Supporting Long Term Evolution in an Internet Architecture

Dongsu Han

CMU-CS-12-144

December 20, 2012

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Srinivasan Seshan, Chair
Peter Steenkiste,
David Andersen,
Konstantina (Dina) Papagiannaki, Telefonica Research
Aditya Akella, University of Wisconsin-Madison

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by Intel and the National Science Foundation under grant numbers CNS-0435382, CNS-0520192, CNS-0721857, CNS-1032226, and CNS-1040801. Dongsu Han was in part supported by the Korea Foundation for Advanced Studies.

Keywords: Future Internet Architecture, Network Evolution, Transport Evolution, Congestion Control, Content-Oriented Networking, Service-Oriented Networking

For Yoomin and YoungEun

Abstract

Networking is fundamentally changing from the traditional model of a dumb data pipe to a system that combines computation and storage with the delivery of data. The Internet has transformed dramatically from its initial design and has grown far beyond its original vision. A future Internet design must facilitate this movement and, at the same time, accommodate dramatic evolution we might experience in the future.

In this dissertation, we address the fundamental challenges in designing an Internet architecture that supports long-term evolution. We argue that interesting interactions between the network and the end-points can happen in the new environment and that they should be better managed and supported by the network architecture. To achieve this, we propose an integrated architecture that facilitates such interactions by supporting the evolution of the network's service model and systematically explore three different aspects of designing a network architecture that supports evolution.

First, we present a network design that supports graceful introduction of new network-level functionality in an efficient manner. Unlike the current Internet, this design allows the introduction of new network-level abstractions along with new functionality. Second, we explore the implications of the network design on end-points. We demonstrate that when networks provide new functionality, some of the features that were traditionally implemented purely at the end-points can now be better handled in cooperation with the network. Therefore, end-points' strategy may have to adapt to the functionality provided by the underlying network. Finally, designing a network architecture often requires designing a set of common behaviors that all routers, switches, and end-points must agree on. However, such common behaviors are especially hard to change once they are deployed, which significantly hinders the evolution in the network. However, we show that we can design a common behavior to support evolution.

In summary, this dissertation demonstrates that a future network architecture can benefit from a design that supports evolution and accommodates diversity at various layers of the network without sacrificing the performance and efficiency.

Acknowledgments

This dissertation was made possible by support from my advisor, colleagues, mentors, friends, and family members who provided invaluable guidance, technical contributions, advice, and encouragement. I would like to truly thank everyone of them for their commitment.

I am greatly indebted to my advisor and mentor Srini Seshan. Without his support, guidance, and contributions, this dissertation would not be what it is. I feel very fortunate to have him as my academic advisor. Over the course of my research career, his intuition, intellectual insight, and positive attitude was tremendously helpful in tackling hard problems and finding problems worth solving. I have also benefited from his encouragement, constructive criticism, and positive attitude. I aspire to do the same to my own students or mentees in the career ahead of me.

Aditya Akella, David Andersen, and Peter Steenkiste were my shadow advisors and mentors who provided invaluable lessons and demonstrated first-hand the quality of excellent researchers. Aditya Akella helped push my research forward by focusing on the key issues and paying attention to the details at the same time. David Andersen, with his curiosity and expertise, provided invaluable feedback in systems design and evaluation. Peter Steenkiste provided guidance and support in all possible ways and his intuition helped me focus on core problems.

I also would like to thank my mentors Dina Papagiannaki and Michael Kaminsky. Not only they led me into interesting research problems in the early years in the program, but they were also supportive and fun to work with. Thanks to them, I had the memorable experience of writing my first research paper.

I also want to thank my colleges that contributed to this dissertation and my other research. John Byers, Hui Zhang, Onur Mutlu, Mor Harchol-Balter, and Adrian Perrig provided invaluable technical feedback. I was fortunate enough to work with many wonderful colleagues including Hyeontaek Lim, Matthew Mukerjee, David Naylor, Suk-bok Lee, Ashok Anand, Robert Grandl, Fahad Dogar, Boyan Li, Michel Machado, Arvind Mukundan, Wenfei Wu, Umar Javed, Jeffery Pang, George Nychis, Yoongu Kim, and Aditya Agarwala. I also thank researchers at AT&T research including Jeff Pang, Ramon Caceres, and Alex Varshavsky for their support.

I want to thank my friends and colleagues including Kaushik Lakshminarayanan, George Nychis, Bin Fan, Xi Liu, Xin Zhang, Jeffery Pang, Vyas Sekar, Vijay Vasudevan, Amar Phanishayee, Soo-bum Lee, U Kang, Xiaohui Wang, Iulian Moraru, JongHyup Lee, Boyan Lee, Hyeontaek Lim, Matthew Mukerjee, David Naylor, Suk-bok Lee, Ashok Anand, Robert Grandl, Fahad Dogar, Boyan Li, Michel Machado, Yilin Mo, Yoongu Kim, Seungil Huh, B. Aditya Prakash, Il-Chul Moon, Mukesh Agrawal, Athula Balachandran, Jin Kyu Kim, Junchen Jiang, and Gunhee kim. I thank everyone I met during the Open House and the Immigration Course. I also must acknowledge all the administrative support I had at Carnegie Mellon University, Angela Miller, Deborah Cavlovich, Catherine Copetas, Kathy McNiff, and the rest of the excellent staff.

Finally, I thank my beloved wife and son for their being; parents, parents-in-law, and members of KCCP for their love, support, and prayer; and friends for helping me get through this long journey.

Contents

1 Introduction **1**

1.1 Problem and Scope 2

1.1.1 The Fundamental Movement towards Resource Integration 2

1.1.2 The State-of-the-art Practice and Challenges 4

1.2 Thesis Statement and Approach 6

1.2.1 Thesis Statement 6

1.2.2 Thesis Approach 7

1.3 Thesis Overview 8

1.3.1 Contributions 9

1.3.2 Thesis Outline 9

2 The Integration of Computation and Storage into Networking **11**

2.1 In-network Computation 11

2.2 In-network Storage 13

2.2.1 Reasons for Caching 13

2.2.2 Trends in In-network Caching 15

2.3 Summary 18

3 An Evolvable Internet Architecture **19**

3.1 Summary of Key Approach 21

3.2 Foundational Ideas of XIA 22

3.3 XIP 23

3.3.1 Principals 24

3.3.2 XIP Addressing 25

3.3.3 XIP Header and Per-Hop Processing 29

3.4 XIA Addresses in Action 32

3.4.1 Bootstrapping Addresses 32

3.4.2 Simple Application Scenarios 34

3.4.3 Support for Richer Scenarios 36
3.5 Implementation 37
3.6 Evaluation 38
3.6.1 Router Design and Performance 39
3.6.2 Application Design and Performance 43
3.6.3 Scalability 46
3.7 Related Work 48
3.8 Chapter Summary 51

4 On End-point Evolution 55
4.1 Identification and End-point Adaptation 56
4.1.1 Background and Current Practice 56
4.1.2 Explicit Invocation and Identification of Functionality 57
4.1.3 End-point Adaptation 59
4.2 A Case for End-point Adaptation 60
4.3 Current Loss Recovery Schemes 62
4.4 Redundant Packet Transmission: Design 63
4.4.1 Basic Idea 63
4.4.2 Key Features 64
4.4.3 Scheduling Redundant Packets 66
4.4.4 Comments on RT/RPT 67
4.5 RPT with Congestion Control 69
4.6 RPT in Various Networks 71
4.7 Evaluation 73
4.7.1 Evaluation Framework 74
4.7.2 End-to-end Video Performance 75
4.7.3 Parameter Selection and Sensitivity 78
4.7.4 Impact of RPT on the Network 82
4.7.5 TCP-Friendly RPT 86
4.7.6 Evaluation Summary 88
4.8 Chapter Summary 89

5 Designing a Common Behavior to Support Evolution 91
5.1 Evolution Support in a Congestion Control Framework 92
5.2 Related Work 94
5.3 Requirements and Principles of Design 96
5.3.1 Requirements for evolution 96
5.3.2 Principles of design 97

5.4 Design 98
5.4.1 Flexible Control Framework 100
5.4.2 Evolution 103
5.4.3 Budget Management 107
5.5 Practical Issues 109
5.6 Evaluation 110
5.6.1 Performance 111
5.6.2 End-point Evolution 118
5.6.3 Network Evolution 121
5.7 Chapter Summary 124

6 Conclusion and Future Work **125**
6.1 Contributions 125
6.1.1 Evolvable Network Design 126
6.1.2 End-point Evolution 126
6.1.3 Designing a Common Behavior 127
6.2 Implications and Outlook 128
6.3 Future Work 129

Bibliography **133**

List of Figures

1.1 Moore's Law and Networking: The increase in the speed of computational power is much faster than the increase in the speed of network I/O [34]. 3
1.2 DRAM density and cost: Target reduction of DRAM's production cost per bit is x60 in the next 12 years [100]. 3
1.3 Design space for an integrated network system design. Three sub-domains are highlighted in red. 6

2.1 Projection of the global Internet traffic per month [11] 14

3.1 XIP packet header. 29
3.2 Simplified diagram of an XIP router. 30
3.3 Bank of the Future example scenario. 34
3.4 Block diagram of our XIA prototype. 37
3.5 Packet forwarding performance of a software router. The forwarding table has 351 K entries. 39
3.6 Fast-path processing smooths out total cost. 41
3.7 In-memory packet processing benchmark (no I/O). 41
3.8 Parallel and serial route lookup cost. 42
3.9 XIP Header Size 42
3.10 Content transfer. 44
3.11 Service migration. 45
3.12 Client mobility. 45

4.1 Redundant Packet Transmission in a redundancy elimination router. 57
4.2 RPT and FEC under 2% random loss. 64
4.3 Sequence of packets sent by RPT 67
4.4 Typical Deployment of WAN optimizers 71
4.5 Bandwidth use 76
4.6 Snapshot of the video 76
4.7 Video quality and loss rate for real traces 77

4.8 RPT's performance is much less sensitive to its parameter setting. 78
4.9 Percent data loss rate and overhead: RPT greatly outperforms FEC with small group size. 79
4.10 Pattern of loss: Small bursts in actual loss are more common than the uniform random loss. 80
4.11 Video quality under 1 to 8% loss. RPT(3) steadily delivers high quality even under high loss. 81
4.12 Bursty loss increases the data loss especially when the delay parameter is small. . 81
4.13 Breakdown of bottleneck link utilization: RPT flows do not impact the link utilization. RPT flows are prioritized over competing TCP flows. 83
4.14 Response time and size for short HTTP flows (long flows omitted for clarity). . . 84
4.15 Impact on loss rate due to RPT flows. 84
4.16 Queuing delay is reduced with RPT flows. 85
4.17 *No Harm:* Bandwidth use on a non-RE link in the no harm case. 85
4.18 TFRC RPT under random loss. 86
4.19 TFRC RPT exhibit TCP friendliness. 87
4.20 We adapt the encoding rate to TFRC's rate. 88
4.21 Video quality comparison. 89

5.1 Design Overview 99
5.2 Network generates the price feedback. 99
5.3 Budget management: (a) Aggregate congestion control using dynamic value translation. (b) Value translation between domains. 107
5.4 Convergence dynamics of a) FCP, b) XCP, and c) RCP: FCP achieves fast convergence and accurate rate control. When new flows arrive, XCP's convergence is slow, and RCP's rate overshoot is significant. 111
5.5 Short flows: Comparison of various statistics. AFCT stands for average flow completion time. 111
5.6 Short flows: Average flow completion time versus flow size. Mean flow size is 30 KB. 111
5.7 FCP's fair-share is determined by the budget. 113
5.8 The algorithm is stable under random perturbation. 113
5.9 Preloading allows fast convergence and accurate rate estimation. 114
5.10 Performance with long-lived flows and short-lived flows. 115
5.11 Performance statistics with long-lived flows and short-lived flows. 115

5.12 FCP's convergence dynamics with multiple links. Flow 0's round trip delay is 100 ms. Each column presents a different scenario: 1) (a) and (b) shows the dynamics of the sending rate and price of the two links when Flow 0 starts up first. 2) (c) and (d) shows the dynamics when Flow 0 starts up after Flow 1 and Flow 2. 116
5.13 A misbehaving flow negatively impacts the link utilization. 117
5.14 Topology 119
5.15 Equal-budget flows 119
5.16 Equal-throughput flows 119
5.17 Max-throughput flows 119
5.18 Theoretical (a) throughput and utility (b) versus budget assignment *X* for flow S1's top flow. 120
5.19 Background flows only take up the spare capacity. The sending rate is averaged over 40 ms. 121
5.20 Support for bandwidth stability 122
5.21 Multicast flow (red line) adapts to congestion. 122
5.22 Deadline flows 122
5.23 Aggregate control 122
5.24 Topologies and flow patterns of experiments 123

List of Tables

3.1 Router Hardware Specification 39
3.2 Forwarding table size and public key store size of an AD with 100 million hosts. . 47

4.1 Comparison of FEC and RPT over CCN 72
4.2 User perception versus PSNR 74
4.3 Average video quality (PSNR) 75
4.4 Maximum one-way delay of RT and FEC 82
4.5 *Harm:* Bandwidth use on a RE-link. 86

Chapter 1

Introduction

Networking is fundamentally changing from the traditional model of a dumb data pipe to a system that combines computation and storage with pure data delivery. Behind this trend are economic motivations as well as technological advances that allow physical and logical integration of computation and storage into networking. For business reasons, network service providers want to provide value added services, such as application acceleration and content delivery, to generate more revenue. The exponential growth in technology, represented by Moore's Law, enabled the integration. Examples of this integration include middleboxes, Software-Defined Networking (SDN), and content distribution networks. Middleboxes introduce in-network computation and storage in the network's data path and have become nearly ubiquitous in today's networks [165]. SDN incorporates computation into networking by creating an open programming environment atop a universal abstraction of networking hardware in an attempt to simplify the network management and foster innovations in the network control plane [85]. Finally, many of today's networked systems from Internet-scale content distribution networks to local deployments of WAN optimizers use in-network caching for efficient content delivery.

The movement towards integrated resources enables rich forms of interaction between the network and the end-points that were traditionally not envisioned. This is a radical departure from the dumb pipe abstraction and, therefore, has profound implications on how we should design the network architecture as a whole. In particular, this trend creates three new challenges that we have not considered in the original Internet design:

1. Along the movement towards this resource integration, new functionality will continue to be introduced into the network. However, the current design of the Internet has significant limitations that prevent new service models to be introduced. IP, for example, does not allow new functionality or per-hop behavior to be introduced at the network layer nor permits its service model to evolve over time.

2. When the network merely provided data delivery, the rest of the functionality had to be implemented purely at the endpoints, resulting in a design where end-points implement almost all of the interesting features. However, when networks provide new functionality, it changes the balance. In fact, some of the features can now be handled more efficiently with the help from the network. Therefore, we must reconsider how traditional features are designed in new environments where networks provide new functionality. Depending on the functionality provided by the underlying network, end-points must also adapt to the environment to effectively leverage new network functionality.

3. Designing a network architecture often requires designing a set of common behaviors that all routers, switches, and end-points must agree on. However, such common behaviors are especially hard to change once they are deployed, which significantly hinders the evolution in the network. Therefore, they must be carefully designed not to hinder evolution.

In this dissertation, we address the fundamental challenges in designing an Internet architecture that supports long-term evolution. The challenges listed above, in fact, address different aspects of designing a network architecture that supports evolution. The central tenet of this dissertation is that we should not design an architecture solely based on what is feasible today and what we must support today because technology will always advance and requirements for the network will also change as networked applications and their requirements will further diversify. Instead, we argue that the network architecture must embrace evolution in technology as well as evolution in the network's usage model. In addressing the challenges listed above, this dissertation provides solutions to fundamental problems in designing a network that supports evolution and takes an important step forward in understanding the implications of resource integration in the network.

1.1 Problem and Scope

This section outlines the scope of our dissertation research in more detail. We first motivate the problem by looking at the fundamental changes in networking and examine the key challenges that we face today.

1.1.1 The Fundamental Movement towards Resource Integration

The traditional goal of networking was to provide connectivity between two end-hosts. As a result, the network itself was considered a "dumb data pipe". The network layer was supposed to perform minimal packet processing to deliver packets from one end to the other, and any sort of interesting interactions happened between two end-points, e.g., at the application layer. In fact,

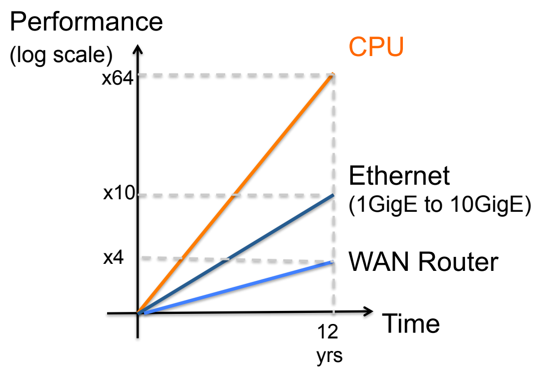


Figure 1.1: Moore's Law and Networking: The increase in the speed of computational power is much faster than the increase in the speed of network I/O [34].

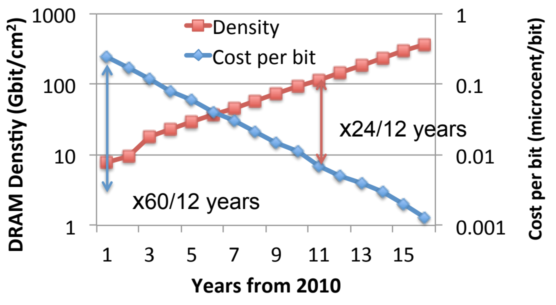


Figure 1.2: DRAM density and cost: Target reduction of DRAM's production cost per bit is x60 in the next 12 years [100].

many protocols in IP were just designed to provide this abstraction in an arbitrary network topology. For example, link layer protocols, such as the spanning tree protocol (STP), TRILL [178], and point-to-point protocol (PPP), are used to establish a path between two end-points in a network with an arbitrary interconnection of links. Routing protocols, such as BGP, OSPF, and RIP, and IS-IS, were designed to provide connectivity across (sub-)networks in an interconnected network. Transport protocols, such as UDP, TCP, and SCTP [143], provide an abstraction of a data pipe between two processes at the end-points.

However, today networks do much more than pure data delivery in order to support networked applications more efficiently. As Internet technology advanced, networked applications have become increasingly diverse. They also operate in a wide variety of environments, such as mobile networks, cloud infrastructures, and enterprise networks. To better support these applications, today's networks integrate computation and storage along with pure data delivery in a wide range of environments. Wide-area Internet service providers (ISPs) and content distribution networks (CDNs) commonly provide application acceleration or/and in-network caching in the wide-area environment. Enterprise networks also utilize various forms of application acceleration and in-network caching, such as virtual application streaming, SSL acceleration and WAN optimization [165]. Also, application-specific acceleration has been explored in other environments, such as data-center and sensor networks [59, 118, 127].

This is a dramatic departure from the original design of IP, which treated the network as a dumb data pipe. Today, packets receive much more processing that are often specialized to the nature of the communication. Two long term and fundamental trends are behind this movement:

- Business and administrative incentives: For business reasons, ISPs want to provide value-added services, such as content distribution and application acceleration, to generate more revenue. This is accelerated by the fact that the revenue for data delivery per byte has been constantly decreasing [163]. ISPs are constantly looking for opportunities for value added services and new sources of revenue. For administrative reasons, network operators want to place control elements, such as load balancers or intrusion detection, at strategic locations in the network to exert better control over the network.

- Long-term technological trend: Moore's law suggests that computational power doubles roughly every two years. However, the increase in the speed of networking I/O has been much slower. Figure 1.1 shows the increase in the computational power compared to the speed of networking I/O.[1] The gap between the two lines arises from fundamental differences between scaling computational power and scaling the I/O speed. The speed of I/O depends on the number of pins whereas the computational power depends on the number of transistors. Due to the limitations in chip packaging technology, it is much harder to increase the number of I/O pins [34]. As the gap increases, sophisticated packet processing becomes even more viable. Furthermore, Moore's law makes storage and computation increasingly cheap, which strongly incentivizes ISPs to integrate these resources to provide value added services. Figure 1.2 shows the technological road map for DRAM production cost and density published by the International Technology Roadmap for Semiconductors (ITRS). The data provides a projection or target improvement for the next 15 years based on the historical improvement achieved by the DRAM industry [99]. The projection shows that DRAM cost and density are expected to improve much faster than the speed of networking I/O; DRAM cost per bit target based on the historical data is set to decrease by 60 times over the next 12 years, and its density is expected to improve by a factor of 24 during the same period.

As we have seen, the movement towards the integration of storage, computation, and data delivery is driven by strong and fundamental trends. Therefore, we conclude that this movement is very likely to continue into the future.

1.1.2 The State-of-the-art Practice and Challenges

This resource integration, however, is not well supported by the current IP design. Even though many new functions have been introduced into the network, much of the network functionality was introduced through middleboxes, an entity that was not envisioned by the original design of IP [54]. In fact, the definition of middleboxes clearly states this fact: "any intermediary box performing functions apart from normal, standard functions of an IP router on the data path

[1]The original form of Figure 1.1 appears in the keynote presentation at NANOG given by A. Bechtolsheim [34].

between a source host and destination host" [48]. Many middleboxes introduce optimizations or new functionality in a transparent fashion. Therefore, applications and end-points did not need to change. This was a clear benefit of middleboxes, which facilitated incremental deployment.

However, middleboxes introduce functionality in a haphazard fashion violating the principles of the original network design. Middleboxes often introduce functionality in a transparent way, manipulate packets on the fly, and redirect or intercept packets [48]. For example, transparent caching performs deep packet inspection to infer the intent of an application even though the packet is not addressed to the cache. The middlebox approach as has been criticized by many [183] and has shown to create many problems; they violate end-to-end principles, impact application performance [132], cause poor interactions with applications and end-hosts [132, 185], and in some cases require applications and protocols to be designed to handle them explicitly [48, 183].

The more fundamental problem to the middlebox approach is related to the principles of system design: we do not know how to design a network that allows the evolution of its per-hop behavior in a more principled fashion. We have been introducing new network functionality haphazardly without any design principle or support from the network architecture. The introduction of new functionality creates the need for a new network-level abstraction, but IP only provides the abstraction of host-based communication. When introducing new functions, in systems design, it is commonsensical to introduce new abstractions that go with new functionality. However, we have been introducing new network-level functionality without a careful consideration of its abstractions. We never had an option to think about new network-level abstractions because the underlying IP network only provided the abstraction of host-based communication. However, it is obvious that more sophisticated functionality will likely to be introduced into the network as the reasons driving this movement are long-term, fundamental trends. Thus, the problems resulting from a haphazard introduction of new functionality will not go away, but only exacerbate in the future.

We summarize two important challenges resulting from this observation:

- It is clear that we can benefit from a more principled approach to introducing new network-level functionality. However, how we should design the network in this new environment and the principles we should adopt in doing so are not well understood. In particular, introducing new service models to the Internet has long been a challenge.

- The integration of resources has profound implications on not just the network design but also on the entire networked system including the end-points and applications. Traditional networked applications are built on top of the model where the network is treated as a dumb data pipe and any interesting interactions only happened between the end-points. However, when networks provide more sophisticated service models, interesting interactions may happen between the end-point and the network infrastructure. Nevertheless,

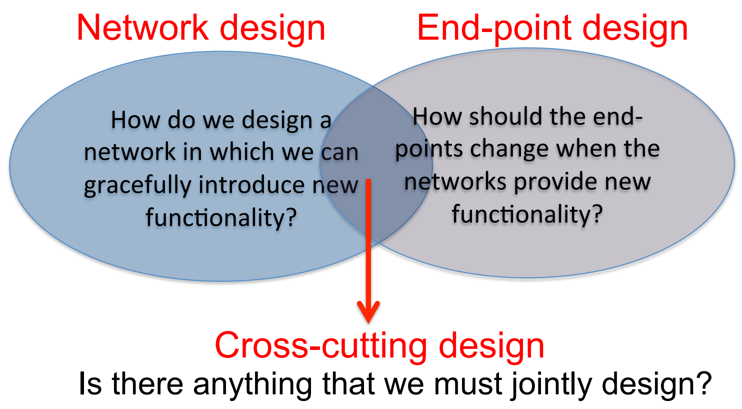

Network design
End-point design
How do we design a network in which we can gracefully introduce new functionality?
How should the end-points change when the networks provide new functionality?
Cross-cutting design
Is there anything that we must jointly design?

Figure 1.3: Design space for an integrated network system design. Three sub-domains are highlighted in red.

the implications of introducing new service model on end-points' protocols are not fully understood.

1.2 Thesis Statement and Approach

1.2.1 Thesis Statement

In summary, this dissertation shows how we can introduce new network-level functionality in a more principled fashion and addresses related challenges that must be considered when designing an architecture that supports long-term evolution.

This dissertation claims that *designing an evolvable network architecture requires an integrated system design that examines multiple aspects of the architecture to incorporate diverse interactions between the end-points and the network.*

To substantiate this claim, we systematically explore the problem space from the ground up. We split the problem space into three sub-domains as shown in Figure 1.3. First, we explore how we should design the network to allow graceful introduction of new functionality. Then, we study the implication of this design on the end-points. Finally, we look at how to design a common behavior in the network without undermining the evolvability of the network.

This dissertation tackles a fundamental issue and solves a problem that has not been previously addressed in each of the three areas:

1. How can the network facilitate the introduction of the new functionality and how do we introduce a new function into the network in a graceful manner? We argue that we need

to be able to introduce new network-level abstractions with new functions. Therefore, we explore a network design that allows evolution in the service model (i.e., a network-level abstraction and functionality). With new abstractions, our network architecture allows the end-points to explicitly invoke the functionality that they want in the network. This, in turn, allows the network to apply the new network functionality to packets without having to infer the application's intent and eliminates the need for haphazard packet processing, such as deep packet inspection.

2. An important implication of the integration on end-point design is that some of the most essential networking features that were previously handled purely at the end-points can now be implemented in cooperation with the network. In fact, we show that unexpected performance benefits can be gained by adapting the end-points' behavior to better take advantage of the network's new service model. Our network design also facilitates dynamic adaptation of end-points' behaviors based on the functionality provided by the underlying network because it allows end-points to explicitly invoke the functionality that they want in the network and identify the functionality supported by the network. To study the benefits of this dynamic adaptation more concretely, we use redundancy elimination as an example new functionality and explore a novel end-point strategy for real-time communication. This provides a valuable insight on what application and networking stack developers must consider when developing protocols that run on new types of networks.

3. Finally, we show how we might a design of a common behavior of end-points and routers without undermining the network's ability to evolve. Some of the most essential functionality of the network requires a design of a common behavior that must be applied to all packets. Examples of such functionality relates to resource allocation, such as mechanisms for denial-of-service (DoS) prevention and congestion control. The challenge in supporting such functionality is that once it is designed it is very hard to change. Therefore, these features require a careful joint design not to hinder the evolution within the network. However, the design of these features is not well studied in the context of an evolvable network design, and the solution to this problem is not well understood. We take congestion control as an example and perform a case study to draw important lessons on how we should design such components.

1.2.2 Thesis Approach

We highlight what we mean by supporting evolution and describe our evaluation methodology used in this dissertation.

What do we mean by supporting evolution? This thesis addresses the problem of supporting evolution in the networking layer (Chapter 3) and in congestion control (Chapter 5). We note that many previous studies have used many different notions of network evolvability. Here, we discuss the definitions used by a number of representative studies and clarify what we mean by supporting evolution. Ratnasamy *et al.* [152] explored mechanisms for making the current Internet more evolvable by enabling the deployment of new IP versions on top of older versions. A more concrete notion of evolvability was use by Active Networks [187], where they aim to provide multiple service models in the same network. Plutarch [61] advocates accommodating heterogeneity in its inter-networking protocol to allow each network to evolve independently to each other. Others [148] have referred to as evolvability the flexibility or extensibility of a specific set of features such as naming and redirection. Finally, OPAE [78] takes a much more broader view of architectural evolvability, which they refer to as the ability to make fundamental changes in the entire architecture. By network evolvability, we mean the ability to introduce a new service model, which is a combination of a new per-hop behavior and a new abstractions of the network. In Chapter 3, we present a design of an architecture that supports a diverse and evolvable set of the service models within the same network. Chapter 5 explores supporting evolution in congestion control. Unlike supporting evolution in the inter-networking layer which focuses on concurrently supporting multiple service models, our congestion control evolution primarily focuses on accommodating diversity within a single congestion control framework.

How do we show that an architecture or a protocol is evolvable? Proving that an architecture is evolvable and showing the limit of its evolvability is very difficult. We are unaware of any established methodology of proving that an architecture is evolvable. We, therefore, rely on the empirical methodology of "proof by demonstration", where we demonstrate that an architecture or a protocol is flexible enough to accommodate many different features and requirements using cases studies. Note that many studies on network evolution have taken this approach [78, 148, 152, 187]. In Chapter 3, we show that our network architecture supports evolution by demonstrating that it enables the introduction of new service models and allows multiple service models to coexist. In Chapter 5, we show that our congestion control algorithm is flexible enough to accommodate diverse application requirements.

1.3 Thesis Overview

We summarize the contributions of this dissertation and present the outline of this dissertation.

1.3.1 Contributions

We make four major contributions in this dissertation:

1. This dissertation presents a comprehensive study on the design of a networked system in a new environment where networks integrate storage and computation and demonstrates how to incorporate rich forms of interaction between the network and the end-point in the design of an Internet architecture.

2. We present a design and evaluation of a network architecture that allows the evolution of network-level abstractions and functionality. Our network architecture, XIA, allows new service models to be gracefully introduced into the network and solves the fundamental problem of supporting incremental deployment of new service models [90].

3. We show that when networks expose new functionality, unexpected performance benefits can be gained by adapting the end-points' behavior. In particular, some of the functions that were traditionally implemented at the end-point can now be handled much more efficiently in cooperation with the network. To demonstrate this, we present a design and performance analysis of a novel reliability scheme for time-critical communication in content-aware networks [89]. To the best of our knowledge, this is the first work to demonstrate that real-time communication can benefit from in-network caching.

4. Finally, we demonstrate how to design a common behavior in the network without significantly hindering evolution. As a case study, we present a clean-state congestion control framework for evolvable transport that accommodates diverse application requirements and ensures coexistence of many different features. To achieve this, our design incorporates a simple invariant for end-points to ensure coexistence and introduces a flexible abstraction to allow diverse resource allocation schemes.

1.3.2 Thesis Outline

The remainder of this dissertation is organized as follows:

- In Chapter 2, we present background on the integration of storage, computation, and networking to understand how networking has moved beyond its traditional model.

- In Chapter 3, we present a new network design that facilitates this integration by supporting the evolution of network-level functionality. In particular, we present a network architecture in which one can gracefully introduce new network-level functionality along with a new abstraction.

- The network design has significant implications on the end-point design. Chapter 4 describes how end-points can now explicitly invoke a function they want in the network and identify whether or not the underlying network supports the function. Furthermore, we show that end-points can now dynamically adapt their strategies based on the functionality provided by the network. To highlight this, we perform an in-depth study of robust real-time data delivery that utilizes the network's ability to suppress duplicate data transfer with in-network caching.

- Chapter 5 demonstrates how to design a common behavior without hindering evolution. We take congestion control as an example and demonstrate how to build an evolvable congestion control framework that accommodates diverse requirements and ensures coexistence of different variants to support multiple styles of communication.

- Finally, we conclude in Chapter 6 with future work and a summary of contributions.

Chapter 2

The Integration of Computation and Storage into Networking

This chapter provides a qualitative survey that highlights the movement towards the integration of computation and storage into networking. We show that the in-network computation and caching is widely used today and deployed almost ubiquitously in many environments. We first look at the integration of computational resources in Section 2.1. Then, we study the integration of storage in to networking in Section 2.2.

2.1 In-network Computation

Computation has been integrated into the network's data plane as well as the control plane. We find that many forms of in-network computation have become prevalent in today's network. We look at two different approaches in introducing in-network computation: middlebox approach and software-defined networking.

Middleboxes: In-network computation has been traditionally introduced through middleboxes. Middleboxes are deployed at strategic points of control to exert better control over the network. Traditional examples of middleboxes typically includes firewalls, intrusion detection, web proxies, and network address translators. These appliances were traditionally deployed locally at a small scale.

However, today, we see many different forms of middleboxes including WAN optimizers, VPN, SSL accelerators, load balancers, media gateways for video/audio conferencing, and application gateways or accelerators. These devices are widely deployed in many of today's networks including mobile networks [10, 185], enterprise networks [160, 165], data-center networks [8, 53, 97, 140], and inter-datacenter networks [118].

We highlight the deployment of middleboxes in two different environments.

- *Enterprise networks:* A number of recent studies performed a detailed survey on the number of middleboxes in enterprise networks [160, 165]. These studies show that the total number of middleboxes is comparable to the number of routers in enterprise networks. For example, Sekar et al. [160] showed that, in a large enterprise network that serves more than 80K users, the number of router was around 900, but the number of middleboxes were more than 600. Sherry et al. [165] presented a more comprehensive survey conducted across 57 enterprise network administrators from the NANOG network operators group. They show that regardless of the size of the networks, the number of middleboxes in an enterprise network is on par with the number of routers. For example, their survey shows that on average small networks have 7.3 L3 switches and 10.2 middleboxes. This shows the degree to which computation has been integrated into today's networks. Middleboxes started out as local deployments but are now scattered all throughout the network.

- *Wide-area networks:* Today's ISPs widely deploy infrastructure for value added services, such as application acceleration, in their network. AT&T, for example, provides a so-called Cloudlet service [58], which performs application-specific acceleration at the edge using the computational power of small clouds near the edge. These services target enterprise applications such as Microsoft Exchange Server and Oracle E-Business Suite [105], as well as consumer mobile applications that are available through open market places, such as the Android Market. Customer-facing ISPs are leveraging their advantage of eye-ball networks to deliver application acceleration closer to the customers. For example, AT&T provided application acceleration and dynamic caching for AccuWeather and was able to locally serve 75% of requests from AT&T's Cloudlet infrastructure. This significantly reduced the traffic to the AccuWeather's headquarters, and also provided dramatic improvement in the response time to its users [155]. We believe that this trend will accelerate in the future and ISPs will provide more value added services by combining computation and data delivery.

Software Defined Networking (SDN): Software Defined Networking (SDN) is another important trend in this context that tries to integrate generic computation into networking. OpenFlow [144], for example, integrates generic computation into the network's control plane. To achieve this, it introduces a universal abstraction of the underlying hardware and the API provided by the "network operating system" or the new control plane allows a secure access to the flow table inside each network switch. Using this API, network administrators can write a program in a high-level language that controls the operation of the network.

The approach taken by OpenFlow sets itself apart from the middlebox approach in that it provides a new abstraction that allows the applications to effectively leverage the power of modern networking hardware. OpenFlow provides a much more principled approach in introducing computation into the network's control plane. With the new abstraction provided by the control plane, it supports a logically centralized intelligence in the control plane. Thus, OpenFlow supports network-wide computation in the control plane.

2.2 In-network Storage

We observe that in-network caching have also become nearly ubiquitous. It is becoming increasingly popular to cope with the ever-increasing demand for content. We take a look at the motivation for and trends in in-network caching to show the inevitable trends towards prevalent caching.

2.2.1 Reasons for Caching

Motivation: Traffic demand on the Internet is rapidly growing at an unprecedented rate and scale. The global Internet traffic has increased eightfold over the past 5 years. In 2009 alone, IP traffic grew by 45%. Market reports project that this trend will continue and the total Internet traffic will grow nearly fourfold from 2011 to 2016 with an expected compound annual growth rate (CAGR) of 29% [11]. Figure 2.1 illustrates this trend. By 2016, the annual global IP traffic is projected to exceed the zettabyte landmark [11].

This growth is largely driven by multimedia content. Multimedia traffic is increasing 76% every year on average [151]. Cisco projects that "the sum of all forms of video (TV, video on demand, Internet, live video communication, and P2P) will account for over 91% of the global consumer traffic by 2013" [3]. Many factors contribute to the growth in multimedia traffic, including the desire for higher quality video, the increase in the number of video titles, and changes in viewing patterns. Users now expect to watch any video at their convenience, and "time-shifted" viewing patterns are becoming the norm [151] with the use of digital video recorders (DVRs) and on-line video services such as YouTube and Hulu.

This explosive growth poses an important challenge to the network— network capacity simply cannot keep up with the exponentially growing demand. In an attempt to design a network that scales with the demand for content, *in-network caching* have become popular.

Benefits of caching: We summarize three main benefits of caching.

- *Reducing bandwidth consumption by eliminating redundant data transfer:* Caching popular content reduces the bandwidth consumption of the network. This is especially attractive when it is expensive to upgrade the capacity of parts of the network. All caching

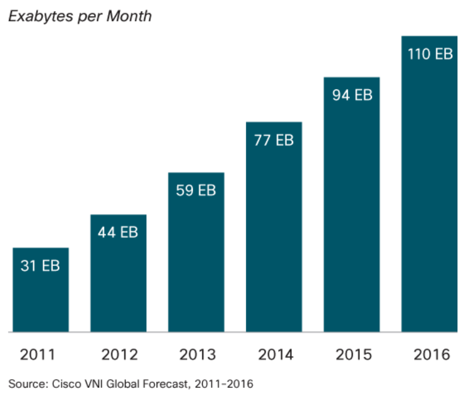


Figure 2.1: Projection of the global Internet traffic per month [11]

systems provide this benefit and many of them are deployed to reap this benefit, including transparent caching, content distribution networks, and peer-to-peer caching. WAN optimization or redundancy elimination [170] highlights this benefit. WAN optimizers are used to suppress redundant data transfer across a wide-area network which is often the bandwidth-constrained and expensive.

- *Scalable content delivery:* Caching (or replication) provides performance scaling. When a server is overloaded, adding more servers with replicated content is a simple solution to scale the performance of the system. Also, distributed caching systems, such as memcached, also sit in front of a back-end database with relatively lower throughput and cache the result to deliver high performance. With the rapidly increasing demand for content delivery, caching has become also popular as it provides an effective solution to cope with such demand.

- *Improved user experience:* Caching data near the edge eliminates much of the variability in the network and reduces response times. The user perceived quality is very important to many applications such as web browsing, on-line shopping, and on-demand video streaming. On the content provider side, it is important to provide better quality for business reasons. It has been shown that the quality of experience is correlated with user engagement [66], which impacts the content provider's revenue. Therefore, content providers utilize content distribution services to deliver better quality. On the ISP side, they have their own incentives to cache. ISPs today often perform transparent caching to popular over-the-top (OTT) content to reduce their own bandwidth consumption and to deliver better quality to their customers [153, 182].

As a result, caching is becoming prevalent at various levels in today's networks. Traditional application proxies, such as Web proxies, employ application-specific caching where the application layer explicitly interacts with the caches. Similarly, ISPs maintain caches at strategic locations to provide value added services such as networked digital video recording for IPTV services [41]. On a larger scale, content distribution networks (CDN) replicate application-level data at different geographic locations for performance and availability. As network technologies advance, storage elements are also being increasingly integrated into the network. For example, packet cache has been incorporated in routers [19] and end-points [13] to remove duplicate transfers of content over the same link. Commercial WAN optimizers [157] use similar techniques, and market reports indicate that 56% of large North American enterprises employ WAN optimization [137]. Such systems create a content-aware network by transparently caching the packet content at the network layer.

We believe that the aforementioned benefits of caching and the motivation for providing value-added services for network service providers will continue to drive the trend of nearly ubiquitous caching.

Two pieces of solid evidence support the future outlook:

- The trend towards ubiquitous caching extends to future Internet designs. Many future Internet architectures, such as XIA [90], MobilityFirst [28], and CCN [102], promote content-addressable networks and facilitate caching within the network. Some proposals, such as CCN [102], require content-aware caching on all routers, while others use caching as an optimization. Such prevalent use of in-network caching has created a new class of networking, called Information Centric Networking (ICN).

- Recently, many traditional telcos and ISPs are entering the content distribution network (CDN) market. Telcos that are already in the market include Telefonica, AT&T, BT, Level 3, Telus, Deutsche Telekom, and Singtel. Many of these ISPs have started offering the service very recently. For example, AT&T, Telefonica, and Level 3 all started their CDN service after 2010. Currently, these ISPs are trying to create a federated infrastructure [175]. While the federated CDN is in its early stage of conception, one of the core ideas is to form an open exchange for content distribution. An important implication is that this approach will create even more widely distributed caching infrastructure in both physical and administrative fashion.

2.2.2 Trends in In-network Caching

The use of storage within the network has dramatically evolved over the last few decades. For example, in the early 90's, caching was typically done at the end-points or near the end-points (e.g., Web proxies). However, recent trends show tighter integration of caching with the network, as evidenced by packet-level redundancy elimination networks that cache packet-level data on each router.

Summary of the trends: We highlight three key trends with a brief survey of the literature and practice:

- Local caching to widely distributed caching: Traditionally caching was done locally, but today we see large-scale distributed caching systems.
- Penetration of caching into lower layers: Traditionally application layer handled explicit interaction with caches (e.g., Web proxies), but today even lower layers closely interact with caching.
- Caching as a part of the core infrastructure: Caching was viewed as a local optimization, but today it is a core part of the infrastructure.

In the remaining section, we describe the three trends in more detail.

In-network caching that started out as a local optimization is now widely distributed in large scales in the Internet. From the early years of the Internet, caching started out as a local optimization. Local caching has been used in the context of Web caching and distributed file systems to reduce latency, save bandwidth, improve load balancing, and support disconnected operations. The most rudimentary form of local caching is the browser cache. It stores previously visited web pages to allow non-stale cached entries to be retrieved from the local disk. Local caching has been applied within a local domain or to a small set of users. Web proxies serve users behind the proxy, and early studies have demonstrated the bandwidth savings and latency reduction with caching and prefetching of Web objects [72, 116]. Reverse proxies are placed close to Web servers for scalability and load balancing. Distributed file systems, such as AFS [95] and Sprite FS [138], also use local caching for increased performance. CODA [112], in particular, employs *hoarding*, which prefetches files and metadata to support disconnected operations.

