A Practical System for Centralized Network Control

Hong Yan

CMU-CS-10-149
Nov 2010

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Hui Zhang, Chair
David Andersen
Srinivasan Seshan
David A. Maltz, Microsoft Research
T. S. Eugene Ng, Rice University

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

This research was sponsored by the NSF under ITR Awards ANI-0085920, ANI-0331653, and NeTS Grant CNS-0520187. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Microsoft, NSF, or the U.S. government.

Keywords: Centralized Network Control, Clean-slate Network Design, Network Dissemination Plane, Network Management

Abstract

IP networking is a spectacular success, catalyzing the diffusion of data networking across academic institutions, governments, businesses, and homes worldwide. Yet, despite the fundamental importance of this infrastructure, today's networks are surprisingly fragile and increasingly difficult to configure, control, and maintain. As our dependence on data networking grows, so do the risks of security breaches, large-scale outages, and service disruptions.

We believe that the root of these problems lies in the complexity of the control and management planes—the software and protocols coordinating network elements—and particularly the way the decision logic and the distributed-systems issues are inexorably intertwined. The research community advocates a complete refactoring of the functionality and proposes a new architecture which they call "4D," after the architecture's four planes: decision, dissemination, discovery, and data. The 4D architecture pulls decision-making logic out of the network elements to create a logically centralized decision plane, where network-level objectives and policies are specified and enforced by direct configuration of state on individual network elements.

While the 4D vision is conceptually appealing, it has raised a wide range of practical concerns related to robustness, flexibility, scalability, and security. Our thesis is that "*it is actually possible to build a 4D network that is as scalable and robust as traditional IP networks but greatly simplifies network control and management*". To prove this thesis, we must address the following technical challenges:

1. What kind of decision-plane framework will enable the centralization and composition of multiple network control functions for sophisticated network control?

2. How can we provide reliable connectivity to remotely manage distributed network elements without relying on the communication services that are being managed?

3. Is there an efficient way to disseminate control messages from central decision servers to a large number of network elements?

We believe that answering the above questions is key to the successful deployment of 4D networks. In this dissertation, we tackle those challenges by building a 4D network control platform called Tesseract and demonstrating that Tesseract enables both simpler protocols that do not have to embed decision-making logic, and more powerful decision algorithms for implementing sophisticated goals. The main target of our work is to turn the revolutionary 4D concept into a practical working system.

To my parents

Acknowledgements

I owe my deepest gratitude to my advisor, Hui Zhang. This thesis would not have been possible without his patient guidance and extraordinary support. Hui teaches me to do good research, write good research papers, and give good research talks. But I've also learned far more from him. He gives me the strength and courage to overcome difficulties and achieve life goals. The journey to a PhD is not easy, and the life ahead is certain to be filled with even greater challenges, but I believe that I'm better prepared than I was seven years ago.

I would also like to thank my committee members, David Maltz, Eugene Ng, David Andersen, and Srinivasan Seshan. David Maltz and Eugene Ng have made tremendous contributions to the ideas presented in my thesis, and to my growth as a researcher. I can't remember how many times I practiced my first conference talk, but I do vividly recall the dry run in the NSDI hotel room with David and Eugene. I will also never forget the sleepless days and nights we spent together before paper deadline, and the joys and frustrations we shared in the process of pushing the new concept of direct network control.

David Andersen and Srinivasan Seshan have provided extremely valuable comments and suggestions on my thesis. They force me put the thesis work in a broader context, and to rethink it from fresh perspectives. They ask challenging questions, but try hard to help me find the answers. Thank you!

Over the past years, I have been very lucky to work with prestigious researchers – Albert Greenberg, Jennifer Rexford, David Garlan, my intern mentor, Kay Sripanidkulcha, as well as super-smart fellow students Vyas Sekar, Andy Myers, Zheng Cai, Hemant Gogineni, Suman Nath, Yinglian Xie, and many others. They have generously given their advice and help when I most needed it.

Finally, I want to thank the Computer Science Department for providing a relaxing academic environment and the freedom to pursue my research interests. In particular, I thank Sharon Burks for convincing me to join CMU; Deborah Cavlovich for continually reminding me to submit my thesis; Catherine Copetas for getting the technical report number for my thesis ten seconds after I requested it; and Kathy McNiff for making sure everything was ready when I proposed and defended.

Contents

Chapter 1

Introduction

Although IP networking has been wildly successful, there are serious problems lurking "under the hood". IP networks exhibit a defining characteristic of unstable complex systems – a small local event (e.g., misconfiguration of a routing protocol on a single interface) can have severe, global impact in the form of a cascading meltdown. In addition, individual Autonomous Systems (ASes) must devote significant resources to "work around" the constraints imposed by today's protocols and mechanisms to achieve their goals for traffic engineering, survivability, security, and policy enforcement. In seeking cures, the research community is exploring a clean slate re-design of the Internet architecture that greatly simplifies network control and management by centralizing architectural intent and directly expressing operational constraints. While the clean slate architecture is conceptually appealing, a substantial number of technical challenges (e.g., scalability) must be addressed before the concept can be proved and widely accepted. This thesis demonstrates that *it is actually possible to build a flexible centralized control system for a single network domain that is as scalable and robust as traditional IP networks but greatly simplifies network control and management*.

