved as an atomic unit, which leads to sub-optimal path choices. 3) Online adaptation of tunnel reservation size (called "Auto-Bandwidth") does not directly consider congestion in

Dest.	NH	Cost	Split
LA	SV	2	100%
DV	DV	1	100%
	SV	2	0%
NYC	DV	5	60%
	SV	6	40%
...			

(a) FIB for **Seattle**

(b) Abilene Topology

Figure 5.1: Example of HBH Traffic Engineering. Depending on the destination, Seattle uses either 1 or 2 nexthops. The cost denotes the shortest path (in hops) of using a certain nexthop. The split ratio can be adapted to react to link failures or congestion.

the network, which can lead to inefficiencies.

In this work, we adopt Hop-by-Hop Traffic Engineering (HBH TE) to avoid the issues of source routing, with the goal to create an overall much simpler design. The basic idea of HBH TE is as follows: 1) A multipath routing protocol equips each router with a list of loop-free nexthops per destination, ranked by their routing distance (see Figure 5.1a). This list and ranking is fixed, i.e., will not change based on traffic load. 2) Each router maintains a split ratio (or "forwarding percentage") for each nexthop in the FIB. Routers will adapt this split ratio based on the traffic load to meet the goals of traffic engineering.

The main two questions are which nexthops to choose and how to set the split ratio? We leave the first question to prior work (discussed in Section 5.5.1). For the second, we use the goal of what we call "latency-optimality" (Section 5.3): packets should take the shortest path (by propagation delay) unless the shortest path's capacity is insufficient to meet traffic demand; then packets should take longer paths to maximize application throughput. Surprisingly, most existing work on traffic engineering chooses a different goal (Section 5.7): minimizing the maximum link

utilization or cost. These different goals lead to quite different designs. For example, while most existing work periodically measures link utilization, we only need to measure link *congestion*, that is, the state when link utilization approaches capacity.

Two obvious disadvantages of HBH TE compared to source routing TE need to be addressed. First, source routing can choose arbitrary paths, while HBH TE's paths are restricted by the requirement of loop-free hop-by-hop routing without per-packet state; that is, by which nexthops can be put in the FIB without risking loops. We show that this restricted path choice does indeed lead to sub-optimal traffic distribution, but only in rare cases, and that the omitted paths are many times longer than the shortest path (Section 5.6.3). Second, routers make decisions based on a local view of the network, as opposed to the global view achieved by source routing with a centralized controller. We show that, perhaps surprisingly, the lack of global knowledge makes little difference. By avoiding the granularity problem of bandwidth reservations together with path probing, our design comes close to the theoretical optimum (Section 5.6.3). Divergence from the optimum is caused much more by the limited path choice than by local decision making.

Our specific design, *Hop-by-Hop Congestion-Aware Traffic Engineering* (HCTE), works as follows: it starts forwarding packets on the shortest path, splits traffic only when these paths become congested (detected by ECN congestion marks), probes the congestion level of new paths before use (and only adds paths with a lower congestion level than the current), adds new paths gradually, and equalizes the congestion price (based on queue occupancy) in equilibrium (see Section 4.3).

Earlier designs of HBH TE are quite rare (we only found two of them – see Section 5.7) and not widely deployed yet. Hence, with this work, we want to make a renewed case for Hop-by-Hop Traffic Engineering. The main difference over the two existing works is our different definition of optimality (Section 5.3), and a novel way of probing the path congestion price to adapt the split ratio.

Compared to source routing, a strong selling point is that HCTE aims to be vastly

simpler: It does not require 1) estimating traffic matrices, 2) pre-selecting a good set of paths (only a good set of potential nexthops), or 3) reserving bandwidth for subsets of traffic. It aims to work out of the box for many topologies, link capacities, and traffic demand patterns. We show that it comes close to the theoretically optimal performance in terms of average completion time and user utility. Moreover, we show that loop-free HBH routing mostly finds the same paths as optimal arbitrary source routing with a global view of traffic demand and link utilization. And that the omitted paths are many times longer than the shortest path (Section 5.6).

5.2 The History of Traffic Engineering

5.2.1 Link Weight Tuning

Sending packets on the shortest path to the destination is not always the best way to use network resources. Some of these shortest paths may be congested, while other, longer paths remain underutilized. Consider Figure 5.2a, where Seattle (SE) sends traffic to Kansas City (KC) over the shortest path SE→DV→KC. If link (SE, DV) or link (DV, KC) are congested, one would want to reroute traffic over the paths SE→SV→DV→KC, or respectively SE→SV→LA→HOU→KC.

One can achieve this sort of *traffic engineering* by tuning the link-weights of conventional routing protocols (like OSPF or IS-IS) [FT00, FRT02, ICBD04], often combined with Equal-Cost Multipath (ECMP). For example, changing the weights of (SE, DV) and (DV, KC) to 2 allows ECMP to use all three paths simultaneously (Figure 4.2b).

However, using ECMP for paths of different length (= physical distance or propagation delay), comes with undesirable side-effects: The only way ECMP can use these paths is to split traffic on them *evenly*, i.e., every path receives about 33% of traffic (Figure 5.2b). As long as link utilization is below capacity, this is wasteful: some traffic that could have used the shortest path (SE→DV→KC) is needlessly

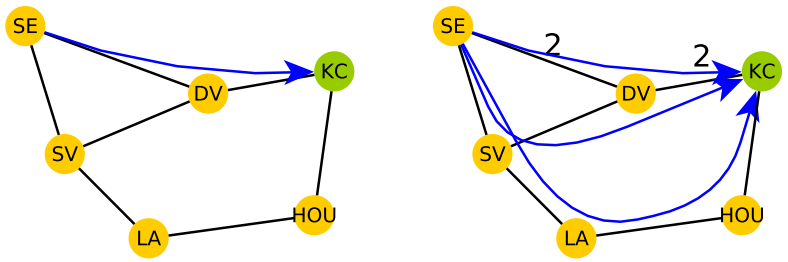


Figure 5.2: Example of ECMP + Link weight tuning. a) Base topology that allows only one equal-cost path. b) Topology with changed link weights to create three equal-cost paths.

sent over a longer path. A better approach would be to keep traffic on the shortest path until demand exceeds its capacity, and only then *gradually* switch to the longer path (see Section 5.3). However, ECMP + link weight tuning cannot achieve this, as it mandates an even split between all equal-cost paths.

ECMP + link-weight tuning has further problems: 1) in larger topologies, it is hard to find link weights that create the desired ECMP paths, 2) an equal split may not be what's needed to create equal link utilization (e.g. when links differ in their capacity), and 3) changing link-weights can have hard-to-predict network-wide side-effects, i.e., causing congestion in other parts of the network.

5.2.2 Source-Based Explicit Routing

A solution to these issues is MPLS/Segment Routing (SR) traffic engineering, or more generally, *source-based explicit routing* through tunnels between network edge routers. In source routing, ingress routers set up a number of tunnels (label-switched paths – LSPs) towards any given egress router. Each tunnel will *reserve* a specific amount of bandwidth and can be set up automatically via the Constrained Shortest Path First (CSPF) algorithm, i.e., finding the shortest path that will also satisfy the bandwidth requirement. Hence, source-based routing solves the problems of

link-weight tuning: It can arbitrarily split up traffic over multiple non-shortests paths/tunnels, without causing network-wide side-effects.

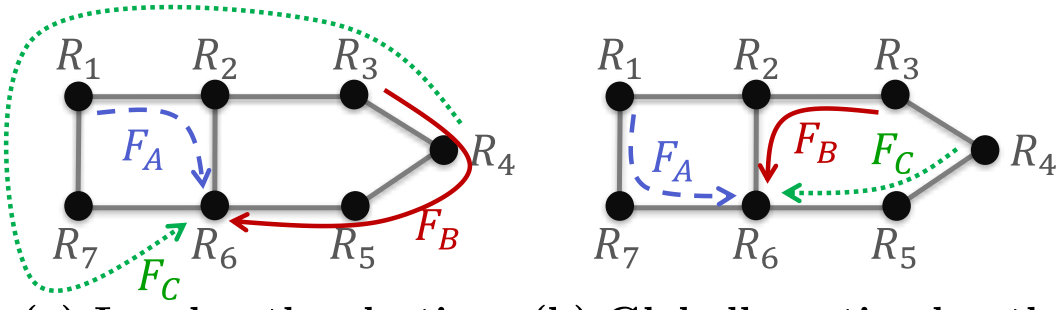
Segment Routing (SR) [FPG+18] is an important update to MPLS (using RSVP-TE [ABG+01]), as it removes the need for pre-establishing paths and maintaining state inside the network. SR puts all the state needed to route a packet into labels attached to the packet itself. However, for our discussion, the similarities between MPLS and SR are more important than the differences: forwarding decisions are made only at the network edge, as both protocols send packets on explicit paths/tunnels between ingress and egress routers. Moreover, both use LSP bandwidth reservations as means of traffic engineering.

Even though explicit source routing avoids the problems of link-weight tuning, it faces a number of operational problems itself [Ste13] [NKK12]:

Manual choice of LSPs (number & which ones) As Jennifer Rexford points out [Rex09], there is no systematic way of determining which paths should be used between two endpoints. Using the k shortest paths may not give enough disjointness to deal with failure or congestion. Using the k maximally disjoint paths may lead to paths that are unnecessarily long. As Rexford puts it, there is an "art" involved in finding k short paths that are also reasonably disjoint.

Granularity: Large tunnels don't fit into small pipes LSPs can only be moved as an atomic unit. If an LSP's bandwidth reservation is a large fraction of the bottleneck link capacity, it might not fit on the optimal path, or not fit on any path at all [Ste13]. In case the LSP does not fit, traffic will still flow, but not be accounted by MPLS bandwidth reservations, which can lead to problems like congestion later on.

Hong et al. [HKM+13] give an illustrative example of the granularity problem (Figure 5.3). Assuming that links have equal capacity and can transfer exactly



(a) Local path selection **(b) Globally optimal paths**

Figure 5.3: Inefficient routing in MPLS TE [HKM+13]

one flow, flows F_A, F_B, and F_C are subsequently added using CSPF. This greedy allocation leads to undesirable results: F_B and F_C take much longer paths then necessary (Figure 5.3a). Hong et al. attribute this problem to MPLS' "local, greedy resource allocation", which they solve via an SDN controller with global knowledge.