With increased deployment of local proxies and caches, cooperative use of distributed caches has been extensively explored in distributed Web caching systems [190] and distributed file systems [25]. Distributed Web caching has shown to further improve the benefit of local caching [190]: increased bandwidth savings, reduced latency, and being more fault-tolerant. Distributed file caching provides increased I/O performance and improves scalability by reducing the load on file servers [25].

More recently, we see caching in large-scale distributed systems and observe systems that specialize in caching being widely deployed. On one hand, content distribution networks have grown into a global scale distributed system. The use of peer-to-peer content distribution networks or hybrid CDNs [76, 192, 195] creates even larger systems of distributed caches. Redundancy elimination networks [19] even introduce caching on every router. On the other hand, distributed caching has been also integrated into the infrastructure for web applications and services. For example, memcached is a widely used distributed caching system for scaling the performance of web services by caching results returned from a backend database.

Caching is being integrated with the network more tightly by penetrating into lower layers. Traditionally, the application layer handled the end-points' interaction with caches. For example, in the case of Web caching, HTTP requests are intercepted and handled by the proxies. In BitTorrent, the application also schedules and handles transfer from multiple content replicas.

However, recent trend shows that storage has also penetration into lower layers. Systems, such as Data Oriented Transfer (DOT) [177] and Internet Backplane Protocol (IBP) [35], integrate storage into the network infrastructure. These systems expose a common interface to the application layer and handle the interactions with storage and caches. For example, DOT provides a transport service, at a layer just above the traditional transport, that effectively utilizes end-point [177] and/or in-network caches [67]. Similarly, IBP "exposes storage resources [to the application layer] for general use in the same way that the Internet now exposes transmission bandwidth for shared use." [35]

Storage elements have also been integrated into routers. For example, packet cache has been incorporated in routers [19, 170] and end-points [13] to remove duplicate transfers of content over the same link. They eliminate byte-level redundancy across packets that traverse the same link and offer protocol independent operation. Commercial WAN optimizers [157] use similar techniques. This technology is mature enough that it has been deployed in many enterprises and can be deployed in high-speed links. Market reports indicate that 56% of large North American enterprises employ WAN optimization solutions and the market for WAN optimization is growing rapidly [137]. It's forwarding speed can also scale up to OC48 speed [19]. High-end commercial products offer 10 to 20 Gbps of WAN optimization throughput [70, 98].

Caching is now an essential part of the core Internet infrastructure. Many players in the Internet and the Internet itself rely on systems that essentially provide caching. Content distribution networks are a part of today's core Internet infrastructure. Distributed caching, such as memcached, has been also integrated into the infrastructure for web applications and services. Many global service and content providers including YouTube, Facebook, Flickr, and Wikipedia use memcached.

We expect caches will continue to be part of the core network infrastructure in the future. Recently, ISPs are seeking to develop their own infrastructure CDN and to interconnect and federate such CDNs through an Operator Carrier Exchange (OCX) [175]. This means that ISPs and telcos that traditionally provided data pipes will introduce caching to their infrastructure. Simultaneously, ISP are employing transparent caching on their own either to remain competitive or to handle increasing traffic demand [182]. This trend is also evidenced by the growth of the transparent caching market—its market is expected to grow at a compound annual growth rate of 39.3% from 2010 to 2014 [153]. Also, traditional CDN players, such as Akamai and Limelight Network, now offer licensed CDN products or managed CDN services [111, 154]. CDN players are trying to partner with ISPs to deploy their service within the ISP's network. As such, caching will continue to penetrate into the core infrastructure.

This trend also holds in the context of future Internet architectures. Future Internet architectures, such as XIA and MobilityFirst, and proposals for Information Centric Networking [102, 115] either accommodate caching with in the infrastructure or advocate universal caching.

2.3 Summary

Motivated by the observations made in Chapter 1, we reviewed the practice of and literature on the integration of computation and storage into networking. We observed that this integration is now widely spread within today's networks and showed evidence that suggest this trend will continue over time. This change, however, was certainly not expected when the original Internet was designed. Therefore, we argue that we need to revisit the design of the network in light of this movement.

Chapter 3

An Evolvable Internet Architecture

The “narrow waist” design of the Internet has been tremendously successful, helping to create a flourishing ecosystem of applications and protocols above the waist, and supporting diverse media, physical layers, and access technologies below. However, the Internet, almost by design, does not facilitate a clean, incremental path for the adoption of new capabilities. In other words, the per-hop behavior of an IP router is fixed and not allowed to change. This is a significant shortcoming of IP in the new environment where networks integrate storage and computation to provide rich functionality.

The shortcoming has significant short-term and long-term costs that we often do not realize. The shortcoming of IP resulted in haphazard and ad-hoc solutions and pervasive deployments of middleboxes which violate the architectural principles and complicates the task of understanding, operating, and managing the network. It limits innovation in practice, which is best illustrated by the 15+ year deployment history of IPv6 and the difficulty of deploying primitives needed to secure the Internet. More broadly, this renders many of the principle taught in networking courses useless and makes it hard to educate who has a broad and deep understanding of the field. On a longer term, the lack of a principled approach hinders the advancement of scientific research and advancement as the practice of networking in many cases is about applying ad-hoc solutions driven by real-world constraints which are often found after the fact.

We argue that these problems are fundamentally tied to the difficulty of changing the network-level abstraction, or an agreement between hosts and routers: it is nearly impossible to modify in any deep way the information shared between end-points and routers that informs routers how to act upon packets. In other words, we have been introducing new functionality (i.e., different types of packet processing) in the network without introducing new network-level abstractions.

We make the case for a new Internet architecture, called the eXpressive Internet Architecture or XIA, to address these problems from the ground up. We provide a more principled way and a clean path for incremental deployment of introducing new abstractions and functionality in the architecture. XIA differs from today's Internet in three key areas.

First, while the Internet architecture and protocols only support the abstraction of host-to-host communication, XIA supports multiple network-level abstractions. To support multiple abstractions, XIA is centered on the notion of a *principal*: a named originator or recipient of a packet, such as a host, a service, or a piece of content. Each *type* of principal type (e.g., host, service or content) represents a new abstraction of the network and is associated with its own semantics, enabling routers to apply different per-hop processing. This provides *expressiveness* in the sense that applications can more precisely specify their intent and invoke the functionality they want in the network by choosing the appropriate principal types for communication. It also allows the network to more aggressively optimize communication operations for a particular communication style by applying new functionality, e.g., caching and dynamic load balancing.

Second, whereas the current Internet's narrow waist is fixed, the set of network-level abstractions and functionality supported in XIA can be extended over time. This provides *evolvability* in two ways. First, new application paradigms and usage models can be supported more effectively by adding native support for the appropriate principals. However, ubiquitous support for a new principal type is unlikely to occur overnight. As a result, the XIA architecture provides support for incremental deployment of new principal types, allowing applications to benefit from even partial deployment of router support for the principal. Second, advances in computing and storage technology can be leveraged to enhance network support for a given principal (e.g., allow incremental deployment of principal-specific optimizations) in a clean, transparent fashion.

Taken together, these two aspects enable XIA to provide many, if not all, of the goals of alternative architectures such as, content-centric [102, 179] networks that better support various forms of content retrieval, and service-centric [140, 158] networks that provide powerful primitives such as service-level anycast. The XIA architecture enables each of these important communication styles—and those that will emerge in the future—to be supported natively to the degree that it makes sense to do so at a given point in time.

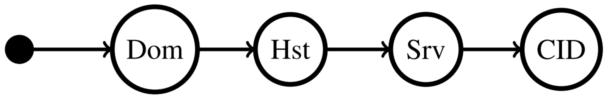
Finally, XIA guarantees principal-specific *intrinsic security* properties with each communication. This allows an entity to validate that it is communicating with the correct counterpart without needing access to external databases, information, or configuration. Intrinsic security is central to reliable sharing of information between hosts and routers and to ensuring correct fulfillment of the contract between them. Also, it can be used to bootstrap higher level security mechanisms.

In this chapter, we present the design of the XIA architecture, which addresses the above three requirements: expressiveness, evolvability, and intrinsic security. In addition to the details of our key design features, we also present case studies of interesting usage scenarios enabled by XIA.

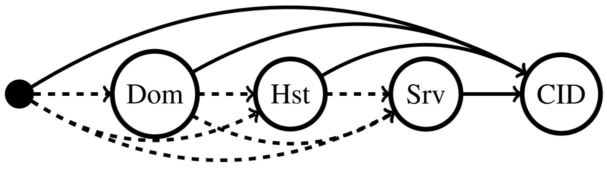
Chapter outline. We summarize the key approach of the architecture in Section 3.1. We highlight key design ideas of the architecture that supports evolution (Section 3.2), and systematically incorporate them into the eXpressive Internet Protocol (XIP) (Section 3.3). We describe optimizations for high-speed per-hop packet processing, which can achieve multi-10Gbps forwarding speed. Then, through concrete examples, we show how networks, hosts, and applications interact with each other and benefit from XIA's flexibility and evolvability (Section 3.4). We briefly describe our prototype implementation in Section 3.5. Using the prototype, we show how applications can benefit, and demonstrate the practicality of XIP and the architecture under current and (expected) future Internet scales and technology (Section 3.6). We discuss related work (Section 3.7) and close with a conclusion and a list of new research question that XIA raises (Section 3.8). The results we present in this chapter are based on our work appeared in [21, 90].

3.1 Summary of Key Approach

The key architectural element that XIA adds to improve evolvability is one we call expressing *intent*. XIA's addresses can simultaneously express both a "new" type of address (or addresses), and one or more backwards compatible pathways to reach that address. This notion is best explained by example: Consider the process of retrieving a particular piece of content (CID) using a network *that provides only host-to-host communication*, much like today's Internet. The source would send a packet destined to a destination network (Dom); the destination network would deliver it to a host; the host would deliver it to a process providing a service (Srv) such as HTTP; and the process would reply with the desired content, as such:



XIA makes this path explicit in addressing, and allows flexibility in expressing it, e.g., "The source really just wants to retrieve this content, and it does not care whether it goes through Dom to get it." As a result, this process of content retrieval might be expressed in XIA by specifying the destination address as a *directed acyclic graph*, not a single address, like this:



By expressing the destination in this way, senders *give flexibility to the network to satisfy their intent*. Imagine a future network in which the destination domain supported routing directly to services [140] (instead of needing to route to a particular host). Using the address as expressed above, this hypothetical network would already have both the *information* it needs (the service ID) and *permission* to do so (the link from the source directly to the service ID). A router or other network element that does not know how to operate on, e.g., the service ID, would simply route the packet to the furthest-along node that it does know (the domain or the host in this example). Note that the sender can use the same address before and after support for service routing is introduced.

XIA terms the types of nodes in the address *principals*; examples of principals include hosts, autonomous domains (analogous to today's autonomous systems), services, content IDs, and so on. The set of principals in XIA is not fixed: hosts or applications can define new types of principals and begin using them *at any time*, without waiting for support from the network. Of course, if they want to get anything done, they must also express a way to get their work done in the current network. We believe that the ability to express not just "how to do it today", but also your underlying intent, is key to enabling future evolvability.

The second difference between XIA and today's Internet comes from a design philosophy that encourages creating principals that have *intrinsic security*: the ability for an entity to validate that it is communicating with the correct counterpart without needing access to external databases, information, or configuration. An example of an intrinsically secure address is using the hash of a public key for a host address [23]. With this mechanism, the host can prove that it sent a particular packet to any receiver who knows its host address. Intrinsic security is central to reliable sharing of information between principals and the network and to ensuring correct fulfillment of the contract between them. It can furthermore be used to bootstrap higher level security mechanisms.

We build security into XIA by requiring that *all* principals are *intrinsically secure* [23]: it should be possible for an entity to validate that it is communicating with the correct principal without needing access to external databases, information, or configuration. Content naming based upon the cryptographic hash of the content is one such intrinsically secure identifier (the receiver of the content can verify that its hash matches the desired name); so also is naming hosts based upon the hash of their public key. Intrinsically secure XIA identifiers directly prevent some security problems (e.g., address spoofing and route hijacking), and can help create effective defenses for others (e.g., network DDoS and cache poisoning attacks). While security innovations are not a focus of this dissertation, intrinsic security is a core architectural theme.

3.2 Foundational Ideas of XIA

We now explore key design ideas that were briefly described in the previous section. XIA is based upon three core ideas for designing an evolvable and secure Internet architecture:

1. Principal types. Applications can use one or more principal types to directly express their intent to access specific functionality. Each principal type defines its own "narrow waist", with an interface for applications and ways in which routers should process packets destined to a particular type of principal.

XIA supports an open-ended set of principal types, from the familiar (hosts), to those popular in current research (content, or services), to those that we have yet to formalize. As new principal types are introduced, applications or protocols may start to use these new principal types at any time, *even before the network has been modified to natively support the new function*. This allows incremental deployment of native network support without further change to the network endpoints, as we will explore through examples in Section 3.4.2 and Section 3.4.3.

2. Flexible addressing. XIA aims to avoid the "bootstrapping problem": why develop applications or protocols that depend on network functionality that does not yet exist, and why develop network functionality when no applications can use it? XIA provides a built-in mechanism for enabling new functions to be deployed piecewise, e.g., starting from the applications and hosts, then, if popular enough, providing gradual network support. The key challenge is: how should a legacy router in the middle of the network handle a new principal type that it does not recognize? To address this, we introduce the notion of a *fallback*. Fallbacks allow communicating parties to specify alternative action(s) if routers cannot operate upon the primary intent. We provide details in Section 3.3.2.

3. Intrinsically secure identifiers. IP is notoriously hard to secure, as network security was not a first-order consideration in its design. XIA aims to build security into the core architecture as much as possible, without impairing expressiveness. In particular, principals used in XIA source and destination addresses must be *intrinsically secure*, i.e., cryptographically derived from the associated communicating entities in a principal type-specific fashion. This allows communicating entities to more accurately ascertain the security and integrity of their transfers; for example, a publisher can attest that it delivered specific bytes to the intended recipients. While the implementation of intrinsic security is not a focus of this paper, we briefly describe the intrinsic security of our current principal types in Section 3.3.1, as well as the specification requirements for intrinsic security for new principal types.

3.3 XIP

XIA facilitates communication between a richer set of principals than many other architectures. We therefore split both the design and our discussion of communication within XIA into two components. First, the basic building block of per-hop communication is the core eXpressive Internet Protocol, or XIP. XIP is principal-independent, and defines an address format, packet header, and associated packet processing logic. A key element of XIP is a flexible format for specifying multiple paths to a destination principal, allowing for "fallback" or "backwards-compatible" paths that use, e.g., more traditional autonomous system and host-based communication.

The second component is the per-hop processing for each principal type. Principals are named with typed, unique eXpressive identifiers which we refer to as XIDs. We focus on host, service, content, and administrative domain principals to provide an example of how XIA supports multiple principals. We refer to the above types as HIDs, SIDs, CIDs, and ADs, respectively. The list of principal types is extensible and more examples can be found in our prior work [21].

Our goal is for the set of mandatory and optional principals to evolve over time. We envision that an initial deployment of XIA would mandate support for ADs and HIDs, which provide global reachability for host-to-host communication—a core building block today. Support for other principal types would be optional and over time, future network architects could mandate or remove the mandate for these or other principals as the needs of the network change. Or, put more colloquially, we do not see the need for ADs and HIDs disappearing any time soon, but our own myopia should not tie the hands of future designers!

3.3.1 Principals

The specification of a principal *type* must define:

1. The semantics of communicating with a principal of that type.

2. A unique XID type, a method for allocating XIDs and a definition of the intrinsic security properties of any communication involving the type. These intrinsically secure XIDs should be globally unique, even if, for scalability, they are reached using hierarchical means, and they should be generated in a distributed and collision-resistant way.

3. Any principal-specific per-hop processing and routing of packets that must either be coordinated or kept consistent in a distributed fashion.

These three features together define the *principal-specific support* for a new principal type. The following paragraphs describe the administrative domain, host, service, and content principals in terms of these features.

Network and *host* principals represent autonomous routing domains and hosts that attach to the network. ADs provide hierarchy or scoping for other principals, that is, they primarily provide control over routing. Hosts have a single identifier that is constant regardless of the interface used or network that a host is attached to. ADs and HIDs are self-certifying: they are generated by hashing the public key of an autonomous domain or a host, unforgeably binding the key to the address. The format of ADs and HIDs and their intrinsic security properties are similar to those of the network and host identifiers used in AIP [23].

Services represent an application service running on one or more hosts within the network. Examples range from an SSH daemon running on a host, to a Web server, to Akamai's global content distribution service, to Google's search service. Each service will use its own application protocol, such as HTTP, for its interactions. An SID is the hash of the public key of a service. To interact with a service, an application transmits packets with the SID of the service as the destination address. Any entity communicating with an SID can verify that the service has the private key associated with the SID. This allows the communicating entity to verify the destination and bootstrap further encryption or authentication.

In today's Internet, the true endpoints of communication are typically application processes—other than, e.g., ICMP messages, very few packets are sent to an IP destination without specifying application port numbers at a higher layer. In XIA, this notion of processes as the true destination can be made explicit by specifying an SID associated with the application process (e.g., a socket) as the intent. An AD, HID pair can be used as the "legacy path" to ensure global reachability, in which case the AD forwards the packet to the host, and the host "forwards" it to the appropriate process (SID). In Section 3.4, we show that making the true process-level destination explicit facilitates transparent process migration, which is difficult in today's IP networks because the true destination is hidden as state in the receiving end-host.

Lastly, *the content principal* allows applications to express their intent to retrieve content without regard to its location. Sending a request packet to a CID initiates retrieval of the content from either a host, an in-network content cache, or other future source. CIDs are the cryptographic hash (e.g., SHA-1, RIPEMD-160) of the associated content. The self-certifying nature of this identifier allows any network element to verify that the content retrieved matches its content identifier.

3.3.2 XIP Addressing

Next, we introduce key concepts for XIP addresses that support the long-term evolution of principal types, the encoding mechanism for these addresses, and a representative set of addressing "styles" supported in XIP.

Core concepts in addressing

XIA provides native support for multiple principal types, allowing senders to express their intent by specifying a typed XID as part of the XIP destination address. However, XIA's design goal of evolvability implies that a principal type used as the intent of an XIP address may not be supported by all routers. Evolvability thus leads us to the architectural notion of **fallback**: intent that may not be globally understood *must* be expressed with an alternative backwards compatible route, such as a globally routable service or a host, that can satisfy the request corresponding to the intent. This fallback is expressed *within* an XIP address since it may be needed to reach the intended destination.

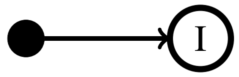
XIP addressing must also deal with the fact that not all XID types will be globally routable, for example, due to scalability issues. This problem is typically addressed through scoping based on network identifiers [23]. Since XIA supports multiple principal types, we generalize scoping by allowing the use of XID types other than ADs for scoping. For example, scaling global flat routing for CIDs may be prohibitively expensive [31, 172], and, thus, requests containing *only* a CID may not be routable. Allowing the application to refine its intent using hierarchical **scoping** using ADs, HIDs, or SIDs to help specify the CID's location can improve scalability and eliminate the need for XID-level global routing. We explore the effectiveness of using this more scalable approach in Section 3.6.3.

The drawback of scoping intent is that a narrow interpretation could limit the network's flexibility to satisfy the intent in the most efficient manner, e.g., by delivering content from the nearest cache holding a copy of the CID, rather than routing to a specific publisher. We can avoid this limitation by combining fallback and scoping, a concept we call (iterative) **refinement** of intent. When using refinement of intent, we give the XID at each scoping step the opportunity to satisfy the intent directly without having to traverse the remainder of the scoping hierarchy.

Addressing mechanisms

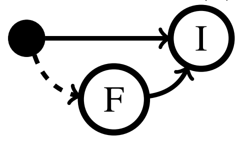
XIA's addressing scheme is a direct realization of these high-level concepts. To implement fallback, scoping, and iterative refinement, XIA uses a restricted directed acyclic graph (DAG) representation of XIDs to specify XIP addresses. A packet contains both the destination DAG and the source DAG to which a reply can be sent. Because of symmetry, we describe only the destination address.

Three basic building blocks are: intent, fallback, and scoping. XIP addresses must have a *single* intent, which can be of any XID type. The simplest XIP address has only a "dummy" source and the intent (I) as a sink:



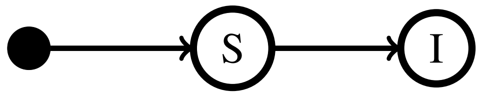
The dummy source (•) appears in all visualizations of XIP addresses to represent the conceptual source of the packet.

A fallback is represented using an additional XID (F) and a “fallback” edge (dotted line):



The fallback edge can be taken if a direct route to the intent is unavailable; we allow up to four fallbacks.

Scoping of intent is represented as:



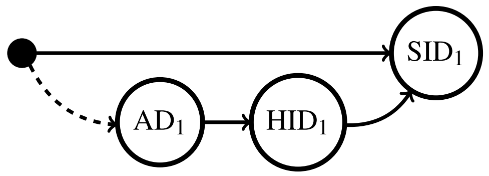
This structure means that the packet must be first routed to a scoping XID S, even if the intent is directly routable.

These building blocks are combined to form more generic DAG addresses that deliver rich semantics, implementing the high-level concepts in Section 3.3.2. To forward a packet, routers traverse edges in the address in order and forward using the next routable XID. Detailed behavior of packet processing is specified in Section 3.3.3.

Addressing style examples

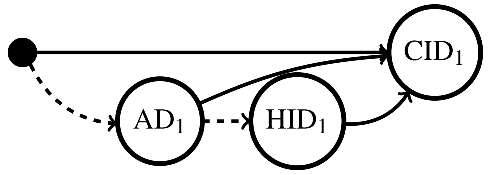
XIP’s DAG addressing provides considerable flexibility. In this subsection, we present three (non-exhaustive) “styles” of how it might be used to achieve important architectural goals.

Supporting evolution: The destination address encodes a service XID as the intent, and an autonomous domain and a host are provided as a fallback path, in case routers do not understand the new principal type.



This scheme provides both fallback and scalable routing. A router outside of AD_1 that does not know how to route based on intent SID_1 directly will instead route to AD_1.

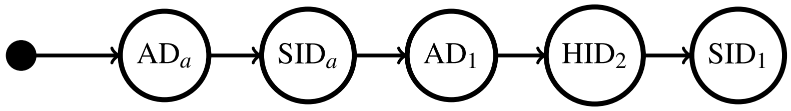
Iterative refinement: In this example, every node includes a direct edge to the intent, with fallback to domain and host-based routing. This allows iterative incremental refinement of the intent. If the CID_1 is unknown, the packet is then forwarded to AD_1. If AD_1 cannot route to the CID, it forwards the packet to HID_1.



An example of the flexibility afforded by this addressing is that an on-path content-caching router could directly reply to a CID query without forwarding the query to the content source. We term this *on-path interception*. Moreover, if technology advances to the point that content IDs became globally routable, the network and applications could benefit directly, without changes to applications.

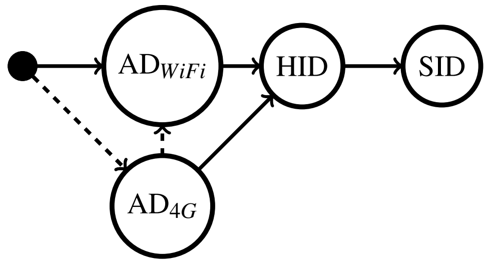
Service binding: DAGs also enable application control in various contexts. In the case of legacy HTTP, while the initial packet may go to any host handling the web service, subsequent packets of the same "session" (e.g., HTTP keep-alive) *must* go to the same host. In XIA, we do so by having the initial packet destined for: $\bullet \rightarrow AD_1 \rightarrow SID_1$. A router inside AD_1 routes the request to a host that provides SID_1. The service replies with a source address bound to the host, $\bullet \rightarrow AD_1 \rightarrow HID_1 \rightarrow SID_1$, to which subsequent packets can be sent.

Loose Source Routing can be accomplished using a linear chain of addresses. The example below, inspired by scenarios from DOA [183], routes packets through a service running in a third party domain, $\bullet \rightarrow AD_a \rightarrow SID_a$:



One concrete case, in use today, occurs when the owner of AD$_1$ subscribes to VeriSign's DoS mitigation filter [181] that "scrubs" packets before they are sent to AD$_1$. (For clarity we show the true destination, but in practice, the final address would be modified so that only the filter can send to AD$_1$.) Today, VeriSign provides this service using BGP, but DAG addresses provide a more general mechanism for source routing and delegation [31].

Multi-homing. Having multiple paths to a sink node allows routers to choose among possible paths if the preferred one fails. An address in this style can arise naturally in datacenters or multi-homed smartphones (e.g., WiFi and 4G).



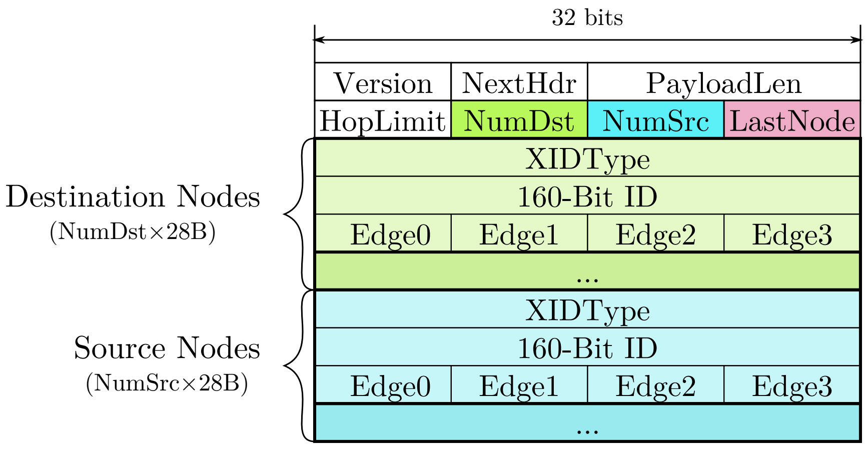


Figure 3.1: XIP packet header.

3.3.3 XIP Header and Per-Hop Processing

This section describes the XIP packet format and per-hop processing that routers perform on packets. Later, in Section 3.6, we show that this design satisfies both the requirements for flexibility and efficient router processing.

Header format

Figure 3.1 shows the header format. Our header encodes a source and a destination DAG, and as a result our address is variable-length—`NumDst` and `NumSrc` indicate the size of the destination and source address. The header contains fields for version, next header, payload length, and hop limit.

Each element in an adjacency list requires 28 bytes: 4 bytes to identify the principal type (XIDType), 20 bytes for the principal's ID, and an array of four 1-byte pointers that represent the out-edges from the node. By topological ordering, the final node in each address (i.e. the one at offset ND or NS) is guaranteed to be a sink node. By definition, each sink has out-degree of zero; thus (to save space) the edge array of the final sink node stores the edges of the *entry node* of the address. Other sinks explicitly store zero out-edges.

Our header stores a pointer, `LastNode`, to the previously visited node in the destination address, for routers to know where to begin forwarding lookups. This makes per-hop processing more efficient by enabling routers to process a partial DAG instead of a full DAG in general.

DAGs are stored as adjacency lists. Each node in the adjacency list contains three fields: an XID Type; a 160-bit XID; and an array of the node indices that represent the node's outgoing edges in the DAG. 8 bits are allocated to represent an outgoing edge. However, the first bit is

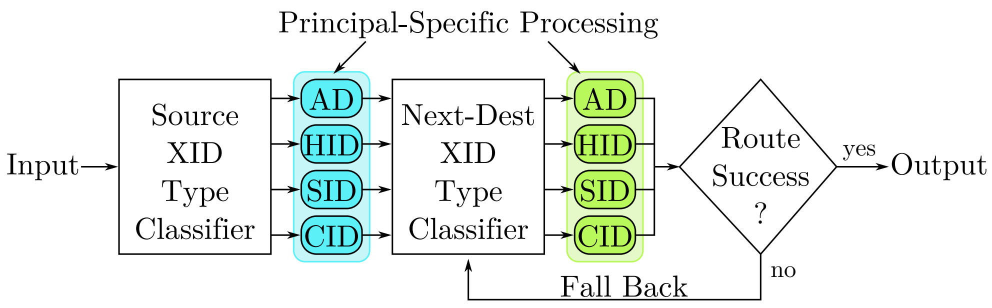

Principal-Specific Processing
Input
Source XID Type Classifier
AD
HID
SID
CID
Next-Dest XID Type Classifier
AD
HID
SID
CID
Route Success ?
yes
Output
no
Fall Back

Figure 3.2: Simplified diagram of an XIP router.

reserved for the purpose of indicating whether the edge was traversed, and the remaining 7 bits represents the index of the node pointed to by the edge. When routers process the packet, it marks the edge that was traversed for diagnosis and path identification purposes.

The adjacency list format allows at most four outgoing edges per node (`Edge0...Edge3`). This choice balances: (a) the per-hop processing cost, overall header size, and simple router implementation; with (b) the desire to flexibly express many styles of addressing. However, we do not limit the degree of expressibility; one can express more outgoing edges by using a special node with a predefined `XIDType` to represent indirection.

Note that our choice of 160-bit XID adds large overhead, which could be unacceptable for bandwidth or power-limited devices. We believe, however, that common header compression techniques [120] can effectively reduce the header size substantially without much computational overhead.

Per-hop processing

Figure 3.2 depicts a simplified flow diagram for packet processing in an XIP router. The edges represent the flow of packets among processing components. Shaded elements are principal-type specific, whereas other elements are common to all principals. Our design isolates principal-type specific logic to make it easier to add support for new principals.

When a packet arrives, a router first performs source XID-specific processing based upon the XID type of the sink node of the source DAG. For example, a source DAG $\bullet \rightarrow AD_1 \rightarrow HID_1 \rightarrow CID_1$ would be passed to the CID processing module. By default, source-specific processing modules are defined as a no-op since source-specific processing is often unnecessary. In our prototype, we override this default only to define a processing module for the content principal type. A CID sink node in the source DAG represents content that is being forwarded to some destination. The prototype CID processing element opportunistically caches content to service future requests for the same CID.

The following stages of processing iteratively examine the outbound edges of the last-visited node of the DAG in priority order. We refer the node pointed by the edge in consideration as the next destination. To attempt to forward along an adjacency, the router examines the XID type of the next destination. If the router supports that principal type, it invokes a principal-specific component based on the type, and if it can forward the packet using the adjacency, it does so. If the router does not support the principal type or does not have an appropriate forwarding rule, it moves on to the next edge. This enables principal-specific customized forwarding ranging from simple route lookups to packet replication or diversion. If no outgoing edge of the last-visited node can be used for forwarding, the destination is considered unreachable and an error is generated.

Optimizations for high performance

The per-hop processing of XIA is more complex than that of IP, which raises obvious concerns about the performance of routers, especially in scenarios using more complex DAGs for addressing. In this section, we show that, despite those concerns, an XIP router can achieve comparable performance to IP routers by taking advantage of well-known optimizations, such as parallel processing and fast-path evaluation.

Packet-level parallelism: By processing multiple packets concurrently, *parallel packet processing* can speed up XIP forwarding. Fortunately, in XIP, AD and HID packet processing resembles IP processing in terms of data dependencies; the forwarding path contains no per-packet state changes at all. In addition, the AD and HID lookup tables are the only shared, global data structure, and like IP forwarding tables, their update rate is relatively infrequent (once a second or so). This makes it relatively straightforward to process packets destined for ADs and HIDs in parallel. While SID and CID packet processing may have common data structures shared by pipelines, any data update can be deferred for less synchronization overhead as the processing can be opportunistic and can always fall back into AD and HID packet processing. This makes CID and SID packet processing parallelizable.

Packet-parallel processing may result in out-of-order packet delivery, which disrupts existing congestion control mechanisms in TCP/IP networks [125]. One solution is to preserve intra-flow packet ordering by serializing packets from the same flow and executing them in the same pipeline processor, or alternatively by using a reordering buffer [82, 191]. An alternative solution is to design to ensure that congestion control and reliability techniques deployed in XIP networks are more tolerant of reordering [38].

Intra-packet parallelism: As discussed earlier, a DAG may encode multiple next-hop candidates as a forwarding destination. Since the evaluation of each candidate can be done in parallel, this address structure also enables *intra-packet parallel processing*. While the different next-hops can be evaluated in parallel, the results of these lookups must be combined and only the highest priority next-hop candidate with a successful lookup should be used. Note that this synchronization stage is likely to be expensive in software implementations and this type of parallelism may be most appropriate in specialized hardware implementations.

Fast-path evaluation: Finally, the XIP design can use *fast-path* processing to speed commonly observed addresses—either as an optimization to reduce average power consumption, or to construct low cost routers that do not require the robust, worst-case performance of backbone routers. For example, our prototype leverages a look-aside cache that keeps a collision-resistant fingerprint of the destination DAG address[1] and the forwarding result (the output port and the new last-node value). When a packet's destination address matches an entry in the cache, the router simply takes the cached result and skips all other destination lookup processing. Otherwise, the router pushes the packet into the original slow-path processing path. Since the processing cost of this fast path does not depend on the address used, this performance optimization may also help conceal the impact of packet processing time variability caused by differences in DAG complexity.

In Section 3.6.1, we show that the combination of these optimizations enables XIP routers to perform almost as well as IP routers. In addition, we show that hardware implementations would be able to further close the gap between XIP and IP performance.

3.4 XIA Addresses in Action

We elaborate how the abstractions introduced in previous sections can be put to work to create an XIP network. The following subsections explain how addresses are created and obtained, and show how XIA's architectural components can work together to support rich applications.

3.4.1 Bootstrapping Addresses

We assume the existence of autonomous domains and a global inter-domain routing protocol for ADs, e.g., as discussed in [23]. We walk through how HIDs, SIDs, and CIDs join a network, and how communication occurs.

[1]The use of collision-resistant hash eliminates the need to *memcmp* potentially lengthy DAGs. The fingerprint is a collision-resistant hash on a portion of the XIP address, which consists of the last-visited node and a few next-hop nodes, effectively representing forwarding possibilities of the flow of the packets. Such an operation is shown to scale up to 222 Gb/s [169] in hardware. We assume this is implemented in the NIC. Modern network interfaces already implement hash-based flow distribution for IPv4 and IPv6 for receiver-side scaling and virtualization.

Attaching hosts to a network: Each host has a public/private key pair. As a first step, each host listens for a periodic advertisement that its AD sends out. This message contains the public key of the AD, plus possibly "well-known" services that the AD provides such as a name resolution service. Using this information, the host then sends an association packet to the AD, which will be forwarded by the AD routers to a service that can proceed with authentication based on the respective public keys.

Advertising services and content: We have designed an XIA socket API which is described in detail in our technical report [22]. We describe here how hosts can advertise services or content using this API.

To advertise a service, a process running on a host first calls `bind()` to bind itself to a public key of the service. This binding inserts the SID (the hash of the service's public key) into the host's routing table so that the service is reachable through the host. Likewise, `putContent()` stores the CID (the hash of the content) in the host's routing table. Since at this point services and content are only locally routable, request packets will have to be scoped using the host's HID, e.g., $\bullet \rightarrow AD \rightarrow HID \rightarrow CID$, to reach the intent.

For a service or a content to be reachable more broadly, the routing information must be propagated. For example, an AD can support direct routing to services and content within its domain using an intra-domain routing protocol that propagates the SIDs and CIDs supported by each host or in-network cache to the routers. Global route propagation can be handled by an inter-domain routing protocol, subject to the AD's policies and business relationships. We leave the exact mechanism, protocol, and policy for principal-specific routing (e.g., content or service routing) as future research.

Obtaining XIP addresses: Now we look into how two parties (hosts, services or content) can obtain source and destination XIP addresses to communicate. As in today's Internet, obtaining addresses is the application's responsibility. Here, we provide a few example scenarios of how XIP addresses can be created.

Source address specifies the return address to the specific instance of the principal (i.e., a bound address). Therefore, when a principal generates a packet, the source address is generally of the form $\bullet \rightarrow AD \rightarrow HID \rightarrow XID$. The AD-prefix is given by the AD when a host attaches to the network; the HID is known by the host; and the XID is provided by the application, allowing the socket layer to create the source address. XIDs will often be ephemeral SIDs. In this case, the socket layer can automatically create an SID when `connect()` is issued to an SID socket without calling `bind()`. This is similar to the use of ephemeral ports in TCP/IP.

Destination address can be obtained in many alternative ways. One way is to use a name resolution service to resolve XIP addresses from human readable names. For example, a lookup of "Google search" in the name resolution service can return a DAG that includes the intent

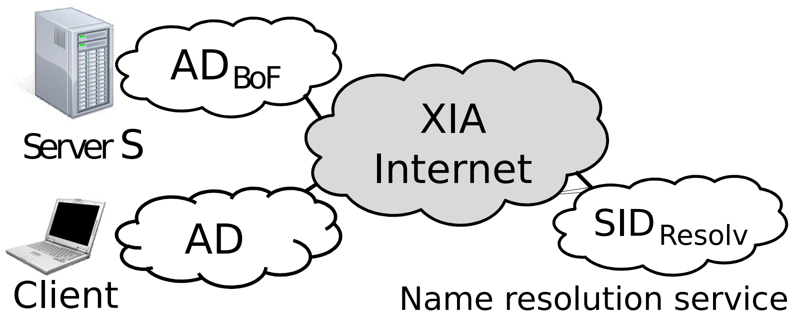


Figure 3.3: Bank of the Future example scenario.

SID along with one or more fallback ADs that host (advertised) instances of the service (similar to today's DNS SRV records). Alternatively, a Web service can embed URLs in its pages that include an intent CID for an image or document, along with the original source ($\bullet \rightarrow AD \rightarrow HID$) or content-distribution SIDs as fallbacks. This information can be in the form of a "ready-to-use" DAG, or as separate fields that can be assembled into a destination address (e.g., iterative refinement style) by the client based on local preferences. For example, the client could choose to receive content via a specific CDN based on the network interface it is using.

Note that we intentionally placed the burden of specifying fallbacks to the application layer. This is because a fallback is an authoritative location of an intent that the underlying network may not know about. Name resolution systems and other application-layer systems are more suitable to provide such information in a globally consistent manner. On the other hand, network optimizations can be applied locally in a much more dynamic fashion. Networks may choose to locally optimize for intent by locally replicating the object of intent and dynamically routing the intent.

A final point is that client ADs may have policies for what addresses are allowed. For example, it may want to specify that all packets entering or leaving the AD go through a firewall. This can be achieved by inserting an $SID_{Firewall}$ in the address, e.g., $\bullet \rightarrow AD \rightarrow SID_{Firewall} \rightarrow HID \rightarrow SID$ for a source address.

3.4.2 Simple Application Scenarios

In this section and the next, we use the example of online banking to walk through the lifecycle of an XIA application and its interaction with a client. Our goal is to illustrate how XIA's support for multiple principals and its addressing format give significant flexibility and control to the application.

In Figure 3.3, Bank of the Future (BoF) provides a secure on-line banking service hosted at a large data center on the XIA Internet. The service runs on many BoF servers and it has a public key that hashes to SID_{BoF}. We assume that all components in BoF's network natively support service and content principals. We focus on a banking interaction between a particular client host HID_C, and a particular BoF process P_S running on server HID_S.

Publishing the service: When process P_S starts on the server, it binds an SID socket to SID_{BoF} by calling `bind()` with its public/private key pair. This SID binding adds SID_{BoF} to the server's (HID_S) routing table, and the route to SID_{BoF} is advertised in the BoF network AD_{BoF}. The service also publishes the association between a human readable service name (e.g., "Bank of the Future Online") and $\bullet \rightarrow AD_{BoF} \rightarrow SID_{BoF}$ through a global name resolution service (SID_{Resolv}).

Connection initiation and binding: When a client wants to connect to the service, it first contacts the name resolution service SID_{Resolv} to obtain the service address. It then initiates a connection by sending a packet destined to $\bullet \rightarrow AD_{BoF} \rightarrow SID_{BoF}$ using the socket API. The source address is $\bullet \rightarrow AD_C \rightarrow HID_C \rightarrow SID_C$, where AD_C is the AD of the client, and SID_C is the ephemeral SID. This packet is routed to AD_{BoF} and then to an instance of SID_{BoF}. After the initial exchange, both processes will establish a session, which includes, for example, establishing a symmetric key derived from their public/private key pairs. Because of this session state, the client needs to continue to communicate with the same server, not just any server that supports SID_{BoF}. To ensure this, the client changes the destination address to $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_{BoF}$, where HID_S is the server that it is currently talking to. After the server binds the service ID to the location of the service, $\bullet \rightarrow AD_{BoF} \rightarrow HID_S$, then communication may continue between $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_{BoF}$ and $\bullet \rightarrow AD \rightarrow HID_C \rightarrow SID_C$.

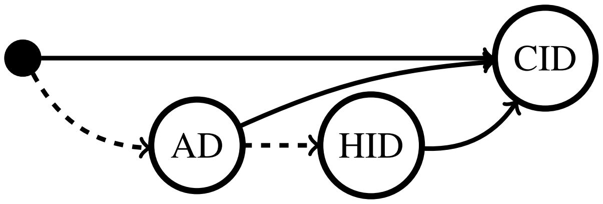
Content transfer: The client can now download content from the on-line banking service. For convenience, we assume that the content being transferred is a Web page. Let us consider both static (faq.html) and dynamic content (statement.html), both of which may contain static images. For *static (cacheable) content*, the SID_{BoF} service will provide the client with the CID_{faq} of the static Web page faq.html along with the CIDs of the images contained in it. The client can then issue parallel requests for those content identifiers, e.g., using $\bullet \rightarrow AD_{BoF} \rightarrow CID_{faq}$ as the destination address for the Web page. The request for *dynamic (non-cachable) content*, e.g., a list of recent bank transactions, is directly sent to SID_{BoF}.