1.1 Problems

The Internet architecture bundles control logic and packet handling into the individual routers distributed throughout an AS. As a result, each router[1] participates in distributed protocols that implicitly *embed* the decision logic. For example, within an IP network domain the path-computation logic is governed by

[1]We use the terms "network element", "router", and "switch" interchangeably throughout the thesis.

distributed protocols such as OSPF, IS-IS, and EIGRP. The routing protocols dictate not only how the routers learn about the topology, but also how they select paths. Similarly, in Ethernet networks, the path-computation logic is embedded in the Spanning Tree protocol [1]. Many of today's data networks must support network-level objectives and capabilities far more sophisticated than best-effort packet delivery. These ever-evolving requirements have led to incremental changes in the control-plane protocols, as well as complex management-plane software that tries to "coax" the control plane into satisfying the network objectives. The resulting complexity is responsible for the increasing fragility of IP networks and the tremendous difficulties people face when trying to understand and manage their networks.

In data networks, the functionality that controls the network is split into three main planes: (i) the *data plane* that handles the individual data packets; (e.g., packet forwarding, packet filtering, queue management, and link scheduling); (ii) the *control plane* that implements the distributed routing algorithms across the network elements (e.g., routing protocols, path-computation algorithms, and logic for merging multiple routing tables into a single forwarding table); and (iii) the *management plane* that monitors the network and configures the data-plane mechanisms and control-plane protocols. While the original IP control plane was designed to have a *single* distributed algorithm to maintain the *forwarding* table in the data plane, today's IP data, control, and management planes are far more complex. The data plane needs to implement, in addition to next-hop forwarding, functions such as tunneling, access control, address translation, and queuing. The states used to implement these functions are governed by multiple entities and must be configured through a rich set of individual, interacting commands. Even for the forwarding state, multiple routing processes usually are running on the same router/switch.

While there are many dependencies among the states and the logic updating the states, most of the dependencies are *not* maintained automatically. For example, controlling routing and reachability today requires complex arrangements of commands to tag routes, filter routes, and configure multiple interacting routing processes, all while ensuring that no router is asked to handle more routes and packet filters than it has resources to cope with. A change in any one part of the configuration can easily break other parts.

The problem is exacerbated as packet delivery cannot commence until the routing protocols create the necessary forwarding tables, and the management plane cannot reach the control plane until the routing protocols are configured. Resolving this Catch-22 dilemma requires installing a significant amount of configuration information on IP routers before deployment.[2] Studies of production networks show them requiring hundreds of thousands of lines of low-level configuration commands distributed across all the routers

[2]This problem is so profound that, whenever possible, remote routers are plugged into telephone modems so that the Public

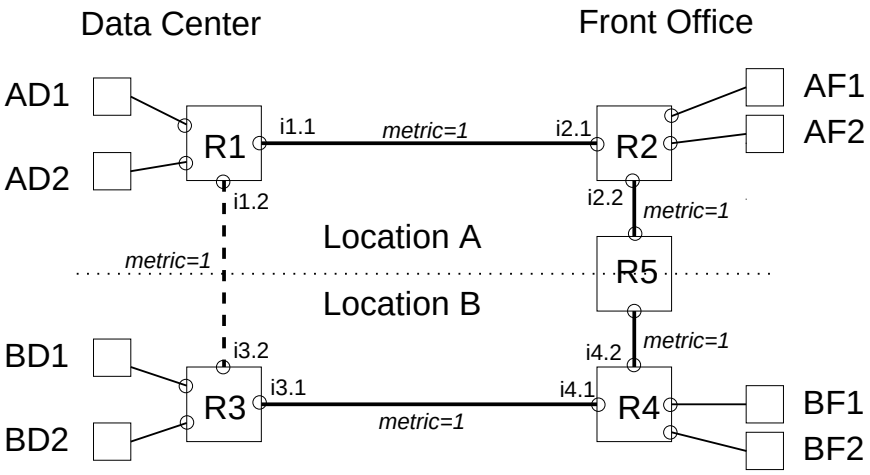

Figure 1.1: Enterprise network with two locations, each location with a front office and a data-center.

in the network [2]. These configurations and the dynamic forwarding state they generate require a myriad of ad hoc scripts and systems in the management plane to validate, monitor, and update. The result is a complex and failure-prone network.

We present two examples that illustrate the network fragility caused by today's complex and unwieldy control and management infrastructure. The examples illustrate how the lack of coordination between routing and security mechanisms can result in a fragile network, and how today's control and management infrastructure makes it difficult to properly coordinate the mechanisms.

Reachability Control in Enterprises Today, many enterprise networks attempt to control which hosts and services on the network can communicate (i.e., reach each other) as part of their security strategy [2]. They implement their strategies using a combination of routing policy and packet filters, but this approach is fraught with peril even in simple networks.

Consider the example enterprise network in Figure 1.1. The company has two locations, A and B. Each location has a number of "front-office" computers used by the sales agents (AF1-2 and BF1-2). Each location also has a data center where servers are kept (AD1-2 and BD1-2). Initially, the two locations are connected by a link between the front-office routers, R2 and R4, over which inter-office communications flow. The Interior Gateway Protocol (IGP) metric for each link is shown in italics.