But the problem could just as well be solved locally, if the *granularity* constraint was removed. R_4 (locally) knows that the path over $R_4 \to R_2 \to R_1 \to R_7 \to R_6$ is suboptimal and only chooses it, because the other paths are already fully utilized. Without granularity constraints, R_4 could shift over a fraction of the traffic of F_C to path $R_5 \to R_6$, which will then cause R_3 to shift some of F_B to $R_2 \to R_6$, which will cause R_1 to shift some of F_A to $R_7 \to R_6$, slowly converging to an equilibrium of the globally optimal solution. This is indeed what happens in HCTE (see Section 5.6).

Auto-BW does not measure congestion directly There are two ways to adjust an LSP's bandwidth assignment: 1) Offline calculation and 2) "Auto-Bandwidth", meaning an online adjustment based on the observed traffic load. [Ste13, NKK12]. Since offline calculation usually acts on much slower timescales, we focus on auto-bandwidth. Auto-BW measures the traffic load over one interval (in the order of minutes), and uses the maximum observed traffic to adjusts the reservation for the next interval (see Figure 5.4). This allows Auto-BW to use shorter latency paths

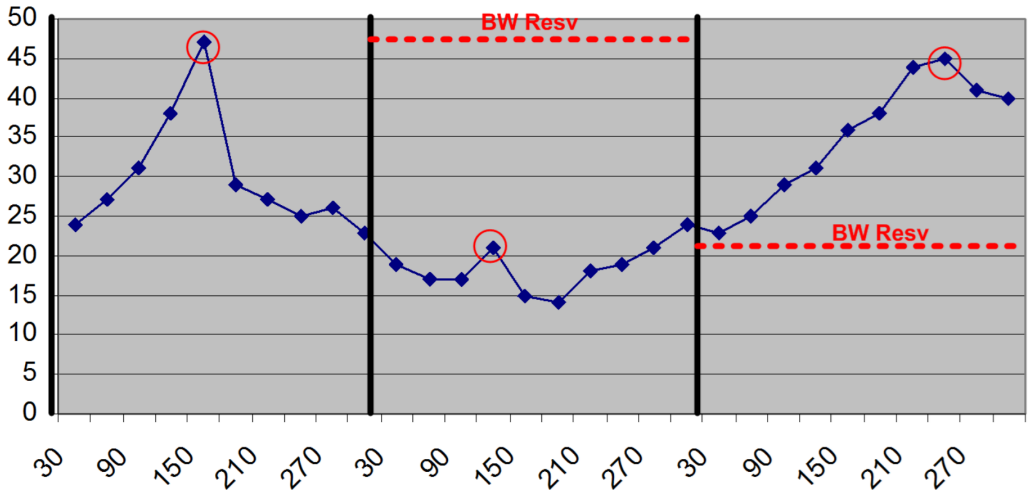


Figure 5.4: When Auto-Bandwidth Doesn't Work Well [Ste13]

first, and also react to bursts of traffic in near real-time.

However, if the traffic fluctuates quickly relative to the adjustment interval, Auto-BW may over-reserve or under-reserve bandwidth. See the example in Figure 5.4, where the adjustment interval is 300s. Over-reserving bandwidth is a problem since it might block other LSPs from using certain paths that have sufficient capacity for them. Under-reserving can lead to even worse outcomes, since it can create congestion on paths that are not seen by the Auto-BW mechanism! [Ste13]. Creating congestion will have TCP endpoints slow down, reducing the traffic demand seen by Auto-BW, which then sees no need to increase the LSPs size. This is a vital flaw of a system that is not congestion-aware.

In summary, there are many conditions where SR/MPLS TE does not lead to optimal results and either human operator intervention or proprietary solutions are necessary [Ste13].

5.3 What is Optimal?

We adopt the following definition of optimality: "In optimal routing, traffic from a source-destination pair is split at *strategic points* allowing *gradual shifting* of traffic among alternative paths. [...] For low input traffic, a source-destination pair tends to use *only one path* (which is the fastest in terms of packet transmission time), while for increasing input traffic, additional paths are used to avoid overloading the shortest path." [QAY+15] To differentiate from other definitions of optimality, we call this "latency-optimal routing" [1].

Even though latency-optimal routing makes intuitive sense, most of the traffic literature uses a different definition of optimality: minimizing the maximum link utilization or link cost [FT00, EJLW01, KKDC05, XCR07, XCR11, MTX13, MT15], called *MinMax* routing. However, MinMax routing in itself has no benefits for application performance (applications work just as well with links utilized at 60% than at 40%), and can even hurt delay-sensitive applications by using longer paths than necessary [SMKV11]. MinMax routing's fundamental problem is that it models application utility based only on throughput but not on latency, and thus optimizes for maximal headroom with no regards to path length.

Leaving some headroom is indeed valuable in order to deal with traffic spikes. As Gvozdiev et al. point out, there is a trade-off between using shorter paths (latency-optimal routing) and leaving maximal headroom (MinMax routing) [GVKH18b]. In practice, one might want to leave a defined amount of headroom (e.g. 10% of link capacity) and aim for latency-optimal routing with that constraint. See Section 4.3 for how we consider this requirement in our design.

[1] Note that this optimization goal also works for link cost choices other than latency. E.g. wireless links can be assigned a cost higher than their propagation delay, to discourage their use. However, in most cases, latency is the most sensible choice.

5.4 Hop-by-Hop Traffic Engineering

We propose a new way of traffic engineering that considers both the problems of link-weight tuning and source routing, and optimizes for latency rather than headroom. Our approach, *Hop-by-Hop Congestion-Aware Traffic Engineering* (HCTE), makes routing decisions at each *hop:* Instead of having edge routers determine the *entire path* from ingress to egress, every router inside the network determines the *nexthop* of a packet, based on the destination address, topology information from the routing protocol, and from probing of potential paths/nexthops. It uses *HBH Multipath Routing:* each router is given a set of loop-free nexthops by the routing protocol, and will set a fine-grained split ratio among them (see Figure 5.1). HCTE is also *congestion-aware:* The split ratio is based on the congestion level of the paths following the potential nexthops.

HCTE solves the problems of link-weight tuning: 1) It avoids unpredictable network-wide side-effects, since one router changing its split ratio will not cause other nodes to reroute, as can happen when changing link weights. 2) It provides "latency-optimal routing": it starts on the shortest path and only splits when necessary (paths are becoming congested). 3) It can set non-equal and fine-grained split ratios, using the available link capacity more effectively than ECMP.

HCTE also solves the problems of source routing: 1) It does not have to choose an arbitrary set of paths (e.g. k-shortest), but rather chooses from small number of nexthops at each step, which leads to an exponential number of possible paths, used on demand. 2) It does not face the same granularity issues: Split ratios can be chosen arbitrarily fine-grained. For practical purposes, it can still be useful to keep all packets of a given TCP flow on the same path (e.g. via flowlet hashing [SKK04]), but these flows are typically much smaller than MPLS/SR LSPs. 3) It is directly congestion-aware, which avoids the problem of over- or under-reserving bandwidth. Paths can no longer be blocked by underused LSPs, and traffic will only be split

when actual utilization approaches link capacity.

Mainly, HCTE aims to be vastly *simpler:* It does not require 1) estimating traffic matrices, 2) deciding on a good set of paths, or 3) reserving bandwidth for subsets of traffic. It aims to work out of the box for many topologies, link capacities, and traffic demands, with close to optimal (by our definition) performance. HCTE uses only *one main parameter:* the amount (percentage) of adjustment of the forwarding split ratio per probing interval. All the other parameters have intuitive default values. The probing interval, for example, should be on the order of the maximum possible round-trip time of the network.

5.5 Design Specifics

Following our earlier definition of "latency-optimal routing", we start out by sending all traffic on the shortest path (= lowest propagation delay). HCTE then has to answer the following questions:

1. When to start splitting traffic? What triggers traffic to be sent on paths other then the shortest path?

2. Where to split traffic? Which router should split and how to coordinate routers?

3. What determines the equilibrium split ratio?

Splitting traffic away from the shortest path creates a trade-off between using shorter paths and achieving higher throughput through longer paths. *When to split* is thus intuitively answered: whenever the shortest path capacity is insufficient to meet demand. It also suggest an *equilibrium split ratio:* as long as demand is not met, try to equalize the congestion price on all paths, which will maximize throughput. *Where to split* is als crucial, since it is possible to choose a longer path that goes back to the same shared bottleneck link, increasing latency without adding throughput. Consider Figure 5.5, and the shortest path SE→DV→KC→IN where link (KC, IN)

is the bottleneck. If SE shifts traffic to its nexthop SV, the second shortest path would be SE→SV→DV→KC→IN, still using the same bottleneck link! This problem of *shared bottlenecks* is especially pressing for Hop-by-Hop traffic engineering, since routers don't determine the full path of packets. Our solution is to only split traffic close enough to the bottleneck link, so that traffic on the longer path can never return to that link. Hence, a quick summary of HCTE is as follows. For each destination in the FIB:

1. Start on the shortest path.
2. When a link gets congested: *gradually* shift traffic away from this link at the router *directly adjacent,* or if not possible, at the *closest possible* router. This avoids the problem of returning to a shared bottleneck.
3. Before splitting traffic, probe alternative nexthops/paths to make sure they have free capacity.
4. In equilibrium: try to equalize the congestion price of all active paths.
5. If demand falls below link capacity: Gradually shift traffic back to the shortest path.

5.5.1 Loop-Free Multipath Routing

We use a multipath link-state routing protocol to fill the FIB with viable nexthops, sorted by their cost (see Figure 4.1a). Cost here is defined as the distance of the shortest path to the destination through this particular nexthop. We set the link metric to the propagation delay. Since in modern high-speed routers (assuming functional traffic engineering), propagation delay vastly exceeds queuing delay and processing delay, this allows routers to have a basic estimate of the latency of the shortest path through each of their nexthops towards the destination. This latency

can be computed efficiently and locally by running Dijkstra's shortest algorithm not just for the node itself, but also once for every neighbor [YW06].