Trust management The above example provides a strong "chain of trust" for obtaining content once the client knows what SID it should contact: the SID is used to validate the content CIDs, which can validate the received content. However, reaching this chain of trust requires bridging the gap from semantically meaningful identifiers ("Bank-of-future") to self-certifying SIDs or CIDs. By creating a foundation by which the rest of a network transaction can be completed securely, XIA's self-certifying principals provide a "narrow waist of trust" in which multiple mechanisms, such as secure DNS, trusted search providers, web-of-trust/PGP models, or physical exchange could all serve as the trust root for subsequent communication.

3.4.3 Support for Richer Scenarios

Using the example above, we now show how XIA's addressing format can support more challenging scenarios such as evolution towards content networking, process migration, and client mobility. Section 3.6.2 provides an evaluation of these example scenarios.

Network evolution for content support: The previous section described how the client can specify static content using scoped intent. Switching to the iterative refinement style (Section 3.3):



means that routing through the specified *AD* or *HID* is optional, and opens the door for *any set of XIA routers to satisfy the client's request for static content.*

The DAG address format supports incremental deployment of content support in an XIA Internet. As a first step, BoF can deploy support for CIDs internally in its network. Even if no ISPs support CIDs, the above address will allow the delivery of the above request (using the *AD*) to the BoF network, where the intent CID can be served.

As the next step, some ISPs may incrementally deploy *on-path* caches. The above address allows them to opportunistically serve content, which may allow them to cut costs by reducing their payments to upstream service providers [15]. Over time, as support for content-centric networking expands, ISPs may make bilateral agreements to enable access to each other's (cached) content. XIA can help leverage such *off-path* caches as well; of course, it would require ISPs to exchange information about cached content and update router forwarding tables appropriately.

Process migration: XIA's addressing can also support seamless process (service) migration through re-binding. Suppose that the server process P_S migrates to another machine, namely HID_T, as part of load balancing or due to a failure; we assume that appropriate OS support for process migration is available. At the start of migration, the route to SID_{BoF} is removed from the old server and added to the new server's routing table. Before migration, the service communication was bound to $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_{BoF}$. After migration, the OS notifies

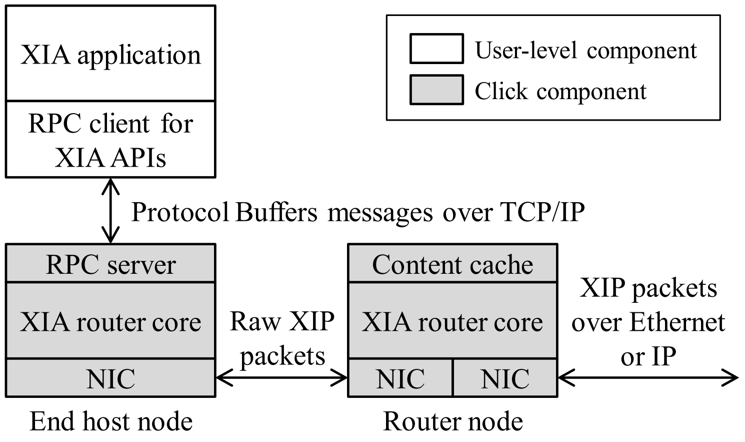


Figure 3.4: Block diagram of our XIA prototype.

the ongoing connections of the new HID, and the binding is changed to $\bullet \rightarrow AD_{BoF} \rightarrow HID_T \rightarrow SID_{BoF}$ at the socket layer. Notification of the binding change propagates to the client via a packet containing the message authentication code (MAC) signed by SID_{BoF} that certifies the binding change. When the client accepts the binding change message, the client updates the socket's destination address. To minimize packet drops, in-flight packets from the client can be forwarded to the new server by putting a redirection entry to SID_{BoF} in the routing table entry of the old server.

Client mobility: The same re-binding mechanism can be used to support client mobility in a way that generalizes approaches such as TCP Migrate [168]. When a client moves and attaches to another AD, AD_{new}, the new source address of the client becomes: $\bullet \rightarrow AD_{new} \rightarrow HID_C \rightarrow SID_C$. When a rebind message arrives at the server, the server updates the binding of the client's address.

Although the locations of both the server and the client have changed in the previous two examples, the two SID end-points did not change. The intrinsic security property remains the same because it relies on the SID. Both the server and the client can verify that they are talking to the services whose public keys hash to SID_{BoF} and SID_C.

3.5 Implementation

Our prototype contains almost the entire architecture. We prototyped XIA routers, principal-specific components, socket APIs for service and content, user-level libraries and a server for name resolution, and applications. We prototyped XIA using the Click modular router [114] and custom built software. Figure 3.4 describe the main components of our prototype. The Click component processes XIA packet forwarding and also contains a simple transport layer for end-hosts. We, therefore, use the Click component for implementing routers as well as end-hosts. At an end-host, Click runs as a process and interfaces with application using our XIA socket API. We implemented an XIA socket API as a user-level library as shown in Figure 3.4.

XIA-enabled end hosts (left) implement user-level applications, and XIA socket API, which invoke Click-based XIA functions over a TCP connection, with API calls encapsulated as Protocol Buffers [81] messages. The XIA host stack is implemented as a Click module consisting of an RPC server, the XIA routing table, and a configurable NIC. XIA routers (right) implement NICs, the XIA router core, as well as a content cache. Our prototype consists of $\sim$4400 SLOC of C/C++ excluding the original Click code. The prototype can run either across the network, encapsulating XIP packets within IP or Ethernet packets, or within a single Click process, in which case the hosts and routers communicate using raw XIP packets. The extensible XIA router core currently performs routing, forwarding, and per-hop processing for ADs, HIDs, CIDs, and SIDs.

It currently supports routing and forwarding for the network and host principal (AD and HID), the content principal (CID), the service principal (SID), and IPv4 principal type for incremental deployment of XIA over IP. The modular design for each principle allows easy addition of future principal types in our prototype router. The IPv4 principal type, for example, was recently added to demonstrate how DAGs can help the incremental deployment of XIA over IP networks [83].

Our prototype is made publicly available at `http://www.github.com/xia-project/xia-core`. Online living documentation is also available at `http://www.xia.cs.cmu.edu/wiki`.

3.6 Evaluation

We evaluate the following three important aspects of XIA:

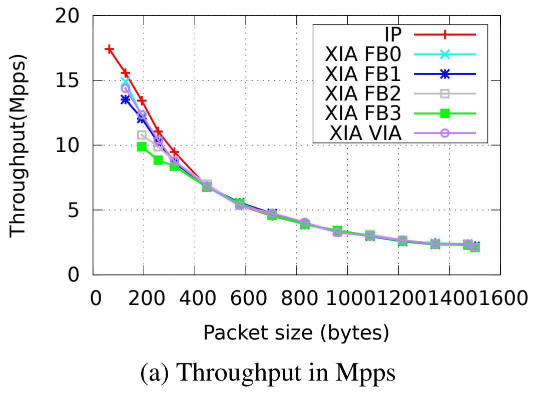
(i) Router processing: Can we scale XIA's packet forwarding performance? In Section 3.6.1, we show that using techniques borrowed from fast IP forwarding, the speed of an XIA router can be made comparable to that of an IPv4 router.

(ii) Application design: How does XIA benefit application design and performance? In Section 3.6.2, we show that use of multiple principal types can simplify application design, and that applications can benefit from the type-specific in-network optimizations allowed by the architecture.

(iii) Scalable routing on XIA: How does forwarding and routing scale with the number of XIA identifiers in use? In Section 3.6.3, we show that XIA routing can scale well beyond support for today's network requirements to more extreme hypothetical scenarios.

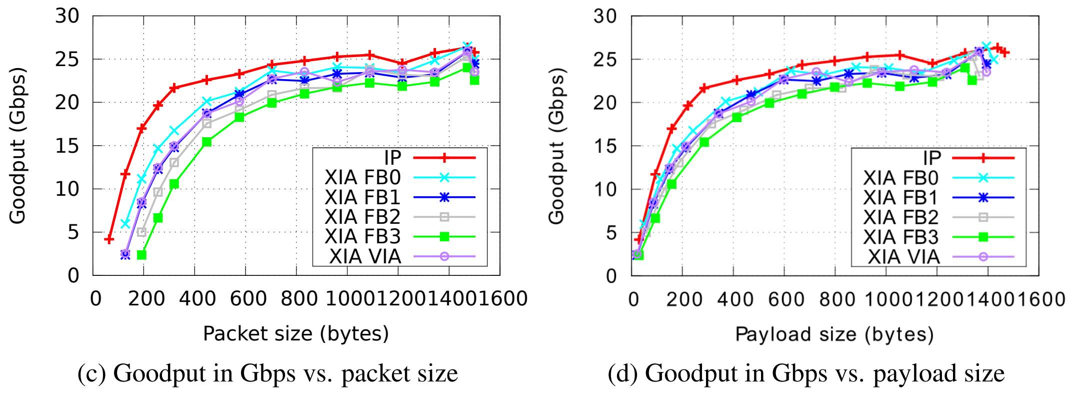
CPU	2x Intel Xeon L5640 2.26 GHz (12MB Cache, QPI 5.86 GT/s)
NIC	2x Intel Ethernet Server Adapter X520-T2
Motherboard	Intel Server Board S5520UR

Table 3.1: Router Hardware Specification



(a) Throughput in Mpps

(b) Throughput in Gbps



(c) Goodput in Gbps vs. packet size

(d) Goodput in Gbps vs. payload size

Figure 3.5: Packet forwarding performance of a software router. The forwarding table has 351 K entries.

3.6.1 Router Design and Performance

We first demonstrate that XIA's forwarding is fast enough for a practical deployment. We show that the packet processing speed of an XIA router is comparable to that of an IP router, and various techniques can be leveraged to further close the performance gap.

Implementation: To measure the packet processing speed, we set up a software router and a packet generator to exploit packet-level parallelism by leveraging their NIC's receiver-side-scaling (RSS) function to distribute IP and XIP packets to multiple CPU cores[2]. The implementation uses the Click modular router framework [114] for processing IP and XIP packets, and PacketShader's I/O Engine (NIC driver and library) [92] for sending and receiving packets to and from the NICs. Table 3.1 provides the specification of the machines.

Forwarding performance: We used a forwarding table of 351K entries based on the Route Views [142] RIB snapshot on Jan 1, 2011. IP uses this table directly; XIA pessimistically uses 351K entries for the AD forwarding table, associating each CIDR block with a distinct AD.

To measure XIA packet processing performance, we generate packets using five different DAGs for the destination address: `FB0`, `FB1`, `FB2`, `FB3`, and `VIA`. `FB0` is the baseline case where no fallback is used. `FB`*i* refers to a DAG which causes the XIA router to evaluate exactly i fallbacks and to then forward based on the $(i+1)$-th route lookup. To force this, we employ a DAG with i fallbacks: the intent identifier and the first $i-1$ fallback identifiers are not in the routing table, but the final fallback is. The last DAG, `VIA`, represents the case where an intermediate node in the DAG has been reached (e.g., arrived at the specified AD). In this case, the router must additionally update the last-visited node field in the packet header to point to the next node in the DAG before forwarding, unlike the other scenarios. Identifiers are generated based on a Pareto distribution over the set of possible destination ADs (shape parameter: 1.2) to mimic a realistic heavy-tailed distribution.

Figure 3.5 (a) and (b) respectively show the impact of varying packet size[3] on packet forwarding performance in packets and bits per second. Figure 3.5 (c) and (d) show the actual goodput achieved excluding the header in each of the above experiments versus packet size including the header and payload size excluding the header. The results are averaged over ten runs each lasting two minutes. Large packet forwarding is limited by I/O bandwidth, and therefore XIA and IP show little performance difference. For small packets of 192 bytes, XIA's `FB0` performance (in pps) is only 9% lower than `IP`. As more fallbacks are evaluated, performance degrades further but is still comparable to that of IPv4 (e.g., `FB3` is 26% slower than `IP`). However, the goodput is much lower due to XIA's large header size. We believe that in cases where the goodput is important, header compression techniques will be an effective solution.

[2]To enable RSS on XIP, we prepend an IP header when generating the packet, but immediately strip the IP header after packet reception.

[3]We include a 14 byte MAC header in calculations of packet size and throughput. We only report the performance of packet sizes in multiples of 64 bytes because of limitations of our underlying hardware. When packet sizes do not align with the 64 byte boundary, DMA performance degrades significantly; we suspect this is triggering a known defect with our Intel hardware. This along with the additional IP and MAC header (34 bytes) and XIP's larger header size (minimum 64 bytes) resulted in a minimum packet size of 128 bytes for XIA.

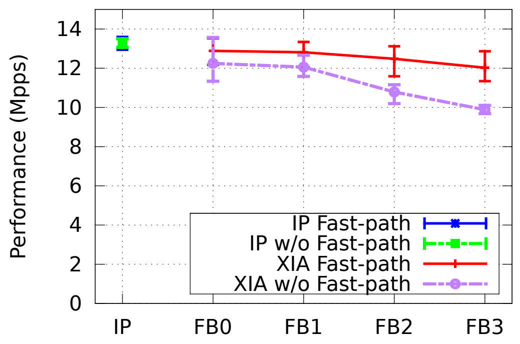


Figure 3.6: Fast-path processing smooths out total cost.

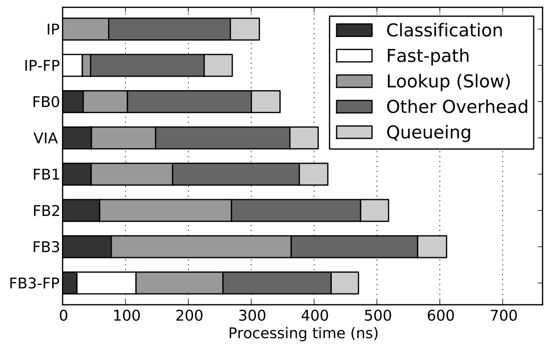


Figure 3.7: In-memory packet processing benchmark (no I/O).

Fast-path processing: We can further reduce the gap between IP forwarding and XIP forwarding using fast-path processing techniques outlined in Section 3.3.3. Our fast-path implementation uses a small per-thread table, which caches route lookup results. The key used for table lookups is a collision-resistant hash of the partial DAG consisting of the last visited node and all its outbound edges. Our choice of hash domain aligns with the fact that a given router operates only on a partial DAG. We assume that the NIC hardware performs this task upon packet reception and the driver reads the hash value along with the packet (Section 3.3.3). We emulated this behavior in our evaluation. We generate 351K unique partial DAGs, and each thread holds 1024 entries in the lookup table. In total, the table holds 14% of these partial DAGs. Figure 3.6 shows the result with and without this fast-path processing for packet size of 192 bytes. Without the fast-path, the performance degrades by 19% from `FB0` to `FB3`. However, with fast-path this difference is only 7%. With a marginal performance gain of IP fast-path, the gap between `FB3` and `IP` fast-path is reduced to 10%.

Intra-packet parallelism: Note that the fast-path optimizations do not improve worst-case performance, which is often critical for high-speed routers that must forward at line speed. High-speed IP routers often rely on specialized hardware to improve worst-case performance. Although we do not have such specialized hardware for XIP, we use micro-benchmarks to estimate the performance that might be possible. The micro-benchmark results in Figure 3.7 show that the route lookup time is dominant and increases as more fallbacks are looked up. Fallbacks within a packet can be processed in parallel using special-purpose hardware to further close the gap. Figure 3.8 shows a comparison between serial and four-way parallel lookup costs without the fast-path processing in our software router prototype, where we deliberately exclude the I/O and synchronization cost in the parallel lookup. The reason we exclude these costs is that

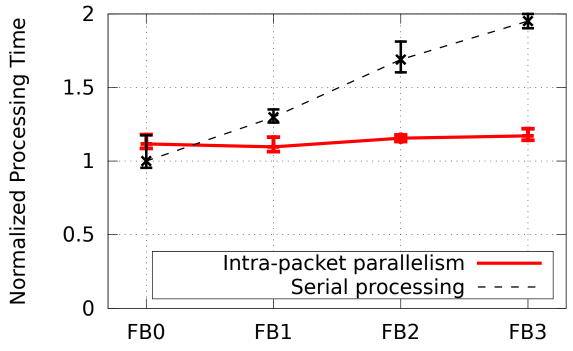


Figure 3.8: Parallel and serial route lookup cost.

	Header	Size
Breakdown	Common header	16 bytes
	Per XID part	28 bytes
Total Size	with 4 XIDs	128 bytes
	with 6 XIDs	184 bytes
	with 8 XIDs	240 bytes

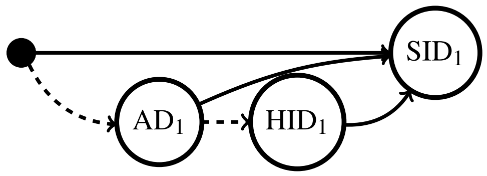
Figure 3.9: XIP Header Size

while the overhead of intra-packet parallelism is high in our software implementation, we believe that such parallel processing overhead will be minimal in hardware router implementations or highly-parallel SIMD systems such as GPU-based software routers [92]. The figure shows that the performance gap between `FB0` and `FB3` can be eliminated with specialized parallel hardware processing.

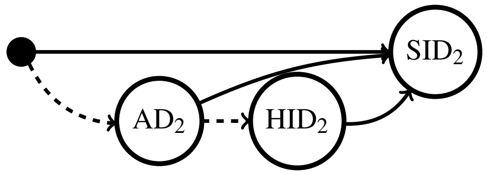
In summary, our evaluation shows that DAG-based forwarding can be implemented efficiently enough to support high speed forwarding operations. We conclude our evaluation on router performance with a discussion on the overhead of the XIA header.

Discussion on header overhead: Table 3.9 shows the header size of XIA headers depending on the number of XIDs in a packet. For example, when there are 6 XIDs in the header the XIP header size is 184 bytes. We believe this will be one of the common cases. An example source and destination address for this case look like the following.

Source Address:



Destination Address:



XIA headers are much longer than IP headers because a) XIDs are long (20-bytes each) and b) there may be multiple XIDs in an address. The former is mainly due to our support for intrinsic security, and the latter for evolution (i.e., fallback). We noted earlier that on bandwidth constrained links header compression will be effective. However, in the core network where high-speed processing is required, header compression may not be desirable. Thus, we perform a back-of-the-envelope calculation on the bandwidth overhead XIA header might introduce. We naively estimate the traffic overhead if XIA were to replace the current Internet today with the following assumptions. 1) Assume the XIA core network sees the same traffic as the IP core. 2) Assume all XIA packets have 6 XIDs in total (3 for dest and 3 for source). 3) Average packet size on the network is 402 bytes [14]. Note that we do not account for any traffic reduction (or overhead) due to the use of content caching or other in-network optimizations. We also do not account for any possible changes in name resolution and application layer protocols such as HTTP. With these simple assumptions, our calculation show that XIA increases the traffic by 40% compared to IPv4 and 36% to IPv6.

3.6.2 Application Design and Performance

We now evaluate XIP's support for applications by implementing and evaluating several application scenarios. While it is hard to quantitatively measure an architecture's support for applications, we demonstrate that XIA is able to retain many desirable features of the current Internet and subsume the benefits of other architectures [102, 115, 140]. Through this exercise, we show that XIP's flexible addressing and support for multiple principals simplifies application design, accommodates network evolution, and accelerates application performance.

Implementation: Using our XIA socket API, and our Click implementation of the XIP network stack, we implement in-network content support, service migration, and client mobility. To closely model realistic application behavior, we built a native XIA Web server that uses service and content principals, and a client-side HTTP-to-XIA proxy that translates HTTP requests and replies into communications based on XIP service and content principals. This proxy allows us to run an unmodified Web browser in an IP network. Implementing the proxy and server using our socket API required only 305 SLOC of Python code, and in-network content support took 742 SLOC of C++, suggesting that the principal-specific socket API and router design facilitates software design.

We now evaluate scenarios described in Section 3.4.2 and Section 3.4.3.

Content transfer and support for evolution: We created a wide-area testbed spanning two universities. The service side (CMU) operates the XIA Web server, and the client side (UWisc) performs Web browsing. The server serves dynamic and static Web pages, each of which consists of either a dynamic or static HTML file and a static image file (15 KB in each Web page). We use the addressing style described in Section 3.4.3.

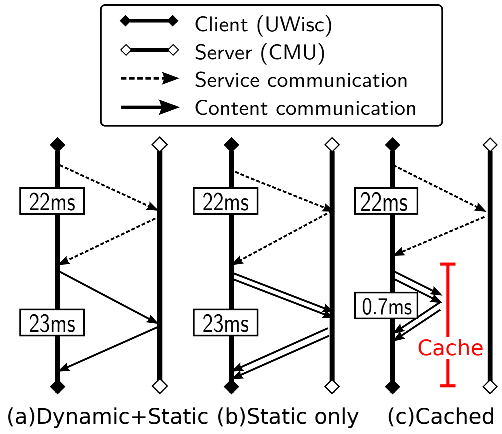


Figure 3.10: Content transfer.

Baseline content transfer: To highlight the application's use of multiple principals, we first show the baseline case where the content request takes the fallback path and the content is fetched from the origin server. Figure 3.10 (a) and (b) respectively show the steps and time taken to retrieve the dynamic and static Web page. When the document is dynamic (a), the server uses service-based communication to directly send the document; it also transmits a CID for the client to access the static image. In the static case (b), the service sends CIDs for both the document and the image, and the client fetches them using content communication. In both cases, it takes two round-trip times (45 ms) to retrieve the content.

Evolution: To highlight XIA's support for evolution, we consider two scenarios: 1) An in-network cache is added within the client's network without any changes to the endpoints, and 2) endpoints do not use XIA's fallback; clients first send the request to the primary intent, then redirect the intent to the original server after a timeout.

Figure 3.10(c) shows that content retrieval becomes faster (22.7 ms) just by adding an in-network cache without any changes (i.e., request packets in (b) and (c) are identical). This is enabled by iterative refinement-style addressing, which permits the intent to be satisfied at any step, while allowing it to fallback to the original server when it's not.

In the second scenario, the source does not use a fallback address and only uses a CID in its DAG address. The completion time becomes much worse (87 ms, not shown in the figure). The initial content request is dropped by the network because an intermediate router does not know how to route to the content. After a timeout, the application redirects the content request to the original server using the address: $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow CID$.

Service migration: As shown in Section 3.4.3, XIA supports seamless service migration with re-binding. To evaluate our service migration support, we run service SID_{BoF} on a virtual machine, which initially resides in a host machine (HID_S), and later migrates to another host (HID_T). We

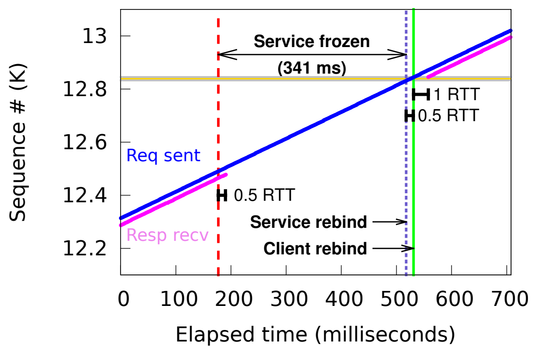


Figure 3.11: Service migration.

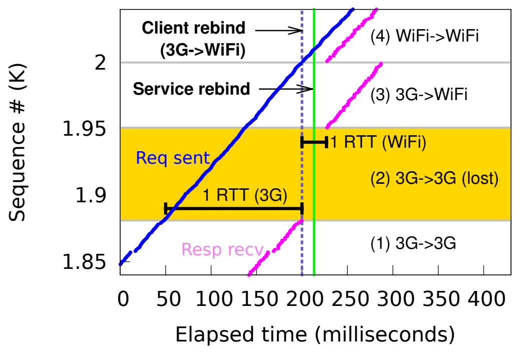


Figure 3.12: Client mobility.

use KVM's live migration [1] to implement stateful migration of a running process; we move the VM along with the process running the service.

Figure 3.11 shows the timeline of service migration. The client makes continual requests to the service, who responds to these requests. Initially this session is bound to: $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_C$. When live migration is initiated (not shown in the figure), the service continues to run on HID_S, but the underlying VM starts copying its state to the new host in the background. When most of the state is transfered to HID_T, SID_{BoF} (and the VM) is *frozen* to perform the final state transfer.

After the final state transfer, the VM and the service resume at HID_T. At this time, the service rebinds to HID_T (`Service rebind`). However, the client is still directing queries to HID_S, because it is not aware of the rebinding. The service then notifies the client of the new binding: $\bullet \rightarrow AD_{BoF} \rightarrow HID_T \rightarrow SID_C$. When the client receives this message, it rebinds to the new DAG after verification (`Client rebind`). The rebound client then starts sending subsequent requests to the new address. After one round-trip time, the client receives responses from the service. The communication is interrupted in between the rebinds. The duration of this interruption due to XIA address rebinding is minimal (the shaded region; about 1.5 RTT). The downtime due to VM migration (freezing) is much longer (341 ms) in our experiment. In a more sophisticated migration implementation, packet drops can be eliminated by buffering all packets received during the entire service interruption period at HID_S and redirecting them to the service at HID_T when it completes rebinding.

Client mobility: In this scenario, a client moves from a 3G network (RTT=150ms) to a WiFi network (RTT=25ms). The client is using a simple ping-like echo service on a server during the move. After connecting to the WiFi network, the client sends a cryptographically signed rebind message, notifying the server of the new binding: $\bullet \rightarrow AD_{new} \rightarrow HID_C \rightarrow SID_C$.

Figure 3.12 shows the timeline of events and the sequence numbers of packets from the echo service; blue dots indicate the time and the sequence number of the requests sent by the client, and purple dots those of the responses received by the client. Different regions (regions 1 to 4) indicate the network from which the request is sent from the client and to which the response is sent by the service. Only the packets in flight at `Client rebind` are lost (shaded area, region (2)) since they are sent to the 3G network, to which the client is no longer connected. However, after `Service rebind`, responses are sent to the WiFi network. Note that packet loss can be eliminated if the 3G network forwards these packets to the new location; this can be done by updating the routing entry for the client's HID on the 3G network. One round-trip time after `Client rebind`, the client begins receiving responses from the service. However, due to the difference in the round-trip times of the 3G and the WiFi network, packet reordering happens at the server-side, and the service responds to requests made from the 3G network prior to client rebind as well as from the WiFi network after client rebind for a short period of time (regions (3) and (4)).

In summary, XIP's flexible addressing supports seamless network evolution and provides adequate application control, while keeping the application design simple.

3.6.3 Scalability

We now turn to a discussion of scalability. First, we demonstrate that XIA works as a "drop-in" replacement for IP, using AD and HID routing much as today's network uses CIDR blocks and ARP within subnets. We then examine scaling to plausible near-term scenarios, such as deploying on-path or near-path content caching, in which routers maintain large lists of content chunks stored in a nearby cache. We conclude by looking at long-term deployment ideas that would stretch the capabilities of near-term routers, such as flat routing within a large provider such as Comcast; and those requiring further advances to become plausible, such as global flat routing.

XIA at today's Internet scale. IP relies on hierarchy to scale: The "default-free" zone of BGP operates on only a few hundred thousand prefixes, not the 4 billion possible 32-bit IP addresses. XIA's AD principals can provide the same function, and we expect that at first, core routers would provide only inter-AD routing. Similar to AIP, the number of ADs should be roughly close to that of today's core BGP routing tables. Section 3.6.1 demonstrated that handling XIP forwarding with a routing table of this size is easy for a high-speed software router.

HID routing within a domain is likely to have a larger range, from a few thousand hosts, to several million. Many datacenters, for example, contain 100K or more hosts [134]. We expect that extremely large domains might split into a few smaller domains based upon geographic scope. XIP routing handles these sizes well: A 100K-entry forwarding table requires only 6.2MB in our (not memory optimized) implementation. Boosting the number of table entries from 10 K

	Forwarding Table	Public Key Store	Content Store
HIDs (100M)	6.25 GB	50 GB	-
SIDs (2 Billion)	125 GB	1 TB	-
CIDs (YouTube 2009)	21 TB	-	168 PB
CIDs (WWW 2010)	At least 227 TB	-	-

Table 3.2: Forwarding table size and public key store size of an AD with 100 million hosts.

to 10 M decreases forwarding performance by only 8.3% (for Pareto-distributed flow sizes with shape=1.2) to 30.9% (for uniformly distributed flows); this slowdown is easily addressed by adding a small amount of additional lookup hardware.

While we believe naive flat forwarding will work in many networks, highly scalable systems, such as TRILL [178] and SEATTLE [110] can also be used.

Supporting tomorrow's scale. XIP provides considerable flexibility in achieving better scalability. First, XIP is not limited to a single level of hierarchy; a domain could add a second subdomain XID type beneath AD to improve its internal scalability. Second, other identifier types can be used to express hierarchy. For example, related content IDs can be grouped using a new principal (e.g., into an object ID). Doing so requires no cooperation from end-hosts: they must merely change the DAG address returned by naming to reflect the new hierarchy.

PortLand [134] suggests that a new layer-2 routing and forwarding protocol is needed to support millions of virtual machines (VMs) in data centers. In XIA, we can create a $HID_{Host} \rightarrow HID_{VM}$ hierarchy[4] that reduces the number of independently routable host identifiers and the forwarding table size by 1 to 2 orders of magnitude (we can also omit HID_{VM} by exposing all services in guest VMs to the host if the host's forwarding table is not overloaded, as in the service migration example of Section 3.6.2).

Hierarchy reduces forwarding table size at the cost of more bandwidth for headers; in XIA, this cost is modest: adding an extra XID requires 28 bytes/packet, but this addition might greatly simplify network components. Also, adding hierarchy in XIA does not hinder evolution. Using iterative refinement addressing, later networks could choose to ignore the hierarchy and route directly to the intended destination. This is the case even for hosts—were memory ever to become so cheap that global host-based routing was practical. More likely, however, this allows optimizations: A network might be willing to store a "redirect pointer" for a recently departed mobile host, similar to some optimizations proposed for Mobile IP. These pointers would operate

[4]For readers concerned about the performance penalty for having to go through a host, a simple filter can be inserted in modern NICs for direct access to VMs. Similar functionality for IPv4 matching is already implemented in many server-class NICs.

on the host ID independent of hierarchy, but would be limited in number and duration.

On-path content caching and interception is a concrete example of opportunistic in-network optimization. A router forwards content (chunk) responses to a cache-engine, which caches the chunk. The router then intercepts the requests for chunks that are in the cache, and forwards these requests to the cache-engine, which serves the content. The forwarding table size can be small in this case, since it only contains information about the local cache (a 4 GB cache with an average object size of 7.8 KB [49] requires a forwarding table of size $<$32 MB.).

Hypothetical extremes. We now look at some extreme scenarios to better understand what kind of advances that are needed to make them feasible in XIA.

Some of the largest organizations have tens of millions of hosts. The largest cable operator has about 23 million customers [57], and Google is preparing to manage 10 million servers in the future [80]. YouTube (2009) has 1.8 billion videos [50] (large objects), and the World Wide Web (2010) has at least 60 billion pages (small objects) [65]. Table 3.2 shows the space needed to route on HIDs and SIDs in an AD with 100 million hosts, and CIDs for YouTube and the Web[5]. Even though for large organizations, the HID and SID tables can fit in DRAM, its cost might be prohibitive if all devices had such a large table.

XIA does not make these extreme designs possible today; they may require non-flat identifiers or inexact routing [196], or techniques that have not yet been developed. Instead, XIP's flexible addressing makes it possible to take advantage of them if they are successfully developed in the future.

3.7 Related Work

Substantial prior work has examined the benefits of network architectures tailored for specific principal types. We view this work as largely complementary to ours, and we have drawn upon it in the design of individual principal types. The set of relevant architectural work is too large to cite fully, but includes proposals for content and service-centric networks, such as CCN [102], DONA [115], TRIAD [84], Serval [140], and many others. Many innovations linger within this prior work that we have not yet incorporated from a desire to err on the side of supporting multiple principals over an exhaustive implementation of a single principal type.

[5] We estimate the number of SIDs by assuming that each host uses up to 20 ephemeral SIDs at a time on average. For large objects, the forwarding table is 0.0125% of the content size assuming a chunk size similar to BitTorrent. For the Web, the average object size is (7.8KB) [49]. We then double the size assuming a 50% load factor hash table for storage.

On network protocol evolution: EvoArch [16] presents an abstract model for evolution of layered protocol stacks. It argues that an evolvable network architecture should have multiple surviving protocols at the waist and suggest three ways to accomplish this through modeling. We believe XIA incorporates mechanism to satisfy all three requirements for an evolvable architecture: 1) XIA decreases the lethal rate (z) of protocols by making protocols to function without the deployment within the network; i.e. principals can survive only at the end-points. 2) XIA encourages principals of non-overlapping functionality to be defined, which increases the chance of multiple protocols' or principals' coexistence. 3) Principals capture common usage models of applications, which increases the generality.

Extensibility through indirection: One approach used in prior work to support multiple principal types devises solutions that leverage indirection, such as through name resolution or via overlays. For example, the Layered Naming Architecture [31] resolves service and data identifiers to end-point identifiers (hosts) and end-point identifiers to IP addresses. *i3* [172] uses an overlay infrastructure that mediates sender-receiver communication to provide enhanced flexibility. Like XIA, these architectures improve support for mobility, anycast, and multicast, but at the cost of additional indirection. Of course, an advantage of these approaches that leverage indirection over XIA is their ease of deployment atop today's Internet. DONA eliminates the cost of indirection by forwarding a packet in the process of name resolution. Like DONA, XIA separates the name (intent) from its locations. However, XIA differs in two key aspects: 1) XIA makes translation from name to location as part of packet forwarding and combines it with principal-specific processing. 2) DONA relies on the network for correct translation from a name to its location, but XIA relies on the backwards-compatible paths provided by the application.

Extensibility through programmability has been pursued through many efforts such as active networks [176], aiming to ease the difficulty of enhancing already-deployed networks. The biggest drawback to such approaches is resource isolation and security. In contrast, XIA does not make it easier to program new functionality into existing routers—although an active networks approach could potentially be applied in tandem.

Architectures that evolve well have been a more recent focus. OPAE [78] shares our goals of supporting evolution and diversity, but their design focuses primarily on improved interfaces for inter-domain routing and applications, whereas XIA targets innovation and evolution of data plane functionality within or across domains. OPAE enables a domain to adopt any architecture independent to others and thus allows heterogeneous architectures to coexist by separating inter-domain addressing from intra-domain addressing. However, it does not specify how to evolve a domain incrementally, while XIA's fallback mechanism allows incremental adoption of new principal types by design. OPAE also allows multiple inter-domain service models to coexist.

However, it does not provide any mechanism for incremental deployment of inter-domain service models. Plutarch [61] is similar to OPAE in that it also allows heterogeneous networks to coexist, but they require an explicit translation mechanism called an interstitial function between two networks that peer with each other. The interstitial function is similar to fallbacks in that ensures packets to be exchanged between two different networking models. However, our fallbacks are hints and the actual per-hop behavior of a packet dynamically adapts to the deployment status of a principal type and route availability. Role-based architecture [43] promotes a non-layered design where a role provides a modularized functionality—similar to XIA's principal-specific processing. However, it is unclear how it allows incremental deployment of new functionality like XIA does.

Ratnasamy *et al.* propose deployable modifications to IP to enhance its evolvability [152]; but does not admit the expressiveness afforded by XIA. Others have argued that we should concede that IP (and HTTP) are here to stay, and simply evolve networks atop them [148]. However, this is not a solution in the long run; EvoArch [16] points out that merely pushing the narrow waist from layer 3 to layer 5 would result in yet another "ossified" layer.

Finally, *virtualizable networks* admit evolution by allowing many competing Internet instances to run concurrently on shared hardware [24, 166, 186]. Clark *et al.* present a compelling argument for the need to enable competition at an architectural level [56], which we internalized in our support for multiple principals. We believe that there are substantial benefits to ensuring that all applications can communicate with all other applications using a single Internet instance, but virtualizable networks offer the potential for stronger isolation properties and to support farther-reaching architectural changes than XIA (e.g., such as moving to a fully circuit-switched network). Substantial research remains in moving these architectures closer to fruition and in comparing their strengths.

Borrowed foundations: *Self-certifying identifiers* were used in Host Identity Protocol (HIP) [93], as well as systems such as the Self-Certifying File System (SFS) [130]. AIP [23] used self-certifying identifiers for both network and host addresses, as XIP does, to simplify network-level security mechanisms. Several content-based networking proposals, such as DONA [115], use them to ensure the authenticity of content. Serval [140] similarly names services based upon the hash of a public key. These works demonstrate the substantial power of these intrinsically secure identifiers, which XIA generalizes to an architectural requirement.

Addressing schemes: Our DAG encodes multiple identifiers in to a single address. This idea appears in other designs. The flexibility of DAG-style addressing has been used elsewhere, notably Slick Packets [139]. Our addressing scheme uses this concept in a new way, to provide support for network evolution.

MobilityFirst [162] propose to use multiple identifiers as an address similar to our design. Qualitatively, our XID maps to their Service ID and GUID. Service is similar to our XID type (principal type) in that it defines how the packet should be treated in the network. For example, it tells whether the GUID is a type of content or a service. The GUID is a globally unique identifier that identifies an entity in the network. The difference is that they do not deal with incremental deployment of new functionality in the network.

3.8 Chapter Summary

In this chapter, we explored a clean-state network design that allows graceful introduction of new network-level functionality. We argued that in order to introduce new functionality in a principled way, we must support the evolution of network-level abstractions. We presented XIA, an architecture that addresses the problem of evolution from the ground up.

XIA builds upon the TCP/IP stack's proven ability to accommodate technology evolution at higher and lower network layers by incorporating evolvability directly into the narrow waist of the network. XIA supports expressiveness, evolution and trustworthy operation through the use of an open-ended set of principal types, each imbued with intrinsic security. The centerpiece of our design, XIP, is a network-layer substrate that enables network innovation, and has the potential to support and amalgamate diverse sets of ideas from other clean-slate designs.

To achieve evolution, XIA supports diversity and incremental deployment of network-level functionality and abstractions. Principal types allow us to define a new abstraction and per-hop packet processing in the network. Our flexible addressing scheme allows incremental deployment of new abstractions and functionality. Finally, through a prototype implementation and evaluations, we showed that XIA supports evolution without sacrificing efficiency. In other words, XIA's basic packet forwarding can scale up to multiple-10Gbps similar to the performance of IP.

Limitations and Future Research: Today's Internet is a global-scale ecosystem of business and consumer entities with complex interactions that have evolved significant over time. Today's best practices are the result of innovations that went through the test of time and market efficiency. Many seemingly good ideas and businesses have flourished, but many of them also did not prosper. Many technical and non-technical problems are ahead in transforming the current IP to a more intelligent infrastructure that effectively integrates various types of resources. This chapter presented the design and evaluation of an evolvable network architecture. The dissertation also studies important related problems in this space.

However, this dissertation does not address all problems of the design and deployment of future Internet architectures. We outline three major areas that need to be studied in the future.

- In-depth exploration of sub-components: This dissertation explores core design principles in designing an integrated network architecture that accommodates the evolution in technology as well as the usage model of the network. This chapter explored the key design and evaluation of the core network architecture that allows evolution of the service model. However, this dissertation does not fully cover the details of all extensible sub-components of the architecture. Many of them deserve full attention on their own. For example, substantial research remains to address issues such as crafting an intelligent content distribution network that takes advantage of distributed content caching; devising a more complete and scalable name resolution service for XIA; incorporating intelligence in service-oriented communication for better load distribution; adapting or engineering suitable intra- and inter-domain routing protocols for HIDs and ADs; and incorporating trustworthy protocols that leverage intrinsic security. We view the large scope of future work as an architectural strength, showing that XIA enables a wealth of future innovations in routing, security, transport, and application design, without unduly sacrificing performance in the pursuit of flexibility.

- Finding the killer application: Another key challenge for XIA and future Internet research is to come up with a new principal type or new functionality that enables a “killer application”. While there are many proposals for future Internet architectures, there is no common agreement in what are the killer applications for the future Internet, at this stage. XIA itself does not make innovation itself straightforward and easy. Substantial research is required to find a “killer application” for the future Internet. The contribution of this chapter, however, is likely to be still valuable regardless because it provides a clean path towards the evolution of the network. XIA has a unique advantage in that it is designed for evolution and, therefore, can incorporate innovative ideas yet to be learned. More specifically, XIA can provide an aid in solving the problem in two ways: 1) XIA’s evolvable per-hop processing allows us to experiment with radical new ideas. 2) The nature of our network design allows us to incorporate ideas from other research in future Internet. Thus, we can provide a single architecture that supports multiple killer applications.

- Large-scale deployment: We, a team of XIA researchers, have built a full prototype of the architecture including the router, socket API, name resolution service, sample applications, and a simple transport layer. It is made publicly available at `https://github.com/xia-project/xia-core`. So far, the evaluation of the architecture was limited to small-scale local deployments, micro-benchmark, WAN deployment over IP and on GENI. However, to fully evaluate an Internet architecture a large-scale deployment is necessary. We expect to bring the architecture into a mature level by identifying and solving some of the practical problems that might surface during large-scale deployment and experimentation.

We envision a two-track approach in XIA's deployment. First, we plan to recruit lager user base for academic or research users. We have already released source codes as well as virtual machine packages that users can easily install on their machines. We also plan to provide an XIA backbone and essential services such as a global name resolution service. Also, future research is going on to facilitate XIA's deployment over IP. Next, as a promising direction for large-scale deployment and future research, we would like to explore domain-specific application of XIA in different environments. Potential domain-specific instantiation of XIA may include Supervisory Control and Data Acquisition (SCADA) networks, mobile networks, and data-center networks. We believe they are more agile and, in many cases, have a clear boundary or separation from the wide-area IP network. Two main features of XIA makes it attractive in these environments: 1) The intrinsic security may provide easy ways to bootstrap many of the security problems in SCADA networks. 2) XIA's flexible addressing and type-specific processing can help in dealing with service provisioning/migration and in-network computation within data-center networks.