Switched Telephone Network provides a management communication path of last resort. Before making configuration changes to the router over the Internet via Telnet or ssh, operators often double-check that the modem connection is still functioning, lest an unfortunate configuration mistake leave them with no other way to contact the router, short of physical access to the console.

The company's security policy is for front office computers to be able to communicate with other locations' front-office computers and the local data center's servers, but not the data center of the other location. Such policies are common in industries such as insurance, where the sales agents of each location are effectively competing against each other even though they work for the same company.

The security policy is implemented using packet filters on the routers controlling entrance to the data centers to drop packets that violate the policy. Interface i1.1 is configured with a packet filter that drops all packets from the BF subnet, and interface i3.1 drops all packets from the AF subnet.

The network functions as desired, until the day when the data-center staff decides to add a new, high-capacity dedicated link between the data centers (shown as a dashed line between R1 and R3 – perhaps they have decided to use each other as remote backup locations). It seems reasonable that with packet filters protecting the entrances to the data centers, the new link between data centers should not compromise the security policy. However, the new link changes the routing such that packets sent from AF to BD will travel from R2 to R1 to R3 to BD – completely avoiding the packet filter installed on interface i3.1 and violating the security policy. When the designers eventually discover the security hole, probably due to an attack exploiting the hole, they would typically respond by copying the packet filter from i3.1 to i3.2, so that it now also drops packets from AF. This filter design does plug the security hole, but it means that if the front-office link from R2 to R4 fails, AF will be unable to reach BF. Even though the links from R2 to R1 to R3 to R4 are all working, the packet filter on interface i3.2 will drop the packets from subnet AF.

In this example, the problems arise because the ability of a network to carry packets depends on the routing protocols and the packet filters working in concert. While routing automatically adapts to topology changes, there is no corresponding way to automatically adapt packet filters or other state. It could be argued that a more "optimal" placement of packet filters, or the use of multi-dimensional packet filters (i.e., filters that test both source and destination address of a packet) would fix the problems shown in this example. However, as networks grow in size and complexity from the trivial example used here for illustrative purposes, finding these optimal placements and maintaining the many multi-dimensional packet filters they generate requires developing and integrating entirely new sets of tools into the network's management systems. Since these tools will be separate from the protocols that control routing in real time, they will perpetually be attempting to remain synchronized with routing protocols by trying to model and guess the protocols' behavior.

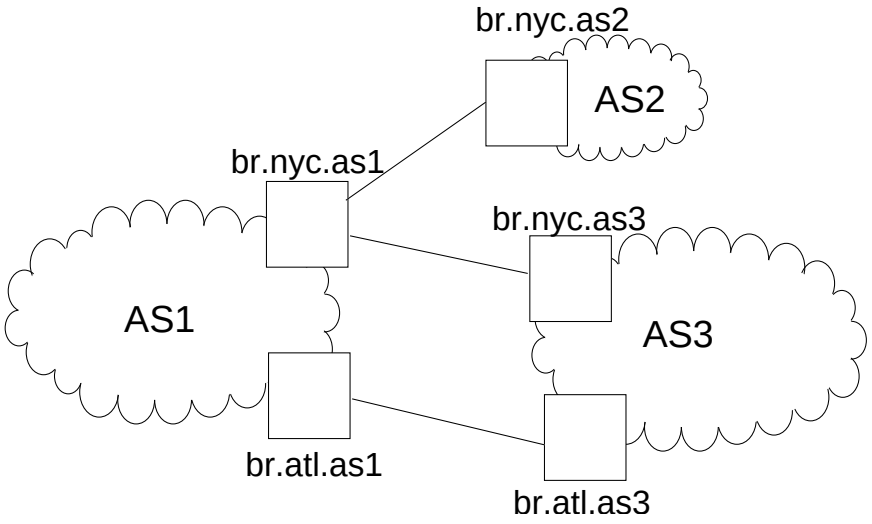

Figure 1.2: Autonomous Systems (ASes) peering with each other via external BGP (eBGP) sessions. AS1 must place packet filters on its ingress links to prevent AS3 from sending packets to destinations for which AS1 has not agreed to provide transit.

Peering Policies in Transit Networks Routing policy is based on the premise that a router that does not announce a route to a destination to a peer will not be sent packets for that destination by that peer. However, the routing system does nothing to prevent an unscrupulous peer from sending packets to that destination anyway. Enforcing routing policy is nearly impossible with today's control and management planes.

Figure 1.2 shows an example of three Autonomous Systems (ASes) peering with each other via three external BGP sessions (one eBGP session along each of the links shown in the figure). Assume that AS1 is a major transit network, and it announces a route to destination d in its eBGP session with AS2. If AS1's policy is to not provide AS3 with transit service for d, it does not announce d in its eBGP sessions with AS3. However, if AS3 wishes to be unscrupulous (e.g., use AS1 for transit service without paying), it can assume AS1 does know a way to d (e.g., so that AS1's own customers can reach d). If AS3 sends packets for d to br.nyc.as1, they will definitely be delivered, as br.nyc.as1 must have a route to d in order to handle legitimate traffic from AS2.