The multipath routing protocol should calculate which nexthops to put in the FIB, with the constraint to 1) avoid loops and 2) allow as many and as diverse paths as possible. An obvious solution is to use only "Downward Paths", that is, nexthops that lead closer to the destination, as measured by the link metric. Another option is MARA [OIVM09] which creates a directed acyclic graph per destination. However, we use our earlier work, LFID (see Chapter 4) [SZB19], since it allows a greater number of nexthops.

5.5.2 Congestion Price & Active Queue Management

We answer the question *When to start splitting traffic?* with "as soon as links on the shortest path start getting congested". We measure this congestion directly at each router queue via Active Queue Management (AQM). The idea of AQM schemes like Random Early Detection (RED) [FJ93] is to drop or mark packets early to signal TCP endpoints to slow down before the queue buffer overflows. This can be viewed as charging a link *price* that is a function of the queuing delay, which in itself is determined by the link utilization. For most utilizations the price will be 0, but will sharply increase once utilization approaches link capacity, that is, the time the AQM starts marking/dropping packets. When using AQM in the marking mode, it will set a 1-bit ECN mark in the packets, to which TCP will react by reducing the sending rate. The ratio of marked to unmarked packets can be interpreted as an implicit signal of the congestion price [ALLY01].

For our design, we use both the implicit price of ECN marks and an explicit congestion price retrieved via probing. Traffic splitting will start once a router receives the first marked packet for a given destination node (ECN packets are carried back on TCP Acknowledgements). It will then start probing this destination to gain a more accurate explicit congestion price for each of its nexthops. We calculate the

congestion price as an exponential moving average (EMA) of packets marked by the AQM per time interval. We chose the smoothing factor α to model a simple moving average over an interval of 5 seconds, with the well-known approximation $\alpha = 2/(N+1)$ (e.g., N = 5000, with one EMA update per millisecond). The length of this interval (here 5 seconds) does not need to be tuned exactly, since we are comparing *relative* prices of paths, rather than absolute ones.

Setting the price to ECN marks per second has the benefit of leaving a wide choice of AQM algorithms to use. We chose the CoDel AQM [NJ12b] for its simplicity and 0-parameter design, but it is easy to swap out, should a better AQM become available.

One may want to split traffic earlier than AQMs signal congestion, in order to leave some headroom for traffic spikes. We suggest two ways to do so in HCTE: First, one could simply calculate the price based on the current link utilization, using a convex price function that sharply increases at the desired level of headroom (e.g. 90% of link utilization). Second, one can use an AQM scheme with a *virtual queue*, which simulates how many packets would be marked/dropped for a (virtual) link of a smaller capacity [KS01]. Both of these ideas require to add a new 1-bit field to packets that indicates congestion to routers (to start initial probing), but not to TCP endpoints.

5.5.3 Probing Paths & Shifting Traffic

After receiving the first congestion mark, routers will probe new paths before using them. Specifically, after the router adjacent to the congested link receives the first congestion mark, it sends out an *initial* probing request packet to all (or a predetermined number of) its nexthops (from there, the probing requests will take the shortest path to the destination). To reduce overhead, this initial probing can be limited in frequency; we limit it to once per 10 seconds. The probing packets will travel on the *primary path*, that is, the path with the highest split ratio (by default:

the shortest path) to the destination, where they are answered by probing replies in the reverse directions. For every link that these probing packets traverse, the AQM will add its current congestion price to the packet. Hence the router that sent the original probe will receive the sum of the prices for the primary paths through all of its nexthops.

After receiving the price for all nexthops, they are sorted in ascending order, first by their congestion price, and on a tie by their routing distance. The first in the list is called *minPriceNexthop*, the last *maxPriceNexthop*. A router will then shift a small percentage (by default we use 0.1%) of traffic from the maxPriceNexthop towards the minPriceNexthop. If there are multiple nexthops with a price of 0 (a common case), the one with the shortest distance is chosen as minPriceNexthop. Hence, if all nexthops have a price of 0 (common when demand falls below capacity), traffic shifts back to the shortest path.

If the minPriceNexthop had not been used earlier, it will now be added to the set of *active* nexthops. We define as active all nexthops with a forwarding percentage above 0%.

If there is more than one active path/nexthop, all active paths will be probed *periodically*. The probing interval should be set at least as high as the maximum RTT in the network; by default we use 200 ms. The split ratio adjustment works as described above, shifting from minPriceNexthop to maxPriceNexthop. The periodic probing interval (200 ms) together with the adjustment per interval (0.1%) can be used to calculate the maximal convergence speed to shift one nexthop from 0% to 100%: 200 seconds. We found this to be an acceptable convergence time (lower than what's typical in MPLS Auto-BW), which leads to very little oscillation of the split ratio (<3%) in equilibrium state. These 3% are per active destination node, and having traffic of multiple destinations at one router will smooth traffic out further. The amount of adjustment per interval can be tuned further to achieve a different trade-off between convergence speed and oscillation in equilibrium.

Combining rare (once/10s) initial probing of all nexthops with more frequent periodic probing of active nexthops will *gradually add paths/nexthops* (from short & low congestion to long & high congestion) until all of their congestion prices exceed 0, meaning their bottleneck link utilization is at capacity. In the equilibrium state all active nexthops will have an *equal congestion price*. Nexthops that have a higher congestion price than this equilibrium (e.g. are used by a larger number of sources with different destination) will not be used for this destination. This will maximize the aggregate throughput, as new paths with free capacity (or lower congestion) are added, but more congested paths are avoided.

In most cases, probing and splitting will happen only at the router *directly adjacent* to the congested link. This router will mark the packet with a 1-bit flag as "handled", so that routers further back in the network will ignore the set congestion mark. One exception is when this router only has a single viable nexthop towards the destination, hence cannot perform any splitting: then it will not mark the packet as handled, and a router further back will do the splitting. Only reacting directly adjacent (or as close as possible) to the congested link has a vital benefit: it avoids unnecessary splitting over paths that still use this link. Consider the Abilene topology in Figure 5.5. If link (KC, IN) is congested, splitting only at node KC will ensure that whatever second path is used, it will never include link (KC, IN). When splitting at a node further back, say SV, one could end up with the new path SV→LA→HOU→KC→IN, still of them using the congested link (KC, IN) !

The total traffic overhead of probing is very small. Probe requests and replies need to include an ID of the probing origin (2 Byte), of the destination (2 Byte), and for the congestion price (4 Byte float). Even combined with L2 and IPv6 headers, the packets will be less than 100 Bytes in size and sent once every 200 ms per *active* destination (probing will stop if a router observes no traffic for a given destination). Even if a router sees traffic towards 100 active destination nodes, the total probing traffic will be less than 400 Kilobits/s, negligible compared to modern link capacities.

5.5.4 Fairness & Competing Traffic

Since the split ratio is determined per destination, often traffic for multiple destinations will compete for the same link capacity. For this case, should we enforce some notion of fairness between multiple destinations? Make sure that every destination gets at least some of the capacity?

The answer is no. We think it is best to leave fairness up to the endpoints. TCP endpoints already enforce a fair sharing of bandwidth and have a much more accurate knowledge of the traffic demand. For example, one destination might be shared by 100 TCP flows, while another contains only one flow. Leaving the outcome up to TCP results in a much fairer outcome than enforcing an equal amount of capacity per destination.

There are known problems with leaving fairness to endpoints, e.g., it is very easy to cheat by being less responsive to congestion [Bri07], but these problems also exist in current networks and are out of scope for this work.

5.6 Evaluation

5.6.1 Implementation

For the implementation of HCTE, we chose Named Data Networking (NDN) [ZAB+14], since it has a mature codebase and built-in support for multipath forwarding at each hop. However, we do not use any NDN features other than those described earlier in this work: a FIB with multiple nexthops per destination that can store some state (the split ratio and probing information) for each of these nexthops. Thus, HCTE can be implemented on IP routers by adding only these features.

We evaluate HCTE against related work, using the ns-3 network simulator [HLR+08] with the ndnSIM module [AMZ+12].

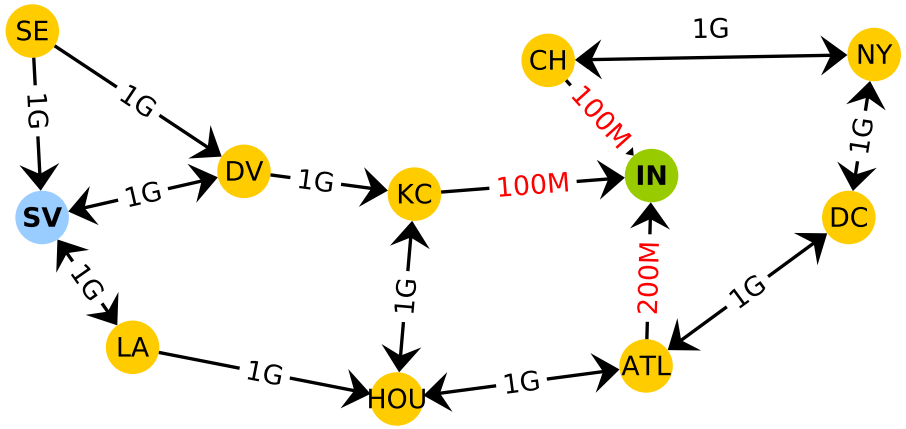


Figure 5.5: Abilene Topology with routing for destination IN

5.6.2 Forwarding Adjustment & Failure Resilience

First, we use a small scenario to show the basic functionality of HCTE over time. Specifically, we show its behavior for low traffic demand, high demand, and link failures. We use the well-known Abilene topology (Figure 5.5) with traffic from Sunnyvale (SV) to Indianapolis (IN). The arrows in Figure 5.5 show the viable nexthops, as computed by the multipath routing protocol. We calculate the link propagation delays based on the geographical distance (obtained from the dataset of the Internet Topology Zoo [KNF+11]) with a packet speed of 0.6c, which gives results closer to real-world RTTs than using the full speed of light. The resulting one-way link delays range from 1.46 ms (CH, IN) to 12.3 ms (LA, HOU). These delays are also used as link weights, which determine the initial shortest path. The link capacities (100 Mbps to 1 Gbps) are chosen so that the bottleneck will be at the three links adjacent to Indianapolis.