Chapter 4

On End-point Evolution

As networks provide new functionality with integrated resources, complicated interactions can occur between the network and the end-points. In the previous chapter, we presented a network architecture that incorporates diverse interactions. The resource integration also has significant implications on end-points. In the old model, where the network was a dumb data pipe, all other interesting features had to be handled purely at the end-points. However, when the network provides new functionality, this is no longer true. In fact, some of the functions traditionally implemented at the end-points can now be handled much more efficiently in cooperation with the network. This shows that when networks evolve to provide new service models, end-points also have to evolve to better take advantage of the new network. To highlight this implication, this chapter offers a compelling case study.

We present two major ideas in this chapter:

1. We must reconsider how we implement some of the most essential networking features at the end-points when networks provide new functionality. Adapting the behavior of end-points to the underlying network can provide significant performance and usability benefits.

2. XIA facilitates the adaptation of end-points' behaviors by allowing end-points to identify the functionality provided by the network.

Chapter outline: This chapter offers a case study for end-point adaptation in an environment where networks provide new functionality. In Section 4.1, we first look at how we can introduce an example new function of redundancy elimination into an XIA network. We then demonstrate that XIA facilitates identification of new functionality and adaptation of end-points based on the functionality provided by the underlying network. Section 4.2 presents a motivation for the end-points' adaptation using a case study of real-time data delivery. Section 4.3 provides a summary of related work. We then explore the core design ideas for a novel reliability scheme for real-time data delivery in Sections 4.4 and 4.5. In Section 4.6, we explore how this new reliability scheme can be applied to non-XIA networks and highlight the differences. We present extensive evaluation results in Section 4.7 and summarize the chapter in Section 4.8. The results we present in this chapter are based on our work appeared in [89, 90].

4.1 Identification and End-point Adaptation

This chapter explores the implications of resource integration in the context of XIA. This chapter focuses on exploring two major implications of XIA on end-points:

- XIA enables the end-points to explicitly invoke new network functionality and identify the functionality supported by the path. This is because XIA allows new network functionality to be introduced explicitly with new abstractions. To effectively demonstrate this, we first take as an example an existing in-network function, redundancy elimination, and show how we can introduce the new function in an "XIA way". We further show that end-points can now identify the functionality provided by the path.
- The end-points' ability to identify new functionality, in turn, facilitates end-points' adaptation of their behavior to the underlying network. Continuing on with our exploration on redundancy elimination, we demonstrate how we can adapt end-points' strategy when the underlying network supports this functionality. In particular, we look at how we can redesign the reliability scheme of end-points for robust real-time communication.

4.1.1 Background and Current Practice

We start by providing background on redundancy elimination (RE). RE has been proposed by Spring et al. [170] and commercialized by the industry in the product form of WAN optimization. Recently, Anand et al. [19] proposed the use of redundancy elimination as a network-wide primitive, creating a redundancy elimination network. In Anand et al's [19] design, RE is deployed across individual ISP links. An upstream router remembers packets sent over the link in a cache (each cache holds a few tens of seconds' worth of data) and compares new packets against cached packets. It encodes new packets on the fly by replacing redundant content (if found) with pointers to the cache. The immediate downstream router maintains an identical packet cache, and decodes the encoded packet. RE is applied in a hop-by-hop fashion.

RE encoding and decoding are deployed on the line cards of the routers as shown in Figure 4.1. Decoding happens on the input interface before the virtual output queue, and encoding

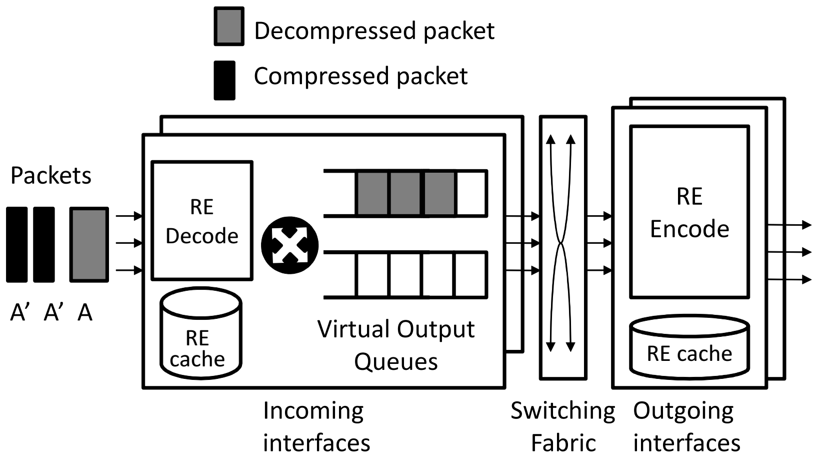


Figure 4.1: Redundant Packet Transmission in a redundancy elimination router.

happens on the output interface. The router's buffers (virtual output queues) contain fully decoded packets.

Today, RE is deployed as middleboxes that introduce this new functionality in a transparent way. WAN optimization, its product form, is reportedly used by 56% of large north American enterprise networks [137]. In such deployments, end-points do not know whether they are traversing an RE network or not. However, we argue that in-network functionality should be introduced explicitly. In this section, we look at how we can introduce RE in an XIA network and explore the implication of this explicit introduction.

4.1.2 Explicit Invocation and Identification of Functionality

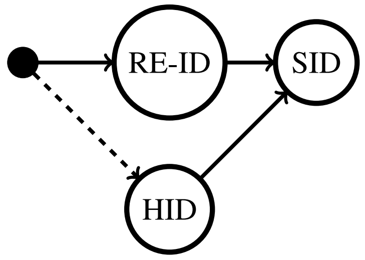
XIA allows us to introduce new functionality in an explicit way by defining a new principal type, which enables explicit invocation and identification of network-level functionality by the end-points. First, we show how we can define an RE principal type to introduce redundancy elimination more explicitly. Then, we explore how end-points can use the RE functionality using our flexible addressing. Finally, we discuss potential benefits of this approach and the possibility of an end-point's strategy adaptation to realize this benefit.

RE as a new principal type: As described in Section 3.3.1, in order to introduce a new principal type we need to define its three features: semantics, a method for allocating XIDs, and principal-specific per-hop behavior.

We suggest the following definitions of the RE principal type:

- *Semantics:* The semantics of an RE-type is similar to the host principal type in that sending a packet to an RE principal means that you want to send a packet to the host that RE principal is referring to.
- *XIDs:* RE principal type has its own unique type. Each host has a unique XID within the RE principal type, but shares its XID with the host principal type for convenience. In other words, XID of the RE principal type (RE-ID) is the same as HID of the host.
- *Principal-specific per-hop processing:* Per-hop processing for the RE principal type applies redundancy elimination to packets on every hop.

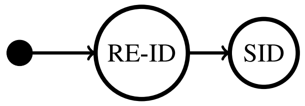
End-point behavior: End-points can now explicitly invoke the functionality that they want in the network. For example, end-points can use the following DAG to invoke the new RE function:



The DAG requests the network to apply RE in forwarding this packet to the specified host. Once the packet gets to the host, it is delivered to a service. However, since the underlying network may not support RE, we use the regular host identifier as a fallback.

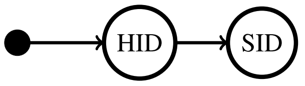
Once the packet gets to the destination, the destination host can inspect the packet header to identify the path taken. As described in Section 3.3, our packet header indicates which of the edges are taken in the network. The destination can feedback this information to the original sender. There may be three different outcomes:

1. Fully RE-enabled path: If the edge between the dummy source and the RE-ID is taken and the fallback edge between the dummy source and the HID is not taken, this means that the path between the original sender and the destination is fully RE-enabled. Once it is determined that the underlying path is RE-enabled, subsequent packets then can be sent without the fallback:



2. Non-RE path: If the fallback edge between the dummy source and the HID is taken and the edge between the dummy source and the RE-ID is not taken, this means that the path

between the original sender and the destination does not support RE. Once it is determined that the underlying path does not support RE, subsequent packets then can be sent only using the host identifier:



3. Partially RE-enabled path: If both edges are taken, this means that the path between the original sender is partially RE-enabled. Unless otherwise noted, we ignore this case in this chapter and treat as a non-RE path. Later, we revisit this scenario in Section 4.7.

4.1.3 End-point Adaptation

In the middlebox approach, the newly introduced functionality is transparent to the end-points, and therefore it is hard to identify how packets are being processed in the network. Unlike the traditional middlebox deployment, XIA allows applications to identify the functionality supported by the path. This difference has a significant implication on end-point design. End-points can now dynamically adapt their strategies based on the functionality provided by the path. We show that when networks provide new functionality, end-points can benefit by adapting their strategy to better take advantage of the network.

In the rest of this chapter, we show how end-points can adapt their behaviors when the path is RE-enabled. To this end, we present Redundant Transmission [89], a novel end-point strategy that takes advantage the RE functionality for robust real-time data delivery. Redundant Transmission (RT) leverages the RE functionality and fundamentally redesigns the reliability scheme to intelligently sends multiple copies of the same packet. While this may seem high-overhead, we show that RT on content-aware networks can effectively support a variety of time-critical applications with far lower overhead and higher degree of robustness compared to existing loss recovery schemes for real-time communication such as Forward Error Correction (FEC). Since RT only works well when the underlying path is RE-enabled, we adapt the end-points' behavior: in case of 1) fully RE-enabled path, we use RT; otherwise, we use FEC.

FEC represents traditional designs in which any interesting interaction happens between two end-points and the network does pure data delivery. Specifically, FEC performs decoding and encoding of redundancy using complex algorithms such as Reed-Solomon coding at the end-points. On the other hand, RT represents a new class of design in which end-points interact with the network to achieve their goals. In the remainder of the chapter, we explore the new loss recovery design that better supports robust real-time communication when the path is content-aware or RE-enabled. Using RT as an example, we demonstrate that end-points must adapt their strategy to better take advantage of the new functionality and that XIA facilitates such dynamic adaptation.

4.2 A Case for End-point Adaptation

Motivation: A variety of current and future Internet applications require time critical or low latency communication. Example applications include delay-sensitive live/interactive video streams, online games, and video-based calls (e.g., Apple's FaceTime), all of which send real-time data. Studies of real-time systems [101, 145] suggest that the maximum tolerable one-way delay is around 150ms for real-time interaction. Within a data center, many soft real-time applications that interact with users require low latency communication [18]. Certain classes of inter-datacenter transfers, such as mirroring financial data, also require real-time communication [32].

The central challenge in supporting such delay-sensitive real-time applications is protecting them from network loss. One set of conventional approaches—acknowledgment-based retransmission protocols—are not appropriate for real-time communication as retransmissions triggered by timeouts can take several RTTs and violate applications' timing constraints [122, 156]. Another set of approaches—redundancy-based schemes such as Forward Error Correction (FEC)—suffer from a fundamental tension between robustness and the bandwidth overhead [42, 64], making them either difficult to tune or inefficient in practice. These techniques have been tuned to provide the best performance tradeoffs possible in traditional networks.

A case for an alternative design: In contrast, the focus of the remainder of this chapter is to show that better protection against congestion losses may be possible in *content-aware networks*, such as redundancy elimination networks. Content-aware processing is seeing ever-growing adoption in a variety of settings, including mobile and cellular networks [10], data centers [97], cloud computing [8], and enterprise networks [4]. The most popular of such content-aware network devices are the WAN optimizers [2, 6, 9] that are typically placed at branch and main offices or between data-centers to reduce the traffic between them.

Our core assumption is that content-aware network devices will be widely deployed across a variety of links in future networks. Given this setting, we ask: *(i)* How do we re-architect loss protection for delay sensitive applications operating in this new context? *(ii)* Does content-awareness help simplify or further complicate the issues that existing loss protection schemes face? And, why?

We show that taking content-awareness into account challenges the conventional wisdom on the trade-offs of redundancy in protecting against losses in time-critical and delay-sensitive applications. In particular, we show that it is now possible to use redundancy in a simple yet clever fashion to ensure robustness against congestion losses while imposing little or no impact on the network or on other existing applications. Equally importantly, we show that it is now far easier to integrate loss protection with other design constraints such as adhering to tight delay bounds.

We believe that the duplicate suppression actions in content-aware frameworks provide a tremendous opportunity to use redundancy-based protection schemes. However, redundancy must be introduced in the right way to ensure: *(a)* the network can eliminate it optimally to provide the desired efficiency and *(b)* the impact on other applications can be controlled.

Summary of our approach: We describe Redundant Transmission (RT) – a loss protection scheme that intelligently sends multiple copies of the same data. The basic idea of RT is to expose the redundancy directly to the underlying content-aware network. The simplest form of RT is to send multiple copies of the same packet. When packets are not lost, the duplicate transmissions in RT are compressed by the underlying network and add little overhead. In contrast, when the network is congested, the loss of a packet prevents the compression of a subsequent transmission. This ensures that the receiver still gets at least one decompressed copy of the original data. In some situations, the fact that packet losses do not directly translate to less bandwidth use may raise the concern that RT streams obtain an unfair share of the network. However, existing congestion control schemes, with some critical adjustments to accommodate RT behavior, can address this concern.

In essence, RT signals the network the relative importance of a packet by transmitting multiple copies. RT requires almost no tuning; this stands in stark contrast with the difficulty of fine-tuning FEC-based approaches for traditional networks. Finally, RT decouples redundancy from delay and easily accommodates application timing constraints; in comparison, FEC schemes today closely tie delay guarantees with redundancy encoding since the receiver cannot reconstruct lost packets until the batch is complete. In effect, RT on content-aware networks can effectively support a variety of time-critical applications far better than existing approaches for traditional networks.

Contributions: To illustrate the benefits of RT concretely, we use as an example RPT, a simple variant of RT for real-time video in a redundancy elimination network. Our evaluation of RPT, using a combination of real-world experiments, network measurements and simulations, shows that RPT decreases the data loss rate by orders of magnitude more than FEC schemes applicable to live communications. As a result, it achieves better video quality than FEC for a given bandwidth budget, or uses up to 20% less bandwidth than FEC schemes to deliver the same video quality.

We make the following contributions:

1. We highlight the need to reconsider the design of loss protection for content-aware networks. We show that network content-awareness enables vastly simpler and more effective approaches to loss protection (Section 4.4).

2. We describe a redundancy scheme, RT, that can provide a high degree of robustness at low overhead, and require minimal tuning (Section 4.4 and Section 4.5).

3. Through extensive experiments and simulations, we find that RT can improve the robustness of real time media applications with strict timing constraints (Section 4.7).

In the remaining sections, we review related work (Section 4.3), discuss realistic deployment examples as well as general implementation of RT on other content-aware networks (Section 4.6), and finally conclude in Section 4.8.

4.3 Current Loss Recovery Schemes

Packet losses are often inevitable on the Internet, especially across heavily-loaded links, such as cross-country or trans-continental links. Many prior works use *timeout-based retransmission* to recover lost data [36, 71, 122, 146, 156] on traditional networks. However, retransmission causes large delays [156] which are often difficult to hide. Also, the performance depends on correct timeout estimation [122] which is often non-trivial [36, 122]. Because of these intrinsic limitations, more sophisticated enhancements such as selective retransmission [71, 146], play-out buffering [146], and modification to codecs [156] are often required to augment retransmission based loss recovery.

Another option is *redundancy-based recovery*, with FEC being an example framework that is widely used today. While coding provides resilience, the use of FEC is constraining in many ways in practice: *(1)* In FEC, the receiver cannot recover lost packets until the batch is complete. This limits the size of the batch for delay-sensitive applications. For example, at most 5 packets are typically batched in video chat applications such as Skype [184]. *(2)* Small batch size makes FEC more susceptible to bursty loss. For example, adding a single coded FEC packet for every five original data packets is not enough to recover from two consecutive lost packets. Therefore, in practice, the amount of redundancy used is high (e.g., 20% to 50% [42, 64, 184]), which is much higher than the underlying packet loss rate. *(3)* Furthermore, FEC needs to adapt to changing network conditions [39, 197], which makes parameter tuning even more difficult. Many studies [77, 197] have shown that fine tuning FEC parameters within various environments is non-trivial.

More sophisticated redundancy-based recovery schemes such as fountain codes [46], rateless coding with feedback [87], and hybrid ARQ [121] introduce redundancy incrementally. However, fountain codes, rateless coding have been mostly used for bulk data transfer or non-real-time streaming. Hybrid ARQ has been mostly used in local wireless networks where the round-trip time is much smaller compared to real-time delay constraints. When the round-trip time is comparable to real-time delay constraints, incremental redundancy schemes degenerate to FEC. Many other sophisticated schemes such as multi-description coding [149] also use FEC to scale the video quality proportional to the bandwidth. While these schemes relax some of the above limitations of FEC, the fundamental limitation of small batch size is inherent to delay-sensitive applications.

4.4 Redundant Packet Transmission: Design

We now describe the design of Redundant Packet Transmission (RPT), a simple variant of RT that sends fully redundant packets for delivering interactive video streams. We envision a scenario in which real-time video traffic and other traffic coexist, with no more than 50% of the traffic on a particular link being interactive, real-time traffic. We picked this scenario because it is representative of forecasts of future network traffic patterns [3].

To simplify exposition, throughout this section, we assume that the RPT flows travel through a network with hop-by-hop Redundancy Elimination (RE) enabled [19]. Later, in Section 4.6, we explore RT in content-aware networks of various other forms. As stated earlier, we assume that packet losses happen only due to congestion.

4.4.1 Basic Idea

As explained earlier, the basic idea of redundant packet transmission (RPT) is to send multiple copies of the same packet. If at least one copy of the packet avoids network loss, the data is received by the receiver. In current network designs, transmitting duplicate packets would incur large overhead. For example, if two duplicates of every original packet are sent, the overhead is 200% and a 1Mbps stream of data would only contain 0.33Mbps of original data. However, in networks with RE, duplicate copies of packets are encoded into small packets, and this overhead would be significantly reduced.

Figure 4.1 illustrates how RPT works with redundancy elimination. From the input link, three duplicate packets are received. The first packet is the original packet A, and the other two packets A', are encoded packets which have been "compressed" to small packets by the previous hop RE encoder. The compressed packet contains a reference (14 bytes in our implementation) used by the RE decoder of the next hop. At the incoming interface, the packets are fully decoded, generating 3 copies of packet A. They are then queued at the appropriate output queue. The figure illustrates a router that uses virtual output queuing. When congestion occurs, packets are dropped at the virtual output queue. Only packets that survive the loss will go through to the RE encoder on the output interface. When multiple packets survive the network loss, the first packet will be sent as decompressed, but the subsequent redundant packets will again be encoded to small packets by the RE encoder.

In this manner, multiple copies of packets provide robustness to loss and RE in the network reduces bandwidth overhead of additional copies.

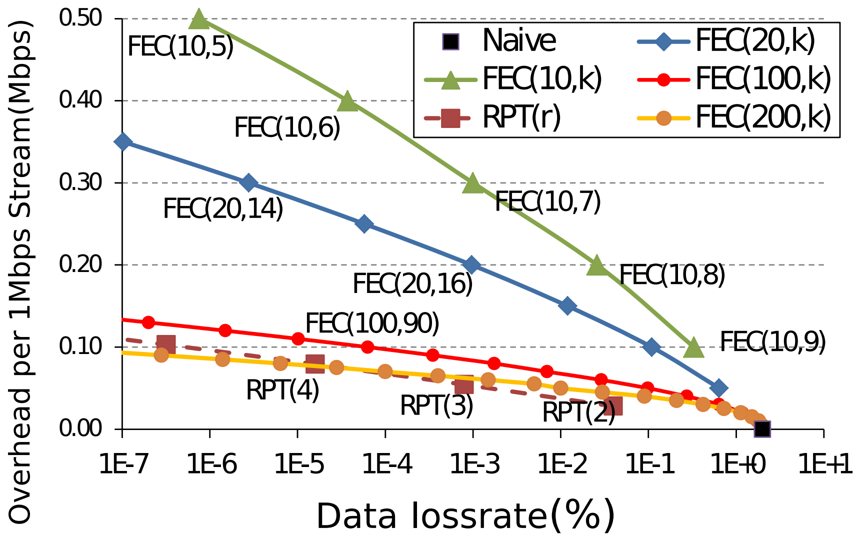


Figure 4.2: RPT and FEC under 2% random loss.

4.4.2 Key Features

Next, we discuss three practically important properties of RPT: *high degree of robustness with low bandwidth overhead*, *ease of use and flexibility for application developers*, and *flow prioritization in the network.*

Low Overhead and High Robustness

As discussed in [19], the packet caches in the RE encoder and decoder are typically designed to hold all packets sent within the last tens of seconds. This is much longer than the timescale in which redundant packets are sent (∼60ms). Thus, all redundant packets sent by the application will be encoded with respect to the original packet. The extra bandwidth cost of each redundant packet is only the size of the encoded packet (43 bytes in our implementation[1].) The overhead of an extra redundant packet, therefore, is less than 3% for 1,500 byte packets, which is 7 to 17 times smaller than the typical FEC overhead for a Skype video call [42, 64].

To compare RPT with FEC, we model RPT and FEC under a 2% uniform random packet loss and analytically derive the data loss of a 1Mbps RPT and FEC streams in an RE network. Figure 4.2 shows the resulting overhead and data loss rate. All flows operate on a fixed budget but splits its bandwidth between original data and redundancy. The overhead (*y-axis*) is defined as the amount of redundancy in the stream, and the data loss (*x-axis*) as the percentage of data that cannot be recovered. RPT(r) denotes redundant streaming that sends *r* duplicate packets.

[1]Our implementation does not encode IP and transport layer headers.

FEC flows with various parameters are shown for comparison. FEC(n,k) denotes that k original packets are coded in to n packets. For FEC, we use a systematic coding approach (e.g. Reed-Solomon) that sends k original packets followed by $n-k$ redundant packets. While both schemes introduce redundancy, only the redundancy introduced by RPT gets minimized by the network unlike FEC which does not introduce redundancy in a way that the network understands; thus FEC over RE networks is identical in performance to FEC over traditional networks.

FEC schemes, especially with a small group size (n), incur large overheads, and are much less effective in loss recovery. For example, FEC(10,8), which adds 0.2 Mbps of redundancy, has similar data loss rates as RPT(2), which only adds 0.03Mbps of redundancy. FEC with group size (n=200) performs similar to RPT. However, it takes 2.4 seconds to transmit 200 1,500 byte packets at 1Mbps. This violates timing constraints of real-time communications because a packet loss may only be recovered 2.4 seconds later in the worst case. Thus, in practice, RPT provides high robustness against packet loss at low overhead.

Ease of Use and Control

Application developers can easily tailor RPT to fit their needs. Three unique aspects of RPT help achieve this property:

1) Detailed parameter tuning is not necessary.

2) RPT allows per-packet redundancy control.

3) Delay and redundancy are decoupled.

Ease of parameter selection: With FEC, the sender has to carefully split its bandwidth between original and redundant data in order to maximize the video quality. If the amount of redundancy is larger than the amount of network loss, the stream tolerates loss. However, this comes at the cost of quality because less bandwidth is used for real content. If the amount of redundancy is too low, the effect of loss shows up in the stream and the quality degrades. This trade-off is clear in FEC(10,k)'s performance in Figure 4.2. Determining the optimal parameters for FEC is difficult and adapting it to changing network conditions is even more so [77].

A unique aspect of RPT is that even though the actual redundancy at the sender is high, the network effectively reduces its cost. Therefore, the sender primarily has to ensure that the amount of redundancy (r) is high enough to tolerate the loss and worry much less about its cost, which makes RPT simple and easy to use. We show in §4.7.3 that only small amount of redundancy ($r = 3$) is good enough for a wide range of loss rates (1% to 8%), and a sub-optimal overshoot (i.e. unnecessary, extra redundancy) has very little impact on actual video quality.

Packet-by-packet redundancy control: RPT introduces redundancy for each packet as opposed to groups of packets, enabling packet-by-packet control of the extent of redundancy. More important packets, e.g., those corresponding to I-frames, could simply be sent more repeatedly than others to increase robustness. In essence, RPT enables fine-grained unequal error protection (UEP) [94]. Thus, RPT is simple to adapt to application-specific needs and data priorities.

Each encoded packet can be viewed as an implicit signal to the network. Importance of the data is encoded in the number of encoded packets, $r-1$. When an original packet gets lost, routers try to resend the original packet when the signal arrives. As such the network tries up to r times until one original copy of the packet goes through.

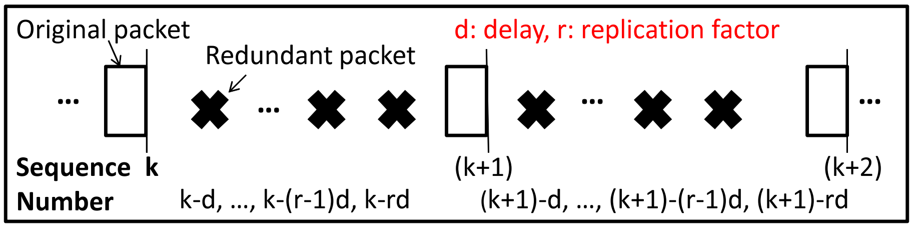
Decoupling of delay and redundancy: Unlike FEC, RPT separates the redundancy decision from delay. FEC schemes closely tie timing with the encoding since the receiver cannot reconstruct lost packets until the batch is complete. In contrast, RPT accommodates timing constraints more easily. For example, sending 3 redundant packets spaced apart by 5 ms is essentially asking every router to retry up to 3 times every 5 ms to deliver the original packet. This mechanism lends itself to application specific control to meet timing constraints. We further discuss the issues in controlling delay in §4.4.3.

Flow Prioritization

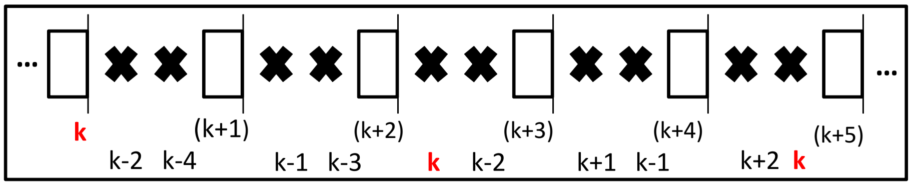
A unique property of RPT-enabled traffic is that it gets preferential treatment over other traffic under lossy conditions. RPT flows do not readily give up bandwidth as quickly as non-RPT flows. This is because for RPT flows packet losses do not directly translate into less bandwidth use due to "deflation" of redundant packets; subsequent redundant packets cause retransmission of the original packet when the original packet is lost. Therefore, *RPT flows are effectively prioritized in congested environments.* As a result, RPT could get more share at the bottleneck link. We believe that this is a desirable property for providing stronger guarantees about the delivery rate of data, and analyze this effect in §4.7. However, this preferential treatment may not be always desirable. In case where fair bandwidth-sharing is desired, RPT is flexible enough to be used with existing congestion control mechanisms while retraining its core benefits. In §4.5, we provide an alternative solution that retains other two benefits of RPT except flow prioritization.

4.4.3 Scheduling Redundant Packets

We now discuss detailed packet sequencing, i.e. how RPT interleaves redundant packets with original packets. Each original packet is transmitted without any delay, but we use two parameters to control the transmission of redundant packets: redundancy (r) and delay (d).



(a) Sequence of packets sent by RPT(r)



(b) Sequence of packets sent by RPT(3) with d=2

Figure 4.3: Sequence of packets sent by RPT

Figure 4.3a shows the packet sequence of an RPT(r) flow. Original packets are sent without any delay, and $r-1$ redundant packets are sent compressed in between two adjacent original packets. Thus, compared to a non-RPT flow of the same bitrate, r times as many packets are sent by a RPT(r) flow.

The delay parameter (d) specifies the number of original packets between two redundant packets that encode the same data. The first redundant packet of sequence number n is sent after the original packet of sequence number $(n+d)$. If the loss is temporally bursty, having a large interval between two redundant packets will help. However, extra delay incurs extra latency in recovering from a loss. So, delay (d) can be adjusted to meet the timing requirements of applications.

Figure 4.3b shows an example with $r = 3$ and $d = 2$. Three copies of packet k is sent, each spaced apart by two original packet transmissions. In §4.7.3, we evaluate RPT's sensitivity to parameter selection.

4.4.4 Comments on RT/RPT

Is this link-layer retransmission? Conceptually, RT is similar to hop-by-hop reliability or link-layer retransmission. However, RT fits better with the end-to-end argument-based design of the Internet by giving end-points an elegant way to control the retransmission behavior inside the network. In contrast, hop-by-hop reliability schemes make it hard for applications to control the delay or to signify the relative importance of data. Similarly, in a naive hop-by-hop retransmission scheme, packets are treated equally and can be delayed longer than the application-specific limit. RT exploits network's content-awareness and provides a signaling mechanism on top of such networks to achieve robustness against packet loss.

Why not make video codecs resilient? In the specific context of video, prior works have proposed making video codecs more resilient to packet loss. Examples include layered video coding [131], H.264 SVC, various loss concealment techniques [174] and codecs such as ChitChat [184]. However, greater loss resilience does not come for free in these designs; these designs typically have lower compression rate than existing schemes or incorporate redundancy (FEC) in order to reconstruct the video with arbitrary loss patterns. Also they are often more computationally complex than existing approaches, which makes them difficult to support on all devices [174].

Our scheme is agnostic to the choice of video codec and the loss concealment schemes used. Of course, the exact video quality gains may differ based on the loss rates, loss patterns and codec used.

How does RT compare to more sophisticated coding? Many sophisticated video coding schemes, such as UEP [94], priority encoding transmission [17], and multiple description coding [51, 149], typically use FEC (or Reed-Solomon codes) as a building block to achieve graceful degradation of video quality. Similarly, we believe that RT can be used as a building block to enable more sophisticated schemes. For example, one can send more important blocks of bits within a stream multiple times. Furthermore, since RE networks also eliminate sub-packet level redundancy, a partially redundant packet may also be used. We leave details of such techniques as future work. In this work instead, we focus on understanding the core properties of RT by comparing a basic form of RT with the most basic use of FEC.

What about wireless errors? RT/RPT can be extended to protect against categories of losses other than those due to congestion, e.g., link layer losses and partial packet errors due to interference and fading. A naive solution is to leverage link layer acknowledgements and override automatic retransmission to adapt the retransmission behavior. Wireless links, such as 802.11 and Bluetooth, have link layer ACKs. Since timing is important for real time data delivery, we assume that we disable the automatic retransmission as in Medusa [161] When an ACK for a data packet is not received, subsequent redundant packets can be sent as the original, decompressed form. When an ACK is received, subsequent redundant packets can either be discarded or sent as compressed. This approach treats redundant packets as explicit signals that control the retransmission in case of packet losses. The benefit of this to schemes over media gateway approach, such as Medusa [161], is that they require media gateways to intercept packets and infer the timing information by inferring the video sequence contained in the packets. In contrast, RT approach uses a simple form of explicit signaling that is controlled by the end-points. Of course, RT approach can also use the gateway approach to terminate RT and switch to other protection schemes.

A more efficient solution that implements the idea of RT in this context is to retransmit a partial packet with errors. Such partial packet recovery [104] has already been explored. Combining RT with partial packet recovery is left as a future work.

In addition, the fact that wireless devices are often mobile introduces new challenges. This is challenging because the packet caches in the mobile device and the access points may not be synchronized. We do not yet know how RT/RPT can work efficiently in mobile environments where access points that a client is talking to may change over time. A naive approach is to send a signal (e.g., NACK) when the caches are not in sync. This signal can be used to immediately retransmit the original packet.

Finally, we note that there a variety of schemes that aim to provide robust performance in such situations, some with a focus on video (e.g., the schemes in [103, 161] for wireless links). However, RT/RPT's explicit focus on congestion losses means that our approach is complementary to such schemes.

4.5 RPT with Congestion Control

As explained earlier, RPT flows are effectively prioritized in congested environments[2]. However, in some environments, fair bandwidth sharing may be more desirable. In such cases, the sending rate should adapt to the network conditions to achieve a "fair-share". To meet this goal, we apply TCP friendly rate control [74] (TFRC) to RPT flows. However, this raises surprisingly subtle problems regarding the transmission rate and loss event rate estimation that are germane to TFRC. We describe these challenges and our modifications to TFRC below.

Packet transmission: In TFRC for RPT, we calculate the byte transmission rate from the equation just as the original TFRC. Note that RPT(r) must send an *original* packet and $r-1$ duplicates. To match the byte sending rate, we adjust the length of the packet so that equal number of bytes are sent by TFRC RPT as the original TFRC in calculating the throughput. Thus, given a computed send rate, TFRC RPT(r) sends r times as many packets. Note that each packet, original or duplicate, carries an individual sequence number and a timestamp for TFRC's rate calculation purposes.

Loss event rate estimation: In the original TFRC, the sending rate is calculated given the loss event rate p, where loss event rate is defined as the inverse of the average number of packets

[2]We further verify this later in §4.7.4.

sent between two loss events. A loss event is a collection of packet drops (or congestion signals) within a single RTT-length period.

Ideally, we would want TFRC RPT(r) to have the same loss event rate as the original TFRC, as that would also make TFRC RPT obtain a TCP friendly fair share of bandwidth. However, the observed loss event rate for RPT depends on the underlying packet loss pattern. For ease of exposition, we look at the two extremes of loss patterns: one that is purely random and the other that is strictly temporally correlated.

Purely random packet drops may occur in a link of a very high degree of multiplexing. On the other hand, in a strictly temporal loss pattern, losses occur during specific intervals. One might see such a loss pattern when cross traffic fills up a router queue at certain intervals. In reality, the two patterns appear inter-mixed depending on source traffic sending patterns and the degree of multiplexing.

Next, we discuss how the two loss patterns impact loss event rate estimation and the transmission rate:

- *Uniform random packet loss:* In this setting, TFRC RPT(r) behaves in a TCP friendly manner without any adjustment to loss estimation. This is because the number of packets sent between two loss events does not change even though the packet sending rates change.
- *Temporal packet loss:* In this setting, packets are lost at specific times. During the time between loss, TFRC RPT(r) sends r times as many packets. Thus, the observed loss event rate for TFRC RPT(r) is only $\frac{1}{r}$ of that of the original TFRC. Therefore, TFRC RPT would send more traffic.

Adjusting the loss event rate: As stated earlier, in practice, the two extreme patterns appear inter-mixed. We therefore want to choose an adjustment factor α so that when the loss event rate is adjusted to α times the measured loss event rate p, TFRC RPT(r) is TCP friendly. As seen in the two extreme cases, α has values 1 for uniform random losses and r for temporal losses, respectively. So, in practice, α should be between 1 and r to achieve exact TCP-friendliness. A larger value of α makes the TFRC-RPT react more aggressively to congestion events, and smaller value less aggressive than a TCP flow. This means, even in the worst case, α can be r times off from the value which achieves exact TCP-friendliness. In this case, a TFRC-RPT flow would have performance similar to $\sqrt{r}$ many TFRC flows because the TCP-friendly rate is inversely proportional to $\sqrt{p}$. Therefore, even an incorrect value of α would still make TFRC RPT friendly to a group of TCP connections and still react to congestion events. In practice, we find in §4.7 that with TFRC-RPT(3), $\alpha = 1.5$ closely approximates the bandwidth share of a single TCP flow under wide range of loss rates and realistic loss patterns.

As such, RPT is flexible enough to allow users to adjust the degree of reactivity to congestion

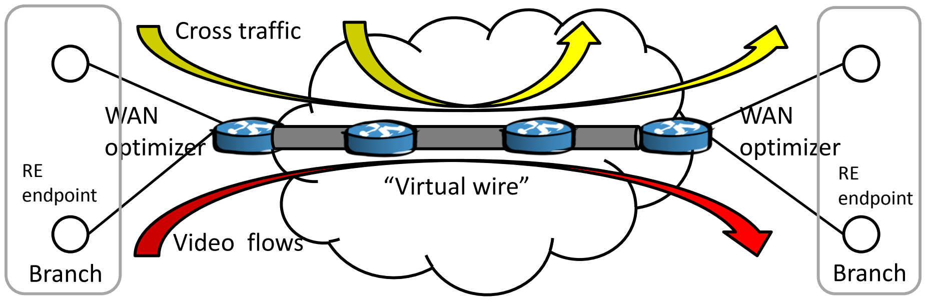


Figure 4.4: Typical Deployment of WAN optimizers

events while being highly robust. Regular RPT does not react to congestion events, and can be used to prioritize important flows. TFRC RPT reacts to congestion events and the reaction degree can be controlled by the parameter α.

4.6 RPT in Various Networks

So far, we have explored RPT on hop-by-hop RE networks as a special case of redundant packet transmission. Here, we look at other deployment scenarios for content-aware devices as well as other content-aware designs.

Corporate networks: WAN optimization is the most popular form of RE deployment in the real world. In a typical deployment, WAN optimizers are placed at branch and main offices or between data-centers to reduce the traffic between them. Example deployments include 58+ customers of Riverbed [9] and Cisco's worldwide deployment to its 200+ offices [6]. While we envision RPT being used in future networks where content-aware devices are widely deployed, RPT can be deployed immediately in such settings.

As shown in Figure 4.4, these sites have low bandwidth connections using leased line or VPN-enabled "virtual" wires. ISPs offering VPN services typically provide bandwidth and data delivery rate (or packet loss) guarantees as part of their SLA [7, 12]. In practice, their loss rate is often negligible because ISPs provision for bandwidth [52]. [3] Thus, the use of VPN and WAN optimizers effectively creates reliable RE "tunnels" on which RPT can operate. Important, real-time data can be sent with redundancy, and compete with other traffic when entering this tunnel. Packets will be lost when the total demand exceeds the capacity of the tunnel, but RPT flows will have protection against such loss. We evaluate this scenario in §4.7.2.

[3]Sprint's MPLS VPN [12] had a packet loss rate of 0.00% within the continental US from Mar 2011 to Feb 2012.

	RPT(3)	FEC(6,5)
Overhead	9%	22%
Data loss rate	8.0 x 10^{-6}	1.9 x 10^{-3}

Table 4.1: Comparison of FEC and RPT over CCN

An alternative is to use traditional QoS schemes such as priority queuing. However, this typically involves deploying extra functionalities including dynamic resource allocation and admission control. For businesses not willing to maintain such an infrastructure, using RPT on an existing RE-enabled VPN would be an excellent option for delivering important, time-sensitive data.

Partial deployment: Not all routers in a network have to be content-aware to use RPT. The requirement for "RPT-safety" is that RE is deployed across bandwidth-constrained links [4], and non-RE links are well provisioned. This is because non-RE links end up carrying several duplicate packets. When such links are of much higher capacity, RPT causes no harm. Otherwise, it impacts network utilization and harms other traffic. In §4.7.2, we explore both cases through examples, and show how the network utilization and the other traffic on the network are impacted when RPT is used in an "unsafe" environment.

To ensure safe operation of RPT, one can detect the presence of RE on bandwidth-constrained links, and use RPT only when it would not harm other traffic. In this section, we outline two possible approaches for this, but leave details as a future work. One approach is to use end-point based measurement: for example, Pathneck [96] allows detection of bottlenecks based on available bandwidth. It sends traceroute packets in between load packets and infers (multiple) bottleneck location(s) from the time gap between returned ICMP packets. Similar to this, we can send two separate packet trains: one with no redundancy and the other with redundancy *r* but with the same bitrate. If all bandwidth constrained links are RE-enabled and RPT is safe to use on other links, the packet gap would not inflate on previously detected bottlenecks and the redundant packet trains would not report different bottleneck links. Another way is to use systems, such as iPlane [128] and I-path [135], which expose path attributes (e.g. available bandwidth) to end-hosts. These systems can easily provide additional information such as RE-functionality for end-hosts to check for RPT safety.

RPT over CCN: RPT also can be integrated with a broad class of content-aware networks, including CCN [102] and SmartRE [20]. In our technical report [88], we explore discuss how

[4]This matches the common deployment scenario for RE [19, 170].

RPT can work atop SmartRE and wireless networks. Here, we focus on applying RPT to CCN.

In CCN, data consumers send “Interest” packets, and the network responds with at most one “Data” packet for each Interest. Inside the network, each router caches content and eliminates duplicate transfers of the same content over any link. CCN is designed to operate on top of unreliable packet delivery service, and thus Interest and Data packets may be lost [102].

We now compare RPT and FEC in CCN. Suppose real-time data is generated continuously, say k packets every 100 ms, and RTT is large. In an FEC-equivalent scheme for CCN, the content publisher would encode k data packets and add $(n-k)$ coded data packets, where $n > k$. The data consumer would then generate n Interest packets for loss protection. The receiver will be able to fully decode the data when more than k Interest and Data packets go through. However, up to n Interest/Data pairs will go through the network when there is no loss. In contrast, RPT does not code data packets, but generates redundant Interest packets. This obviously provides robustness against Interest packet loss. Moreover, when a Data packet is lost, subsequent redundant Interest packet will re-initiate the Data transfer. Since Interest packets are small compared to Data and duplicate Interests do not result in duplicate transfers of the Data, the bandwidth cost of redundancy is minimal. In RPT, at most k Data packets will be transferred instead of n in the FEC scheme.

To demonstrate the benefit more concretely, we take the Web page example from CCN and compare RPT and FEC over CCN. In the CCN-over-jumbo-UDP protocol case [102], a client generates three Interest packets (325 bytes) and receives five 1500-byte packets (6873 bytes) to fetch a Web page [102]. To compare RPT and FEC, we assume in RPT a redundancy parameter of 3 is used and in FEC the server adds one packet to the original data. Table 4.1 shows the overhead and data loss rate of each scheme at the underlying loss rate 2%. The data loss rates is the amount of data that could not be recovered. Even though the overhead of RPT is only 41% of that of FEC, it’s data loss rate is 240 times better. To achieve equal or greater level of robustness than RPT($r = 3$), FEC has to introduce 11 times the overhead of RPT($r = 3$).

4.7 Evaluation

In this section, we answer four specific questions through extensive evaluation:

(*i*) **Does RPT deliver better video quality?** How well does it work in practice?

In §4.7.2, we show that RPT provides high robustness and low bandwidth overhead, which translate to higher quality for video applications.

(*ii*) **Is RPT sensitive to its parameter setting, or does it require fine tuning of parameters**?

In §4.7.3, we show that, unlike FEC, RPT is easy to use since careful parameter tuning is not necessary, and delay can be independently controlled with the delay parameter.

(*iii*) **How do RT flows affect other flows and the overall network behavior**?