Enforcing routing policy requires installing packet filters to drop packets to destinations that have not been announced as reachable. As the announcements are received by an AS, and the AS's own topology changes over time, the announcements sent by the AS will change and the packet filters must be moved correspondingly. Implementing such functionality by adding another ad hoc script to the management plane is essentially impossible today. Even if it were possible to write a script that snoops on the eBGP announcements sent to each neighboring border router and installs packet filters on the ingress interface as appropriate,

the script would be extremely dangerous as it would not properly order the packet filter installation/removal with the BGP announcements. For example, it would be bad to announce to a neighbor border router a route to a destination before removing the packet filters that drop the packets sent to the destination.

Beyond ordering issues, transit networks handle a large number of destinations, and each packet filter applied to an interface consumes forwarding resources and reduces the effective capacity of the interface. It might be desirable to move packet filters into the network whenever possible, away from the ingress interfaces, so that one packet filter can enforce the BGP policy for multiple ingress interfaces.

Enforcing routing policy requires dynamically placing packet filters to respond to the continually changing routes selected by that policy. Correctly and optimally placing the filters requires that the placement be synchronized with the announcement of routing decisions and that the placement algorithms have access to the complete routing topology of the network.

Same Problems, Many Guises There are many data networks, designed and managed by different organizations with different goals. Individual networks serve radically different purposes; in addition to the familiar backbone networks, there are access, metro, enterprise, and data-center networks. In each of these settings, the network administrators struggle to "program" their networks, integrating a diverse set of technologies and protocols, and artfully setting the configurable parameters that determine the network's functionality and dynamics.

While the specific context, technology, and mechanisms may change from network to network, there is commonality among the problems. For example, while Ethernet was initially designed to run on a shared medium, it has since evolved into a networking technology with a full package of data plane, control plane, and management plane to rival IP. Just as IP has many routing protocols to compute the forwarding table, Ethernet has many variations of the spanning tree protocol [3]. Just as IP networks have mechanisms such as MPLS to control the paths that packets take, Ethernet has virtual LANs (and VLANs-in-VLANs). Just as IP networks have needed to implement sophisticated functionality, including traffic engineering, security policies, and fast restoration, these same needs are being required of Ethernet in many contexts, such as enterprises, data centers [4], and metro/access networks [5]. Just as ad hoc management capabilities need to be overlaid on top of the IP control plane, achieving advanced functionality in Ethernet networks has led to increasingly ad hoc and complex management systems. The current architecture forces these systems to operate outside Ethernet's control plane, where they often come into conflict with it.

1.2 Thesis

We argue that the root cause of these problems lies in the control plane running on the network elements and the management plane that monitors and configures them. The key to solving the problems is to create a way for the architectural intent and operational constraints governing the network to be expressed directly, and then automatically enforced by setting data-plane states on the individual routers/switches. Until this occurs, we expect the design and operation of robust networks to remain a difficult challenge, and the state of the art to remain a losing battle against a trend where ever-richer and more complex state and logic are embedded in distributed protocols or exposed through box-level interfaces.

There is an emerging trend for revisiting the division of functionality and advocating an extreme design point that turns a *network's decision logic* from distributed processes that run on individual routers into a *centralized control service*.

In the envisioned control paradigm, a logically centralized entity, called the decision element, is responsible for creating all the state at every router. As a result, any conflicts between the policy objectives can be detected at the time of state creation. With today's multiple independent and distributed mechanisms, these conflicts often only appear *in vivo* after some part of the configuration state has been changed by one of the mechanisms.

The centralized control paradigm also simplifies the router functionality. Because algorithms making control decisions are no longer run at switches, the only distributed functions to be implemented by switches are those that discover the neighborhood status at each switch, and those that enable the control communications between the decision element and the switches. Thus, the router software can be very light-weight. Yet sophisticated control algorithms can easily be implemented with this minimal set of distributed functions.

While the centralized control vision is conceptually appealing, providing a platform to support centralized network control has not been attempted, and is technically challenging. For example, it is generally considered not scalable to compute routing tables and push them to thousands of routers in the network from a single server; remotely configuring a network using the network being configured appears to be a chicken-and-egg problem; and it is unclear whether shifting control logic from distributed routers to a centralized server makes the server too complex to build and operate. The natural question then is: is it actually practical to run a network with centralized control?

In this dissertation we answer this question affirmatively by building a centralized network control platform and showing that the platform can support some of the most representative network control functionality with high performance. Our thesis statement is that "*it is actually possible to build a centralized control platform that is as scalable and robust as traditional IP networks but greatly simplifies network control and management*".

1.3 Contributions

The main contribution of this dissertation is to *provide a robust and scalable system to enable flexible centralized network control*. In particular, we build, extend, and evaluate a centralized network control system called *Tesseract* to answer the following questions:

1. **How to make centralized control flexible?** *Flexibility* represents the ability of the system to support different types of control logic and network technologies. To answer this question, we use experiments to demonstrate that the Tesseract system can seamlessly integrate most useful control functionality, such as shortest path routing, packet filtering, and traffic engineering. We also show that the same control logic can operate on both IP and Ethernet networks. In particular, we apply shortest path routing to Ethernet to increase throughput performance without compromising the Ethernet plug-and-play feature.