This scenario includes three distinct parts:

- 0-50s: The application demand (~30 Mbps) is below the capacity of the shortest path.
- 50-300s: The demand rises above the SP capacity.

- 300-450: Link (ATL, IN) has failed.

The results are shown in Figure 5.6. First, as long as the demand is below link capacity, all packets stay on the shortest path SV→DV→KC→IN.

As soon as the demand exceeds capacity, router KC will see congestion marks from IN, start probing nexthop HOU, and then start shifting over traffic to a second path: KC→HOU→ATL→IN until the KC's split ratio reaches about 66% towards nexthop HOU (based on the difference in bottleneck capacity of the two paths). At that point, ATL will see congestion from IN, probe its neighbors, and start splitting traffic towards a third path ATL→DC→NY→CH→IN. The final split ratios will be KC ⇒ {IN: 25%, HOU: 75%}; ATL ⇒ {IN: 66%, DC: 33%}. This is the optimal for this scenario, since the application throughput approaches the combined capacity of all three bottleneck links (400 Mbps). The average RTT increases corresponding to the longer paths that are used.

At 300s, the link (ATL, IN) fails, and ATL immediately shifts all of its traffic to nexthop DC. Afterwards, KC will re-balance its forwarding split ratio to match the new optimum of {IN: 50%, HOU: 50%}.

5.6.3 Traffic Split in Equilibrium

Next, we evaluate HCTE's equilibrium traffic distribution for a larger, more diverse topology. Following our definition of latency-optimal routing, we need to evaluate two cases: 1) When traffic demand stays below link capacity, HCTE should only choose the shortest path. 2) When traffic demand exceeds capacity, HCTE should add longer paths, maximizing the aggregate throughput or the sum of user utilities.

The first case is obviously met, since traffic stays on the shortest path unless congestion marks are observed, implying that demand has exceeded link capacity. Hence, we focus our evaluation on the second case, and indeed use an *infinite* demand between each source-destination pair.

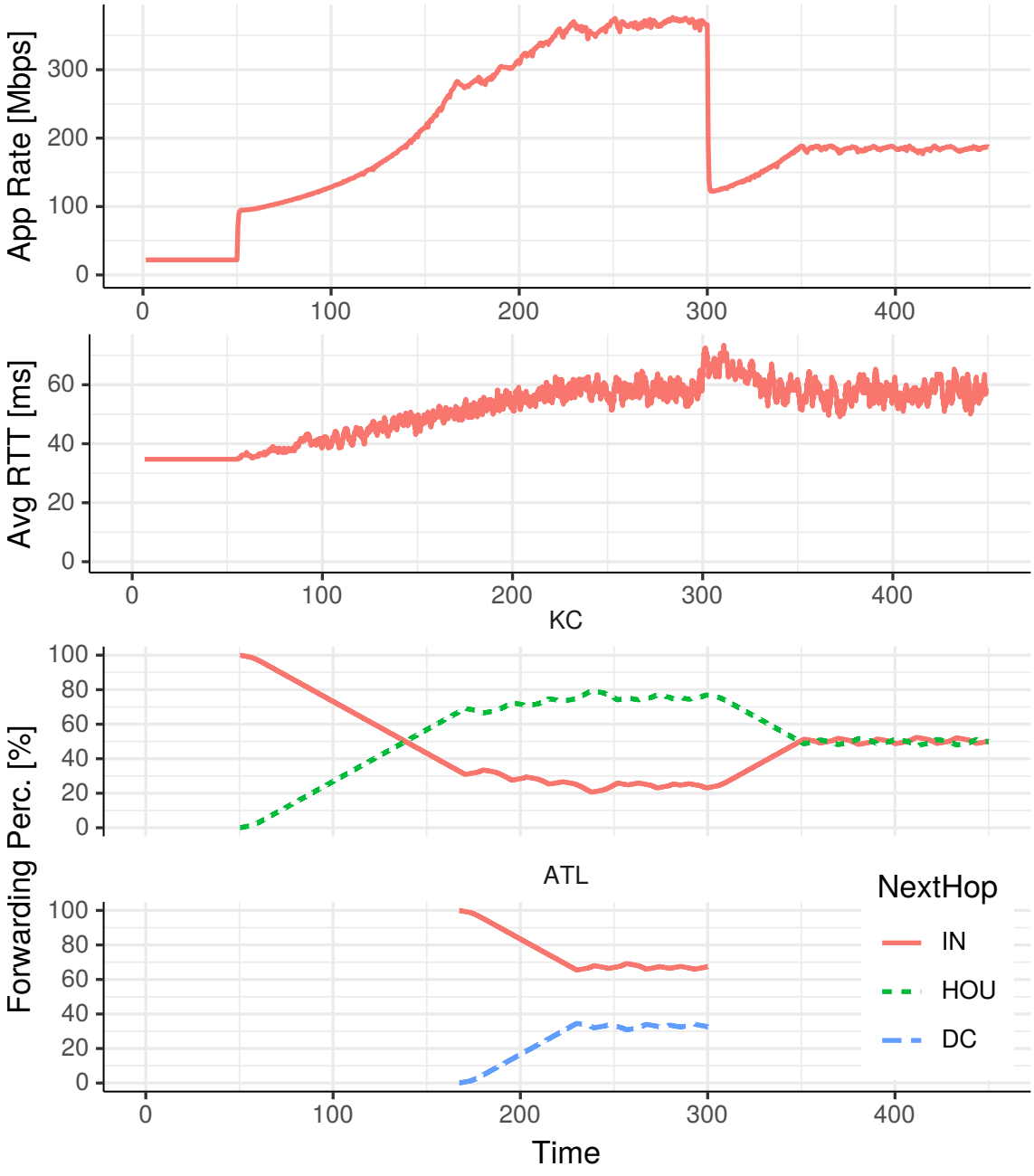


Figure 5.6: Results for the Abilene Topology

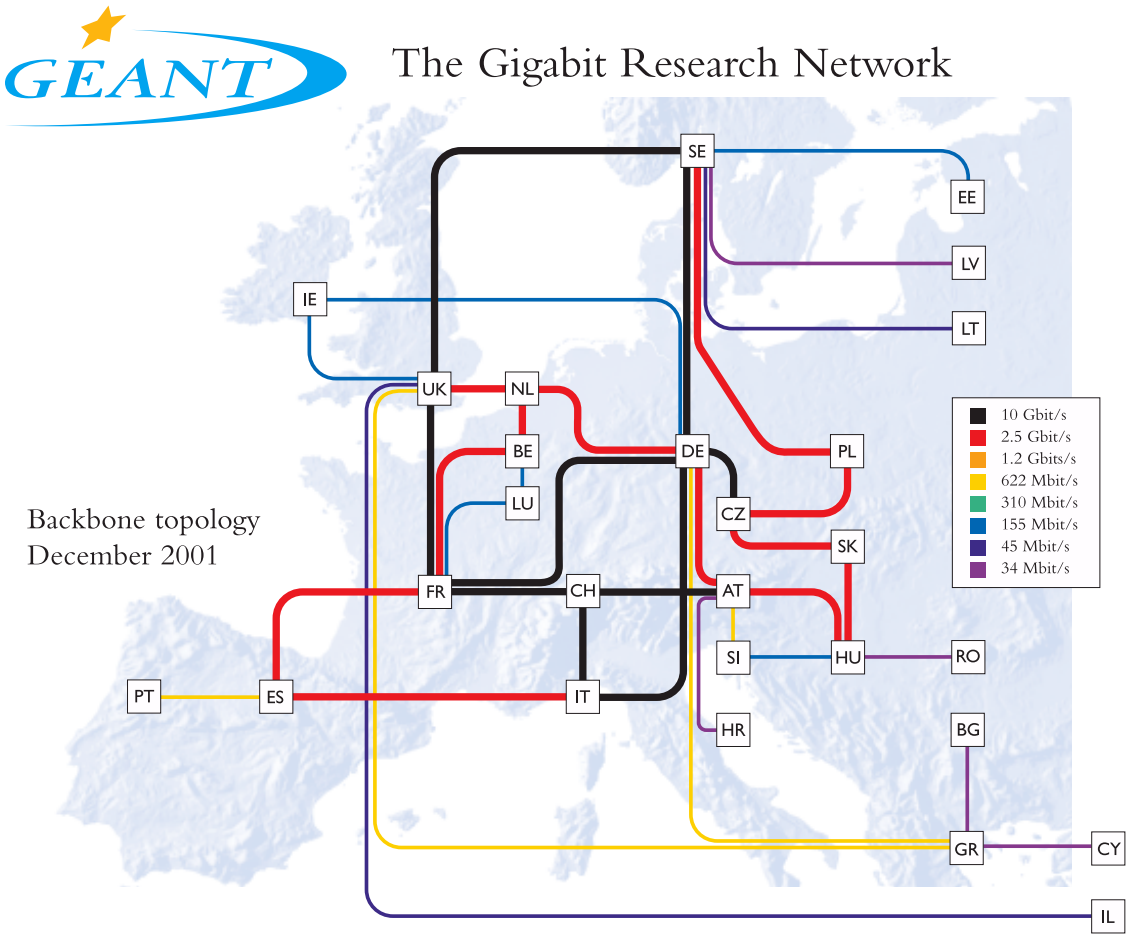


Figure 5.7: Geant Backbone Topology 2001 [Gea01]

We use the Geant topology (Figure 5.7) since it is larger than the Abilene topology, has a higher average node degree, and also a large range of link capacities (from 34 Mbit/s to 10 Gbit/s). We calculate the link delays based on geographical distance (similar to above), which range from 0.90 ms (SK, HU) to 20.0 ms (IL, UK). We compare HCTE with related work from both IP and NDN multipath routing/traffic engineering:

- **SP:** Shortest path routing with link weights set to the propagation delay.
- **InvCap:** Shortest path routing with link weights set inversely to the link capacity. This is a simple and very common heuristic to distribute the link load.
- **PI:** Setting the forwarding split ratio in order to equalize the number of pending Interest packets (= content chunk requests) at each router. This is similar to the forwarding adaptation in [CGM16].
- **HCTE:** Our algorithm, as described in this work.
- **OPT:** The optimal result, obtained from solving the following optimization problem in *Matlab*:

$$\max_{\mathbf{x},\mathbf{R}} \quad \sum_{i\in\mathcal{U}} U_i(x_i) - \gamma \sum_{l\in\mathcal{L}} D_l(f_l) \tag{5.1}$$

$$\text{s.t.} \quad \mathbf{Rx} = \mathbf{f} \preceq \mathbf{c}, \quad \mathbf{x} \succeq \mathbf{0}, \tag{5.2}$$

  Where the utility $U_i(x) = -\frac{1}{x}$ is based on throughput x for each application i, and $D_l(f_l)$ represents the incurred link propagation delay (see details in Appendix 5.8). By choosing a small positive value for γ, the solution will maximize the utility with the shortest possible paths.