In §4.7.4, we demonstrate that RT flows are effectively prioritized over non-RT flows on congested links and may occupy more bandwidth than their fair-share.

(*iv*) **Can we make RT flows adapt to network conditions and be TCP-friendly**?

We show in §4.7.5 that RT can also be made TCP-friendly, while retaining the key benefits.

4.7.1 Evaluation Framework

We use a combination of real-world experiments, network measurements and simulations. We implemented an RE encoder and decoder using Click [114], and created a router similar to that of Figure 4.1. Using this implementation, we create a hop-by-hop RE network in our lab as well as in Emulab. These serve as our evaluation framework.

We use implementation-based evaluation to show the overall end-to-end performance of RPT, and simulations to unravel the details and observe how it interacts with other cross traffic. To obtain realistic packet traces and loss patterns from highly multiplexed networks, we performed active measurements to collect real-world Internet packet traces. We also created background traffic and simulated RPT and FEC flows in a hop-by-hop RE network using the ns-2 simulator. These video flow packet traces are then fed into *evalid* video performance evaluation tool [113] to obtain the received video sequence with loss. For video, we used the *football* video sequence in CIF format, taken from a well-known library [5]. We used H.264 encoding with 30 frames per second. I-frames were inserted every second and only I- and P-frames were used to model live streams.

RE implementation: We implemented the Max-Match algorithm described in [19]. We further modified it to only store non-redundant packets in the packet cache. Therefore, sending redundant packets does not interfere with other cached content. We use a small cache of 4MB. The implementation of the encoder encodes a 1500 byte fully redundant packet to a 43 byte packet[5]. We also implemented RE in ns-2.

Quality	Excellent	Good	Fair	Poor	Bad
PSNR (dB)	> 37	$31 \sim 37$	$25 \sim 31$	$20 \sim 25$	< 20

Table 4.2: User perception versus PSNR

[5]We do not compress network and transport layer headers. Thus, the packet may be compressed even further in practice.

	Encoded	Received
RPT	37.3 dB	**37.1 dB**
FEC	36.9 dB	**35.3 dB**
Naive	37.5 dB	**31.4 dB**

Table 4.3: Average video quality (PSNR)

Evaluation metric: We use the standard Peak-to-Signal-to-Noise Ratio (PSNR) [159] as the metric for the video quality. PSNR is defined using a logarithmic unit of dB, and therefore a small difference in PSNR results in visually noticeable difference in the video. The MPEG committee reportedly uses a threshold of PSNR = 0.5dB to test the significance of the quality improvement [159]. Typical values for PSNR for encoded video are between 30 and 50 dB. Table 4.2 maps the PSNR value to a user perceived video quality [113].

4.7.2 End-to-end Video Performance

In this section, we evaluate the end-to-end performance of RPT and examine key characteristics.

Experimental setting: First, we use our testbed based on our hop-by-hop RE implementation, and create a streaming application that uses redundant packet transmission. We create a topology where an RE router in the middle connects two networks, one at 100Mbps and the other at 10Mbps. To create loss, we generate traffic from the well-connected network to the 10Mbps bottleneck link.

We generate a 1Mbps UDP video stream and long-running TCP flows as background traffic. We adjust the background traffic load to create a 2% loss on the video flow. We then compare the video quality achieved by RPT, Naive, and FEC that use the same amount of bandwidth. We use RPT that has 6% overhead ($r = 3, d = 2$), and FEC(10,9) with 10% overhead, which closely match in latency constraints with comparable overhead. Naive uses UDP without any protection.

Figure 4.5 shows the sending rate and the received data rate after the loss. The RPT and FEC senders respectively use about 6% and 10% of their bandwidth towards redundancy, while the Naive sender fully uses 1Mbps to send original data. The sending rates of the three senders are the same, within a small margin of error (1%). The Naive receiver loses 2% of the data and receives 0.98Mbps because of the loss. The FEC receiver only recovers about 66% of the lost data due to the bursty loss pattern. On the other hand, the RPT receiver receives virtually all original data sent. Note that only the amount of redundancy has slightly decreased. This is because when an original packet is lost, a subsequent redundant packet is naturally expanded inside the network.

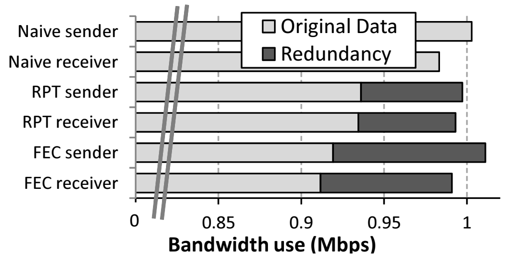
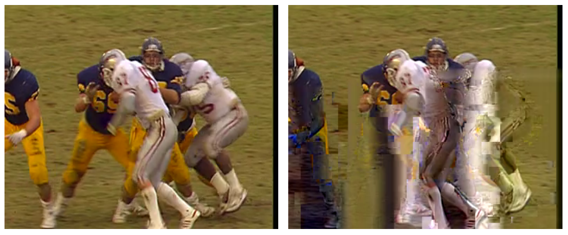


Figure 4.5: Bandwidth use



(a) RPT flow (b) Naive flow

Figure 4.6: Snapshot of the video

As a result, the RPT flow gives much higher video quality. Figure 4.6 shows a snapshot of the video for RPT and Naive flows. Table 4.3 shows the video quality of an encoded video and the received video. The encoded video column shows the quality of video generated at the sender before packet loss. When RPT and FEC are used, the encoded video quality is slightly lower because of the bandwidth used towards redundancy. However, **because the RPT flow is highly robust against loss, it provides the best video streaming quality** (1.8 dB better than FEC and almost 6dB better than Naive).

RE-enabled Corporate VPN of §4.6 is the most common deployment scenario of RE in today's networks. To demonstrate the feasibility of this scenario, we set up a network of four routers in Emulab [69] and created an RE-enabled VPN tunnel that isolates the traffic between two remote offices similar to that of Figure 4.4. The physical links between two remote offices have 100Mbps capacity, and carries traffic from other customers. We generate cross traffic over the physical links that carries the VPN traffic so that the physical links experience congestive loss. We allocate 5Mbps of bandwidth to the VPN-enabled "virtual" wire, which is emulated using the priority queuing discipline from the Linux kernel's traffic control module. We introduced a 1Mbps video traffic and 5 TCP connections between the two remote offices, and compare RPT(3) and FEC(10,9) whose bandwidth overhead best matches to that of RPT(3), while adhering to the latency constraint. The video stream experiences a loss rate of around 2.7% and the tunnel's link utilization was nearly 100% in both cases. The resulting PSNR of the RT flow and FEC were 37.1dB and 34.1dB respectively. This result shows that RT also works well on the most common form of today's content-aware networks.

Real traces: To study the performance of RPT in a realistic wide-area setting, we collected a real-world packet trace. We generated UDP packets from a university in Korea to a wired host connected to a residential ISP in Pittsburgh, PA. The sending rate was varied from 1Mbps to 10Mbps, each run lasting at least 30 seconds. The round-trip-time was around 250ms, which indicates that retransmissions would violate the timing constraint of an interactive stream.

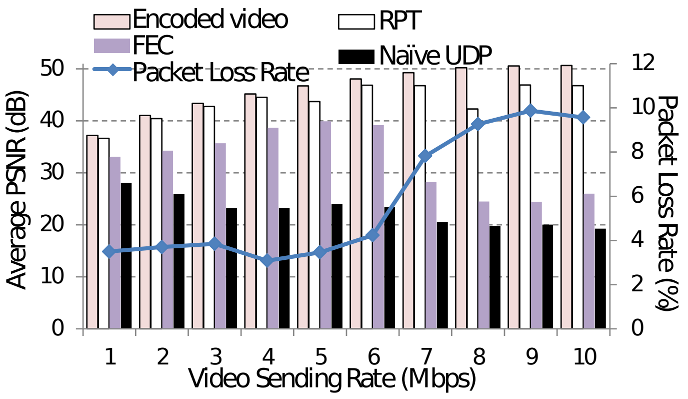


Figure 4.7: Video quality and loss rate for real traces

Assuming that the packet loss rates do not change significantly with RPT[6], we apply the loss pattern obtained from the measurement to an RPT flow, an FEC flow and a Naive UDP flow. For RPT, we use a redundancy parameter of $r = 3$ and a delay parameter $d = 2$. For FEC, we choose the parameters so that the overhead matches closest to that of RPT, while the additional latency incurred by FEC at the receiver does not exceed 150ms, which results in different parameters for different sending rates. Figure 4.7 shows the video quality for each scheme. The solid line shows the packet loss rate. The `Encoded video` bar shows the ideal PSNR without any loss when all the bandwidth is used towards sending original data, presented as a reference. As the sending rate increases, the quality of the encoded video increases. However, the loss rate from Korea to U.S. was 3.5% at 1Mbps but increased to 9.8% at 10Mbps as the sending rate increases. Because of the high loss rate, the naive UDP sender performs poorly (PSNR well under 30dB). FEC achieves better performance than naive, but much worse than RPT especially under high loss rates. In contrast, **RPT gives the best performance, closely following the quality of the encoded video until the loss rate is about 8%**. Even at the higher loss rates, the impact on quality is much less than the FEC scheme. This is because RPT gives much better protection against loss than FEC at similar overhead. [7]

[6]We later verify this and see how RPT affects the loss rate in §4.7.4.

[7]Large drop in PSNR at 8 Mbps is an artifact of the video's resolution being too small compared to its encoding rate and a particular pattern of bursty data loss. When the video compression gets nearly lossless, even a small data loss causes PSNR to drop sharply. In addition, two original packets that are close together in sequence were lost by coincidence in 8Mbps RPT. This had a more detrimental effect on the PSNR value because the lost data belonged to the same video frame. The actual data loss rate of the 8Mbps RPT flow was 0.165%, which is less than 0.174% of the 9Mbps RPT flow.

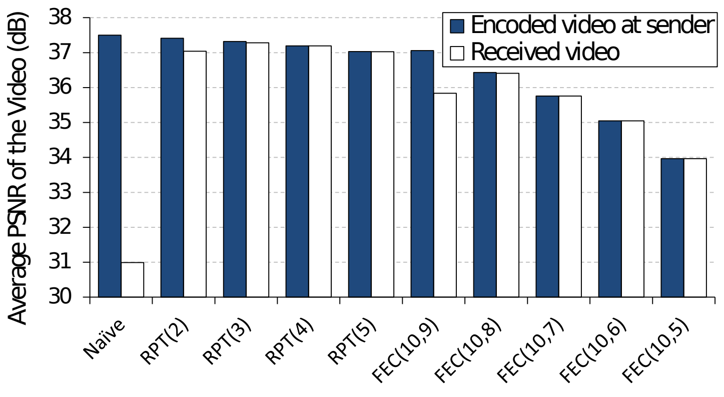


Figure 4.8: RPT's performance is much less sensitive to its parameter setting.

4.7.3 Parameter Selection and Sensitivity

In this section, we provide an in-depth performance evaluation of RPT. In particular, we compare RPT and FEC's parameter sensitivity using simulations that produce packet loss patterns of highly multiplexed networks with realistic cross traffic.

Simulated RE Network: We use the ns-2 simulator to create a realistic loss pattern by generating a mix of HTTP and long-running TCP cross traffic. We use a dumbbell topology with the RE-enabled bottleneck link capacity set to 100Mbps, and simulate a hop-by-hop RE network and RPT flows. We generate 100 long-running TCP flows and 100 HTTP requests per second. We used the packmime [47] module to generate representative HTTP traffic patterns. We also generate ten video flows each having 1Mbps budget regardless of the loss protection scheme it uses. The results presented are averages of ten runs with each simulating five minutes of traffic. We first look at the final video quality seen by the end receiver under different parameter settings, and then analyze the underlying loss rate and overhead.

How do RPT flows and FEC flows perform with different redundancy parameters? For RPT, we vary the redundancy parameter r from 2 to 5, while fixing the delay parameter d to 2. For FEC, we use a group size $n = 10$ to meet the latency constraints and vary the number of original data packets k from 5 to 9.

Figure 4.8 shows the quality of the video seen by the receiver compared to the encoded quality at the sender. The quality of encoded video decreases as the overhead of redundancy is increased. For example, the encoded video of FEC(10,5) because sender only uses half the bandwidth to encoded the video. Both the overhead and loss impacts the video quality. A video sender has to split the total bandwidth between redundancy and original data. If the overhead is large, then the

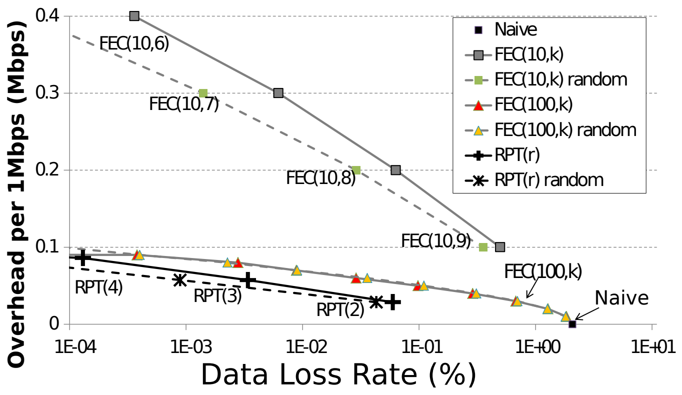
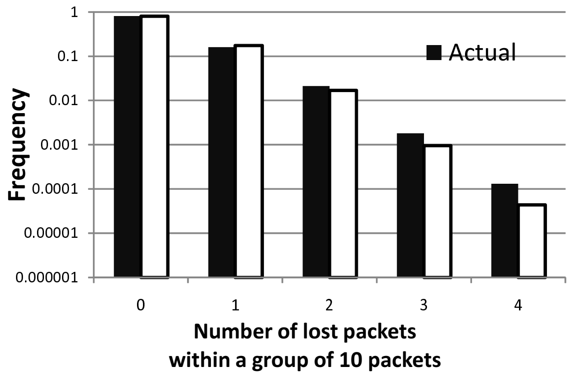


Figure 4.9: Percent data loss rate and overhead: RPT greatly outperforms FEC with small group size.

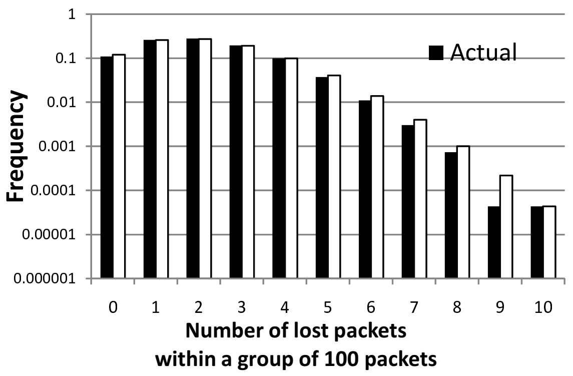
quality of the encoded video suffers. Once the video is encoded and sent across the network, loss degrades the quality of the video seen by the end receiver. The gap between the encoded video and the received video within the same scheme represents the effect of loss. Since the Naive sender does not have any protection against loss, the resulting video quality is severely degraded from the original video. The result shows that **RPT performs better than FEC's best, and its performance is stable across different parameter settings.** In contrast, FEC's performance is highly sensitive to the parameter selection. Therefore, with FEC, the sender has to carefully tune the parameter to balance the amount of redundancy and encoding rate.

Figure 4.9 shows the underlying data loss rate and overhead of the video flows. The *x*-axis shows the data loss rate in log-scale, and the *y*-axis shows the amount of overhead. All video flows experience $\sim$2% packet loss. For comparison, the performance of RPT and two FEC families (n=10, 100) under uniform random loss (dotted lines) are also shown. RPT(4)'s data loss rate is several orders of magnitude lower than the loss rate of FEC(10,9) whose overhead is similar. RPT(4) even performs better than FECs with large group size, such as FEC(100,91), whose latency exceeds the real-time constraint. FEC(10,7) achieve similar data loss rate to RPT(3) but has 6 times the overhead, which translated to 2dB difference in PSNR.

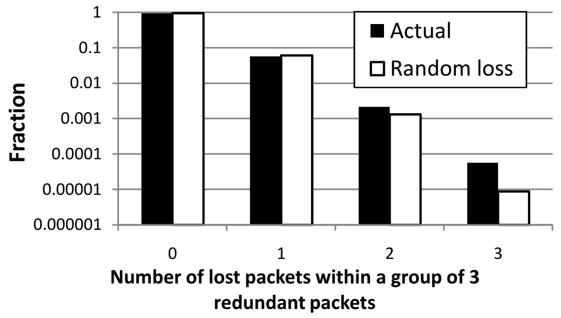
The gap between the uniform loss and actual loss lines in Figure 4.9 represents the effect of bursty loss performance. FEC(10,k) and RPT show a relatively large gap between the two lines. Figure 4.10 explains how the difference in the loss pattern affects the performance. Figure 4.10a and 4.10b respectively show the frequency of multiple packet losses within a group of 10 and 100 packets. They show that the loss pattern is bursty. Analyzing the underlying loss pattern, we observe that within a group of 10 packets, losses of 2 to 4 packets appear more frequently in the actual loss pattern. This shows that the underlying traffic is bursty. On the other hand, loss bursts of more than 5 occur less frequently in the actual pattern because TCP congestion control



(a) Frequency of packet loss in a group of 10 packets



(b) Frequency of packet loss in a group of 100 packets



(c) Frequency of packet loss in a group three redundant packets

Figure 4.10: Pattern of loss: Small bursts in actual loss are more common than the uniform random loss.

eventually kicks in. Figure 4.10c shows the frequency of multiple redundant packet losses in RPT(3). When all three packets are lost, the data is lost. It shows that three packet losses appear more frequently in the actual loss pattern, which explains the gap in Figure 4.9.

We now show how the parameter should be set in RPT.

How should we choose parameters in RPT? To answer this question, we study how loss rate and burstiness of loss affect the performance of RPT. First, we use the random loss pattern and vary the packet loss rate from 1% to 8%. For RPT, we vary the redundancy parameter from 2 to 4, but fix the delay parameter at 2. For each loss rate, the average PSNR of a naive sender and an RPT sender is shown in Figure 4.11. It shows that video quality of RPT(3) is virtually immune to a wide range of packet losses; the average PSNR for RPT(3) under 8% loss only decreased by

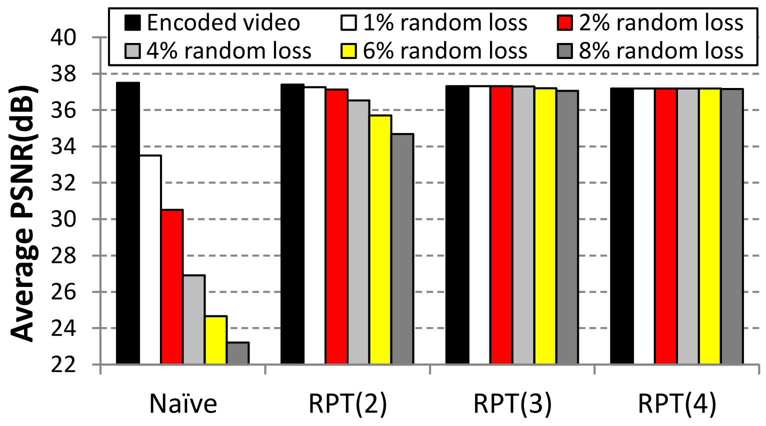


Figure 4.11: Video quality under 1 to 8% loss. RPT(3) steadily delivers high quality even under high loss.

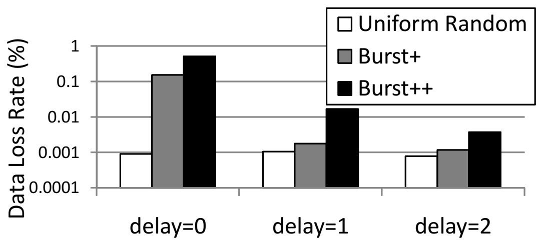


Figure 4.12: Bursty loss increases the data loss especially when the delay parameter is small.

0.25dB compared to the zero-loss case. We, therefore, use $r = 3$ in the rest of our evaluation.

Second, we look at the role of the delay parameter under bursty loss. For reference, we generate a 2% random loss, which on average has 1 lost packet every 50 packets. We then create bursty loss patterns by reducing the number of packets between losses by up to 15 and 35, while keeping the average loss rate the same. The three cases are named as `Uniform random`, `Burst+`, and `Burst++` respectively. Figure 4.12 shows the data loss rate of RPT with different delay parameters under the three loss conditions. An increase in burstiness negatively impacts the data loss rate, but as delay is increased the negative impact is decreased. In general, a large delay parameter gives more protection against bursty loss, but incurs additional latency. **We use $d = 2$ and $r = 3$ for RPT because of its superior performance in wide range of loss conditions**.

How much latency is caused by RPT and FEC flows? A loss might be recovered by subsequent redundant packets; here we quantify the delay in this. In RPT(r) with delay d, the receiver buffer must hold $d \cdot r$ packets. So the delay in RPT is $d \cdot r \cdot intv$, where $intv$ is the interval between

	FEC(10,6)	FEC(100,92)	RPT(3)
No sender buffering	240ms	2400ms	60ms
Sender buffering	180ms	1300ms	-

Table 4.4: Maximum one-way delay of RT and FEC

packets. The RPT sender needs no additional buffering, as it transmits the packet as soon as a packet is generated from the encoder. In FEC, two alternatives exist where one does sender buffering to pace packets evenly and the other doesn't but further delays the transmission of redundant coded packets [88]. Table 4.4 shows the delay caused by RPT and FEC for 1Mbps RPT(3) with $d = 2$ and FEC streams that exhibit similar data loss rate from Figure 4.9. We see that **RPT gives a significantly lower delay than FEC schemes of similar strength in loss protection**, and FEC(100,k) is not suitable for delay-sensitive communication.

How do RPT and FEC flows perform under extreme load? One might think that in a highly congested link with a high fraction of RPT traffic, RPT flows would constantly overflow the buffer and the performance would drop. To create such a scenario with increased traffic load, we vary the fraction of video traffic in the link from 10% to 90%, while keeping the number of background TCP connections and bottleneck bandwidth the same. Detailed evaluation is provided in [88]. In summary, we find that **RPT flows achieve close-to-ideal video quality, and better quality compared to the best FEC scheme** in all cases (10% to 80%) except for one very extreme case (90%) with very heavy cross traffic creating loss rate $> 10\%$. The extreme case we portray in our experiment is unlikely to occur in practice for two reasons: 1) The loss rates in practice are likely to be much lower. 2) Even the aggressive estimate suggests that no more than 15% of traffic in future will be real-time in nature [3].

4.7.4 Impact of RPT on the Network

We now examine the effect of RPT flows on other cross traffic and the network. For this evaluation, we use the same simulation setup and topology described in §4.7.3.

How do other TCP flows perform? We look at the impact on two different types of TCP flows: long-running TCP flows and HTTP-like short TCP flows.

To evaluate the *impact on long-running TCP*, we send 100 long-running TCP flows and a varying number of video flows to vary the fraction of video traffic on the bottleneck link (from 10 to 90%). We also vary the redundancy parameter from 0 (Non-RPT) to 5. We use a small router buffer size of $\frac{2 \cdot B \cdot RTT}{\sqrt{100}}$ [26] to maximize the negative impact.

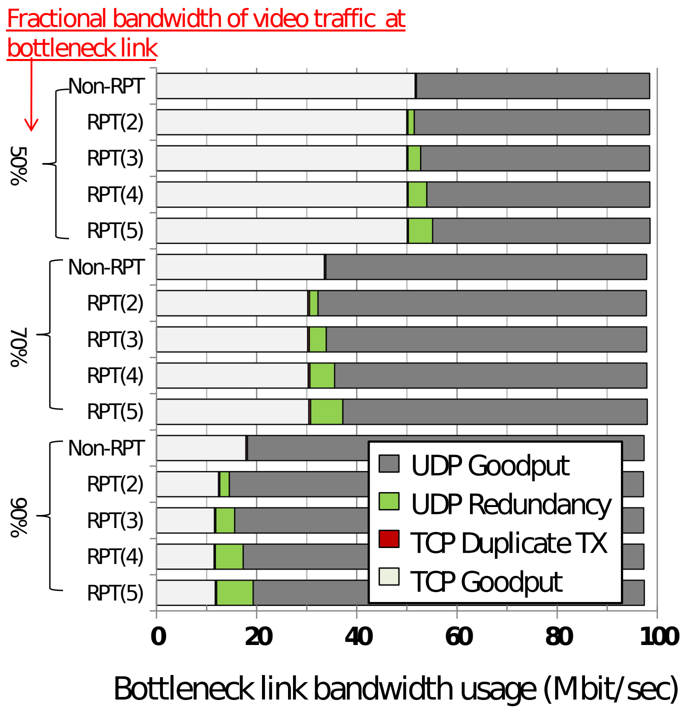


Figure 4.13: Breakdown of bottleneck link utilization: RPT flows do not impact the link utilization. RPT flows are prioritized over competing TCP flows.

Figure 4.13 shows the bottleneck link traffic decomposition in four categories[8]: UDP goodput, UDP redundancy, TCP duplicate, TCP goodput. UDP goodput is the bandwidth occupied by the original packet and the UDP redundancy represents the bandwidth occupied by the compressed packets. TCP goodput represents packets contributing to application throughput, and TCP duplicate Tx shows the amount of duplicate TCP packets received.

We observe that **1) the bandwidth utilization is not affected by RPT, and 2) RPT flows are effectively prioritized over non-RPT TCP flows.** In all cases the bottleneck bandwidth utilization was over 97.5%; TCP fills up the bottleneck even if the router queue is occupied by many decompressed redundant packets. TCP throughput, on the other hand, is impacted by the RPT flow especially when the network is highly congested; e.g. in the 90% video traffic case (bottom-most bars), TCP goodput (white region) decreases when RPT is used. This is because when RPT and non-RPT cross traffic competes, even though they experience the same underlying packet loss rates, for RPT flows packet loss do not directly translate into throughput loss. With redundant transmissions, the network recovers the loss through subsequent uncompressed

[8]Only a subset of results (video traffic occupying 50% to 90% of bottleneck) shown for brevity, but all cases confirm the same results.

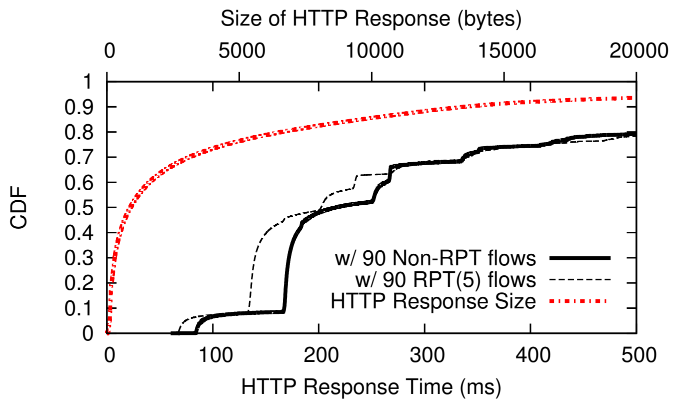


Figure 4.14: Response time and size for short HTTP flows (long flows omitted for clarity).

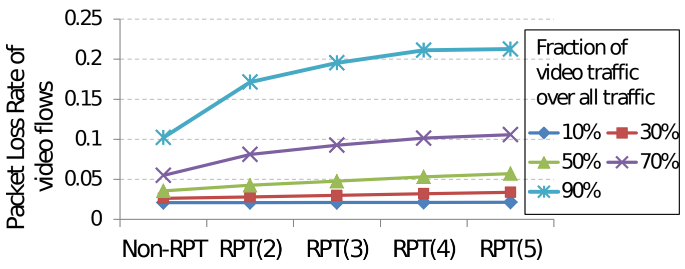


Figure 4.15: Impact on loss rate due to RPT flows.

redundant packets, effectively prioritizing the RPT flows.

To see the *impact on HTTP-type short flows*, we look at how RT changes the response time of short flows under the setup described in §4.7.3. Figure 4.14 shows the CDFs of the size of the response messages, and the response times. To highlight the difference, we only show response times when 90% of the traffic is video and RPT(5) is used, but the trend is visible across all cases.

The response time for short flows decreases in the presence of RPT flows. Since redundant packets in the queue are compressed when they are sent out, the service rate of the queue increases with RT. Therefore the queuing delay is reduced, which results in a decrease in the response time. However, for larger flows (tail end of the figure) the response time actually increases, as they behave more like long-running TCP flows, which obtain less throughput under congestion (Figure 4.13).

How does network behavior change with RPT flows? There are subtle changes in loss rate and queuing delay.

Loss rate: Figure 4.15 shows the packet loss rate of UDP video flows at the bottleneck router with varying amount of RPT traffic. **When the fraction of video traffic is moderate, adding**

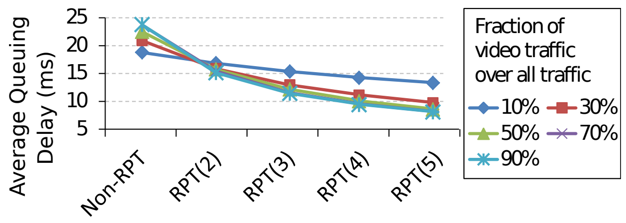


Figure 4.16: Queuing delay is reduced with RPT flows.

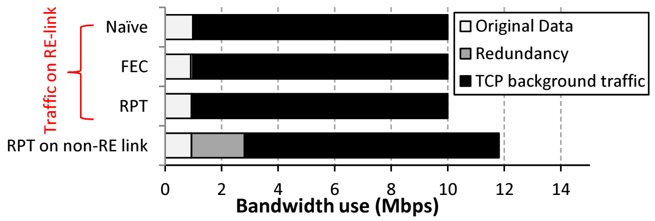


Figure 4.17: *No Harm:* Bandwidth use on a non-RE link in the no harm case.

more redundancy has little impact on the underlying packet loss of the video flow. This is because while redundant packets increase the load, they also increase the service rate of the link. However, we observe that when RPT flows dominate the bottleneck link, the underlying loss rate for video flows goes up as redundancy increases. The underlying reason for increased loss is that under congestion RPT flows compete with each other for bandwidth when most of the traffic is from RPT flows. However, in §4.7.3, we saw that even under such extreme conditions RPT performs better than FEC.
Queuing Delay: Figure 4.16 shows the average queuing delay with varying redundancy parameters and varying number of RPT flows. **Redundant packets decrease queuing delay.** This is because redundant packets appear as decompressed at the router queue, but are sent out compressed at the bottleneck link. Therefore, as the number of redundant packets increase, service rate becomes faster.

What's the impact of partial content-awareness? In §4.6, we noted that RT may cause harm in partially content-aware networks and should be used only after detecting RT-safety. Here, demonstrate both the *no-harm* and the *harm* case, and quantify the impact using our experimental testbed, which has a 10Mbps and a 100Mbps RE link.

To demonstrate *no-harm*, we disabled the RE encoder on the non-bottleneck 100Mbps links of our testbed. We then generated an RPT flow and TCP background flows through this 100Mbps link and the 10Mbps RE-enabled link. Figure 4.17 shows the traffic on both links. The RPT flow occupying 1Mbps on an RE-enabled bottleneck link introduces almost 2Mbps of overhead

	Non-RPT	RPT(3)
TCP traffic (Mbps)	5.7	3.6
Video traffic (Mbps)	4.0	4.0
Total (Utilization)	9.7 (97%)	7.6 (76%)

Table 4.5: *Harm:* Bandwidth use on a RE-link.

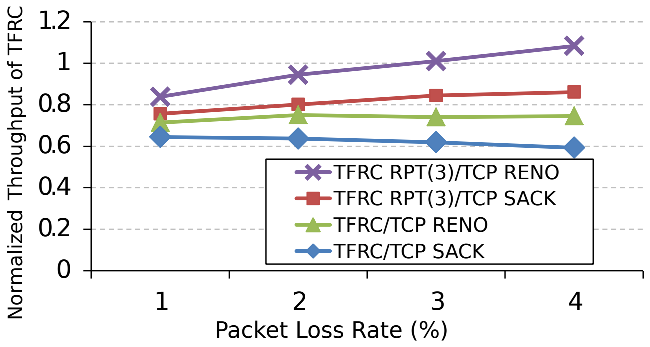


Figure 4.18: TFRC RPT under random loss.

(`redundancy`) on the non-RE 100Mbps link. However, this causes no harm since the 100Mbps link is not bandwidth constrained.

To demonstrate *harm*, we reduced the capacity of the non-RE link to 15Mbps, and introduced four 1Mbps video flows and a TCP flow. Table 4.5 compares bandwidth use on the 10Mbps RE link when the video flows are sent with and without redundancy. When there's no redundancy (`Non-RPT`), the link utilization of the 10Mbps link is 97%. When RPT(3) is used, the 4Mbps video flows occupy 11.2Mbps on the non-RE link. This shifts the bottleneck to be the 15Mbps non-RE link, which forces the 10Mbps RE-link and the network to be under-utilized at 76%. This verifies that in a partial deployment setting, detecting RT-safety is important as discussed in §4.6.

4.7.5 TCP-Friendly RPT

RPT flows do not give up bandwidth as easily under congestion. In Section 4.5, we discussed an alternative that makes RPT flows achieve fair bandwidth sharing using TCP-friendly rate control. In particular, we showed that incorporating TFRC requires careful adjustment of loss event rate, and explained how it should be done in two distinct loss patterns: *Uniform random* and *Temporal* packet loss.

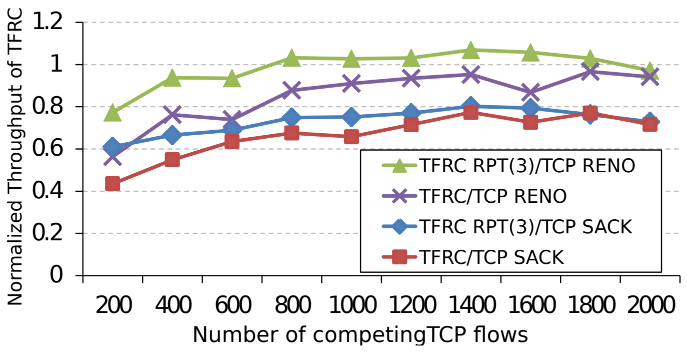


Figure 4.19: TFRC RPT exhibit TCP friendliness.

Is TFRC RPT TCP-friendly? We first evaluate our scheme under the *two extreme loss patterns* created artificially, and evaluate it under a more realistic loss pattern.

Uniform random loss: In this setting, TFRC RPT behaves in a TCP friendly manner without any adjustment in the loss estimation. Figure 4.18 shows the normalized throughput of TFRC and TFRC RPT(3) with respect to TCP Sack and Reno under 1% to 4% random loss. TFRC RPT(3) performs slightly better than TFRC because multiple packet losses within an RTT are counted as one loss event, and therefore the loss event rate for RPT(3) is slightly lower than that of normal TFRC.

Temporal packet loss: Here, we adjust the loss event rate of TFRC RPT(r) to be r times the observed loss event rate. To validate TCP-friendliness, we evaluated the performance of TFRC RPT(3) and TFRC under the same temporal loss pattern. To create such a pattern, we generated the same cross traffic, but artificially modified the router's queue so that redundant packets do not increase the queue length. Indeed, the performance difference of the two was less than 3% with the adjusted loss event rate.

Realistic environment: The two cases appear in an inter-mixed way in practice. As discussed in Section 4.4, an adjustment factor between 1 and r is sufficient for TCP friendliness. To create realistic loss patterns, we ran TFRC with competing TCP flows. The same dumbbell topology with 1 Gbps bottleneck link capacity is used. We vary the number of competing TCP flows from 200 to 2000. Each flow's RTT is randomly selected between 40ms and 200ms. Among the TCP flows, five of them are set to have the same RTT as the TFRC flows. We compare the relative throughput of TFRC flows to the average throughput of TCP flows. Our result shows TFRC RPT(3)'s performance reasonably matches that of TCP when $\alpha = 1.5$ across various loss rates. Figure 4.19 shows the normalized TFRC RPT(3)'s performance with respect to TCP Reno and TCP Sack. The result show that **TFRC RPT is TCP friendly.**

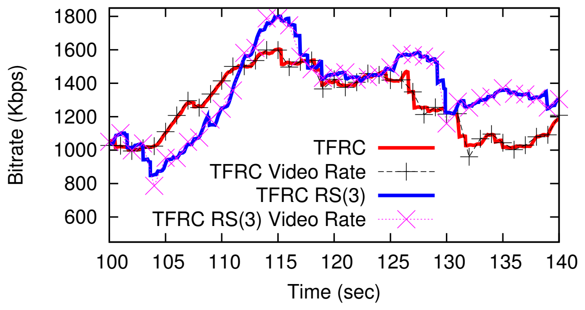


Figure 4.20: We adapt the encoding rate to TFRC's rate.

Adaptive encoding: From the TFRC's sending rate, we generate the video stream in the following way. At every RTT interval, TFRC yields a sending rate based on the average loss event rate over a much longer period of time. However, we adapt the video encoding rate at every second to the current sending rate of TFRC. Since TFRC's sending rate can change during a one second interval, we account for the difference in the next interval. The difference is small as TFRC's sending rate is already a smooth value. Figure 4.20 shows that he video rate sampled every second, closely matches with the TFRC's sending rate.

Video quality: We created video streams using TFRC and TFRC RPT; in either case, we output video according to the TFRC's or TFRC RPT's sending rate.[9] We compare the video quality of normal TFRC, normal TFRC with FEC, and TFRC RPT under the same cross traffic. We vary the cross traffic to create TFRC flows whose throughputs range from 562Kbps to 2.0Mbps. For TFRC with FEC, we choose the parameter which gives the best PSNR with delay under 150ms. Figure 4.21 shows the video quality achieved by the TFRC flows. **We see that TFRC RPT gives the best video quality in all cases.**

4.7.6 Evaluation Summary

In summary, our evaluation highlights three key properties of RPT claimed in Section 4.4.2: *high degree of robustness for applications with low overhead*, *ease of use and flexibility for developers*, and *flow prioritization in the network*. In Section 4.7.2, we showed that RPT provides high robustness and low bandwidth overhead, which translated into higher quality for video applications. In Section 4.7.3, we showed that, unlike FEC, RPT is easy to use since careful parameter tuning is not necessary, and delay can be independently controlled from the redundancy. Section 4.7.4 showed that RT flows are effectively prioritized over non-RT flows in congested links.

[9]For more details, refer to our technical report [88].

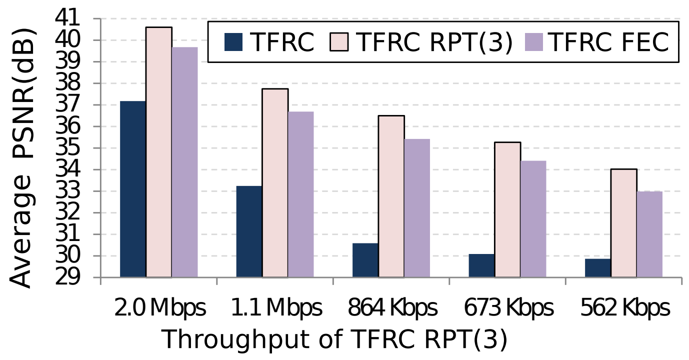


Figure 4.21: Video quality comparison.

In cases where fair-sharing is desired, we show RT can share bandwidth in a TCP friendly way while still retaining other benefits of RPT in Section 4.7.5.

4.8 Chapter Summary

As networks provide new functionality with integrated resources, more complicated interactions occur between the network and the end-points. At the same time, our network design in Chapter 3 allows new network-level functionality to be introduced in an explicit fashion. Therefore, it is important to study the interactions and the implication in the perspective of end-point and application design. In this chapter, we explored the intersection of the two; we presented a case study of a network end-point interaction in conjunction with the implications of our network design. We showed that end-points also have to evolve when networks evolve to provide new functionality and XIA also facilitates end-points' evolution.

XIA's support for end-point adaptation: Using redundancy elimination as an example, we demonstrated that XIA facilitates adaptation of end-points. We showed that applications can explicitly invoke new network-level functionality they want and identify whether or not the path supports the new function. We introduced RE as a new principal type by defining its semantics, identifier, and per-hop behavior. Our flexible addressing allowed end-points to explicitly invoke the desired functionality and identify the path information. This, in turn, allowed the opportunity to adapt end-point's strategy depending on the functionality provided by the path.

Redesign of redundancy in content-aware networks: As an example of end-point adaptation, we presented a case for the redesign of a redundancy-based loss recovery scheme. We explored issues arising from the confluence of two trends – growing importance and volume of real-time traffic, and the growing adoption of content-aware networks. This chapter examined a key problem at this intersection, namely that of protecting real-time traffic from data losses in content-aware networks. We showed that adding redundancy in a way that network understands reduces the cost and increases the benefits of loss protection quite significantly. We referred to our candidate loss protection approach as redundant transmission (RT). Using Redundant Packet Transmission (RPT) in redundancy-elimination networks [19] as an example, we highlighted various features of RT and establish that is a promising candidate to use in several practical content-aware networking scenarios. We showed that RT decreases the data loss rate by orders-of-magnitude more than typical FEC schemes applicable in live video communications, and delivers higher quality video than FEC using the same bandwidth budget. RT provides fine-grained control to signal the importance of data and satisfies tight delay constraints. Yet, it is easy to use as its performance is much less sensitive to parameter selection. We showed that constant bitrate RT flows are prioritized over non-RT flows, but can share bandwidth fairly by incorporating TCP friendly rate control into RPT.

Chapter 5

Designing a Common Behavior to Support Evolution

This chapter studies how we might introduce a common behavior in the network without undermining the network's ability to evolve over time. In designing a new network architecture, we often have to design a set of common behaviors that every router and end-points must agree on. Features supported by these common behaviors often concern resource allocation in the network, and are essential and critical to the operation of the network. For example, denial-of-service prevention mechanisms, such as shut-off [23], require cooperation of end-points and routers and a common agreement between them.

These features have two distinct characteristics:

- These features are commonly applied to all packets regardless of their principal types.
- These features require a consistent processing in the network and sometimes they must be applied on every hop. Examples include explicit congestion control algorithms, quality-of-service features, and active queue management.