2. **How to make centralized control robust?** *Robustness* represents the ability of the system to survive failures and attacks. We answer this question by simulating network failures and attacks and measuring the impact on Tesseract. For example, we take down network links and measure the time it takes the platform to re-compute routes and re-install the new routes on the affected routers. Having realized the importance of having a robust communication channel for network control, we develop a generic remote management communication system based on the Tesseract dissemination component. We call it the Meta-Management System (MMS). The MMS is built in with a *recursive authentication* algorithm and a variety of liveness mechanisms to ensure reachability between the centralized network decision servers and routers in the event of network failures and DDoS attacks.

3. **How to make centralized control scalable?** *Scalability* represents the ability to support the increasing size of networks. We develop an efficient dissemination algorithm to make dissemination load

increase sub-linearly with the number of nodes in its control domain. We leverage an insight that network state pushed to different routers can be similar, and we explore the similarity to significantly improve dissemination scalability.

In answering the above three questions, we use the Emulab [6] testbed to conduct intensive experiments with topologies of real production backbone and enterprise networks. We also build simple analytical models and use theoretical analysis to derive the time complexity of centralized computation. We verify the analytical results using experimental data. For example, in evaluating our recursive authentication algorithm for the Meta-Management System we build a model to derive the time it takes for a single central server to authenticate n network nodes. We run experiments on different-sized network topologies to verify that the experimental results agree with our theoretical analysis.

The most critical step we take to enable extensive experimental evaluation is a Linux implementation of the Tesseract system. We present the implementation details in this dissertation and we open-source the code in http://www.cs.cmu.edu/~4d.

1.4 Acknowledgements

The initial idea of network-wide decision-making is proposed by Jennifer Rexford et al. [7]. The idea is crystallized and turned into the clean slate network design concept by Albert Greenbert et al. [8]. This thesis work is part of the effort to develop a practical system called *Tesseract* [9]. The *Tesseract* project is joint work with David Maltz at Microsoft Research and T. S. Eugene Ng et al. at Rice University.

1.5 Organization

The rest of this dissertation is organized as follows:

Chapter 2 places this thesis work in the context of centralized network control and explains how 4D/Tesseract is different from other centralized network control frameworks.

Chapter 3 describes the basic framework of our solution – the Tesseract system. We use experimental results to show the new system is comparable performance-wise with the traditional distributed network control system while providing a variety of appealing new capabilities.

Chapter 4 addresses the robustness problem of 4D dissemination. We design and implement a Tesseract dissemination subsystem called the Meta-Management System. It is equipped with a recursive authentication algorithm and protective APIs for surviving severe network failures and DDoS attacks.

Chapter 5 addresses the scalability problem of 4D dissemination. We conceive an efficient dissemination algorithm to improve scalability. Experimental results show that the new algorithm reduces server dissemination load by 97%.

Chapter 2

A Taxonomy of Centralized Network Control

The focus of this thesis work is a centralized network control system and its dissemination component, which is designed to also serve other network control systems. This chapter provides the context of this thesis work by giving an overview of developments concerning network control and management, especially centralized network control systems. As outlined in Figure 2.1, we position our system in the context of network control and management and describe how it differs from other existing centralized network control systems. We also compare our work on network control dissemination with prior work to show where our contribution lies.

2.1 State-of-the-art

In a running network, control and management systems are responsible for maintaining distributed state at switches and routers in the presence of dynamic traffic and network conditions. In data networks, the network control system includes the routing protocols and their configuration. The management system is realized by the people and programs that track network behavior and adapt the configuration over time. The control and management systems are where network intelligence lies, and so are fundamental to the operation of the overall network.

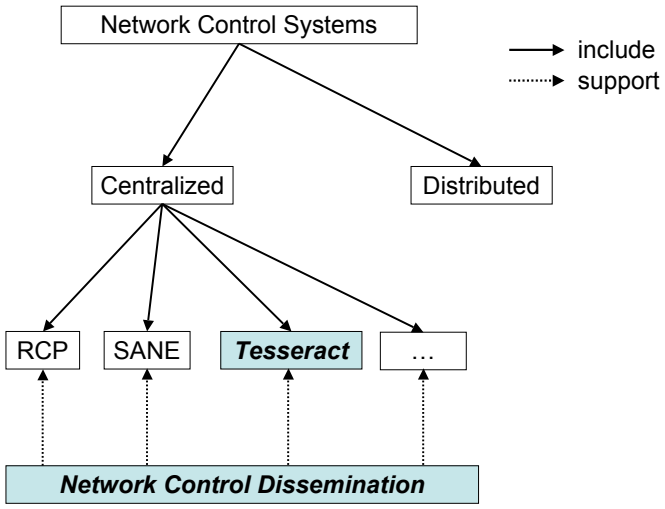

Figure 2.1: A taxonomy of network control systems. This thesis implements the shadowed boxes: Tesseract and Network Control Dissemination. Tesseract is one of the pioneering implementations of the centralized network control architecture. Tesseract distinguishes itself from other prototypes by (i) providing a platform to compose a variety of network control algorithms and to operate different types of networks (e.g., IP and Ethernet); (ii) providing a generic dissemination service for other centralized network control and management systems.