  Note that this optimum is usually not achievable via loop-free hop-by-hop multipath routing (the optimal solution can include paths outside the restrictions

imposed by "downward paths" [YW06] or LFID [SZB19]), and serves to show the theoretical limit of the other schemes.

To compare the routing schemes, for each run we randomly distribute a number of source-destination pairs among the topology. We exclude the nodes that have only one link (BG, CY, IL, PT, HR, RO, LT, LV, EE), since this link is usually the bottleneck (capacities of 34 to 622 Mbps) and thus routing will not make a difference.

We put a number of TCP Cubic-like applications (= 10) at each source node to retrieve packets from the producer application at the destination node. These apps will create infinite demand, that is, fill up the available link capacity. We run two scenarios (1 & 5 src-dst pairs), with 20 runs each. We chose the number of src-dst pairs quite low ($\leq$5), since with an increasing number, the optimal solution (and HCTE) will approximate shortest path routing. If every node requests content with infinite demand, there is little use of routing on longer paths, since those paths will have little free capacity. In reality, it is unlikely that more than a few src-dst pairs will exceed their shortest path capacity. So you can think of the src-dst pairs here as modeling only those high-demand pairs.

We measure the total throughput, average completion time of a 10 Megabyte file, and average RTT over all used paths. Note that minimizing the average completion time (file size / throughput) is equivalent to maximizing the sum of user utility for utility function of $U(x) = -\frac{1}{x}$, where x is the throughput. Thus, this utility function is also called *minimum potential delay fairness* [KS03, Flo08]. We chose the average completion time as the main metric, since 1) it takes fairness into account (one starved application barely changes total throughput, but increases avg. completion time to infinity), and 2) is what TCP implicitly optimizes for [KMT98, Low03]. Hence, we use the same optimization goal for OPT.

The results in Figures 5.8 and 5.9 show a boxplot of 20 runs (median, hinges = 25th & 75th percentiles, whiskers = 1.5*inter-quartile range), with the points showing outliers. Note that some of the outliers of the completion time (mostly SP

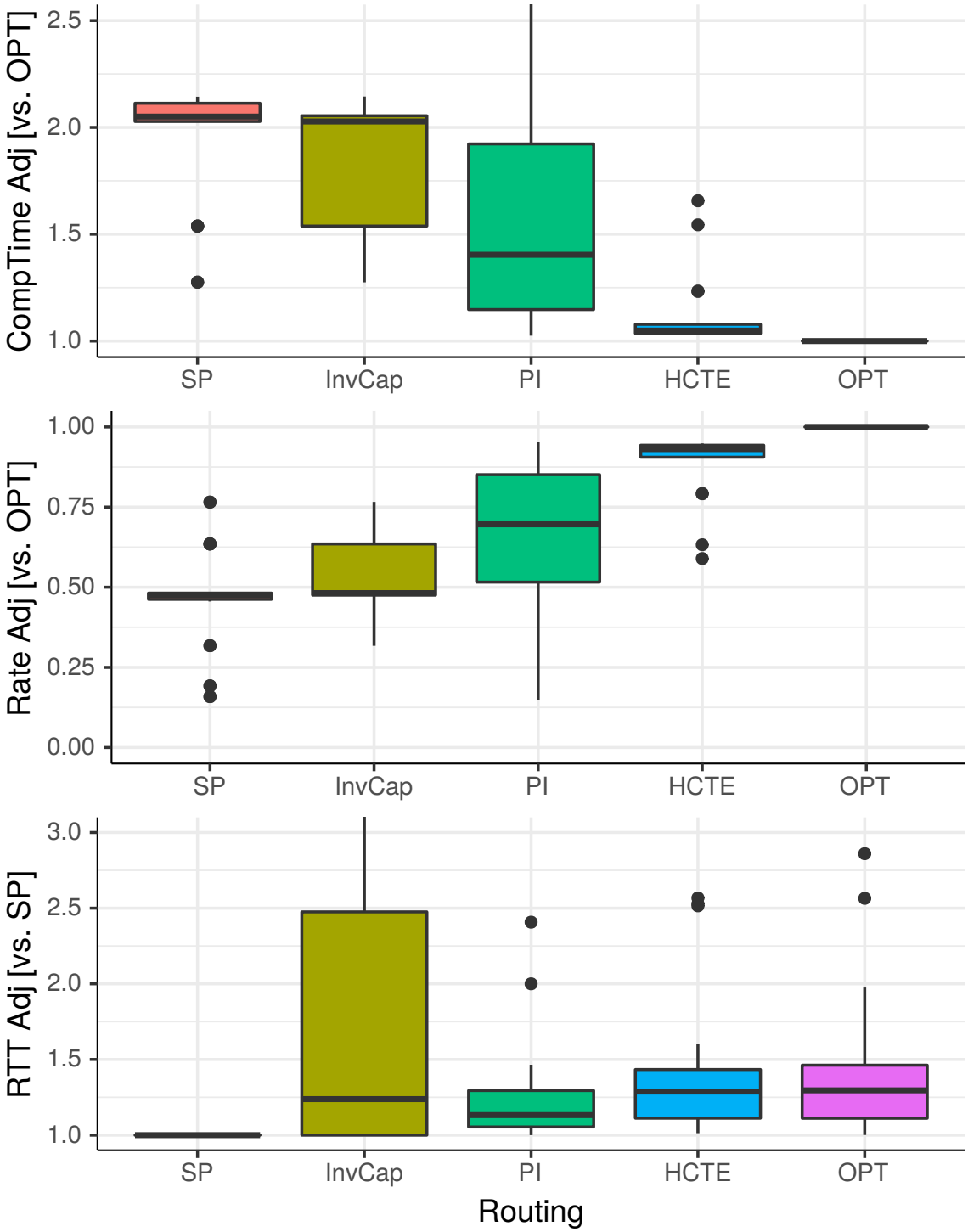


Figure 5.8: Results for 1 randomly distributed source-destination pairs (20 runs).

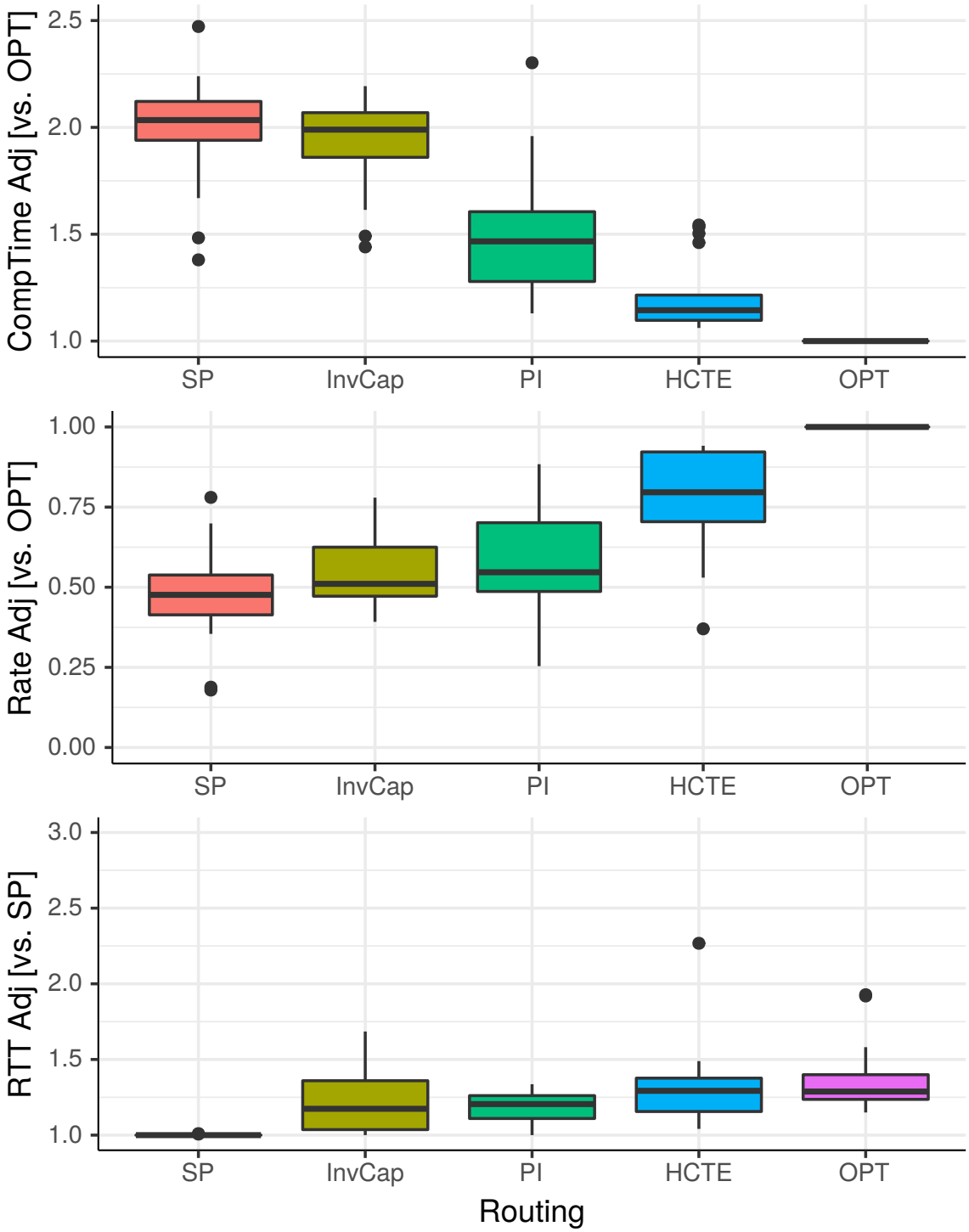


Figure 5.9: Results for 5 randomly distributed source-destination pairs (20 runs).

and InvCap) have been cut off from the top of the plot.