The key challenge in designing such common behaviors is that once they are designed it is very hard to change them. This can significantly hinder evolution and prevent the network to support important features that may be needed in the future. One solution to this problem is to allow multiple behaviors to coexist in the same network similar to our XIP design. However, this is not always possible. For example, two different resource allocation schemes or different styles of congestion control (e.g, TCP and explicit congestion control) often cannot coexist. We take an alternative approach for evolution where we carefully design the common behavior to accommodate diverse requirements within the same framework. Unfortunately, very little research shows how we can design such common behaviors across all routers and end-points without hindering evolution. Therefore, in this chapter, we take congestion control as an example and perform a case study to draw lessons on how we should design such components. In particular, our goal is to design a congestion control algorithm that supports evolution by allowing some key pieces, but not all, to evolve over time. This chapter takes an important step towards designing a common behavior to support evolution and demonstrates that this is indeed possible. Our results show that it is critical to introduce the right abstraction that maximizes the flexibility.

Chapter outline: Section 5.1 introduces the problem of supporting evolution in congestion control and states our goal. We discuss related work in Section 5.2 and explore the requirements in Section 5.3. Section 5.4 presents our design of an evolvable framework for congestion control, and Section 5.5 address remaining practical issues of the design. Section 5.6 evaluates our design and demonstrates that the framework supports evolution. Finally, we summarize the lessons from this chapter in Section 5.7. The results we present in this chapter are based on our work appeared in [91].

5.1 Evolution Support in a Congestion Control Framework

Networked applications often require a wide range of communication features, such as reliability, flow control, and in-order delivery, in order to operate effectively. Since the Internet provides only a very simple, best-effort, datagram-based communication interface, we have relied on transport protocols to play the key role to implementing the application's desired functionality on top of the simple Internet packet service. As application needs and workloads have changed over time, transport protocols have evolved to meet their needs. In general, changes to transport protocols, such as adding better loss recovery mechanisms, have been relatively simple since they require only changes to the endpoints. However, among the functions that transport protocols implement, congestion control is unique since it concerns resource allocation, which requires coordination among all participants using the resource. As a result, two very different styles of congestion control cannot coexist, making evolution of congestion control itself very difficult.

The need for evolution in congestion control becomes apparent when we look at the history of TCP. While TCP's congestion control was not designed with evolution in mind, the development of TCP-Friendliness principles [74] enabled the development of a wide range of congestion control techniques to meet different application requirements; these include support for: streaming applications that require bandwidth guarantees [37, 55, 74, 133], or low-latency recovery [89, 119], non-interactive applications that can leverage low-priority, background transfer [180], applications that require multi-path communication for robustness [189], and Bittorrent-like content transfers that involve in transferring chunks from multiple sources.

The ability for TCP's congestion control to evolve has been enabled by two key aspects of its design: 1) Purely end-point based nature allowed new algorithms to be easily deployed and 2) its AIMD-based congestion avoidance led to the notion of TCP-friendliness.

Unfortunately, recent efforts, such as RCP [68] and XCP [106], rely on explicit congestion feedback from the network. While these designs are far more efficient than TCP, they limit the range of different end-point application behaviors that the network can support. It is not understood whether such explicit congestion control algorithms can be made flexible to support evolution. In this chapter, we explore the design of FCP (flexible control protocol), a novel congestion control framework that is as efficient as explicit congestion control algorithms (e.g., RCP and XCP), but retains (or even expands) the flexibility of TCP-friendliness based solutions.

Problem Scope: Our goal is to design a congestion control framework that supports evolution while retaining the efficiency of explicit congestion control algorithms. Note that this does not address the problem of evolving the congestion control framework itself (e.g., transiting from TCP-based congestion control to an XCP-based congestion control). We know of no existing solution other than providing complete isolation through network virtualization. While it is an open problem, this problem is beyond the scope of this dissertation. We argue that such transition should be a rare event and argue that each congestion control framework should be designed to support evolution. This allows each framework to support diverse application requirements and minimizes the need to undergo radical changes in the network. Thus, we focus on designing a congestion control framework that is as flexible as an end-point-based design (e.g., TCP), but that is also as efficient as explicit congestion control algorithms. Finally, we note that during the rare event of transition relatively heavy weight solution of virtualization can be used.

Summary of Approach and Contributions: FCP leverages ideas from economics-based congestion control [107, 108] and explicit congestion control. In particular, to enable flexible resource allocation with in a host, we allow each domain to allocate resources (budget) to a host, and make networks *explicitly* signal the congestion price. The flexibility comes from the end-points being able to assign their own resources to their flows and the networks being able to aggregate flows and assign differential price to different classes of flows to provide extra functionality. The system maintains a key invariant that the amount of traffic a sender can generate per unit time is limited by its budget and the congestion price. Co-existence of different styles and strategies of rate control is ensured simply by maintaining this key invariant, allowing evolution.

Our primary contribution is showing that explicit congestion control algorithms can be made flexible enough to allow evolution just as end-point based algorithms. To this end, we design an explicit congestion control algorithm, called FCP, and demonstrate that FCP easily allows different features to coexist within the same network; end-hosts can implement diverse styles of rate control and networks can leverage differential pricing and aggregate control to achieve different goals.

Our secondary contribution is to make economics-based congestion control [107, 108, 109] more practical by extending past theoretical work in three critical ways:

- FCP uses "pricing" as a *pure abstraction* and the key form of *explicit feedback*. In contrast, some existing approaches [45, 62, 108, 109] directly associate congestion control to the real-world pricing and modify existing congestion control algorithms to support weighted proportional fairness. Others [27, 117, 129] use active queue management and single-bit Explicit Congestion Notification (ECN), which is less efficient than explicit congestion control algorithms.
- Unlike previous explicit rate feedback designs [68, 107, 123], FCP accommodates high variability and rapid shifts in workload, which is critical for flexibility and performance. This property of FCP comes from a novel *preloading* feature that allows senders to commit the amount of resource it wants to spend ahead of time.
- Finally, we address practical system design and resource management issues, such as dealing with relative pricing, and show that FCP is efficient. In particular, we improve upon Low and Lapsley's preliminary design in [123], which does not address many practical issues.

5.2 Related Work

Our framework builds upon concepts from previous designs to provide a generalized framework for resource allocation.

Economics-driven resource allocation has been much explored in the early days of the commercial Internet [60, 141, 164]. MacKie-Mason and Varian [126] proposed a *smart market* approach for assigning bandwidth. Since then many have tried to combine economics-driven resource allocation with congestion control, which led to the development of economics-based congestion control.

Kelly [107] proved that when users choose the charge per unit time that maximizes their utility, the rate can be determined by the network so that the total utility of the system is maximized, and the rate assignment satisfies the weighted proportional fairness criterion. It also proved that two classes of globally stable algorithms can achieve such rate allocation: 1) The primal algorithm implicitly signals the congestion price, and the senders use additive increase/multiplicative decrease rules to control rate. 2) The dual algorithm uses explicit rate feedback based on shadow pricing.

The primal algorithms use end-point based congestion control [45, 62, 79]. MulTCP [62], for example, adjusts the aggressiveness of TCP proportional to the host's willingness to pay. Gibbens and Kelly [79] show how marking packets (with an ECN bit) and charging the user the amount proportional to the number of marked packets achieves the same goal, and allows evolution of end-point based congestion control algorithms. The dual algorithm uses rate as an explicit feedback, and was materialized by Kelly et al. [109] by extending RCP [68].

FCP also adopts proportional fairness, but is different from these approaches in that 1) it uses congestion pricing as a pure abstraction, decoupling from real-world pricing and billing, and 2) it is an explicit congestion control algorithm that is designed for supporting evolution using price as an explicit feedback. FCP also addresses practical issues of implementation and deployment that others do not address [107, 109], such as differential pricing and accommodating large fluctuations in workload.

Network Utility Maximization: Kelly's economics-based congestion control led to many developments. On the theory side, It was generalized into network utility maximization (NUM) [194]. Many congestion control algorithms and active queue management schemes have been designed based on this framework [27, 117, 123, 129]. In these protocols, every sender tries to maximize its own benefit, utility minus the bandwidth cost, and each link generates its price similar to the original economics-based congestion control. However, the key differences from the original congestion control are that the cost is not associated with any monetary value, and senders agree on a common behavior. With the exception of a preliminary design of explicit price feedback in [123], all other designs utilize ECN-bit to deliver the cost information on top of TCP. Low and Lasley's initial design was a form of explicit congestion control with price feedback. However, it did not address many practical problems that this chapter addresses. For example, they propose a initial slow-start behavior which defeats the purpose of explicit congestion control algorithms. They also suffer from inaccurate feedback similar to RCP when the load frequently changes.

Congestion control with various features: We categorize works that introduce new features into congestion control in §5.3.1. Our work enables such diverse features to coexist by using pricing as the core abstraction in the common congestion control framework and by allowing aggregation and local control, generalizing the idea first explored in CM [29].

Extensible transport: Others focus on designing an extensible [44, 147] or configurable [40] transport protocol. These works either focus on security aspects of installing mobile code at the end-host or take software engineering approaches to modularize transport functionality at compile time. However, the core mechanism for coexistence is TCP-friendliness.

Virtualization: Finally, virtualization [24, 166] partitions a physical network allowing completely different congestion control algorithms to operate within each virtualized network. Although this is good for creating a testbed for new networking technologies and protocols that run in isolation [166], it is not meant to support coexistence of diverse protocols in reality and has a number of practical limitations. Slicing bandwidth creates fragments and reduces the degree of statistical multiplexing, which we rely on to provision resources. Also, this increases the complexity of applications and end-hosts who want to use multiple protocols simultaneously as they have to participate in multiple slices and maintain multiple networking stacks.

5.3 Requirements and Principles of Design

In this section, we first identify two key requirements for evolution and closely look at each of them (Section 5.3.1). We then describe the two key principles that the FCP design uses to satisfy these requirements (Section 5.3.2).

5.3.1 Requirements for evolution

To support evolution, the system must **accommodate diversity** such that different algorithms and policies coexist and be **flexible** enough that new algorithms can be implemented to accommodate (future) changes in communication patterns.

Accommodating diversity: To understand the nature of diversity, we categorize previous work on network resource allocation into four categories:

1. Allocating resources locally: Prior works, such as congestion manager [29], SCTP [136], and SST [75], have shown benefits of flexible bandwidth sharing across (sub) flows that share a common path.

2. Allocating resource within a network: Support for differential or weighted bandwidth allocation across different flows has been explored previously. Generalized AIMD [33, 193], weighted fair-sharing [62, 63], and support for background flows [180] are some examples.

3. Allocating resource in a network-wide fashion: Bandwidth is sometimes allocated on aggregate flows or sub-flows: Multipath TCP (MPTCP) [189] controls the total amount of bandwidth allocated to its sub-flows; distributed rate limiting [150] and assured forwarding in DiffServ control resource allocation to a group of flows. In such systems, flows that do not traverse the same path are allocated a resource that is shared among themselves (e.g., MPTCP is fair to regular TCP at shared bottlenecks). In such designs, one can be more aggressive in some parts of the network at the expense of being less aggressive in other parts [150].

4. Stability in resource allocation: Although strict bandwidth or latency guarantees require in-network support, many transport designs aim to provide some level of performance guarantees [74, 173]. For example, TFRC is designed such that the throughput variation over consecutive RTTs is bounded and OverQoS [173] provides similar but probabilistic guarantees.

Flexibility is the key to accommodating as yet unforeseen communication styles of the future. To achieve this, more control should be exposed to the end-host as well as to the network. Furthermore, a scalable mechanism is needed to exert control over aggregate flows—a scheme that requires per-flow state in the network for control would not work.

Note that apart from these requirements for evolution, there are also common, traditional goals such as high efficiency, fast convergence, and fair bandwidth allocation. Many works [68, 73, 86, 106, 171], such as TCP-cubic, XCP, and core-stateless fair-queuing, fall into this category. Our goal is to design a flexible congestion control framework that accommodates above-mentioned diversity and flexibility, while still achieving these traditional goals.

5.3.2 Principles of design

FCP employs two key principles for accommodating diversity and flexible resource allocation: *aggregation* and *local control*.

Aggregation: FCP assigns resources to a group of flows. Aggregation allows the network to control the amount of resources that the group is consuming in a distributed and scalable fashion while preserving the relative weight of the individual flows within the group. For example, we assign resources (a budget) to a host so that the host can generate traffic proportional to the amount of budget it has. Aggregation also simplifies control and enforcement; the network can enforce that a host is generating traffic within its allocated resources without having to keep per-flow state. For additional flexibility, we allow aggregation at various levels. As networks aggregate flows that come from a host, ISPs can also aggregate traffic coming from another ISP and assign resources. For example, each domain can independently assign weights to the neighboring domains traffic or to any group of flows. As we show in Section 5.4, such aggregation provides a mechanism for the network to ensure fairness while allowing flexibility at the end-host, one of the key ingredients for evolution.

Local control: Local control gives freedom to distribute the resource to individual flows, once resources are allocated on aggregate flows. For example, when the network assigns resources to the aggregate flows from a single host, the host can control how to distribute the resource locally amongst its flows (diversity category 1 in Section 5.3.1). The network then respects the resource (or weight) given to the flow and assigns bandwidth proportional to its weight (category

2 support). A host can also spend more resources on some flows while spending less on others (diversity category 3 support). Various points in the network may also control how their own resources are shared between groups of flows. For example, networks can allocate bandwidth in a more stable manner to certain group of flows (category 4 support).

As such, both the end-host and the network have local control, ensuring flexibility; the network decides how its own resource is used within its domain, and end-hosts decide how to use their own resources given the network constraint.

Feedback design: Designing the form of network to end-point feedback that meets both requirements is challenging. Providing an absolute feedback as in XCP or RCP leaves no local control at the end-point because the feedback strictly defines the end-point's behavior. Using an abstract or implicit feedback, such as loss rate or latency, or loosely defining the semantics of feedback, on the other hand, increases end-point control, allowing a range of end-host behaviors. However, using such feedback typically involves guesswork. As a result, end-hosts end up probing for bandwidth, which in turn sacrifices performance and increases the convergence time (e.g., in a large bandwidth-delay product link). Providing differential feedback for differential bandwidth allocation is also hard in this context. As a result, differential bandwidth allocation typically is done through carefully controlling how aggressively end-points respond to feedback relative to each other [193]. This, in turn, makes enforcement and resource allocation on aggregate flows very hard. For example, for enforcement, the network has to independently measure the implicit feedback that an end-point is receiving and correlate it with the end-point's sending rate.

We take a different approach by leveraging ideas from Kelly [79, 107]. We use pricing as a form of explicit feedback and design an evolvable explicit congestion control algorithm by supporting flexible aggregation and local control. Contrary to the common belief, we demonstrate explicit congestion control can be made flexible to allow evolution.

5.4 Design

We now describe our design in steps.
(1) Each sender (host) is assigned a budget ($\$/sec$), the maximum amount it can spend per unit time. We first focus on how FCP works when the budget is assigned by a centralized entity. Our budget defines the weight of a sender, but is different from the notion of "willingness to pay" in [107] in that it is not a real monetary value. Later, in Section 5.4.3, we extend FCP to support distributed budget management in which each domain can assign budgets completely independently and may not trust each other. The central challenge there is how to value a budget assigned by a different domain. We show that FCP allows flexible aggregate congestion control

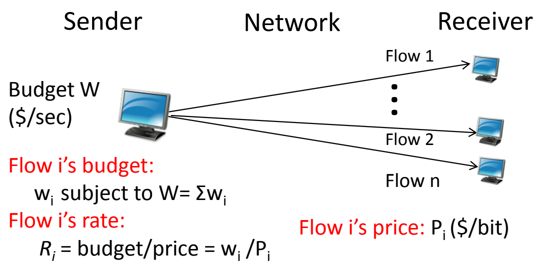

Sender
Network
Receiver
Budget W ($/sec)
Flow 1
Flow 2
Flow n
Flow i's budget:
w_i subject to $W= \Sigma w_i$
Flow i's rate:
R_i = budget/price = w_i / P_i
Flow i's price: P_i ($/bit)

Figure 5.1: Design Overview

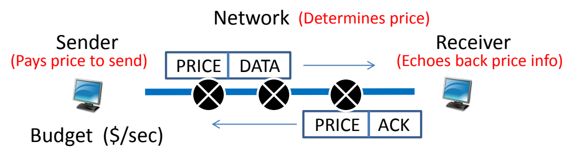


Figure 5.2: Network generates the price feedback.

by dynamically translating the value of a budget assigned by different entities.
(2) At the start of a flow, a sender allocates part of its budget to the flow. This budget now determines the weight of the flow. End-point evolution is enabled at this stage because the sender can implement various strategies of resource allocation to achieve different goals (see Section 5.4.2).

Figure 5.1 illustrates this. The sender has a budget of W, and can distribute its budget to its flows at its will provided that the sum of the flows' budgets, $\sum w_i$, is less than or equal to W. The rate of flow i, R_i, is then defined as w_i/P_i where P_i is the price for the path flow i is traversing. This allows the end-host to control its own resource and achieve differential bandwidth allocation on a flow-by-flow basis, as we show in Section 5.4.2. FCP also allows this budget assignment to change at any time in the lifetime of a flow, providing additional flexibility. As such, we reintroduce the end-point's flexibility of end-point based designs to an explicit congestion control algorithm. Next, we show how the path price is generated.
(3) The network determines the congestion price ($\$/bit$) of each link. In FCP, the sender learns the path price in the form of explicit feedback. The price of path r, P_r, is defined as the sum of link prices: $P_r = \sum_{l \in r} p_l$, where p_l is the price of link l in path r. To ensure efficiency, the price adapts to the amount of budget (traffic times its price) flowing through the system and its capacity. In Section 5.4.1, we first show a uniform pricing scheme in which all packets see the same price. In Section 5.4.2, we show how networks can employ differential pricing and aggregate congestion control to support various features such as quality of service, aggregate resource allocation, and multicast-aware congestion control.
(4) The flow now ramps up to its fair-share, R_i, by spending its budget w_i. Two problems remain at this stage: At the start of the flow the sender does not know the path price, P_i. More importantly, the network does not know the budget amount (w_i) it should expect for accurate price generation. This leads to serious problems when the workload changes rapidly. For example, when a path is not congested, its congestion price will be an infinitesimal value, ε. Any $w_i >> \varepsilon$ will set the sending rate to an infinite value, and overload the network significantly. To address this problem, we introduce *preloading*, a distinct feature of our design that allows

a host to rapidly increase or decrease a flow's budget. Preloading allows the sender to specify the amount of budget increase for the next round in multiples of the current price, allowing the network to adjust the expected input budget and generate prices based on this committed budget. The challenge here is to accurately calculate price using this information and for hosts and routers to update the budget w_i and the price P_i in a coordinated fashion so that w_i is only spent when the network is expecting it. This is especially challenging when the system is asynchronous and feedback is delayed. For this, in FCP, routers update the price on every packet reception, and hosts preload on a packet-by-packet basis when the budget changes (e.g., initial ramp up). In Section 5.4.1, we show how our pricing works with preloading and how the sender updates its budget, w_i.

The sender now ramps up in two steps using preloading:

(4-1) Along with the first packet, we preload how much we want to send in the next RTT. However, at this point we do not yet know the path price. Therefore, we preload a conservative amount. (See Section 5.4.1 for details.)

(4-2) After the price is discovered, the sender can preload the appropriate amount to spend the budget assigned to the flow. Preloading is also used when an active flow's budget assignment changes.

5.4.1 Flexible Control Framework

We now describe the details. In FCP, the system maintains the following *invariant*: For all hosts h,

$$\sum_{s \in Packets} price(s) \cdot size(s) \leq W_h \tag{5.1}$$

where *Packets* is the set of packets sent by host h during unit time, price(s) is the most recent price of the path that packet s is sent through, and W_h is host h's budget.

Explicit price feedback: To satisfy the invariant, senders have to know the path price. As Figure 5.2 illustrates, an FCP header contains a price value, which gets accumulated in the forward path and echoed back to the sender. Each router on the path updates the price in the header as: $price = price + p_l$, where p_l is the egress link price.

Pricing ensures network efficiency by dynamically adapting the price to the amount of incoming budget. We show how each router calculates the link price to achieve this. Each link's price must reflect the amount of incoming budget and the capacity of the link. Each router calculates the link price per bit $p(t)$ ($\$/bit$) at time t as:

$$p(t) = \frac{I(t)}{C - \alpha q(t)/d} \tag{5.2}$$

$$I(t) = \frac{\sum_{s \in (t-d,t]} p(t - rtt(s)) \cdot size(s)}{d} \tag{5.3}$$

Equation 5.2 sets the price as the incoming budget (amount of traffic times its price) over remaining link capacity. $I(t)$, the numerator, denotes the total incoming budget per unit time ($\$/sec$), which is the sum of all packets' prices ($) seen during the averaging window interval (t-d,t]. The denominator reflects the remaining link capacity, where C is the link-capacity, $q(t)$ is the instantaneous queue size, α is a constant, and d is the averaging window. $rtt(s)$ is the RTT of the packet s, and $p(t - rtt(s))$ is the past feedback price per bit. Unlike other explicit congestion control protocols [68, 106], the router calculates the link price at every packet reception, and keeps the time series $p(\cdot)$. We set d as multiples of average RTT (twice the RTT in our implementation).

Equation 5.2 deals with efficiency control by quickly adapting the price based on the incoming budget and the remaining link capacity. Fairness is achieved because everyone sees the same price, and therefore the bandwidth allocation is proportional to budget. In Section 5.6.1, we show simulations in various environments to demonstrate the stability of this algorithm, and perform local stability tests by introducing perturbations.

Equation 5.3 estimates the amount of incoming budget that a router is going to see in the future using recent history. When incoming budget, $I(t)$, is relatively constant over time the price is stable. However, when the input budget constantly changes by a large amount, the price will also fluctuate because the fair-share rate also fluctuates. This, coupled with the inherent delayed feedback, can leave the system in an undesirable state for an extended period. During convergence, the network may see high loss rate, under-utilization, or unfairness. Other state-of-the-art congestion control frameworks also have this problem. For example, when new flows arrive RCP results in periods of high loss, and XCP results in slow convergence to fair-share, as we show in Section 5.6.1.

This problem has been regarded as being acceptable in other systems because changes are viewed as either temporary or incremental. Theory of economics-based congestion control [107] shows that even for a theoretical proof of global stability the analysis requires such an assumption when users can dynamically adapt their willingness to pay. However, this is not the case in our system. One of the key enablers of evolution in FCP is the end-host's ability to arbitrarily assign its budget to individual flows, and we expect that rapid change in the input budget will be the norm. Increasing budget rapidly also allows senders to quickly ramp up their sending rates to their fair-share. However, this is especially problematic when the amount of budget can vary by large amounts between flows. For example, consider a scenario where a link has an incoming budget of 1\$/*sec*. When, a new flow with a budget of 1000 \$/*sec* arrives, this link now sees 1001x the current load. Preventing this behavior and forcing users to incrementally introduce budget will make convergence (to fairness) significantly slower and limit the flexibility of the end-hosts.

Preloading: To address this problem, we introduce *preloading*, a distinct feature of our design that allows a host to rapidly increase or decrease the budget amount per flow. Preloading allows the sender to specify the amount of budget willing to introduce in the next round, allowing the network to adjust the expected input budget and generate prices based on this committed budget.

However, the sender cannot just specify the absolute budget amount that it wants to spend on the path because the path price is a sum of link prices. Given only the total amount, each individual link cannot estimate how much budget is spent traversing it. Instead, we let senders preload in multiples of the current price. For example, if the sender wants to introduce 10 times more budget in the next round, it specifies the preload value of 10. For routers to take this preload into account, we update Equation 5.3 as follows:

$$I(t) = \frac{\sum_{s \in (t-d,t]} p(\cdot)\, size(s)\, (1 + preload(s) \cdot d / rtt(s))}{d} \tag{5.4}$$

The additional preload term takes account the expected increase in the incoming budget, and $rtt(s)$ accounts for the difference between the flow's RTT and the averaging window. Preloading provides a hint for routers to accurately account for rapid changes in the input budget. Preloading significantly speeds up convergence and reduces estimation errors as shown in Section 5.6.

Header: We now describe the full header format. An FCP data packet contains the following congestion header:

RTT	price	preload	balance

When the sender initializes the header, RTT field is set to the current RTT of the flow, price is set to 0, and preload to the desired value (refer to sender behavior below). The balance field is set as the last feedback price—i.e., the price that the sender is paying to send the packet. This is used for congestion control enforcement (Section 5.5). Price and preload value are echoed back by an acknowledgement.

Sender behavior with preloading: Senders can adjust the allocation of budget to its flows at any time. Let x_i be the new target budget of flow i, and w_i the current budget whose unit is \$/*sec*. When sending a packet, it preloads by $(x_i - w_i)/w_i$, the relative difference in budget. When an ACK for data packet s is received, both the current budget and the sending rate are updated according to the feedback. When preload value is non-zero, the current budget is updated as:

$$w_i = w_i + paid \cdot size(s) \cdot preload/rtt$$

where paid is the price of packet p in the previous RTT. The sending rate r_i is updated as: $r_i = w_i/p_i$, where p_i is the price feedback in the ACK packet. Note that preloading occurs on a packet by packet basis and the sender only updates the budget after it receives the new price that accounts for its new budget commitment. Also note that the preload can be negative. For example, a flow preloads -1 when it is terminating. Negative preloading allows the network to quickly respond to decreasing budget influx, and is useful when there are many flows that arrive and leave. Without negative preload, it takes the average window, d, amount of time to completely decay the budget of the flow that has departed.

Start-up behavior: We now describe how FCP works from the start of a flow. We assume a TCP-like 3-way handshake. Along with a SYN packet, we preload how much we want to send in the next RTT. However, because we do not yet know the path price, we do not aggressively preload. In our implementation of FCP, we adjust the preload so that we can send 10 packets per RTT after receiving a SYN/ACK. The SYN/ACK contains the price. Upon receiving it, we initialize the budget assigned to this flow w_i as: $price \cdot size \cdot preload/rtt$. After this, the sender adjusts its flows' budget assignments as described earlier.

5.4.2 Evolution

FCP's common framework enables evolution by providing greater flexibility with local control and aggregation. We show a number of examples that demonstrate end-point and network evolution. Coexistence is guaranteed as long as end-hosts stick to the invariant of Equation 5.1, which defines the host-level fairness. Therefore, it is much easier to design various resource allocation schemes.

End-point evolution: End-hosts can assign budget to individual flows at will. This enables intelligent and flexible policies and algorithms to be implemented. Below we outline several strategies that end-hosts can employ.

- Equal budget: Flow-level fairness between flows within a single host can also be achieved by equally partitioning the budget between flows. For example, when there are n flows, each flow gets $budget/n$. Then the throughput of each flow will be purely determined by the path's congestion level.

- Equal throughput: End-points may want to send equal rates to all parties with which it is communicating (e.g., a conference call). This is achieved by carefully assigning budget (i.e., assigning more budget to expensive flows).

- Max throughput: The goal is to maximize throughput. To maximize throughput, end-points use relatively more budget towards inexpensive paths (less congested) and use little budget for more congested paths. This generalizes the approach used in multipath TCP [189] even to subflows sent to different destinations as we show in Section 5.6.

- Background flows, similar to those in TCP Nice [180], are also supported by FCP. A background flow is a flow that only utilizes "spare capacity". When foreground flows are able to take up all the capacity, background flows should yield and transmit at a minimal rate. This can be achieved by assigning a minimal budget to background flows. When links are not fully utilized, the price goes down to "zero" and path price becomes marginal. Therefore, with only a marginal budget, background flows can fill up the capacity.

- Statistical bandwidth stability: Some flows require a relatively stable throughput [74]. This can be achieved by reallocating budget between flows; if a bandwidth stability flow's price increases (decreases), we increase (decrease) its budget. When the budget needs to increase, the flow steals budget from other flows. This is slightly different from smooth bandwidth allocation given by TFRC in that temporary variation is allowed, but the average throughput over a few RTTs is probabilistically stable. The probability depends on how much budget can be stolen and the degree of path price variation. Such a statistical guarantee is similar to that of OverQoS [173]. Later, we show how we can achieve guaranteed bandwidth stability with network support.

Note that these algorithms are not mutually exclusive; they can coexist as long as the invariant (Eq.5.1) is satisfied. Different hosts can use different strategies. Even within a host, different groups of flows can use different strategies.

Network and end-point evolution: End-points and networks can also simultaneously evolve to achieve a common goal. FCP's local control and aggregation allow routers to give differential pricing to different groups of flows. We show how this may support various features in congestion control. In our examples, algorithms change from the original design including the path price generation, link price calculation, and preloading behavior. However, the invariant of Equation 5.1 remains unchanged.

- Bandwidth stability: We implement a version of slowly changing rate with end-point cooperation and differential pricing: 1) For stability flows, end-points limit the maximum preload to

1 which limits the speed at which flow budget and rates can ramp up. 2) Routers have two virtual queues: stability and normal[1]. The stability queue's price variation during a time window of twice the average RTT is bounded by a factor of two. When the price has to go up more than that amount, it steals bandwidth from the normal queue to bound the price increase of the stability queue. As a result, the normal queue's price increase even more, but the bandwidth stability queue's price is bounded. When the price might go down by a factor of two during a window, it assigns less bandwidth to the stability queue. However, because we allow the stability queue's price to change, at steady state the prices of both queues converge to the same value. Below shows the algorithm in pseudocode.

```
function UPDATEPRICE(Packet p)
    // update average RTT
    // update total input budget, deadline queue's and best effort queue's input budget.

    // Calculate price according to Eq.5.2 using the total input budget
    price = calculatePrice(totalInputBudget, linkCapacity)
    if isOutofRange(price) then
        stabilityPrice = priceLimit(price) // Price increase/decrease is bounded
        // calculate BW needed to bound the price
        requiredBW = stabilityInputBudget/stabilityPrice/avgRTT/d
    else
        // calculate normal queue's price
        normalPrice = calculatePrice(nomalInputBudget, linkCapacity - requiredBW)
    end if
end function

function ISOUTOFRANGE(price)
    return true if the price is too high or too low.
    otherwise return false
end function
```

- Multicast-aware congestion control: FCP allows the network to evolve to support a different service model. For example, FCP can support multicast-aware congestion control by changing how the path price is generated from link prices. We sum up the link price in a multicast tree, by aggregating the price information at the router. The price of a multicast packet is the sum of the price of links that it traverses. To calculate the price, we sum up the price of a packet in a recursive manner in a multicast tree. When a router receives a multicast packet, it remembers its upstream link price, resets the price feedback to zero, and sends out a copy of the packet to

[1]We assume that routers can easily classify packets into these queues

each of its outgoing interfaces along the multicast tree (See pseudocode for ReceiveMulticast below). This allows us to account for the link price only once. Receivers echo back the price information. Upon receiving an ACK, each router computes the price of its subtree and sends a new ACK containing the new price. The price of the subtree is computed by summing up the price of the link from its parent to itself that it remembered previously and all the price feedback from its children in the multicast tree (See pseudocode for ReceiveACK below). Through a recursive process, the sender at the root receives the total price of sending a packet down the multicast tree.

```
function RECEIVEMULTICAST(Packet p, SourceAddress sa, MulticastAddress ma)
    upstremPrice[(sa, ma)] = p.price  // remember upstream link price
    p.price = 0  // reset price
    for all c in children[(sa,ma)] do
        send(p)
    end for
end function

function RECEIVEACK(Packet p, SourceAddress sa, MulticastAddress ma)
    price = 0
    price += upstremPrice[(sa, ma)]
    subTreePrice[previousHop(p)] = p.price
    for all c in children[(sa,ma)] do
        price += subTreePrice[c]
    end for
    p.price = price  // SubTree's price
    sendDelayed(p)  // Limit the number of ACKs to prevent ACK implosion
end function
```

- FCP can also support Paris Metro Pricing (PMP) style [141] differential pricing. The network can assign higher price to a priority service in multiples of the standard price. For example, the priority queue's price can be 10 times the standard queue. The end-host will only use the priority queue when it is worth it and necessary making the priority queue see much less traffic that the standard one.

- D^3[188]-style deadline support can be implemented using differential pricing with two virtual queues: deadline queue and best effort queue. Assume the best effort queue is assigned a small fraction of bandwidth at least. The deadline queue always gives a small agreed-upon fixed price. Flows indicate their desired rate by preloading. The router gives the fixed price only when the desired rate can be satisfied. Otherwise, it puts the packet into the best-effort queue and gives a normal price to control the aggregate rate to the bandwidth allocated to the

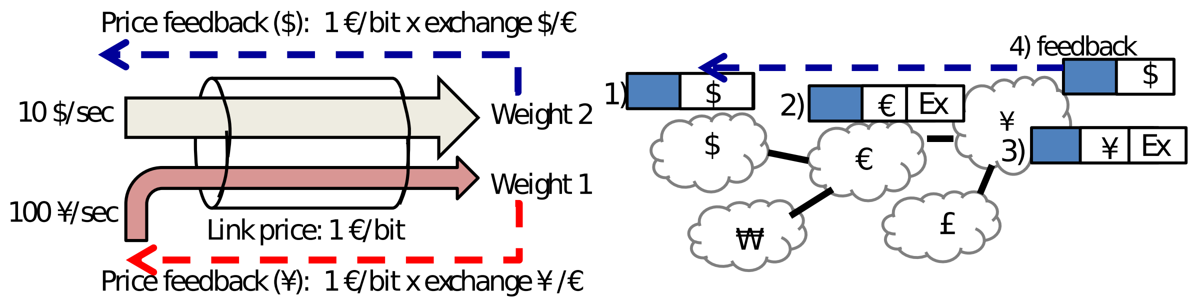


Figure 5.3: Budget management: (a) Aggregate congestion control using dynamic value translation. (b) Value translation between domains.

normal queue. The host first tries deadline support, but falls back to best effort service when it receives a price different from the fixed value after the preloading. Below is the algorithm in pseudocode.

```
function UPDATEPRICE(Packet p)
    (omitted)... // update average RTT

    if p in Deadline flow then
        updateDeadlineInputBudget(p)
    else
        updateBestEffortInputBudget(p)
    end if

    requiredBW = stabilityInputBudget/fixedDeadlinePrice/avgRTT/d
    if requiredBW > linkCapacity then
        subtractDeadlineInputBudget(p)  // Demote the flow to best-effort flow
        updateBestEffortInputBudget(p)
    end if

     // calculate best-effort queue's price
    bestEffortPrice = calculatePrice(bestEffortInputBudget, linkCapacity - requiredBW)
end function
```

5.4.3 Budget Management

Budget management is another form of flexibility that FCP provides. This allows networks to dynamically translate the value of a budget belonging to different flow groups or assigned by different domains. The former allows aggregate congestion control between groups of flows, and the latter enables distributed budget assignment between mutually untrusted domains.

Aggregate congestion control: FCP allows the network to perform aggregate congestion control over an arbitrary set of flows so that each group in aggregate attains bandwidth proportional to some predefined weight. This, for example, can be used to dynamically allocate bandwidth between multiple tenants in a data-center [167].

To achieve this, FCP views each group as having its own currency unit whose value is proportional to the weight of the group, and inversely proportional to the aggregate input budget of the group. When giving feedback, a router translates its price into the group's current currency value (Figure 5.3 (a)). For example, if flow group A has a weight of 2 and B has a weight of 1, and their current input budgets are, respectively, 10 $\$/sec$ and 100 $¥/sec$, A's currency has more value. To estimate the input budget for each group of flows, we use the `balance` field. Each router keeps a separate input budget for each group. It also keeps the aggregate input budget using its own link's price history and updates the price as in the original scheme using Equation 5.2. Thus, we calculate the normalized exchange rate E_G for flow group G as:

$$E_G(t) = \frac{w_G}{\sum_{H \in Group} w_H} / \frac{I_G(t)}{\sum_{H \in Group} I_H(t)}$$

And adjust the price feedback for packet s as:

$$price(s) = price(s) + p(t) \cdot E_G(t)$$

where $p(\cdot)$ is defined in Equation 5.2.

The implementation can be efficient, requiring O(1) computation for every packet arrival. This can be achieved using a timer, where a timer decays the input budget and updates the exchange rate for the queue when a packet expires out of the averaging time window. At every packet reception, we also update the input budget and exchange rate of the queue that the incoming packet belongs to.

Inter-domain budget management: In an Internet scale deployment, budget assignment must be made in a distributed fashion. One solution is to let each ISP or domain assign budget to its customers without any coordination, and rely on dynamic translation of the value. Here, we outline this approach, and leave the exact mechanism and demonstration as future work. When a packet enters another domain, the value the packet is holding (balance field in the header) can be translated to the ingress network's price unit (Figure 5.3 (b)). One can use a relatively fixed exchange rate that changes slowly or a dynamic exchange rate similar to the previous design. This allows the provider (peering) network to assign bandwidth to its customers (peers) proportional to their weight. Only one thing needs to be changed in the feedback. For the price feedback to be represented in the unit of the sender's price, the data packet also carries an exchange rate field, `Ex`. This mechanism also protects the provider's network from malicious customer networks who intentionally assign a large budget in an attempt to get more share because their budget's actual value will be discounted.

5.5 Practical Issues

Section 5.4 addressed the first order design issues. We now address remaining issues in implementation and deployment.

Real number to floating point: The first thing that has to be addressed in implementing Equation 5.2 is that price cannot be arbitrarily small as it must be represented as a floating point. However, in our relative pricing, the price can be an infinitesimal value. To address this problem, we define a globally agreed upon minimum price. We treat this value as zero when summing up the price of each link, i.e., for any price P, $P = P + MININUM$. Now, an uncongested path has the minimum price. When a new flow joins, the sender preloads to its target budget. If the target budget is larger than the bottleneck capacity over *MINIMUM*, the flow can saturate the link. For a host with a unit budget to be able to saturate any link, this minimum price has to be sufficiently lower than unit budget over any link's capacity. We use the minimum price of $10^{-18}\$/byte$ or $\$1/Exabyte$ in our implementation.

Computational complexity: To calculate price, routers need to keep a moving window of input budget and update the average RTT. When packets arrive at the router, the router updates these statistics and assigns a timer for the packet so that its value can expire when it goes out of the window. It requires roughly 20 floating point operations for each packet.[2] We believe high-speed implementation is possible even with software implementations on modern hardware.[3]

Security and Enforcement: FCP easily lends itself to enforcement of fair-share with only per-user state at the edge router. The network has to enforce two things: 1) the sender is paying the right amount, and 2) the sender is operating within its budget. To enforce 1), we use Re-feedback's [45] enforcement mechanism. The sender set the `balance` field to the amount it is

[2] It requires 6 operations to update intermediate values when packets arrive and 2 operations when packets leave the window. An additional 7 operations are required to update the final price, average RTT, and calculate the feedback.

[3] As a rough estimate, Intel Core i7 series have advertised performance of $\sim$100 GFLOPS. At 100 Gbps, this gives a budget of $\sim$500 FLOPS/packet.

paying, and every router along the path subtracts its link's price for this packet, $p(t - rtt)$. If the amount reaches a negative value, this means that the sender did not pay enough. The egress edge router maintains an average of the balance value. If the value is consistently below zero, the egress router computes the average balance for each sender and identifies who is consistently paying less, and drops its packet. This information is then propagated upstream to drop packets near the source. Details on statistical methods for efficient detection are described in [45]. To enforce 2), the ingress edge router enforces a modified version of invariant that accounts for preloading:

$$\sum_{\text{packet in (t-1,t]}}^{\text{for each}} paid(packet) \cdot size \cdot preload \leq budget$$

This requires a classifier and per-user state at the ingress edge, and ensures that a host does not increase its budget arbitrarily. Ingress edge routers can drop packets once a host uses more budget than it is assigned.

Implementation details: In our experiments in Section 5.6, we set the average window size, d in Equation 5.2, to twice the average RTT. The queue parameter α is set to 2, but to prevent the remaining link capacity (denominator of Equation 5.2) from becoming negative,[4] we bound its minimum value to 1/2 the capacity. Under this setting, we verified the steady-state local stability of the algorithm using simulations in Section 5.6.1.

Finally, we also make our implementation robust to RTT measurement errors. Equation 5.4 uses $p(t - rtt(s))$, however, when price variation is large around time $t - rtt(s)$, small measurement error can have adverse effect on a router's estimate of the feedback price. Or even worse, routers may not store the full price history. We take the minimum of $p(\cdot)$ and *balance* field in the congestion header. This bounds the error between the paid price of the sender and the router's estimation of it, even when routers do not fully remember its past pricing.

5.6 Evaluation

We answer three questions in this section:

1. How does FCP perform compared to other schemes and what are the unique properties of FCP?
2. Does the flexibility allow evolution in the end-point?
3. Does the flexibility allow evolution in the network?

[4]This can happen when the queue capacity is relatively large compared to the current bandwidth delay product.

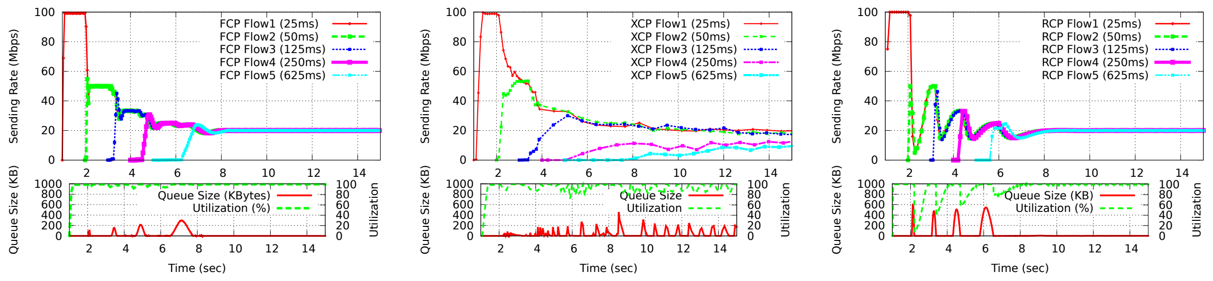


Figure 5.4: Convergence dynamics of a) FCP, b) XCP, and c) RCP: FCP achieves fast convergence and accurate rate control. When new flows arrive, XCP's convergence is slow, and RCP's rate overshoot is significant.