2.1.1 Distributed Control and Centralized Management

There are two important distinctions between the control and management systems. First is the timescale on which they operate. To be responsive, the IP routing protocols exchange messages, re-compute distributed state, and take actions on a relatively fast timescale. The management system, on the other hand, usually operates on a slower timescale. Second is how the functionality of the system is implemented. The control system, due to the requirement for rapid response, is typically implemented in a distributed fashion, where intelligence or control logic resides with each router and protocols are used to coordinate the actions of all routers. The coordinations are fully automatic. In contrast, the management system is usually implemented via a client-server architecture, where a centralized server (which can be made highly available by standard techniques) collects information from all routers, computes and generates actions (possibly with the input of a human operator), and sends management commands to routers.

2.1.2 Problem

Historically, IP networks were designed to provide best-effort service. Initially, support for management was minimal, with most of the intelligence residing within the routing protocols, which were first designed

	Network Type	Domain	Dissemination	Decision
Tesseract	IP/Ethernet	Intra-domain	Logically Out-of-Band	Programmable Platform
RCP	IP	Inter-domain	iBGP	BGP Route Selection
SANE	Ethernet	Enterprise	Spanning Tree	Security Policy Enforcement
SS7	Circuit Switching	Telecommunication	Physically Out-of-Band	Call Management

Table 2.1: Side-by-side comparison between Tesseract and other centralized network control systems.

to provide resiliency under link and router failures. As IP networks became more mainstream, management interfaces such as SNMP were added to routers, but in general the management system remains very primitive. At the same time, a large number of features such as support for traffic engineering, policy routing, network maintenance, VPN, etc., have been added to IP networks, primarily by extending the control system.

Adding features by extending the distributed control system requires enhancing the existing routing protocols in order to (i) propagate more dynamic metrics such as link load in the case of traffic engineering; and (ii) perform more functions, such as computing new paths as the traffic load changes. Despite the conceptual appeal of this approach, overloading routing protocols introduces substantial complexity and state in the control system, which negatively affects scalability, robustness, and performance. Over time, adding features becomes more difficult, as each new feature must interoperate with all the previous ones.

2.2 Exploration of Centralized Control

In exploring a revolutionary solution to enhance network functionality without overloading networks with distributed control state, the researchers propose to investigate centralized network control architectures. The Routing Control Platform (RCP) [10, 11] was the first prototype that explored centralizing network control. The concept of lifting network control logic from distributed network elements into a logically centralized decision element was explicitly called out and presented as the 4D architecture [7, 8]. The main contribution of this thesis is turning 4D into a real system called Tesseract[1].

2.2.1 Prior Centralized Network Control Systems

Centralized network control systems that precede 4D include the Routing Control Platform, Signaling System 7 and the Intelligent Network, and policy-based networking.

[1]The tesseract is the 4-dimensional hypercube.

Routing control platform (RCP): RCP is a solution for controlling inter-domain routing in IP networks, and replaces today's distributed BGP routing computations. RCP computes the BGP routes for an Autonomous System (AS) at centralized servers to give the operators of transit networks greater control over how BGP routing decisions are made.

Traditional telecommunications networks: The concept of centralization is heavily used in many management paradigms for telecommunications networks, usually based on circuit-switched technology [12]. Signaling System 7 (SS7) [13, 14] keeps communication channels for management information isolated from the paths used by user data, and takes the approach of a hard separation between management and user data into separate links or channels. The Intelligent Network (IN) architecture [15] supports extension of network functionality by enabling user data (call placements) to trigger Detection Points that escape out of normal call handling and that invoke per-service code.

Management tools for a distributed control plane: Many tools have been developed to ease the configuration of the existing architecture for control and management, which depends on individually configured switches/routers running a distributed control plane. Some approaches, such as those adopted by Cplane and Orchestream, developed frameworks to solve the problems inherent in configuring large numbers of distributed switches/routers that may use different command languages. Other tools focus on specific operational tasks, such as traffic engineering or mitigation of Denial-of-Service (DoS) attacks. For example, Cariden's MATE [16] and OpNet's SP Guru [17] products can tune OSPF costs or MPLS Label Switched Paths to the prevailing traffic, and ArborNetwork's PeakFlow DoS [18] product detects DoS attacks and generates filters to block the offending traffic. The general approach of policy-based networking (PBN) has been studied to automate provisioning and network management in applications such as QoS [19]. Network-management tools and PBN approaches usually assume the existing control-plane protocols, focus on a small portion of the configuration state (e.g., packet filters, but not routing), and do not consider the interactions among multiple mechanisms.

2.2.2 4D/Tesseract

4D for the first time presents a centralized network control architecture in a systematic way, and Tesseract is a pioneering 4D implementation. Tesseract provides many carefully designed features to achieve the goals we set in 4D, and demonstrates that, as we will show in this thesis, it is actually practical to build a

centralized network control system that is as responsive and robust as existing distributed network control frameworks, while offering many new network functionalities.

4D stands for four key network planes: the *Data, Discovery, Dissemination, and Decision*. In the traditional network architecture, a router generally has only two planes: the *data and control planes*. In contrast to the traditional network, where each individual router runs complicated routing decision logic, the 4D architecture lifts the decision logic from the individual routers' distributed control planes to create a logically centralized decision plane. Routers are significantly simplified; instead of making independent routing decisions, their job is to take instructions from the centralized decision plane and forward packets based on the centrally made decisions. The decision plane has a network-wide view of the topology and traffic, and exerts direct control over the operation of the data plane. No decision logic is hardwired in protocols distributed among the network elements. The output of the decision logic is communicated to routers by the dissemination plane. By pulling all of the decision logic out of the routers, 4D enables both simpler protocols and more sophisticated algorithms for driving the operation of the data plane. The 4D concept outlines a clean-slate approach to data-network control and management.