It may be insightful to discuss the three outliers of HCTE in Figure 5.8. The one with the highest relative completion time (1.65x of OPT) is sending traffic from CH to UK. It will use the shortest path CH→FR→UK and a second slightly longer one CH→AT→DE→NL→UK (roughly 2.5x as long). However, due to restrictions to get loop-free HBH routing, DE cannot use the nexthop SE (the path over SE→UK has very high BW), since packets could take the path SE→PL→CZ→DE, causing a loop. Also, if this path (CH→AT→DE→SE→UK) was available, it would be 5.28x longer than the shortest path. The second highest outlier is traffic from NL to FR. NL can use the shortest path nexthop over UK or a slightly longer (1.6x) path over BE. However, NL cannot choose the path over DE (3.33x longer), since it may cause the loop DE→IE→UK→NL. The third outlier is similarly caused by restrictions in loop-free HBH routing. These outliers show that sometimes HCTE diverges from the optimal. However, they are rare and only very high-delay paths (at least 3x longer than shortest path) are omitted.

In general, for both scenarios (1 and 5 src-dst pairs), HCTE clearly outcompetes the related work in terms of completion time. The median completion time (over 20 runs) is only 6% or respectively 14% higher than the optimal. Note that there are multiple overheads in the ndnSIM evaluation that don't exist in the Matlab calculation of the optimal result: the size of packet headers, probing traffic, and fluctuations in both TCP rate adjustment and router split ratio adjustment.

The total throughput of HCTE is less close to the optimal (median of 93.1% and 79.6% respectively), but still higher than related work. For the average RTT, the optimal solution is higher than either PI or HCTE, since it uses some longer paths that are excluded from Hop-by-hop routing.

5.7 Related Work

One kind of related work is Multipath TCP (MPTCP) [WRGH11], a *transport layer* protocol which runs on end devices to simultaneously multiple access networks, e.g., a smart phone connecting to both WiFi and a cellular network. MPTCP only determines the first hop and routing is left to the corresponding access networks. Thus, MPTCP paths often show significant overlap. In contrast, HCTE works at the *network layer* and does influence routing decisions. One interesting aspect is how both schemes deal with *shared bottlenecks.* In MPTCP, two subflows may share the same bottleneck link and hence would achieve twice the throughput as one competing TCP flow. MPTCP solves this issue by making each subflow proportionally less aggressive than TCP. In HCTE, we instead avoid the problem of shared bottlenecks by rerouting only at those bottleneck links. Since traffic is only split directly adjacent to (or as close a possible to) a congested link (see Section 4.3), rerouted flows will never return to the same bottleneck link.

Two other related works are DEFT and PEFT [XCR07, XCR11], which share our approach of using a link congestion price and hop-by-hop forwarding. The main difference is that DEFT/PEFT uses a fixed exponential split ratio at routers and manipulates link weights. HCTE uses fixed link weights (at the propagation delay) and manipulates the split ratio. We believe that our approach is less vulnerable to the global side-effects of tuning link-weights. Moreover, the fixed exponential split ratio means that traffic will be split even if the shortest path has sufficient capacity, violating our definition of "latency-optimal routing" (see Section 5.3).

Some related works are close to the generic source-routing discussed in Section 5.2.2: TeXCP, MATE, DATE, and COPE. TeXCP [KKDC05] is a distributed (among ingress routers), online traffic engineering scheme that uses MPLS to balance the load among multiple end-to-end paths. It considers potential traffic oscillations when multiple ingress routers adjust to congestion on a single link. TeXCP solves this by

using explicit feedback to assure the traffic increase on both paths will not overshoot the link capacity. Our solution to the oscillation problem is quite different: Only a single entity, the router closest to the congestion, will adjust the split ratio. This reduces oscillations, since one of the main issues is multiple independent entities making the same decision at the same time. Moreover, the router adjacent to congestion has a much quicker feedback loop than edge routers, which may further reduce oscillations. TeXCP mostly has the same issues as the generic source-routing (Section 5.2.2). It uses MinMax routing as optimization goal, which the authors admit "may sometimes increase the delay" [KKDC05]. Their solution is to have the operator manually limit the possible path choices: "Clearly the ISP should not pick a path that goes to LA via Europe" [KKDC05]. In contrast, our traffic engineering solution requires no such intuition or common sense from the operator. Paths will be the shortest possible by default, and longer paths are only considered when demand exceeds the capacity of shorter ones.

MATE, DATE, and COPE are similar to TeXCP in many regards, so we describe only the differences. MATE [EJLW01] is different in that 1) it minimizes a cost function based on a model of queuing delay rather than minimizing maximum link utilization. 2) MATE assumes that ingress nodes have instantaneous knowledge of the whole network state, while TeXCP is fully distributed [KKDC05]. COPE [WXQ+06] provides a worst-case performance guarantee under all traffic demands. DATE [HBCR07] explicitly considers congestion prices and tries to maximize user utility (rather than minimizing link utilization or cost), which is similar to our approach.

Lastly, we discuss two works that are more closely related to HCTE. HALO [MTX13, MT15] (Hop-by-Hop Adaptive Link-state Optimal Routing) also splits traffic at each router and is "adaptive", i.e., does not require to estimate the traffic demand matrix. However, like the work discussed above, the optimization goal is still MinMax routing, which does not account for a path's propagation delay, thus

often sends packets on paths longer than necessary.

Most similar to our work is the work by Kvalbein et al. called "Routing Homeostasis" (RH) [KDM09]. RH uses a distributed hop-by-hop approach, where individual routers decide on their nexthops. It has the same optimality goal as HCTE, of preferring low-latency paths, and it doesn't relying on estimation of traffic matrices. The main differences are that RH works completely local without a way of probing path congestion: routers adjust the split ratio based only on the utilization of their adjacent links, without regard to the path further down in the network. This can lead to congested links further along the path which are not considered, but which would be detected by our path probing mechanism. Kvalbein et al. also currently do not have any packet-level simulator implementation of their scheme, which made it hard to compare against.

5.8 Appendix – Optimization Problem

We consider a multi-commodity flow (MCF) problem that is well-known in the literature [WLLD05, LS06], where a network is modeled as a set of directional links, denoted by $\mathcal{L}$, with finite capacities $\boldsymbol{c} = (c_l, l \in \mathcal{L})$ and propagation delays $\boldsymbol{d} = (d_l, l \in \mathcal{L})$, shared by a set of source-destination pairs $\mathcal{U}$. Let $\mathbf{x} = (x_i, i \in \mathcal{U})$ denote the flow vector and $\mathbf{R} = [R_{li}]$ the routing matrix that obeys the flow conservation rules, i.e., x_i is the data flow of pair $i \in \mathcal{U}$ and R_{li} the fraction of i's flow along link l. The objective is a joint user performance maximization and network cost minimization:

$$\max_{\mathbf{x},\mathbf{R}} \quad \sum_{i \in \mathcal{U}} U_i(x_i) - \gamma \sum_{l \in \mathcal{L}} D_l(f_l) \tag{5.3}$$

$$\text{s.t.} \quad \mathbf{R}\mathbf{x} = \mathbf{f} \preceq \mathbf{c}, \quad \mathbf{x} \succeq \mathbf{0}, \tag{5.4}$$

Where the network cost is represented by the network-wide flow delay cost $D_l(f_l) = d_l * f_l$, where f_l is the flow through link l. Assuming that the user utilities $U_i : \mathbb{R}_+ \to \mathbb{R}$ are concave and the link cost functions $D_l : [0, c_l) \to \mathbb{R}$ are convex, this is a MCF problem with linear constraints and convex objective, and thus can be solved efficiently by various numerical packages.

We chose a utility function inversely proportional to the throughput, since it is implicitly achieved by TCP congestion control [KMT98, Low03]: $U_i(x) = -\frac{1}{x}, \quad \forall i \in \mathcal{U}$

CHAPTER 6

Conclusions

When Named Data Networking (NDN) can improve the efficiency of the current Internet, it is in large parts because of its use of multiple paths and endpoints. NDN can use shorter paths to cached content, use additional paths to less-loaded content providers, or use alternative paths that avoid congested network links.

In this dissertation we showed to how better exploit this multipath transport capability of NDN. First, we focused on the end-point applications and designed a new *congestion control* protocol for data requests that can now be answered by multiple caches, through multiple paths, and by multiple endpoints. Second, we designed a new *multipath routing protocol* to let NDN routers use the maximal possible nexthops while still avoiding loops. Third, we developed a novel *traffic engineering* scheme that fits the NDN semantics by splitting packets at every hop, rather than using end-to-end tunnels like current work.

Since publishing our work on congestion control (PCON) in 2016 [SYZZ16], it has been cited over 55 in the relevant literature. Since then, it has been compared many times [YLF+17, WYW18, TFA+19] against other approaches to congestion control (see Section 3.5 for a summary). The jury is still out which of those is the best approach for which scenario (e.g. wired, wireless, IP-overlay tunnels). We do not claim that PCON is the best solution for all scenarios, but it has at least been proven to be a very popular one; we think this popularity is due to its simpler design and freely available source code (`https://github.com/schneiderklaus`).

The work on multipath routing and traffic engineering is more recent (still under conference submission), but the source code is (for multipath routing) or will soon be (for traffic engineering) published on the GitHub repository mentioned above.

We hope that it will be just as influential as the work on congestion control.