Protocol	AFCT	Loss rate	Avg. queuing delay	Avg. utilization
XCP	0.52 sec	0%	0.13 ms	64%
TCP	0.60 sec	1.6%	12 ms	68%
RCP	0.21 sec	0%	3.2 ms	66%
FCP	0.33 sec	0%	0.027 ms	65%

Figure 5.5: Short flows: Comparison of various statistics. AFCT stands for average flow completion time.

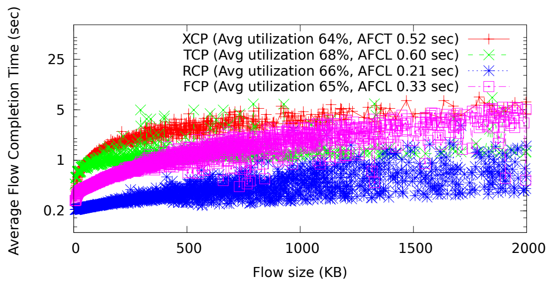


Figure 5.6: Short flows: Average flow completion time versus flow size. Mean flow size is 30 KB.

5.6.1 Performance

We implement FCP's algorithm described in Section 5.4 using ns-2. Using simulations, we study the performance of FCP. First, we compare the performance of FCP with other schemes (TCP, RCP, and XCP). We, then, look at unique characteristics of FCP, including fairness, preloading effectiveness, and its stability. Finally, we look at FCP's performance under a wide range of scenarios. The results show that FCP provides fast convergence while providing more accurate feedback during convergence and is as efficient as other explicit congestion control algorithms in many cases.

Performance comparison

First, we compare the performance of FCP with other schemes (TCP, RCP, and XCP) in various environments.

Long Running Flows: We first compare the convergence dynamics of long-running flows. We generate flows with different round-trip propagational delays ranging from 25 ms to 625 ms. Each flow starts one second after another traversing the same 100 Mbps bottleneck in a dumbbell topology. Each flow belongs to a different sender, and all senders were assigned a budget of 1 \$/sec. For RCP and XCP, we used the default parameter setting. Figure 5.4 shows the sending rate of each flow, queue size, and utilization for a) FCP, b) XCP, and c) RCP over time.[5] FCP's convergence is faster than RCP's and XCP's. XCP flows do not converge to the fair-share even at t=10 sec. In RCP, all flows get the same rate as soon as they start because the router gives the same rate to all flows. However, large fluctuation occurs during convergence because the total rate overshoots the capacity and packets accumulate in the queue when new flows start. Average bottleneck link utilization was 99% (FCP), 93% (XCP), and 97% (RCP).

Short flows: To see how FCP works with short flows, we generate a large number of short flows. We generate a Pareto distributed flows size with mean of 30 KB and shape of 1.2, flows arriving as a Poisson process with a mean arrival rate 438 flows/sec (offered load of 0.7). The bottleneck bandwidth and round trip delay is set to 150Mbps and 100 ms.

All FCP senders have the same budget of 1 \$/sec. However, because the flow is short, using the entire budget is unnecessary. Thus, we only preload to achieve at most $\frac{flowsize}{RTT}$ Bytes/sec. Table 5.5 and Figure 5.6 show the statistics and average flow completion time by flow size. For flow size less than 500KB, FCP performs better than TCP and XCP. However, RCP is faster because it jump-starts with the current rate after the SYN-ACK exchange, whereas FCP has to conservatively preload in the first round. (We assume every flow is going to a different destination.) For larger flows, FCP performance is in between that of XCP and TCP, but RCP still performs better. However, we saw that RCP makes an undesirable trade-off to achieve this. We further study RCP's limitation in Section 5.6.1.

The large variance for FCP and RCP is due to the fair-share rate changing as the load changes. On the other hand, XCP and TCP's rate is largely a function of how long the flow has been active because they slowly reach the fair-share rate. Finally, FCP shows the same zero loss rate as other explicit congestion control schemes, but has a much lower queueing delay. This is because FCP uses preloading to accounts for variations in the workload.

Fairness, stability, and preloading

To better understand FCP, we look at behaviors specific to FCP. In particular, we look at the fairness, stability, and the preloading behavior of FCP. We show that 1) the fair-share in FCP is determined by the budget, 2) FCP is locally stable under random perturbations, and 3) preloading allows fast convergence and accurate feedback.

[5] Sending rate is averaged over a 100 ms. For XCP ($t \geq 3$), we use a 500 ms averaging window.

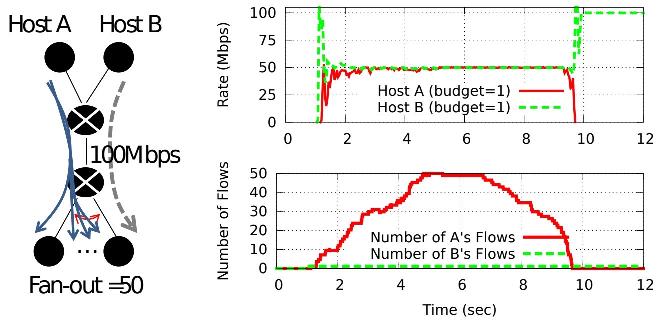


Figure 5.7: FCP's fair-share is determined by the budget.

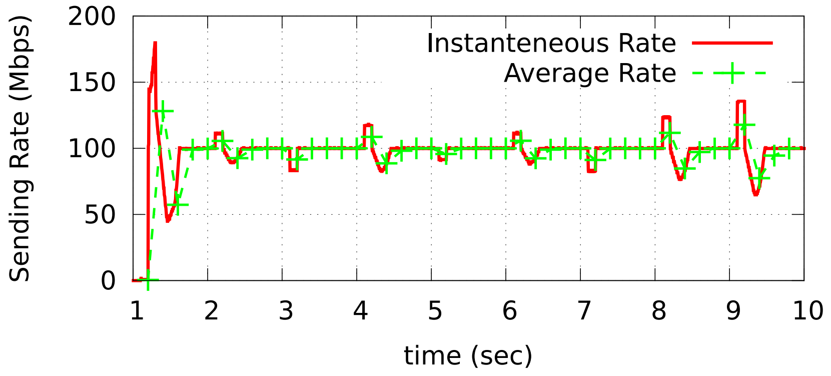


Figure 5.8: The algorithm is stable under random perturbation.

Fairness: Fairness in FCP is very different from traditional flow-level fairness. In FCP, two hosts that have the same budget achieve the same throughput if their traffic goes through the same bottleneck. To show this, we create two hosts A and B of equal budget. Host A generates 50 flows, but B sends only one, which go through a common bottleneck link. Host A's flow size is 1 MB each and they arrive randomly between [1,3]. Host B sends a long-running flow starting at t=1. Figure 5.7 shows the topology (left), the sending rate of each host, and the number of active flows (right). Host A's traffic, when present, gets 50% share regardless of the number of flows.

Local stability: A common method to prove the local stability is to show the stability of a linearized equation of the congestion controller near its equilibrium point [106]. However, it is shown that explicit congestion control algorithms whose equilibrium is at zero queue length is discontinuous at equilibrium and that the above traditional method produces incorrect results [30]. As a result, several studies have used much more complicated methods, such as

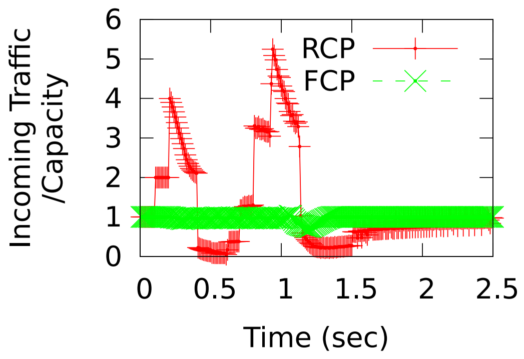


Figure 5.9: Preloading allows fast convergence and accurate rate estimation.

Lyapunov methods, and verified the result with simulations [30, 124].

In this work, we demonstrate the stability using packet-level simulations. In particular, in this experiment, we perform a local stability test by introducing perturbations. The system's stability depends on the stability of the price feedback equation. We, therefore, intentionally introduce errors in the price calculation and observe whether the system is able to restore itself on its own. If the system is stable, it will return to the equilibrium state shortly after the perturbation is introduced. We use a single link with a round trip delay of 100 ms with 100 Mbps of capacity. A long running flow starts at t=1.1 sec with a unit budget. At every second from t=2, we introduce random perturbation of [-30%, 30%] in the router's price calculation for a duration of 100 ms. For example, during the interval, t=[2,2.1] sec, the feedback price off from the correct price by a random fraction between 1 to 30%. Figure 5.8 shows the instantaneous and average rate of the flow. We observe after the perturbation is introduced the system either overshoots or undershoots the capacity. However, the system recovers the equilibrium shortly (i.e., the sending rate stabilizes at 100 Mbps).

How effective is preloading? Preloading is one of the distinguishing features of FCP. So far, we have seen the end-to-end performance of FCP with preloading. Here, we look at the benefit of preloading in isolation.

We demonstrate the benefit of FCP by comparing it with RCP that does not have preloading. We compare a FCP flow that continuously doubles its budget every 100 ms (one RTT) with RCP flows that double in flow count for a duration of 1 second. In both cases, the load (input budget for FCP and flow count for RCP) doubles every RTT.

Figure 5.9 shows the normalized incoming traffic at the bottleneck link while the load is ramping up from 1 to 1000 during the first second. We see that preloading allows fast convergence and accurate rate control; FCP's normalized rate is close to 1 at all times. On the other

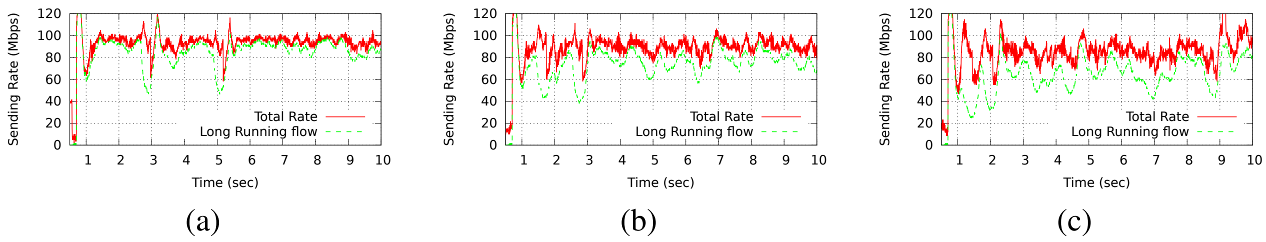


Figure 5.10: Performance with long-lived flows and short-lived flows.

Case	new flows/sec	utilization (%)
(a)	41.6	98.2
(b)	83.2	95.7
(c)	125	81.9

Figure 5.11: Performance statistics with long-lived flows and short-lived flows.

hand, RCP overestimates the sending rate by up to 5 times the capacity because RCP allocates bandwidth in a very aggressive manner to mimic processor sharing. With preloading, however, end-hosts can rapidly increase or decrease a flow's budget without causing undesirable behaviors in the network.

In-depth study of FCP's behavior

Finally, we look at detailed behaviors of FCP with mixed flow sizes, the convergence dynamics with multiple links, and the impact of misbehaving users.

Mixed flow sizes: We now study how FCP performs with long-lived flows and short-lived flows. All flows go through a common bottleneck of 100 Mbps. Long running flows start at t=0.5 sec and introduce a unit budget to the bottleneck link. They run for the duration of the simulation. Short flows arrive as a Poisson process and their size is Pareto distributed as earlier with a mean size of 30 KB. We vary the fraction of bandwidth that short flows occupy from 10% to 30%. Figure 5.10 shows the link utilization and the throughput of the long-lived flows. Figures 5.10 (a), (b), (c) respectively illustrate the case when the short flows account for 10%, 20%, and 30% of the bottleneck. Table 5.11 show the average number of new short flows per second and the average link utilization from t=0.5 sec to t=30 sec for each experiment. We observe that the link utilization becomes lower as the number of new flows increase. This is because when flows terminate, even though they preload a negative value, it takes some time for the price to reflect

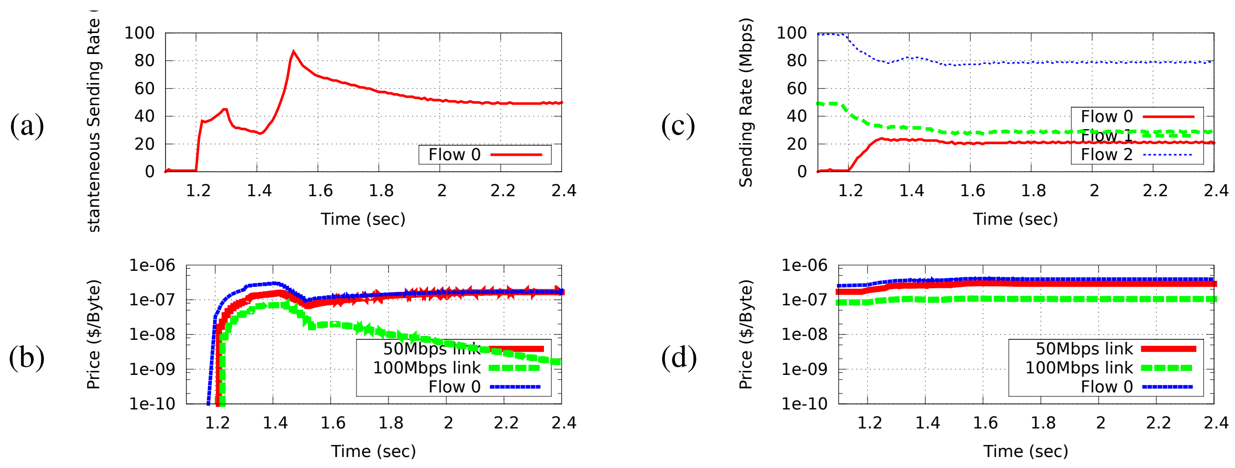
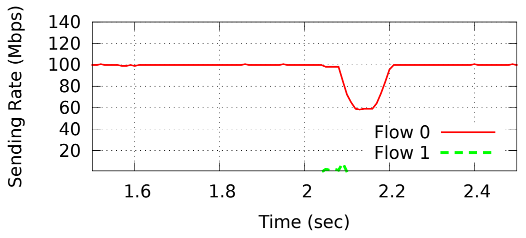


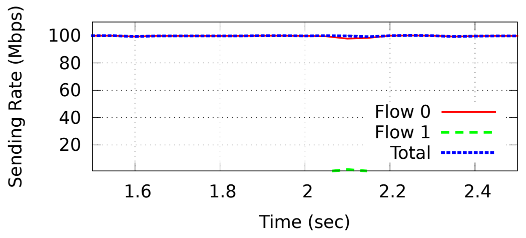
Figure 5.12: FCP's convergence dynamics with multiple links. Flow 0's round trip delay is 100 ms. Each column presents a different scenario: 1) (a) and (b) shows the dynamics of the sending rate and price of the two links when Flow 0 starts up first. 2) (c) and (d) shows the dynamics when Flow 0 starts up after Flow 1 and Flow 2.

due to the averaging window. During this period, it results in a slight underutilization. The utilization has a negative correlation with the amount of input budget variance per unit time. In other words, the utilization is high when a small fraction of input budget changes because of flow arrival and departure. For example, when the average flow size of the short flows is made larger (300 KB) with the fraction of short flows being 30%, the utilization slightly goes up to 90% from 82%. Also, when the long running flows' total budget is high, the utilization also goes up. When the long running flow's budget is set to $10\$/sec$, the utilization goes up from 82% to 99% This shows that when the fraction of stable flows' input budget is relatively high, the system is highly efficient.

Convergence dynamic with multiple links: FCP adopts proportional fairness. FCP's convergence dynamics with multiple links best illustrates how link prices are determined with proportional fairness. To demonstrate this, we use the topology of Figure 5.19 (b) with a larger link latency of 12.5 ms to better observe the dynamics. First, we start a single flow (Flow 0) at t=1.1 sec with a unit budget. Figure 5.12 (a) shows the instantaneous sending rate of the flow. Figure 5.12 (b) shows the how the price of the two links and the price of Flow 0 (the sum of the two prices) change over time. At the start of the flow, the price is at its minimum, $10^{-18}\$/byte$ (not visible in the figure). Because the sum of two minimum prices is the minimum price, the path price is also minimum. Both links' prices go up when Flow 0 preloads to use its entire budget. However, because the path price was at minimum, each link thinks that the flow's budget



(a) A misbehaving flow



(b) A well-behaving short-lived flow

Figure 5.13: A misbehaving flow negatively impacts the link utilization.

is entirely used toward its own link, which results in a temporary over-estimation of the path price. However, soon the price goes down because there is spare capacity. During this process, the system results in a temporary under-estimation of path price because the two links adjust its price independently. However, the path price and the sending rate stabilizes soon after. In particular, only the bottleneck link's price remain above the minimum price; the 100 Mbps link's price eventually falls down to the minimum price as its link is never fully utilized. On the other hand, the 50 Mbps bottleneck's price stabilizes above the minimum price level and dominates the path price. As a result, Flow 0's sending rate stabilizes at 50 Mbps. Our result shows that there is an interaction between links within a path in adjusting the link's price. However, it converges to the correct equilibrium point.

Above, we observed the cold startup behavior of a flow with multiple links. The behavior is slightly different when both links' prices do not start from the minimum price. Two show this we introduce a flow (Flow 0 of Figure 5.19) after Flow 1 and Flow 2 of Figure 5.19 have started. Old flows starts at t=0 sec and the new flow (Flow 0) starts at t=1.1 sec. All flows have a unit budget. Now, Flow 0 traverses two bottleneck links (i.e., the price of both links are not minimum). Figure 5.12 (c) and (d) respectively shows the sending rate of each flow and price of each link. We observe that the convergence is faster and overestimation/underestimation is much smaller. This is because the relative difference in the input budget after the introduction of a new flow is much smaller in this scenario than the cold startup. We also observe that the throughput of Flow 0 is lower than that of flow 1 even though they are sharing the 50 Mbps bottleneck link. This is because Flow 0's price is higher than Flow 1's price; Flow 1 only pays for the 50 Mbps link bottleneck where as Flow 0 pays for both bottlenecks.

In summary, we saw that the system converges to proportional fairness relatively quickly even though the link prices are calculated independently and the feedback only contains the total price. We also observed that only the bottleneck link's price stay above the minimum price level.

Misbehaving users: Misbehaving users may commit to use its own budget, but not live up to its promise. For example, a user may preload to use a budget of 1 $\$/sec$, but actually use nothing. This results in an underutilization because when a user preloads the network update its price reflecting the expected incoming budget. We illustrate this scenario here. We generate two flows, a normal flow and a misbehaving flow, on a common 100 Mbps bottleneck link. The round-trip delay is set to 40 ms. Figure 5.13 (a) shows the sending rate for both flows at every 10 ms period. For comparison, Figure 5.13 (b) shows the correct behavior when Flow 1 is a short-lived flow. In this case, the utilization is close to 100%. This is because a well-behaving flow does not over-preload. It also performs negative preloading at the end of the flow, as explained earlier.

Figure 5.13 (a) shows the misbehavior. A long-running flow (Flow 0) starts up at t=1 sec and starts to saturate the bottleneck link. At time t=2 sec, a user who has a unit budget starts up a misbehaving flow (Flow 1). It preloads its entire budget (1 $\$/sec$) but does not send any traffic after the preload. As a result, the path price goes up and Flow 0 nearly halves its sending rate. The period underutilization ends after the duration of the averaging window because the routers lower the price to achieve full utilization. However, in general, if the misbehaving users preload constantly without actually using the budget in the next round, the network is going to be underutilized constantly. We believe that this is not any worse than sending junk traffic. Even if a user preloads and does not live up to its promise, it can never harm other traffic more than what would have been its fair-share had it generated real traffic. However, the network can easily detect misbehaving users by comparing the expected incoming budget calculated in the previous averaging window to the actual incoming budget of the current averring window. If the former is consistently greater than the latter, the network classify the sender as an attacker and take appropriate measures.

Another type of misbehavior is to perform negative preload, but not decreasing the budget spent. This results in an overload. Similar to the underutilization problem, if a user constantly performs negative preloading, the network will be overloaded constantly. Here, the solution is to perform enforcement at the network similar to the method mentioned above. When the expected incoming budget for a host is consistently less than the actual incoming budget, the network should classify the host as an attacker and block its traffic.

5.6.2 End-point Evolution

In FCP, an end-point can freely distribute its budget to its own flows. This allows evolution in the end-point's resource allocation schemes. Here, we evaluate end-point based algorithms outlined in Section 5.4.2. For easy comparison, we use the topology of Figure 5.14. Sender 0 (S0) has a flow to receiver 0. S1 and S2 each has a flow to receiver 0 and receiver 1. S0 and S2 have a budget of 1 $\$/sec$, and S1 has twice as much.

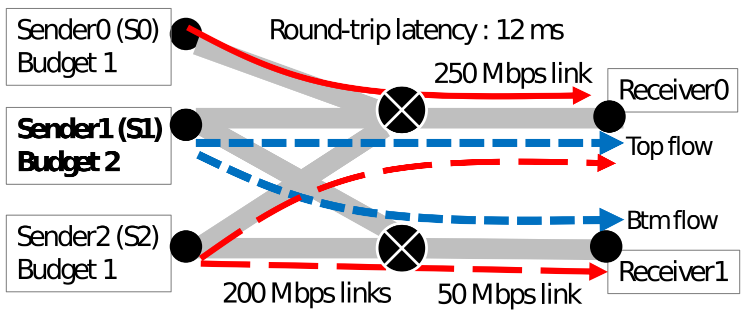


Figure 5.14: Topology

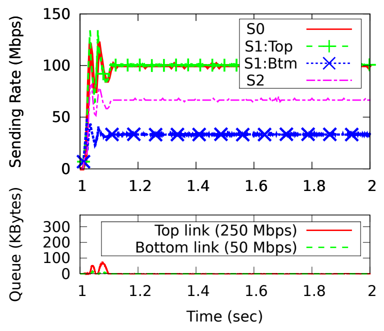


Figure 5.15: Equal-budget flows

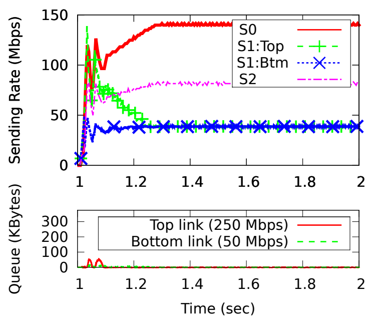


Figure 5.16: Equal-throughput flows

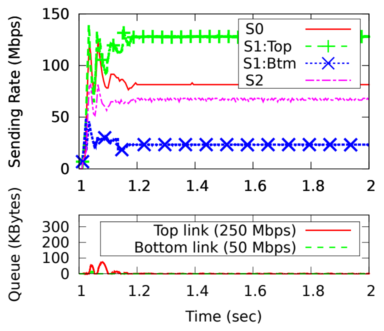


Figure 5.17: Max-throughput flows

Equal-budget (baseline) splits the budget equally among flows within a host. For example, S1 splits its budget in half; the top flow (to receiver 0), and the bottom flow (to receiver 1), each gets 1 $\$/sec$. Figure 5.15 shows the instantaneous sending rate of each sender with S1's rate broken down. It shows FCP achieves weighted bandwidth allocation. The top 250 Mbps bottleneck link's total input budget is 2.5 $\$/sec$. Because S0 and S1's top flow use 1 $\$/sec$, their throughput is 100 Mbps.

Equal throughput: Now, S1 changes its budget assignment to equal-throughput where it tries to achieve equal throughput on its flows, while others still use equal-budget assignment. For this, we start with the equal-budget assignment, and reassign every 2 average RTTs, and increase the budget of a flow whose rate is less than the average rate. Figure 5.16 shows the result. At steady-state, S1's two flows achieve equal throughput of 38.7 Mbps. A budget of 0.27 $\$/sec$ is assigned to the top flow, and 1.73 $\$/sec$ to the bottom.

Max-throughput: S1 now uses max-throughput assignment, where it tries to maximize the its total throughput. Others still use the base-line assignment. We implement this using gradient ascent. S1's flows start with equal budget, and at every 2 average RTT, it performs an experiment

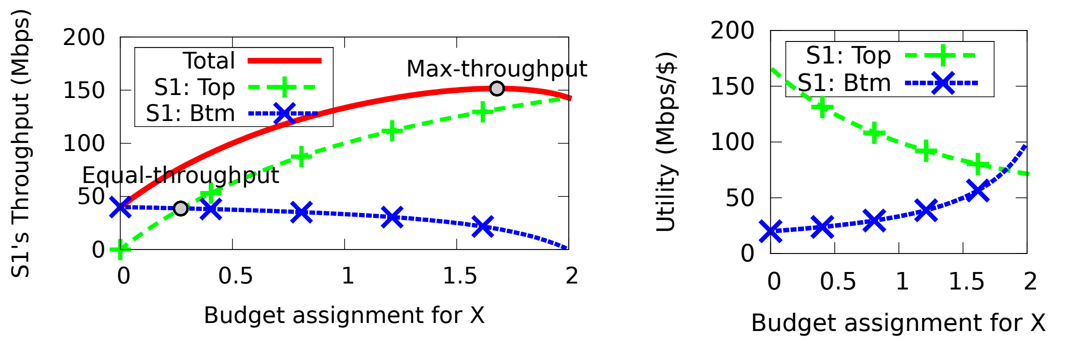
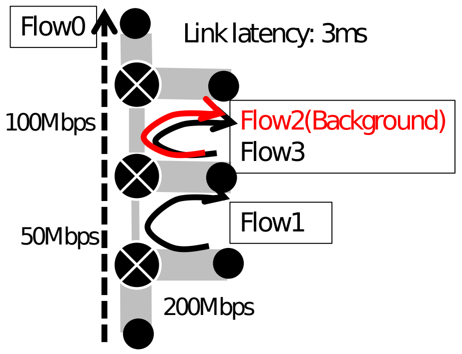


Figure 5.18: Theoretical (a) throughput and utility (b) versus budget assignment X for flow S1's top flow.

to change the budget assignment. It chooses a flow in round robin and increases its budget by 10% while decreasing others uniformly to maintain the total budget assignment. After 2 average RTTs, it compares the current throughput averaged over an RTT with the previous result, and move towards the gradient direction. The algorithm terminates when the throughput difference is less than 0.5%. The algorithm restarts when it observes a change in the price of a flow.

Figure 5.17 shows the result. S1's total throughput converges at 150.6 Mbps, and the assigned budget for the top flow (X) converges at 1.56 $\$/sec$. Figure 5.18 a) shows the theoretical throughput versus the budget assignment, X. The theoretical maximum throughput is 151.6 Mbps at $X = 1.68$ When more (less) budget is spent on X than this, the top (bottom) bottleneck link's price goes up (down), and the marginal utility becomes negative. Figure 5.18 b) shows such non-linear utility (rate per unit budget) curve for the two flows.

Background flows: FCP can also support background flows. A background flow by definition is a long-running flow that only occupies bandwidth if there's no other flow competing for bandwidth. This can be achieved with a flow having a very small assigned budget compared to other long running flow. For this, each host uses 1/10000 of its budget towards background flows. Using the topology in Figure 5.19, we ran a background flow with three other flows with a unit budget. Flow 0 starts at t=1 and immediately occupies the 50 Mbps bottleneck link (Figure 5.19). Flow 1 arrives at t=2 and shares the bottleneck with Flow 0. At t=5, the background flow (Flow 2) starts and occupies the remaining 75 Mbps of the 100 Mbps link. Note this did not influence Flow 0's rate. Flow 3 arrives at t=10 with unit budget and drives out the background flow. Note that now the 100 Mbps link also became a bottleneck, and Flow 0 is getting a smaller throughput than Flow 1. This is because Flow 0 is now paying the price of the two bottlenecks combined.



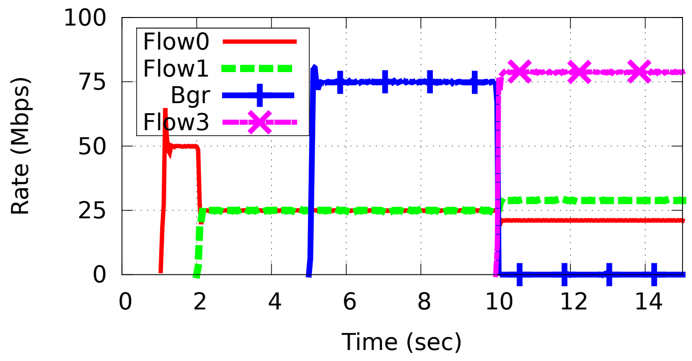


Figure 5.19: Background flows only take up the spare capacity. The sending rate is averaged over 40 ms.

5.6.3 Network Evolution

We now evaluate the network evolution scenarios in Section 5.4.2.

Bandwidth stability: We the implemented bandwidth stability feature described in Section 5.4.2. Figure 5.20 shows the sending rate of a stability flow (`Flow 2`) compared to a normal flow (`Flow 1`) under changing network conditions. Flow 2 starts at t=2 and slowly ramps up doubling its assigned budget at every RTT (100 ms). Cross traffic (Flow 3, dashed blue line) that has 50 times the budget of Flow 1 repeats a periodic on/off pattern starting from t=3. Flow 1's sending rate (solid black line) changes abruptly when the cross traffic join and leave, but the stability flows react slowly because its price does not change by more than twice in any direction during a window of 200 ms (twice the average RTT).

Multicast congestion control: FCP allows the network to evolve to support a different service model. For example, FCP can support multicast-aware congestion control as described in Section 5.4.2. We use the topology of Figure 5.24 (b) and generate a multicast flow from the top node to the bottom four receiver nodes. The link capacity varies by link from 10 to 100 Mbps. We also introduce unicast cross traffic to vary the load, and show that the multicast flow dynamically adapts its rate.

Figure 5.21 shows the result. A multicast flow starts at t=1 and saturates the bottleneck link capacity of 10 Mbps. Three unicast flows then start sequentially at t=2, 2.5, 3. When `Unicast 1` arrives, it shares equally shares the bottleneck bandwidth. As other unicast flows arrive, other links also become a bottleneck and their price goes up. As a result, the multicast flow's price (the sum of all link price) goes up and its sending rate goes down. At steady state, the multicast flow's

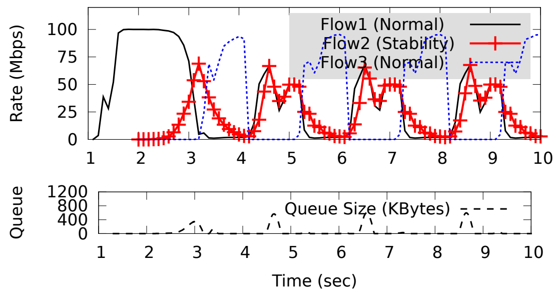


Figure 5.20: Support for bandwidth stability

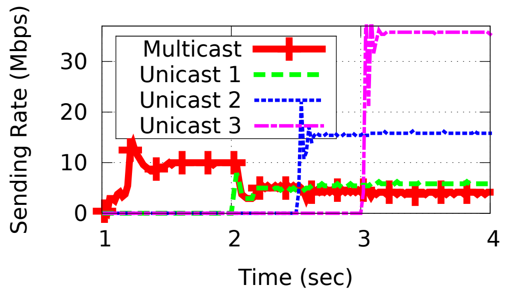


Figure 5.21: Multicast flow (red line) adapts to congestion.

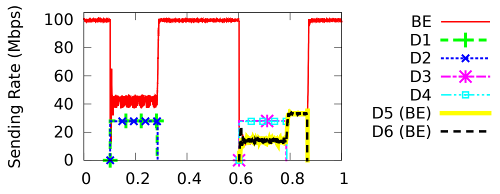


Figure 5.22: Deadline flows

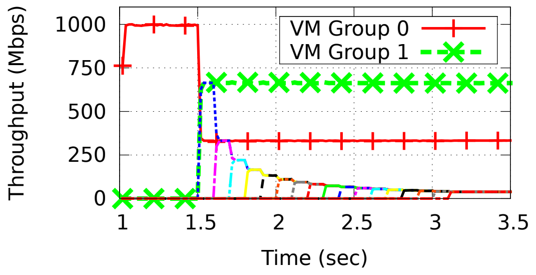


Figure 5.23: Aggregate control

throughput is around 4 Mbps. The unicast flows take up the remaining capacity (e.g., unicast 3's rate is 36 Mbps).

Deadline support: As we described in Section 5.4.2, FCP can offer D^3-style deadline support using two virtual queues (best effort and deadline) and differential pricing. Deadline flows are guaranteed a fixed price when admitted. We use the minimum price (Section 5.5) as the fixed price. A deadline flow at the beginning of the flow preloads the amount required to meet the desired rate R at once. If the routers on the path can accommodate the new deadline flow, they return the fixed price. The deadline flow is then able to send at desired rate and meet the deadline. Otherwise, routers give the best effort pricing and treat the flow as a best effort flow. We run an experiment to show both cases.

Figure 5.22 shows the instantaneous rates of deadline and best effort flows going through a 100 Mbps bottleneck link with a round-trip delay of 2 ms. The queue is set to admit up to 80 Mbps of deadline flows and assigns at least 20 Mbps to best effort flows in a work conserving manner. A best effort (BE) flow starts at the beginning and saturates the link. At t=0.1 sec, two deadline flows (D1 and D2) requiring 30 Mbps of throughput arrive to meet a flow completion time of ∼200 ms. Because the link can accommodate the deadline flows, they both get admitted

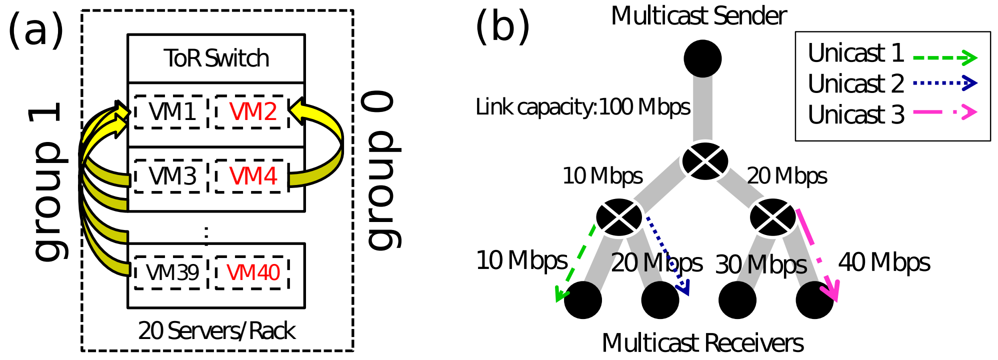


Figure 5.24: Topologies and flow patterns of experiments

and complete within the deadline. At t=0.6 sec, four deadline flows (D3 to D6) arrive with the same requirement. However, the network can only accommodate two deadline flows (D4 and D5). The other two (D5 and D6) receive the best effort pricing and become best effort flows. We then additionally preload to make it achieve its fair-share in the best effort queue. As a result, all the best effort flows achieve the same throughput.

Budget management and aggregate congestion control: We implement aggregate congestion control using the algorithm described in Section 5.4.3. To show how aggregate congestion control may work in a data-center to allocate resources between tenants, we use a flow pattern of Figure 5.24 (a), similar to the example shown in [167]. Within a rack, there are 20 servers and each have two virtual machines (VMs). Each server is connected to the ToR switch at 1 Gbps. Odd numbered VMs (VM group 1) belong to one tenant and even ones (VM group 0) to the other. Each tenant allocates a budget to each of its own VM independently. We assume all VMs have an equal budget. The data-center provider allocated a weight of 1 to group 0, and weight of 2 to group 1. In group 0, there is only one flow. In group 1, there are multiple flows starting one after another, all originating from a different VM (see Figure 5.24 (a)). Figure 5.23 shows the throughput of each group as well as the individual flows. At t=1, a flow starts in the VM group 0. From t=1.5 sec, VM group 1's flows start to arrive, and get twice as much share as group 0. Dotted lines show that the throughput of group 1's individual flows who have equal budget achieve the same throughput. The result highlights two benefits of aggregate congestion control: 1) regardless of the number of flows in the group, the aggregate flows get bandwidth proportional to their weights (group 1 gets twice as much as group 0), and 2) the weight of flows within each group is preserved (individual flows in group 1 receive the same throughput).

5.7 Chapter Summary

This chapter explored an important problem of designing a common behavior that must be applied to all packets. We showed that when introducing such features a careful design is required in order not to hinder evolution. In this context, we explored an open problem of designing a congestion control algorithm that supports evolution. While there have been a number of initial studies on evolvable Internet architectures [78, 90], little research has been done in designing an evolvable transport layer. We argued that existing approaches such as virtualization and TCP-friendly designs have limitations, and these limitations call for a universal framework for congestion control in which various strategies and algorithms can coexist. Moreover, modern congestion control algorithms utilize in-network components to generate feedback. While they may generate more accurate feedback, they do not allow any flexibility at the end-points, which significantly hinders evolution.

To address this problem, we proposed a design based on a pricing abstraction and a simple invariant. We presented an explicit feedback-based algorithm that generates accurate feedback and allows flexibility at the end-points and in the network. We also addressed important practical issues to make the algorithm more practical.

Finally, we conclude by summarizing two important lessons learned:

1. Some of the most fundamental features, such as congestion control, require designing a common behavior that must be agreed upon by many participants of the network and must be applied to all packets regardless of their communication types. In designing such features, supporting diversity and flexibility is important not to hinder evolution.

2. When introducing a new in-network component, introducing the right abstraction provides many benefits. In the case of congestion control, introducing the right abstraction that allows maximum flexibility is critical in supporting evolution.

Chapter 6

Conclusion and Future Work

This dissertation addressed core challenges in designing a future Internet architecture. We started from the observation that networking is fundamental changing from the traditional model of a dumb data pipe to a system that combines computation and storage with the delivery of data. The Internet today has been transformed dramatically from its initial model and has grown far beyond its original vision. Therefore, we argued that future Internet designs must facilitate this movement and, at the same time, accommodate evolution that we might experience in the future. This chapter concludes the dissertation with a summary of the approach and contributions, followed by a discussion on remaining open problems.

6.1 Contributions

In the context described above, this dissertation presented the following thesis: *designing an evolvable network architecture requires an integrated system design that examines multiple aspects of the architecture to incorporate diverse interactions between the end-points and the network.*

We explored three different aspects in designing an architecture that supports evolution:

1. Design of an evolvable Internet architecture: We demonstrated that we can design a network that allows new service models to be introduced.

2. End-point evolution or adaptation: We demonstrated that when networks provide new functionality, end-points may have to adapt their behavior to gain performance benefits.

3. Designing a common behavior that supports evolution: We showed that we can design a common behavior without hindering evolution.

Below, we highlight the contributions made in each of the three design aspects.

6.1.1 Evolvable Network Design

Key Insight and Motivation: Today's Internet Protocol does not allow the per-hop processing to change over time; it is designed to only support the abstraction of the host-based communication. This is a significant shortcoming in this new environment where integrated resources enable more sophisticated packet processing. Moreover, the introduction of new functionality, such as in-network caching, creates several different styles of communication, which are not best served by today's host-based communication model. For example, although many applications require access to a service or content that might be widely replicated, today's IP networks only allow applications to specify a particular host that the service or content can be accessed from. Our solution to this problem is to design a network that allows the introduction of new abstractions along with new functionality and supports the evolution of the service model.

Core Contributions: To achieve evolution, XIA supports diversity and incremental deployment of new network-level functionality and abstractions (i.e., new service models). Principal types allow us to define a new abstraction and per-hop packet processing in the network. The major challenge is to allow the end-host to use the new service model even before the network is fully upgraded to support it. This is crucial for evolution since the deployment of new network functionality does not happen over night.XIA enables incremental deployment of new service models by providing alternative ways for routers to satisfy the new functionality, called fallbacks, that are incorporated in our flexible addressing scheme. Our flexible addressing scheme, thus, enables incremental deployment of new abstractions and functionality. Finally, through a prototype implementation and evaluations, we showed that XIA supports evolution without sacrificing efficiency. In other words, XIA's basic packet forwarding can scale up to multiple-10Gbps similar to the performance of IP.

6.1.2 End-point Evolution

Key Insight and Motivation: The resource integration also has significant implications on end-points. As networks provide new functionality with integrated resources, complicated interactions can occur between the network and the end-points. In the old model, where the network was a dumb data pipe, all other interesting features had to be handled purely at the end-points. However, when network provides new functionality, this is no longer true. In fact, some of the functions traditionally implemented at the end-points can now be handled much more efficiently in cooperation with the network. This suggests that when networks evolve to provide new service models, end-points also have to evolve to better take advantage of the new network. To highlight this implication, we presented a compelling case study in Chapter 4 that shows we can gain unexpected performance benefits by adapting the behavior of end-points to the underlying network.

Core Contributions: We showed that XIA facilitates end-points' interaction with new network functionality. XIA enables the end-points to explicitly to invoke new network functionality and to identify the functionality supported by the path. This , in turn, allows end-points to dynamically adapt to the functionality provided by the underlying network.

Using redundancy elimination (RE) as an example, we demonstrated that end-points can benefit from adapting their strategies to the functionality provided by the network. In particular, we presented Redundant Transmission (RT)—a novel loss protection scheme that intelligently sends multiple copies of the same data when the underlying network is RE-enabled. In real-time data delivery, loss recovery is traditionally handled by introducing cleverly coded redundancy as in Forward Error Correction (FEC). However, when the underlying network is content-aware, we demonstrate that our new approach works better. The basic idea of RT is to expose the redundancy directly to the underlying content-aware network. The simplest form of RT is to send multiple copies of the same packet. When packets are not lost, the duplicate transmissions in RT are compressed by the underlying network and add little overhead. In contrast, when the network is congested, the loss of a packet prevents the compression of a subsequent transmission. This ensures that the receiver still gets at least one decompressed copy of the original data. Using real-time interactive video streaming as an example, we demonstrated the benefits of RT. Our evaluation results show that RT provides much better loss protection, delivers better video quality, and is much easier to use than traditional FEC-based schemes.