4D/Tesseract distinguishes itself from prior centralized network control systems in the following ways.

2.2.2.1 New Concept of Clean-Slate Design

4D advocates redesigning the control and management functions from the ground up. We believe that a clean-slate approach based on sound principles will, at the minimum, provide an alternative perspective and shed light on fundamental trade-offs in the design of network control and management functions. More strongly, we believe that such an approach is *necessary* to avoid perpetuating the substantial complexity of today's control plane. Fortunately, we can make significant, fundamental changes in the control and management of IP networks *without changing the format of the data packets*. This enables network evolution and provides a key lever for substantial innovation in the Internet architecture. A good example of this principle is the Ethernet technology, which has successfully evolved from a shared-medium network to a switched network with new control-plane protocols based on learning and spanning trees, all while leaving the packet format unchanged.

Rather than exploring incremental extensions to today's control and management planes, 4D proposes a *clean-slate* repartitioning of functionality. We believe that a green-field approach based on sound principles is necessary to avoid perpetuating the substantial complexity in today's design. We have developed the 4D

architecture as an *extreme design point* that completely separates the decision logic from the underlying protocols. We deliberately chose an extreme design as we believe that it crystallizes the issues, so that exploring the strengths and weaknesses of this architecture will lead to important network-level abstractions and a deeper understanding of the essential functionality needed in the underlying routers.

2.2.2.2 Sound Design Principles

The rich literature on the complexity of today's control and management planes has led us to the following three principles that we believe are essential to dividing the responsibility for controlling and managing a data network:

Network-level objectives: Each network should be configured via specification of the requirements and goals for its performance. Running a robust data network depends on satisfying objectives for performance, reliability, and policy that can (and should) be expressed separately from the network elements. For example, a traffic-engineering objective could be stated as "keep all links below 70% utilization, even under single-link failures." A reachability policy objective could be stated as "do not allow hosts in subnet B to access the accounting servers in subnet A." Today's networks require these goals to be expressed in low-level configuration commands on the individual routers, increasing the likelihood that the objectives are violated due to semantic mistakes in translating the network-level objectives into specific protocols and mechanisms.

Network-wide views: Our notion of a network-wide view is borrowed from the database community and means having assembled a coherent snapshot of the state of each network component. Timely, accurate, network-wide views of topology, traffic, and events are crucial for running a robust network. The network-wide view must accurately reflect the current state of the data plane, including information about each device, including its name, resource limitations, and physical attributes. Armed with the topology, traffic matrix, state, and inventory information about a network, algorithms can be written to compute how the network should meet its objectives. However, today's control plane was *not* designed to provide these network-wide views, forcing substantial retro-fitting to obtain them. Instead of adding measurement support to the system as an afterthought, we believe that providing the information necessary to construct a complete, consistent, network-wide view should be one of the primary functions of the routers and switches.

Direct control: Direct control means that the control and management system should have both the ability and the sole responsibility for setting all the state in the data plane that directs packet forwarding. The decision logic should not be hardwired in protocols distributed among routers. Rather, only the output

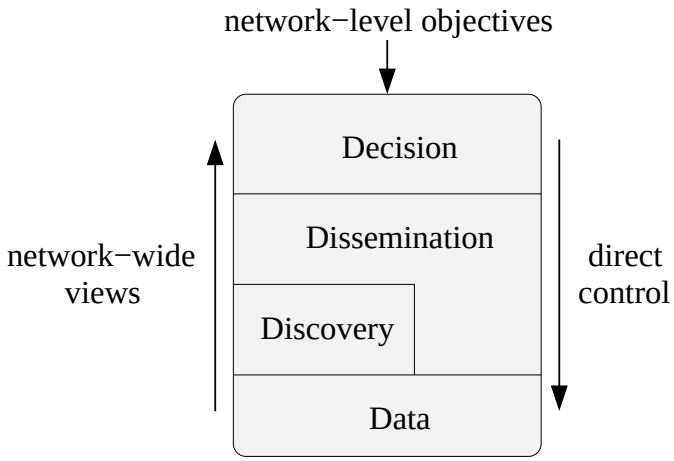

Figure 2.2: New 4D architecture with network-level objectives, network-wide views, and direct control

of the decision logic should be communicated to the network elements. Satisfying network-level objectives is much easier with direct control over the configuration of the data plane. IP and Ethernet originally embedded the path-computation logic in simple distributed protocols that incrementally grew more complicated, as discussed earlier. Because of the difficulty of extending the distributed control protocols to support sophisticated network-level objectives such as traffic engineering or reachability control, the management plane is typically used to implement these additional capabilities. With only indirect influence over the network, today's management plane must replicate the state and logic of the control plane and perform a complex "inversion" of the functionality. The problem would be much easier to solve if the management plane could compute the forwarding tables and install them in the routers. For direct control to be meaningful, it must be complete. If configuration commands or multiple entities can affect the state in the network elements, then yet more entities are required for auditing (and correcting) the settings [20, 21, 22] to ensure the network-level objectives are met.