Lastly, we have implemented the majority of the work in this dissertation into the official NDN Software Prototype – NFD (see `https://redmine.named-data.net/issues/1624`, `https://redmine.named-data.net/issues/4985`), which comes with its own set of real-world complications. Adhering to code style, accessing deep network-interface functions like queuing from the user space (which are usually implemented in the kernel), and supporting multiple operating systems (Linux and MacOS) which differ in their network stack implementation, are just a few examples. However, this effort pays off since it makes our work easily usable for other researchers, and future-proof – the NFD code will keep up to date with new versions of programming languages, libraries, and even operating systems. We hope that this will allow others to use and build on our work. From personal discussions, I noticed that questions about NDN transport layer performance keep coming up again and again, and the work presented here is just a small part of a much larger picture.

"Congestion control is not a project; it's a career."

— David R. Oran

CHAPTER 7

REFERENCES

[ABG+01] Daniel Awduche, Lou Berger, D Gan, Tony Li, Vijay Srinivasan, and George Swallow. Rsvp-te: extensions to rsvp for lsp tunnels, 2001.

[ABK15] Spyridon Antonakopoulos, Yigal Bejerano, and Pramod Koppol. Full protection made easy: the dispath ip fast reroute scheme. *IEEE/ACM ToN*, 2015.

[ALLY01] Sanjeewa Athuraliya, Steven H Low, Victor H Li, and Qinghe Yin. Rem: Active queue management. *IEEE network*, 2001.

[AMZ+12] Alexander Afanasyev, Ilya Moiseenko, Lixia Zhang, et al. ndnsim: Ndn simulator for ns-3. *University of California, Los Angeles, Tech. Rep*, 2012.

[ANO10] Somaya Arianfar, Pekka Nikander, and Jörg Ott. Packet-level caching for information-centric networking. In *ACM SIGCOMM, ReArch Workshop*, 2010.

[ASZ+14] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, Ilya Moi-seenko, Yingdi Yu, Wentao Shang, Yi Huang, Jerald Paul Abraham, Steve DiBenedetto, et al. Nfd developers guide. Technical report, Technical Report NDN-0021, NDN Project, 2014.

[Atl06] Alia Atlas. U-turn Alternates for IP/LDP Fast-Reroute. Internet-Draft draft-atlas-ip-local-protect-uturn-03, IETF, 2006.

[AZ08] Alia K Atlas and Alex Zinin. Basic specification for ip fast-reroute: loop-free alternates. 2008.

[Bar64] Paul Baran. On distributed communications networks. *Communications Systems, IEEE Transactions on*, 12(1), 1964.

[BFA03] Ethan Blanton, Kevin Fall, and Mark Allman. A conservative sack-based loss recovery algorithm for tcp. *RFC 3517*, 2003.

[BMST13] Sebastian Braun, Massimo Monti, Manolis Sifalakis, and Christian Tschudin. An empirical study of receiver-based aimd flow-control for ccn. In *IEEE ICCCN*, 2013.

[BPS13] Stewart Bryant, Stefano Previdi, and Mike Shand. A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses. RFC 6981, August 2013.

[Bri07] Bob Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 2007.

[ccn] Project CCNx™. http://www.ccnx.org/.

[CGM12] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks. In *ACM ICN workshop*, 2012.

[CGM+13] Giovanna Carofiglio, Massimo Gallo, Luca Muscariello, Michele Papalini, and Sen Wang. Optimal multipath congestion control and request forwarding in information-centric networks. In *ICNP*, 2013.

[CGM16] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. Optimal multipath congestion control and request forwarding in information-centric networks: Protocol design and experimentation. *Computer Networks*, 2016.

[CJT09] Carlos E Caicedo, James BD Joshi, and Summit R Tuladhar. Ipv6 security challenges. *Computer*, (2):36–42, 2009.

[CNM+16] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrović, Aurojit Panda, Andrei Gurtov, Aleksander Maidry, Michael Schapira, and Scott Shenker. The quest for resilient (static) forwarding tables. In *IEEE INFOCOM*. IEEE, 2016.

[CRR+12] Raffaele Chiocchetti, Dario Rossi, Giuseppe Rossini, Giovanna Carofiglio, and Diego Perino. Exploit the known or explore the unknown?: hamlet-like doubts in icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 7–12. ACM, 2012.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DPBM15] Andrea Detti, Claudio Pisa, and Nicola Blefari Melazzi. Modeling multipath forwarding strategies in ICN. In *IEEE INFOCOM WKSHPS*, 2015.

[ecm00] Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, November 2000.

[EGR16] Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. Ip fast rerouting for multi-link failures. *IEEE/ACM ToN*, 2016.

[EJLW01] A Elwalid, C Jin, S Low, and I Widjaja. Mate: Mpls adaptive traffic engineering. In *INFOCOM*. IEEE, 2001.

[Fel07] Anja Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1993.

[Flo08] Sally Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166, March 2008.

[FPG+18] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. Segment Routing Architecture. RFC 8402, July 2018.

[FRT02] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *IEEE communications Magazine*, 2002.

[FT00] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing ospf weights. In *INFOCOM 2000*, volume 2, pages 519–528. IEEE, 2000.

[Gea01] Geant – The Gigabit Research Network, Backbone Topology, December 2001. https://geant3plus.archive.geant.net/Resources/Media_Library/Documents/topology_map_2001.pdf, 2001.

[GLA15] JJ Garcia-Luna-Aceves. A fault-tolerant forwarding strategy for interest-based information centric networks. *IFIP Networking 2015*, 2015.

[GLAMB15] JJ Garcia-Luna-Aceves and Maziar Mirzazad-Barijough. Enabling correct interest forwarding and retransmissions in a content centric network. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*, pages 135–146. IEEE Computer Society, 2015.

[Gri15] Brothers Grimm. *Grimm's fairy tales*. McClelland & Stewart, 2015.

[GRJ08] Ivan Gojmerac, Peter Reichl, and Lasse Jansen. Towards low-complexity internet te: the adaptive multi-path algorithm. *Computer Networks*, 2008.

[GVKH18a] Nikola Gvozdiev, Stefano Vissicchio, Brad Karp, and Mark Handley. On low-latency-capable topologies, and their impact on the design of intra-domain routing. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 88–102. ACM, 2018.

[GVKH18b] Nikola Gvozdiev, Stefano Vissicchio, Brad Karp, and Mark Handley. On low-latency-capable topologies, and their impact on the design of intra-domain routing. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018.

[HBCR07] Jiayue He, Ma'ayan Bresler, Mung Chiang, and Jennifer Rexford. Towards robust multi-layer traffic engineering: Optimization of congestion control and routing. *IEEE Journal on Selected Areas in Communications*, 2007.

[HKM+13] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review*. ACM, 2013.

[HLR+08] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demo*, 2008.

[HR08] Jiayue He and Jennifer Rexford. Toward internet-wide multipath routing. *IEEE Network*, 2008.

[IBTD03] Sundar Iyer, Supratik Bhattacharyya, Nina Taft, and Christophe Diot. An approach to alleviate link overload as observed on an ip backbone. In *IEEE INFOCOM*, 2003.

[ICBD04] Gianluca Iannaccone, Chen-Nee Chuah, Supratik Bhattacharyya, and Christophe Diot. Feasibility of ip restoration in a tier 1 backbone. *Ieee Network*, 18(2):13–19, 2004.

[isi90] OSI IS-IS Intra-domain Routing Protocol. RFC 1142, February 1990.

[Jac88] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM CCR*, 1988.

[Jac06a] Van Jacobson. If a clean slate is the solution, what was the problem? http://wiki.fp7-pursuit.eu/uploads/7/7a/Jacobson_cleanslate.pdf, February 2006.

[Jac06b] Van Jacobson. A new way to look at networking. http://www.ccnx.org/395/1/van-jacobsen-at-google/, August 2006.

[JST+09] 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*, pages 1–12, 2009.

[KDM09] Amund Kvalbein, Constantine Dovrolis, and Chidambaram Muthu. Multipath load-adaptive routing: Putting the emphasis on robustness and simplicity. In *ICNP*. IEEE, 2009.

[KGGZ11] Kin-Wah Kwong, Lixin Gao, Roch Guérin, and Zhi-Li Zhang. On the feasibility and efficacy of protection routing in ip networks. *IEEE/ACM ToN*, 2011.

[KKDC05] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *ACM SIGCOMM Computer Communication Review*. ACM, 2005.

[KMT98] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 1998.

[KNF+11] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[KR12] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

[KS01] Srisankar Kunniyur and Rayadurgam Srikant. Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management. In *ACM SIGCOMM Computer Communication Review*. ACM, 2001.

[KS03] Srisankar Kunniyur and Rayadurgam Srikant. End-to-end congestion control schemes: Utility functions, random losses and ecn marks. *IEEE/ACM Transactions on Networking (TON)*, 2003.

[KZ89a] Atul Khanna and John Zinky. The revised arpanet routing metric. *ACM SIGCOMM Computer Communication Review*, 19(4):45–56, 1989.

[KZ89b] Atul Khanna and John Zinky. The revised arpanet routing metric. *ACM SIGCOMM Computer Communication Review*, 1989.

[LCR+07] Karthik Lakshminarayanan, Matthew Caesar, Murali Rangan, Tom Anderson, Scott Shenker, and Ion Stoica. Achieving convergence-free routing using failure-carrying packets. *ACM SIGCOMM CCR*, 37(4):241–252, 2007.

[Leb13] Mike Leber. Global ipv6 deployment progress report. *Hurricane Electric,[Online]. Available: http://bgp. he. net/ipv6-progress-report. cgi. [Accessed: Jun-2015]*, 2013.

[LHLX15] Kai Lei, Chaojun Hou, Lihua Li, and Kuai Xu. A rcp-based congestion control protocol in named data networking. In *CyberC*, 2015.

[LHX+15] Chengcheng Li, Tao Huang, Renchao Xie, Hengyang Zhang, Jiang Liu, and Yunjie Liu. A novel multi-path traffic control mechanism in named data networking. In *IEEE ICT*, 2015.

[Low03] Steven H Low. A duality model of tcp and queue management algorithms. *IEEE/ACM Transactions on Networking (ToN)*, 2003.

[LPS+13] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. Ensuring connectivity via data plane mechanisms. In *NSDI*, 2013.

[LS06] Xiaojun Lin and Ness B Shroff. Utility maximization for communication networks with multipath routing. *IEEE Transactions on Automatic Control*, 2006.