6.1.3 Designing a Common Behavior

Chapter 5 presented the challenges in designing a common behavior in the network. These common behaviors support features that are critical to the operation of the network and that must be applied to all packets. The central challenge in designing a common behavior is that once it is introduced, it is very hard to change. Therefore, it can easily hinder evolution in the network and significantly undermine the evolvability of the network. However, this problem has been often neglected in previous designs for evolvable network architectures. In Chapter 5, we show that we can indeed design a common behavior to support evolution by taking congestion control as an example.

Congestion control algorithms must support diverse application requirements, such as background transfer, bandwidth stability, and multipath transfer, and also need to evolve over time to meet new requirements. For example, TCP has evolved to support diverse application requirements. TCP was much easier to evolve than the network layer because it is a purely end-point based algorithm. However, modern congestion control algorithms, such as RCP [68] and XCP [106], utilize in-network components that rely on in-network components that are hard to evolve. While they provide higher efficiency than TCP due to the use of explicit feedback, it is not clear how they can evolve to support diverse application requirements. Therefore, the core problem is that existing approaches provide either high efficiency or evolvability, but not both.

Core Contributions: In Chapter 5, we showed that we can to achieve both requirements, efficiency and flexibility, in a congestion control algorithm by introducing the abstraction of pricing that has been explored in theoretical context [107]. To this end, we designed an explicit congestion control algorithm, called FCP, that uses price as a form of explicit feedback and demonstrated that FCP easily allows different features to coexist within the same network. In FCP, end-hosts can implement diverse styles of rate control and networks can leverage differential pricing and aggregate control to achieve many different goals.

Our primary contribution is showing that explicit congestion control algorithms can be made flexible enough to allow evolution just as end-point based algorithms. Our secondary contribution is to make economics-based congestion control [107, 108, 109] practical by extending past theoretical work in three critical ways:

1. FCP uses "pricing" as a *pure abstraction* and the key form of *explicit feedback*.
2. Unlike previous explicit rate feedback designs [68, 107], FCP accommodates high variability and rapid shifts in workload, which is critical for flexibility and performance. This property of FCP comes from a novel *preloading* feature that allows senders to commit the amount of resource it wants to spend ahead of time.
3. Finally, we address practical system design and resource management issues, such as dealing with relative pricing, and show that FCP is efficient.

6.2 Implications and Outlook

The trends towards the integration of storage, computation, and networking is a fundamental trend. Trends outlined in Chapter 2 suggest that networks in the future will provide more diverse and richer functionality in order to better support networked applications. Without addressing the problem of the fixed abstraction of the current Internet, it will be increasingly difficult to integrate new functionality and reason about the network's operation. This dissertation provides important lessons and implications in this context.

First, the network architecture presented in Chapter 3 provides a new way of introducing new functionality. We believe that this architecture will foster innovation. Through the process of self-selection, a set of new service models that pass through the test of time and market will survive. However, we do not believe middleboxes will go away because in some cases transparent introduction of functionality might still be valuable. Nevertheless, as networks become more complex, introducing new functionality with abstractions will provide significant benefits.

Second, when networks provide new functionality, interesting interactions may occur between the network and the end-points. The study presented in Chapter 4 is just a small example of this. However, more research is required to better understand how to utilize new abstractions and functionality provided by the network.

Finally, in Chapter 5, we saw that certain functions need to be carefully introduced into the network in order not to hinder evolution. In the future, the networking environment and the functionality provided by the network will be even more diverse and heterogeneous. We believe that introducing a common feature will be increasingly difficult and a careful design of the abstraction that allows evolution of its behavior will be even more critical.

XIA and Software-defined Networking: Software-defined Networking (SDN) is an important related trend in this context. Broadly speaking, XIA is an architecture that enables software-defined networking especially in the data plane. We draw a parallel between XIA and OpenFlow [85], another enabler of SDN, and highlight the common underlying theme. OpenFlow makes an observation that today's networking hardware provides flexible functionality in flow classification and takes advantage of this fact. It exposes a universal abstraction of the underlying hardware's flexible functionality and combines it with generic computation. This was one of the first attempts to introduce a common abstraction when networks provide flexible functions. Despite their explicit focus on the network's control plane, OpenFlow bears some similarity with XIA. Both of them aim to provide extensibility and provide the architectural support for long-term evolution. The motivation for both architectures stems from the observation that the underlying hardware is much more powerful and flexible and that the architecture lacks support for introducing new functionality. OpenFlow targets introducing new functionality in the control plane, while XIA targets the data plane.

The observation suggests that there might be a common theme for future research in networking that shares the motivation and problems. While many challenges and opportunities for future research remain for both, we believe that identifying new abstractions for networking and exploring new use of them will be the key agenda for future Internet research.

6.3 Future Work

There are several directions for future research that this dissertation leaves open.

Evaluation methodology for new Internet architectures: While there have been many proposals for new Internet architectures [28, 31, 43, 61, 78, 84, 102, 115, 172, 176], we know of no systematic study on the methodology of evaluating a new architecture. While many have used standard metrics, such as scalability and performance, some aspects of architectures are very hard to quantify. Evolvability, among others, is not easily quantifiable. Moreover, Chapter 1 points out what people mean by evolvability actually differs across different studies. In many cases, the scope of evolution was also different. Establishing new metrics for evaluating architectures and methodology for comparing two different architectures are important future work. We note two lessons that we learned from this dissertation: 1) For features that are not quantifiable, stating what they mean and how it differs from their use in other studies are important. 2) It is difficult to prove or disprove the degree or limit for these open features such as evolvability. However, this may provide valuable information.

Innovative use of XIA: This dissertation studied a few examples of how XIA can be used to support innovative features. However, XIA does not make innovation itself easier. What new principal types we need and how we can utilize them in support new or existing applications are important open problems. For example, building content distribution networks that uses the new abstraction of content-oriented communication and integrating intelligence into content distribution remain to be studied. Coming up with a new class of applications that XIA can support will be also important in evaluating the strength of the architecture.

Innovation in security is another open area of research. It is well known that the Internet today has many security shortcomings. While XIA provides primitives for bootstrapping security in the network, the problem of utilizing and incorporating them into networking protocols to realize a secure Internet is not addressed by this dissertation. While we believe, XIA's approach will result in a more secure network, new possibilities in this domain should be explored to demonstrate what XIA enables.

Deployment and Domain specific applications of XIA: A relatively large-scale deployment is necessary to thoroughly evaluate a network architecture. While this dissertation looked at different aspects of an integrated design, our evaluations were limited to small-scale deployments of our prototype. A practical approach is to start with a specialized domain, such as Supervisory Control and Data Acquisition (SCADA) networks or datacenter networks. Focusing on these environments will allow us to explore XIA's strength in different environments and allow us to look at different aspects of XIA. For example, high availability and security are often the key requirements for SCADA networks, such as traffic signal control networks and smart grid. For datacenters, providing and efficiently supporting new in-network services for cloud computing will be an interesting area to study. The XIA's ability to combine computation and storage with data delivery may potentially benefit many applications.

Integration with OpenFlow: As discussed previously, software-defined networking shares some common design principles with XIA. The difference, however, is that XIA focuses more on the evolution in the data plane, but OpenFlow exclusively addresses issues in programmable control plane. An interesting avenue of research is to show whether these two can be integrated to create synergy. We believe that OpenFlow can be leveraged to better manage the diverse functionality provided by the network and to dynamically scale the resources used in the network for different types of in-network processing.

One possibility is to leverage the idea of OpenFlow to dynamically route a packet based on its principal type in order to combine different features. For example, content principal types may take a different route that supports in-network caching and dynamically adjust the resources, such as the memory or storage used by the principal type.

Scalable routing and name resolution: Scalable routing is an important part of future Internet research. As shown in Section 3.6.3, the number of routable entities can be very large for content and service. Another significant difference in this environment is that churn is now a norm due to replication, caching, and mobility. In traditional environments, routing updates are mainly caused by rare events such as failures and changes in topology. Therefore, the design of a routing system must change to effectively deal with frequent updates. At the same time, the data plane has to deal with transient failures due to churn. We believe that this problem cannot be solved by a single protocol, but multiple mechanisms and protocols must be coherently designed.

In XIA and MobilityFirst [28], part of the routing (i.e., default fallback route) is done in a separate infrastructure through name resolution. For example, in XIA, when obtaining a mapping from human readable names to network addresses, the name resolution system gives a set of identifiers that applications can use to form a DAG. While this alleviates the problem of mapping flat addresses to topological identifiers, it is not clear whether a distributed name resolution infrastructure can handle frequent updates. An important future work is to design a hybrid routing system that combines name resolution, routing protocols, and data plane driven failure recovery.

Evolution of congestion control: While Chapter 5 presented a congestion control framework that supports evolution, it did not address how we might transition from TCP to FCP. One approach is to provide a complete isolation of shared resources through virtualization and run different congestion control algorithms on different slides. However, virtualization is a heavy-weight mechanism, and many practical problems, such as discovering the path property, still remain. An alternative approach is to explore a light-weight form of virtualization or a signaling mechanism that creates an isolated path in a more dynamic fashion. We note that the granularity of this type of resource reservation and exact mechanisms are interesting avenues for research.

Evolvable transport protocols: Supporting multiple principal types also introduces challenges in designing transport protocols for different communication types. While TCP became the de facto standard in today's Internet for many practical reasons, it has many limitations. Recent proposals for transport protocols also do not address all concerns specific to different principal types. Although, Chapter 5 addressed the problem of designing a congestion control framework that supports evolution, designing transport protocols specialized for different principal types is an open problem that remains. Different principal types may have different requirements; content transfer may utilize multiple sources whereas service principal types often require flow affinity. Short Web content transfer also requires flows to complete fast and can benefit from two-way handshake. Identifying the such requirements for different principal types and designing new transport protocols that support the new requirements are important next steps.

Generic resource allocation and control: While this dissertation explored resource allocation for shared bandwidth, other forms of shared computing resources exist within the network. These resources are locally controlled today (e.g., certain amount of storage is assigned for a certain collection of video). However, there might be a need to dynamically control storage and computational resources in a network-wide fashion or reserve them along an end-to-end path. An interesting research question is to find out whether there is such need, how it might be useful, and how we might perform generic resource allocation if it is necessary.

Bibliography

[1] KVM: Kernel based virtual machine. http://www.linux-kvm.org/page/Main_Page. 3.6.2

[2] Juniper Networks Datasheet. http://www.juniper.net/us/en/local/pdf/datasheets/1000113-en.pdf, 2009. 4.2

[3] Cisco visual networking index: Forecast and methodology, 20092014. http://www.cisco.com/, 2010. 2.2.1, 4.4, 4.7.3

[4] Magic Quadrant for WAN Optimization Controllers. http://www.gartner.com/technology/media-products/reprints/riverbed/article1/article1.html, 2010. 4.2

[5] YUV CIF reference videos. http://www.tkn.tu-berlin.de/research/evalvid/cif.html, 2010. 4.7.1

[6] Cisco Internal WAAS Implementation. http://blogs.cisco.com/ciscoit/cisco_internal_waas_implementation/, 2010. 4.2, 4.6

[7] AT&T Businees Service Guide - AT&T VPN Service. http://new.serviceguide.att.com/portals/sgportal.portal?_nfpb=true&_pageLabel=avpn_page, 2011. 4.6

[8] Riverbed Cloud Products. http://www.riverbed.com/us/products/cloud_products/cloud_steelhead.php, 2011. 2.1, 4.2

[9] Riverbed Customer Stories. http://www.riverbed.com/us/customers/index.php?filter=bandwidth, 2011. 4.2, 4.6

[10] Riverbed Steelhead Mobile. http://www.riverbed.com/us/products/steelhead_appliance/steelhead_mobile/, 2011. 2.1, 4.2

[11] Cisco visual networking index: Forecast and methodology, 20112016. http://www.cisco.com/, 2012. (document), 2.2.1, 2.1

[12] Sprint Network Performance. https://www.sprint.net/sla_performance.php?network=pip, 2012. 4.6, 3

[13] Bhavish Aggarwal, Aditya Akella, Ashok Anand, Athula Balachandran, Pushkar Chitnis, Chitra Muthukrishnan, Ramachandran Ramjee, and George Varghese. EndRE: an end-system redundancy elimination service for enterprises. In *Proc. NSDI*, pages 28–28, Berkeley, CA, USA, 2010. USENIX Association. 2.2.1, 2.2.2

[14] Agilent Technologies. The journal of internet test methodologies, 2012. `http://www.ixiacom.com/pdfs/test_plans/agilent_journal_of_internet_test_methodologies.pdf`. 3.6.1

[15] Patrick Agyapong and Marvin Sirbu. Economic incentives in content-centric networking: Implications for protocol design and public policy. In *Proc. Research Conference on Communications, Information and Internet Policy*, Arlington, VA, September 2011. 3.4.3

[16] Saamer Akhshabi and Constantine Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In *Proc. ACM SIGCOMM*, Toronto, Canada, August 2011. 3.7, 3.7

[17] Andres Albanese, Johannes Blöer, Jeff Edmonds, Michael Luby, and Madhu Sudan. Priority encoding transmission. *IEEE Transactions on Information Theory*, 42, 1994. 4.4.4

[18] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, 2010. 4.2

[19] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *Proc. ACM SIGCOMM*, 2008. 2.2.1, 2.2.2, 2.2.2, 4.1.1, 4.4, 4.4.2, 4, 4.7.1, 4.8

[20] Ashok Anand, Vyas Sekar, and Aditya Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *Proc. ACM SIGCOMM*, pages 87–98, Barcelona, Spain, August 2009. 4.6

[21] Ashok Anand, Fahad Dogar, Dongsu Han, Boyan Li, Hyeontaek Lim, Michel Machado, Wenfei Wu, Aditya Akella, David Andersen, John Byers, Srinivasan Seshan, and Peter Steenkiste. XIA: An architecture for an evolvable and trustworthy Internet. In *Proc. ACM Hotnets-X*, Cambridge, MA, November 2011. 3, 3.3

[22] Ashok Anand, Fahad Dogar, Dongsu Han, Boyan Li, Hyeontaek Lim, Michel Machado, Wenfei Wu, Aditya Akella, David Andersen, John Byers, Srinivasan Seshan, and Peter Steenkiste. XIA: An architecture for an evolvable and trustworthy Internet. Technical Report CMU-CS-11-100, Carnegie Mellon University, February 2011. 3.4.1

[23] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable Internet Protocol (AIP). In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008. 3.1, 3.3.1, 3.3.2, 3.4.1, 3.7, 5

[24] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38, April 2005. 3.7, 5.2

[25] Siddhartha Annapureddy, Michael J. Freedman, and David Mazières. Shark: Scaling file servers via cooperative caching. In *Proc. 2nd USENIX NSDI*, Boston, MA, May 2005. 2.2.2

[26] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proc. ACM SIGCOMM*, 2004. 4.7.4

[27] S. Athuraliya, S.H. Low, V.H. Li, and Qinghe Yin. Rem: active queue management. *Network, IEEE*, 15(3):48 –53, May 2001. 5.1, 5.2

[28] Akash Baid, Tam Vu, and Dipankar Raychaudhuri. Comparing alternative approaches for networking of named objects in the future internet. In *Proceedings of 1st Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN)*, 2012. 2.2.1, 6.3, 6.3

[29] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM*, pages 175–187, Cambridge, MA, September 1999. 5.2, 1

[30] H. Balakrishnan, N. Dukkipati, N. McKeown, and C.J. Tomlin. Stability analysis of explicit congestion control protocols. *Communications Letters, IEEE*, 11(10):823 –825, October 2007. 5.6.1

[31] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the Internet. In *Proc. ACM SIGCOMM*, pages 343–352, Portland, OR, August 2004. 3.3.2, 3.3.2, 3.7, 6.3

[32] Mahesh Balakrishnan, Tudor Marian, Ken Birman, Hakim Weatherspoon, and Einar Vollset. Maelstrom: transparent error correction for lambda networks. In *Proc. USENIX NSDI*, 2008. 4.2

[33] Deepak Bansal and Hari Balakrishnan. Binomial Congestion Control Algorithms. In *IEEE Infocom*, Anchorage, AK, April 2001. 2

[34] Andreas Bechtolsheim. Moore's law and networking. In *Keynote address, NANOG*, June 2012. (document), 1.1, 1.1.1, 1

[35] Micah Beck, Terry Moore, and James S. Plank. An end-to-end approach to globally scalable network storage. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, August 2002. 2.2.2

[36] Ali C. Begen and Yucel Altunbasak. Redundancy-controllable adaptive retransmission timeout estimation for packet video. In *Proc. ACM NOSSDAV*, 2006. 4.3

[37] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), December 1998. 5.1

[38] Ethan Blanton and Mark Allman. On making TCP more robust to packet reordering. *ACM SIGCOMM CCR*, 32:20–30, January 2002. 3.3.3

[39] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Don Towsley. Adaptive FEC-based error control for internet telephony. In *Proc. IEEE INFOCOM*, 1999. 4.3

[40] Christoph Borchert, Daniel Lohmann, and Olaf Spinczyk. CiAO/IP: a highly configurable aspect-oriented IP stack. In *Proc. ACM MobiSys*, June 2012. 5.2

[41] Sem Borst, Varun Gupta, and Anwar Walid. Distributed caching algorithms for content distribution networks(to appear). In *Proc. IEEE INFOCOM*, San Deigo, CA, March 2010. 2.2.1

[42] O. Boyaci, A.G. Forte, and H. Schulzrinne. Performance of video-chat applications under congestion. In *Proc. IEEE ISM*, December 2009. 4.2, 4.3, 4.4.2

[43] Robert Braden, Ted Faber, and Mark Handley. From protocol stack to protocol heap: role-based architecture. *ACM SIGCOMM CCR*, 33, January 2003. 3.7, 6.3

[44] Patrick G. Bridges, Gary T. Wong, Matti Hiltunen, Richard D. Schlichting, and Matthew J. Barrick. A configurable and extensible transport protocol. *IEEE ToN*, 15(6), December 2007. 5.2

[45] Bob Briscoe, Arnaud Jacquet, Carla Di Cairano-Gilfedder, Alessandro Salvatori, Andrea Soppera, and Martin Koyabe. Policing congestion response in an internetwork using re-feedback. In *Proc. ACM SIGCOMM*, 2005. 5.1, 5.2, 5.5

[46] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM*, 1998. 4.3

[47] Jin Cao, W.S. Cleveland, Yuan Gao, K. Jeffay, F.D. Smith, and M. Weigle. Stochastic models for generating synthetic HTTP source traffic. In *Proc. IEEE INFOCOM*, volume 3, 2004. 4.7.3

[48] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), February 2002. URL `http://www.ietf.org/rfc/rfc3234.txt`. 1.1.2

[49] Joachim Charzinski. Traffic properties, client side cachability and cdn usage of popular web sites. In Bruno Mller-Clostermann, Klaus Echtle, and Erwin Rathgeb, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, volume 5987 of *Lecture Notes in Computer Science*, pages 136–150. Springer Berlin / Heidelberg, 2010. 3.6.3, 5

[50] G. Chatzopoulou, Cheng Sheng, and M. Faloutsos. A first step towards understanding popularity in youtube. In *Proc. INFOCOM IEEE Conference on Computer Communications Workshops*, March 2010. 3.6.3

[51] Philip A. Chou, Helen J. Wang, and Venkata N. Padmanabhan. Layered multiple description coding. In *Proc. Packet Video Workshop*, 2003. 4.4.4

[52] Cisco. Deploying guaranteed-bandwith services with mpls. `http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/gurtb_wp.pdf`, 2012. 4.6

[53] Cisco systems. Data center: Load balancing data center. `https://learningnetwork.cisco.com/docs/DOC-3438`, November 2008. Last accessed Sep 2012. 2.1

[54] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. ACM SIGCOMM*, pages 109–114, Stanford, CA, August 1988. 1.1.2

[55] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms. In *Proc. ACM SIGCOMM*, Baltimore, MD, August 1992. 5.1

[56] David Clark, John Wroclawski, Karen Sollins, and Bob Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. ACM SIGCOMM*, pages 347–256, Pittsburgh, PA, August 2002. 3.7

[57] Comcast. Comcast press room - corporate overview. `http://www.comcast.com/corporate/about/pressroom/corporateoverview/corporateoverview.html`, 2011. 3.6.3

[58] Contendo. Cotendo will announce patent-pending cloudlet(tm) platform at tnw 2011, 2011. `http://cotendo.pressdoc.com/19551-cotendo-will-announce-patent-pending-cloudlet-platform-at-tnw-2011`. 2.1

[59] Paolo Costa, Austin Donnelly, Antony Rowstron, and Greg O'Shea. Camdoop: exploiting in-network aggregation for big data applications. In *Proc. 9th USENIX NSDI*, San Jose, CA, April 2012. 1.1.1

[60] Costas Courcoubetis, Vasilios A. Siris, and George D. Stamoulis. Integration of pricing and flow control for available bit rate services in ATM networks. In *Proc. IEEE GLOBECOM*, 1996. 5.2

[61] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: An argument for network pluralism. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Karlsruhe, Germany, August 2003. 1.2.2, 3.7, 6.3

[62] Jon Crowcroft and Philippe Oechslin. Differentiated end-to-end internet services using a weighted proportional fair sharing TCP. *ACM SIGCOMM CCR*, 28, 1998. 5.1, 5.2, 2

[63] Dragana Damjanovic and Michael Welzl. MulTFRC: providing weighted fairness for multimedia applications (and others too!). *ACM SIGCOMM CCR*, 39, June 2009. 2

[64] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Skype video responsiveness to bandwidth variations. In *Proc. ACM NOSSDAV*, 2008. 4.2, 4.3, 4.4.2

[65] Maurice de Kunder. The size of the World Wide Web. `http://www.worldwidewebsize.com/`, January 2011. 3.6.3

[66] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM*, 2011. 2.2.1

[67] Fahad Dogar, Amar Phanishayee, Himabindu Pucha, Olatunji Ruwase, and David Andersen. Ditto - a system for opportunistic caching in multi-hop wireless mesh networks. In *Proc. ACM MobiCom*, San Francisco, CA, September 2008. 2.2.2

[68] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-shen, and Nick Mckeown. Processor sharing flows in the Internet. In *Proc. IWQoS*, 2005. 5.1, 5.1, 5.2, 5.3.1, 5.4.1, 6.1.3, 2

[69] Emulab. Emulab. `http://www.emulab.net/`. 4.7.2

[70] F5 Networks. BIG-IP hardware. `http://www.f5.com/products/hardware/big-ip.html`. 2.2.2

[71] Nick Feamster and Hari Balakrishnan. Packet loss recovery for streaming video. In *Proc. International Packet Video Workshop*, 2002. 4.3

[72] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *Proc. IEEE INFOCOM*, pages 107 –116 vol.1, March 1999. doi: 10.1109/INFCOM.1999.749258. 2.2.2

[73] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1, August 1993. 5.3.1

[74] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM*, 2000. 4.5, 5.1, 4, 5.4.2

[75] Bryan Ford. Structured streams: a new transport abstraction. In *Proc. ACM SIGCOMM*, 2007. 1

[76] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with Coral. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004. 2.2.2

[77] P. Frossard and O. Verscheure. Joint source/FEC rate selection for quality-optimal MPEG-2 video delivery. *IEEE Transactions on Image Processing*, 10(12), December 2001. 4.3, 4.4.2

[78] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James

Wilcox. Intelligent design enables architectural evolution. In *Proc. ACM Workshop on Hot Topics in Networks*, 2011. 1.2.2, 1.2.2, 3.7, 5.7, 6.3

[79] Richard J. Gibbens and Frank P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, pages 1969–1985, 1999. 5.2, 5.3.2

[80] Google. Google: one million servers and counting. `http://www.pandia.com/sew/481-gartner.html`, 2007. 3.6.3

[81] GoogleProtocolBuffers. Protocol Buffers. `http://code.google.com/apis/protocolbuffers`. 3.5

[82] S. Govind, R. Govindarajan, and J. Kuri. Packet reordering in network processors. In *Proc. IEEE IPDPS 2007*, March 2007. 3.3.3

[83] Robert Grandl, Dongsu Han, Suk-Bok Lee, Hyeontaek Lim, Michel Machado, Matthew Mukerjee, and David Naylor. Supporting network evolution and incremental deployment with xia. In *Proc. of the ACM SIGCOMM*, 2012. (Demo). 3.5

[84] Mark Gritter and David R. Cheriton. TRIAD: A New Next-Generation Internet Architecture. `http://www-dsg.stanford.edu/triad/`, July 2000. 3.7, 6.3

[85] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008. 1, 6.2

[86] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42, July 2008. 5.3.1

[87] A. Hagedorn, S. Agarwal, D. Starobinski, and A. Trachtenberg. Rateless coding with feedback. In *Proc. IEEE INFOCOM*, 2009. 4.3

[88] Dongsu Han, Ashok Anand, Aditya Akella, and Srinivasan Seshan. RPT: Re-architecting loss protection for content-aware networks. Technical Report TR-11-117, Carnegie Mellon Univ., 2011. 4.6, 4.7.3, 4.7.3, 9

[89] Dongsu Han, Ashok Anand, Aditya Akella, and Srinivasan Seshan. RPT: Re-architecting loss protection for content-aware networks. In *Proc. 9th USENIX NSDI*, San Jose, CA, April 2012. 3, 4, 4.1.3, 5.1

[90] Dongsu Han, Ashok Anand, Fahad Dogar, Boyan Li, Hyeontaek Lim, Michel Machado, Arvind Mukundan, Wenfei Wu, Aditya Akella, David G. Andersen, John W. Byers, Srinivasan Seshan, and Peter Steenkiste. XIA: Efficient support for evolvable internetworking. In *Proc. 9th USENIX NSDI*, San Jose, CA, April 2012. 2, 2.2.1, 3, 4, 5.7

[91] Dongsu Han, Robert Grandl, Aditya Akella, and Srinivasan Seshan. A framework for an evolvable transport protocol. Technical Report CMU-CS-12-125, Carnegie Mellon

University, May 2012. 5

[92] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: a GPU-accelerated software router. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010. 3.6.1, 3.6.1

[93] HIP. Host Identity Protocol (HIP) Architecture. Interent Engineering Task Force, RFC 4423, May 2006. 3.7

[94] Uwe Horn, K. Stuhlmller, Ericsson Eurolab Herzogenrath, M. Link, and B. Girod. Robust internet video transmission based on scalable coding and unequal error protection. *Signal Processing: Image Communication*, 1999. 4.4.2, 4.4.4

[95] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1), February 1988. 2.2.2

[96] Ningning Hu, Li (Erran) Li, and Zhuoqing Morley Mao. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *Proc. ACM SIGCOMM*, pages 41–54, Portland, OR, August 2004. 4.6

[97] Infineta. Velocity Dedupe Engine. `http://www.infineta.com/technology/reduce`, 2011. 2.1, 4.2

[98] Infineta—Press Release. Infineta systems ships first-ever 10gbps, wire speed wan optimization system for high capacity data center interconnects, June 2011. `http://www.infineta.com/company/news/press-releases/infineta-systems-ships-first-ever-10gbps-wire-speed-wan-optimization`. 2.2.2

[99] ITRS. Itrs 2011 edition: Executive summary, 2011. 1.1.1

[100] ITRS. Itrs 2011 edition: Overall roadmap technology charateristics (ortc) tables, 2011. (document), 1.2

[101] ITU-T. Recommendation G.114 one-way transmission time. 4.2

[102] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proc. ACM CoNEXT*, December 2009. 2.2.1, 2.2.2, 3, 3.6.2, 3.7, 4.6, 6.3

[103] Szymon Jakubczak and Dina Katabi. A cross-layer design for scalable mobile video. In *Proc. ACM MobiCom*, 2011. 4.4.4

[104] Kyle Jamieson and Hari Balakrishnan. PPR: Partial packet recovery for wireless networks. In *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007. 4.4.4

[105] Juniper Networks. The new newtork is application infrastructure. `http://www.juniper.net/us/en/solutions/enterprise/application-`

infrastructure/. 2.1

[106] Dina Katabi, Mark Handley, and Chalrie Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, August 2002. 5.1, 5.3.1, 5.4.1, 5.6.1, 6.1.3

[107] F P Kelly, A K Maulloo, and D K H Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3), 1998. 5.1, 5.2, 5.3.2, 5.4, 5.4.1, 6.1.3, 2

[108] Frank Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 1997. 5.1, 6.1.3

[109] Frank Kelly, Gaurav Raina, and Thomas Voice. Stability and fairness of explicit congestion control with small buffers. *ACM SIGCOMM CCR*, 2008. 5.1, 5.2, 6.1.3

[110] Changoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in SEATTLE: A scalable ethernet architecture for large enterprises. In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008. 3.6.3

[111] Aditya Kishore. Will operators embrace akamai's licensed cdn?, February 2012. http://www.lightreading.com/document.asp?doc_id=218111. 2.2.2

[112] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992. 2.2.2

[113] Jirka Klaue, Berthold Rathke, and Adam Wolisz. EvalVid - a framework for video transmission and quality evaluation. In *Proc. Performance TOOLS*, 2003. 4.7.1, 4.7.1

[114] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000. 3.5, 3.6.1, 4.7.1

[115] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-Oriented (and Beyond) Network Architecture. In *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007. 2.2.2, 3.6.2, 3.7, 3.7, 6.3

[116] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997. 2.2.2

[117] S. Kunniyur and R. Srikant. End-to-end congestion control schemes: utility functions, random losses and ecn marks. In *Proc. IEEE INFOCOM*, volume 3, pages 1323 –1332 vol.3, March 2000. 5.1, 5.2

[118] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *Proc. ACM SIGCOMM*, 2011. 1.1.1, 2.1

[119] A. Li. RTP Payload Format for Generic Forward Error Correction. RFC 5109 (Proposed Standard), December 2007. 5.1

[120] J. Lilley, J. Yang, H. Balakrishnan, and S. Seshan. A Unified Header Compression Framework for Low-Bandwidth Links. In *Proc. ACM Mobicom*, Boston, MA, August 2000. 3.3.3

[121] Hang Liu and M. El Zarki. Performance of H.263 video transmission over wireless channels using hybrid ARQ. *IEEE JSAC*, 15(9), December 1997. 4.3

[122] Dmitri Loguinov and Hayder Radha. On retransmission schemes for real-time streaming in the Internet. In *Proc. IEEE INFOCOM*, 2001. 4.2, 4.3

[123] Steven H. Low and David E. Lapsley. Optimization flow controli: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999. 5.1, 5.2

[124] Zongtao Lu and Shijie Zhang. Stability analysis of xcp congestion control systems. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, April 2009. 5.6.1

[125] Reiner Ludwig and Randy H. Katz. The Eifel algorithm: making TCP robust against spurious retransmissions. *ACM SIGCOMM CCR*, 30:30–36, January 2000. 3.3.3

[126] Jeffrey K. MacKie-Mason and Hal R. Varian. Pricing the internet. Computational Economics 9401002, EconWPA, January 1994. 5.2

[127] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *Proc. 5th USENIX OSDI*, Boston, MA, December 2002. 1.1.1

[128] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *Proc. 7th USENIX OSDI*, Seattle, WA, November 2006. 4.6

[129] Laurent Massoulié and James Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Trans. Netw.*, 10(3):320–328, June 2002. ISSN 1063-6692. 5.1, 5.2

[130] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 124–139, Kiawah Island, SC, December 1999. 3.7

[131] Steven McCanne, Martin Vetterli, and Van Jacobson. Low-complexity video coding for receiver-driven layered multicast. *IEEE JSAC*, 15(6), 1997. 4.4.4

[132] Alberto Medina, Mark Allman, and Sally Floyd. Measuring interactions between transport protocols and middleboxes. In *Proc. of ACM Internet Measurement Conference (IMC)*,

2004. 1.1.2

[133] Greg Minshall, Yasushi Saito, Jeffrey C. Mogul, and Ben Verghese. Application performance pitfalls and TCP's nagle algorithm. *SIGMETRICS PER*, 27, March 2000. 5.1

[134] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer2 data center network fabric. In *Proc. ACM SIGCOMM*, Barcelona, Spain, August 2009. 3.6.3, 3.6.3

[135] Kiyohide Nakauchi and Katsushi Kobayashi. An explicit router feedback framework for high bandwidth-delay product networks. *Comput. Netw.*, 51, May 2007. 4.6

[136] Preethi Natarajan, Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. SCTP: an innovative transport layer protocol for the web. In *Proc. World Wide Web*, 2006. 1

[137] Colin Neagle. Wan optimization market shakeup predicted, January 2012. `http://www.networkworld.com/news/2012/011912-wan-optimization-255076.html`. 2.2.1, 2.2.2, 4.1.1

[138] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the sprite network file system. *ACM Trans. Comput. Syst.*, 6(1):134–154, February 1988. 2.2.2

[139] Giang T. K. Nguyen, Rachit Agarwal, Junda Liu, Matthew Caesar, Brighten Godfrey, and Scott Shenker. Slick packets. In *Proc. SIGMETRICS*, 2011. 3.7

[140] Erik Nordstrom, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedman. Serval: An end-host stack for service-centric networking. In *Proc. 9th USENIX NSDI*, San Jose, CA, April 2012. 2.1, 3, 3.1, 3.6.2, 3.7, 3.7

[141] Andrew Odlyzko. Paris metro pricing: The minimalist differentiated services solution. In *Proc. IWQoS*, 1999. 5.2, 5.4.2

[142] University of Oregon. RouteViews. `http://www.routeviews.org/`. 3.6.1

[143] L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002. URL `http://www.ietf.org/rfc/rfc3286.txt`. 1.1.1

[144] Open Networking Foundation. Open networking foundation. `://www.opennetworking.org/`, 2012. 2.1

[145] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *Proc. NOSSDAV*, 2002. 4.2

[146] Christos Papadopoulos. Retransmission-based error control for continuous media applications. In *Proc. NOSSDAV*, 1996. 4.3

[147] Parveen Patel, Andrew Whitaker, David Wetherall, Jay Lepreau, and Tim Stack. Upgrad-

ing transport protocols using untrusted mobile code. In *Proc. ACM SOSP*, 2003. 5.2

[148] Lucian Popa, Ali Ghodsi, and Ion Stoica. HTTP as the narrow waist of the future Internet. In *Proc. ACM Hotnets-IX*, Monterey, CA, USA., October 2010. 1.2.2, 1.2.2, 3.7

[149] R. Puri and K. Ramchandran. Multiple description source coding using forward error correction codes. In *Proc. Asilomar conference on signals, systems, and computers*, 1999. 4.3, 4.4.4

[150] Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C. Snoeren. Cloud control with distributed rate limiting. In *Proc. ACM SIGCOMM*, 2007. 3

[151] K. K. Ramakrishnan. CPM - adaptive VoD with cooperative peer assist and multicast. Keynote Speech at the IEEE Workshop on Local and Metropolitan Area Networks, September 2008. 2.2.1

[152] Sylvia Ratnasamy, Scott Shenker, and Steven McCanne. Towards an evolvable Internet architecture. In *Proc. ACM SIGCOMM*, Philadelphia, PA, August 2005. 1.2.2, 1.2.2, 3.7

[153] Dan Rayburn. Transparent caching market $142m this year, growing to over $400m by 2014, December 2011. `http://blog.streamingmedia.com/the_business_of_online_vi/2011/12/transparent-caching-market-142m-this-year-growing-to-over-400m-by-2014.html`. 2.2.1, 2.2.2

[154] Dan Rayburn. Limelight launches managed cdn offering for carriers, is a deal with f5 next?, February 2012. `http://blog.streamingmedia.com/the_business_of_online_vi/2012/02/limelight-networks-launches-licensed-cdn-lcdn-offering.html`. 2.2.2

[155] AT&T Press Release. At&t delivering web content faster with new content delivery network solution. `http://www.corp.att.com/emea/insights/pr/eng/cdn_110411.html`. 2.1

[156] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the Internet. In *Proc. ACM SIGCOMM*, Vancouver, British Columbia, Canada, September 1998. 4.2, 4.3

[157] Riverbed. Riverbed. `http://www.riverbed.com/`. 2.2.1, 2.2.2

[158] Umar Saif and Justin Mazzola Paluska. Service-oriented network sockets. In *Proc. ACM MobiSys*, San Francisco, CA, May 2003. 3

[159] D. Salomon. *Data Compression; the Complete Reference*. Springer, 2007. 4.7.1

[160] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. USENIX conference on Networked Systems Design and Implementation*, 2012. 2.1

[161] Sayandeep Sen, Neel Kamal Madabhushi, and Suman Banerjee. Scalable wifi media delivery through adaptive broadcasts. In *Proc. USENIX NSDI*, 2010. 4.4.4

[162] Ivan Seskar, Kiran Nagaraja, Sam Nelson, and Dipankar Raychaudhuri. Mobilityfirst future internet architecture project. In *Proc. of the 7th Asian Internet Engineering Conference*, 2011. 3.7

[163] Chetan Sharma. The mobile operator's dilemma (and opportunity): the forth curve. `http://gigaom.com/2012/07/17/the-mobile-operators-dilemma-and-opportunity-the-fourth-curve/`. 1.1.1

[164] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in computer networks: reshaping the research agenda. *ACM SIGCOMM CCR*, 26(2), April 1996. 5.2

[165] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proc. ACM SIGCOMM*, 2012. 1, 1.1.1, 2.1

[166] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? In *Proc. 9th USENIX OSDI*, Vancouver, Canada, October 2010. 3.7, 5.2

[167] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proc. 8th USENIX NSDI*, 2011. 5.4.3, 5.6.3

[168] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Proc. ACM Mobicom*, pages 155–166, Boston, MA, August 2000. 3.4.3

[169] M.I. Soliman and G.Y. Abozaid. Performance evaluation of a high throughput crypto coprocessor using VHDL. In *Proc. ICCES*, December 2010. 1

[170] Neil Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. ACM SIGCOMM*, Stockholm, Sweden, September 2000. 2.2.1, 2.2.2, 4.1.1, 4

[171] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. In *Proc. ACM SIGCOMM*, 1998. 5.3.1

[172] Ion Stoica, Daniel Adkins, Shelley Zhaung, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, pages 73–86, Pittsburgh, PA, August 2002. 3.3.2, 3.7, 6.3

[173] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004. 4, 5.4.2

[174] Abraham Suissa, Jennifer Mellor, Frantz Lohier, and Patrick Garda. A novel video packet loss concealment algorithm & real time implementation. In *Proc. DASIP*, 2008. 4.4.4

[175] Jon Tee. Carriers to compete more aggressively in content delivery: Cdns hope this will grow the market. `http://innovationobservatory.com/content/carriers-compete-more-aggressively-content`. 2.2.1, 2.2.2

[176] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *ACM Computer Communications Review*, 26(2):5–18, April 1996. 3.7, 6.3

[177] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil. An architecture for Internet data transfer. In *Proc. 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, May 2006. 2.2.2

[178] TRILL. IETF transparent interconnection of lots of links (TRILL) working group. `http://datatracker.ietf.org/wg/trill/charter/`. 1.1.1, 3.6.3

[179] Dirk Trossen, Mikko Sarela, and Karen Sollins. Arguments for an information-centric internetworking architecture. *ACM Computer Communications Review*, 40:26–33, April 2010. 3

[180] Arun Venkataramani, Ravi Kokku, and Mike Dahlin. TCP Nice: a mechanism for background transfers. *SIGOPS Oper. Syst. Rev.*, 36, December 2002. 5.1, 2, 5.4.2

[181] VeriSign. Internet Defense Network, 2010. `http://www.verisign.com/ddos-protection/index.html`. 3.3.2

[182] Verivue. Transparent caching. `http://www.verivue.com/products-carrier-cdn-transparent-cache.asp`. 2.2.1, 2.2.2

[183] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *Proc. 6th USENIX OSDI*, San Francisco, CA, December 2004. 1.1.2, 3.3.2

[184] Jue Wang and Dina Katabi. ChitChat: Making video chat robust to packet loss. Technical Report MIT-CSAIL-TR-2010-031, MIT, July 2010. 4.3, 4.4.4

[185] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang. An untold story of middleboxes in cellular networks. In *Proc. ACM SIGCOMM*, 2011. 1.1.2, 2.1

[186] G. Watson, N. McKeown, and M. Casado. NetFPGA: A tool for network research and education. In *2nd workshop on Architectural Research using FPGA Platforms (WARFP)*, 2006. 3.7

[187] David J. Wetherall. *Service Introduction in an Active Network*. PhD thesis, Massachusetts Institute of Technology, November 1998. 1.2.2, 1.2.2

[188] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than

late: meeting deadlines in datacenter networks. In *Proc. ACM SIGCOMM*, 2011. 5.4.2

[189] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. 8th USENIX NSDI*, Boston, MA, April 2011. 5.1, 3, 5.4.2

[190] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, December 1999. 2.2.2

[191] Wenji Wu, Phil Demar, and Matt Crawford. Sorting reordered packets with interrupt coalescing. *Computer Networks*, 53:2646–2662, October 2009. 3.3.3

[192] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, and Heung keung Chai. A cdn-p2p hybrid architecture for cost-effective streaming media distribution. *Computer Networks*, 44:353–382, 2004. 2.2.2

[193] Y.R. Yang and S.S. Lam. General aimd congestion control. In *Proc. ICNP*, 2000. 2, 5.3.2

[194] Yung Yi and Mung Chiang. Stochastic network utility maximisation - a tribute to kelly's paper published in this journal a decade ago. *European Transactions on Telecommunications*, 19(4):421–442, 2008. 5.2

[195] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *Proc. ACM Multimedia*, 2009. 2.2.2

[196] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. BUFFALO: bloom filter forwarding architecture for large organizations. In *Proc. ACM CoNEXT*, 2009. 3.6.3

[197] Xiaoqing Zhu, Rong Pan, Nandita Dukkipati, Vijay Subramanian, and Flavio Bonomi. Layered Internet video engineering (LIVE): Network-assisted bandwidth sharing and transient loss protection for scalable video streaming. In *Proc. IEEE INFOCOM*, 2010. 4.3