[MAMZ15] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM 2.0: A new version of the NDN simulator for NS-3. Technical Report NDN-0028, NDN, January 2015.

[MCAP18] Luca Muscariello, Giovanna Carofiglio, Jordan Auge, and Michele Papalini. Hybrid information-centric networking. Technical report, Internet-Draft draft-muscariello-intareahicn-00. Internet Engineering Task ..., 2018.

[MEFV08] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. In *ACM SIGCOMM CCR*. ACM, 2008.

[Moy98] John Moy. OSPF Version 2. RFC 2328, April 1998.

[MSWA02] Ratul Mahajan, Neil Spring, David Wetherall, and Tom Anderson. Inferring link weights using end-to-end measurements. In *IMW*, 2002.

[MT15] Nithin Michael and Ao Tang. Halo: Hop-by-hop adaptive link-state optimal routing. *IEEE/ACM Transactions on Networking*, 2015.

[MTX13] Nithin Michael, Ao Tang, and Dahai Xu. Optimal link-state hop-by-hop routing. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2013.

[ndna] Ndn protocol design principles, accessed apr. 2016. http://named-data.net/project/ndn-design-principles/.

[ndnb] Ndn testbed. http://named-data.net/ndn-testbed/.

[NFST15] Dinh Nguyen, Masaki Fukushima, Kohei Sugiyama, and Atsushi Tagami. Efficient multipath forwarding and congestion control without route-labeling in ccn. In *IEEE ICCW*, 2015.

[NIS] NIST. Ndn-dpdk: High-speed named data networking forwarder. https://github.com/usnistgov/ndn-dpdk.

[NJ12a] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Queue*, 2012.

[NJ12b] Kathleen Nichols and Van Jacobson. Controlling queue delay. *ACM Communications*, 2012.

[NJMI16] K Nichols, V Jacobson, A McGregor, and J Iyengar. Controlled delay active queue management: draft-ietf-aqm-codel-03. *RFC draft*, 2016.

[NKK12] Mohan Nanduri, Mark Kasten, and Naoki Kitajima. MPLS Traffic Engineering with Auto-Bandwidth: Operational Experience and Lessons Learned. https://www.apricot.net/apricot2012/__data/assets/pdf_file/0020/45623/MPLS-Traffic-Engineering-with-Auto-Bandwidth.pdf, 2012.

[NST99] Paolo Narvaez, K-Y Siu, and H-Y Tzeng. Efficient algorithms for multi-path link-state routing. 1999.

[OIVM09] Yasuhiro Ohara, Shinji Imahori, and Rodney Van Meter. Mara: Maximum alternative routing algorithm. In *INFOCOM*. IEEE, 2009.

[ORS12] Sara Oueslati, James Roberts, and Nada Sbihi. Flow-aware traffic control for a content-centric network. In *IEEE INFOCOM*, 2012.

[PACS00] Vern Paxson, Mark Allman, Jerry Chu, and Matt Sargent. Computing tcp's retransmission timer. Technical report, RFC 2988, 2000.

[PJK14] Heungsoon Park, Hoseok Jang, and Taewook Kwon. Popularity-based congestion control in named data networking. In *IEEE ICUFN*, 2014.

[PM09] Fernando Paganini and Enrique Mallada. A unified approach to congestion control and node-based multipath routing. *IEEE/ACM ToN*, 2009.

[PPJ11] J. Pan, S. Paul, and R. Jain. A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49(7):26–36, 2011.

[QAY+15] Junaid Qadir, Anwaar Ali, Kok-Lim Alvin Yau, Arjuna Sathiaseelan, and Jon Crowcroft. Exploiting the power of multiplicity: a holistic survey of network-layer multipath. *IEEE Communications Surveys & Tutorials*, 2015.

[Qi] Yan Qi. An implementation of the k-shortest-paths algorithm in cpp. `https://github.com/yan-qi/k-shortest-paths-cpp-version`.

[Rex09] Jennifer Rexford. Rethinking internet traffic management using optimization theory. https://www.microsoft.com/en-us/research/video/rethinking-internet-traffic-management-using-optimization-theory/, 2009.

[RF14] Natalya Rozhnova and Serge Fdida. An extended hop-by-hop interest shaping mechanism for ccn. In *IEEE GLOBECOM*, 2014.

[RN12] Glenn Robertson and Srihari Nelakuditi. Handling multiple failures in ip networks through localized on-demand link state routing. *IEEE TNSM*, 2012.

[RR11] Dario Rossi and Giuseppe Rossini. Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech*, 2011.

[RTEC11] Gábor Rétvári, János Tapolcai, Gábor Enyedi, and András Császár. Ip fast reroute: Loop free alternates revisited. In *INFOCOM*. IEEE, 2011.

[SB10] Mike Shand and Stewart Bryant. IP Fast Reroute Framework. RFC 5714, 2010.

[SCP13] Lorenzo Saino, Cosmin Cocora, and George Pavlou. Cctcp: A scalable receiver-driven congestion control protocol for ccn. In *IEEE ICC*, 2013.

[SJ09] Diana Smetters and Van Jacobson. Securing network content. Technical report, Citeseer, 2009.

[SKK04] Shan Sinha, Srikanth Kandula, and Dina Katabi. Harnessing tcp's burstiness with flowlet switching. In *Hotnets-III*. Citeseer, 2004.

[SMKV11] Abhigyan Sharma, Aditya Mishra, Vikas Kumar, and Arun Venkataramani. Beyond mlu: An application-centric comparison of traffic engineering schemes. In *2011 Proceedings IEEE INFOCOM*. IEEE, 2011.

[SMW02] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM CCR*, 2002.

[Ste13] Richard Steenbergen. MPLS RSVP-TE Auto-Bandwidth: Practical Lessons Learned. https://archive.nanog.org/sites/default/files/tues.general.steenbergen.autobandwidth.30.pdf, 2013.

[SYZZ16] Klaus Schneider, Cheng Yi, Beichuan Zhang, and Lixia Zhang. A practical congestion control scheme for named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 21–30. ACM, 2016.

[SZB19] Klaus Schneider, Beichuan Zhang, and Lotfi Benmohamed. Hop-by-hop multipath routing: Choosing the right nexthop set. *arXiv preprint arXiv:1906.10266*, 2019.

[TFA+19] Adrien Thibaud, Julien Fasson, Fabrice Arnal, Renaud Sallantin, Emmanuel Dubois, and Emmanuel Chaput. An analysis of ndn congestion control challenges. 2019.

[TW13] A.S. Tanenbaum and D.J. Wetherall. *Computer Networks*. Pearson custom library. Pearson, 2013.

[VGLA99] Srinivas Vutukury and JJ Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *ACM SIGCOMM CCR*. ACM, 1999.

[Vil99] Curtis Villamizar. OSPF Optimized Multipath (OSPF-OMP). Internet-Draft draft-ietf-ospf-omp-02, IETF, February 1999. Work in Progress.

[VLK13] Hung Quoc Vo, Olav Lysne, and Amund Kvalbein. Routing with joker links for maximized robustness. In *IFIP Networking*, pages 1–9. IEEE, 2013.

[VMAZ16] Satyanarayana Vusirikala, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. A best effort link layer reliability scheme. Technical report, NDN TR41, 2016.

[VPL13] Matteo Varvello, Diego Perino, and Leonardo Linguaglossa. On the design and implementation of a wire-speed pending interest table. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 369–374. IEEE, 2013.

[VRC01] Arun Viswanathan, Eric C. Rosen, and Ross Callon. Multiprotocol Label Switching Architecture. RFC 3031, January 2001.

[WLLD05] Jiantao Wang, Lun Li, Steven H Low, and John C Doyle. Cross-layer optimization in tcp/ip networks. *IEEE/ACM Transactions on Networking (TON)*, 2005.

[WRGH11] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *NSDI*, 2011.

[WRN+13] 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. *ACM SIGCOMM CCR*, 2013.

[WXQ+06] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. Cope: traffic engineering in dynamic networks. In *Sigcomm*, 2006.

[WYW18] Minxiao Wang, Meng Yue, and Zhijun Wu. Wincm: A window based congestion control mechanism for ndn. In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pages 80–86. IEEE, 2018.

[XCR07] Dahai Xu, Mung Chiang, and Jennifer Rexford. Deft: Distributed exponentially-weighted flow splitting. In *INFOCOM 2007*, pages 71–79. IEEE, 2007.

[XCR11] Dahai Xu, Mung Chiang, and Jennifer Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. *IEEE/ACM Transactions on networking*, 2011.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM*, 2004.

[YAM+13] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A case for stateful forwarding plane. *Elsevier Computer Communications*, 2013.

[YHC+14] Edmund Yeh, Tracey Ho, Ying Cui, Michael Burd, Ran Liu, and Derek Leong. Vip: A framework for joint dynamic forwarding and caching in named data networks. In *ACM ICN*, 2014.

[YLF+17] Yuhang Ye, Brian Lee, Ronan Flynn, Niall Murray, and Yuansong Qiao. Hlaf: Heterogeneous-latency adaptive forwarding strategy for peer-assisted video streaming in ndn. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 657–662. IEEE, 2017.

[YLS+14] Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. Keep forwarding: Towards k-link failure resilient routing. In *INFOCOM*. IEEE, 2014.

[YSC12] Haowei Yuan, Tian Song, and Patrick Crowley. Scalable ndn forwarding: Concepts, issues and principles. In *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. IEEE, 2012.

[YW06] Xiaowei Yang and David Wetherall. Source selectable path diversity via routing deflections. In *ACM SIGCOMM CCR*. ACM, 2006.

[ZAB+14] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 2014.

[ZEB+10] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, Christos Papadopoulos, et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[ZWL+15] Jianer Zhou, Qinghua Wu, Zhenyu Li, Mohamed Ali Kaafar, and Gaogang Xie. A proactive transport mechanism with explicit congestion notification for ndn. In *IEEE ICC*, 2015.

[ZZR+14] Feixiong Zhang, Yanyong Zhang, Alex Reznik, Hang Liu, Chen Qian, and Chenren Xu. A transport protocol for content-centric networking with explicit congestion control. In *IEEE ICCCN*, 2014.