In addition to these three principles, any design must also consider traditional systems requirements, such as scalability, reliability, and consistency. Our three principles attempt to capture the issues specific to the control and management of networks. By separating the network-specific issues from the traditional systems requirements, we can apply existing techniques from other areas of distributed computing research to the traditional systems problems while exposing for closer scrutiny the network-specific ones.

2.2.2.3 New Architecture

Although the three principles could be satisfied in many ways, we have deliberately made the 4D architecture an extreme design point where all control and management decisions are made in a logically centralized fashion by servers that have complete control over the network elements. The routers and switches only have the ability to run network discovery protocols and accept explicit instructions that control the behavior of the data plane, resulting in network devices that are auto-configurable. Our architecture has the following four components, as illustrated in Figure 2.2:

Decision plane: The decision plane makes *all* decisions driving network control, including reachability, load balancing, access control, security, and interface configuration. Replacing today's management plane, the decision plane operates in *real time* on a network-wide view of the topology, the traffic, and the capabilities and resource limitations of the routers/switches. The decision plane uses algorithms to turn network-level objectives (e.g., reachability matrix, load-balancing goals, and survivability requirements) directly into the packet-handling state that must be configured into the data plane (e.g., forwarding table entries, packet filters, queuing parameters). The algorithms in the decision plane may be customized based on knowledge of the network structure (e.g., a network with a ring topology can use a simpler path-computation algorithm than a network with a mesh topology). The decision plane consists of multiple servers called decision elements that connect directly to the network.

Dissemination plane: The dissemination plane provides a robust and efficient communication substrate that connects routers with decision elements. While control information may traverse the same set of physical links as the data packets, the dissemination paths are maintained separately from the data paths so they can be operational without requiring configuration or successful establishment of paths in the data plane. In contrast, in today's networks, control and management data are carried over the data paths, which need to be established by routing protocols before use. The dissemination plane moves management information created by the decision plane to the data plane, and state identified by the discovery plane to the decision plane, but does not create state itself.

Discovery plane: The discovery plane is responsible for discovering the physical components in the network and creating logical identifiers to represent them. The discovery plane defines the scope and persistence of the identifiers, and carries out the automatic discovery and management of the relationships between them. This includes box-level discovery (e.g., what interfaces are on this router? How many FIB entries can it hold?), neighbor discovery (e.g., what other routers does this interface connect to?), and discovery

of lower-layer link characteristics (e.g., what is the capacity of the interface?). The decision plane uses the information learned from the discovery plane to construct network-wide views. In contrast, in today's IP networks, the only automatic mechanism is neighbor discovery between two preconfigured and adjacent IP interfaces; physical device discovery and associations between entities are driven by configuration commands and external inventory databases.

Data plane: The data plane handles individual packets based on the state that is *output* by the decision plane. This state includes the forwarding table, packet filters, link-scheduling weights, and queue-management parameters, as well as tunnels and network address translation mappings. The data plane may also have fine-grain support for collecting measurements [23] on behalf of the discovery plane. Although the 4D architecture can be realized with the data plane available in today's networks, enhancements to the functionality in the data plane could help in simplifying the logic of the decision plane. For example, an integrated mechanism for forwarding, filtering, and transforming packets would simplify the problem of realizing a reachability matrix.

The 4D architecture embodies our three principles. The decision-plane logic operates on a network-wide view of the topology and traffic, with the help of the discovery plane in collecting the measurement data, to satisfy network-level objectives. The decision plane has direct control over the operation of the data plane, obviating the need to model and invert the actions of the control plane. Pulling much of the control state and logic out of the routers enables both simpler protocols that do not have to embed decision-making logic, and more powerful decision algorithms for implementing sophisticated goals.

2.2.2.4 Advantages

Our 4D architecture offers several important advantages over today's division of functionality:

Separates networking logic from distributed systems issues: The 4D architecture does not and cannot eliminate all distributed protocols, as networks fundamentally involve routers distributed in space. Rather, the 4D proposes separating the logic that controls the network, such as route computation, from the protocols that move information around the network. This separation creates an architectural force opposing the box-centric nature of protocol design and device configuration that causes so much complexity today. The 4D tries to find the interfaces and functionality we need to manage complexity—i.e., that factor out issues not unique to networking, and enable the use of existing distributed systems techniques and protocols to solve these problems.

Higher robustness: By simplifying the state and logic for network control, and ensuring the internal consistency of the state, our architecture greatly reduces the fragility of the network. The 4D architecture raises the level of abstraction for managing the network, allowing network administrators to focus on specifying network-level objectives rather than configuring specific protocols and mechanisms on individual routers and switches. Network-wide views provide a conceptually appealing way for people and systems to reason about the network without regard for complex protocol interactions among a group of routers/switches. Moving the state and logic out of the network elements also facilitates the creation of new, more sophisticated algorithms for computing the data-plane state that are easier to maintain and extend.

Improved scalability: The decision plane can introduce new levels of hierarchy that are not available in today's protocols. For example, the decision plane could incorporate structures such as Point-of-Presence, geographic region, and institution that drive so much of network design, without being constrained by protocol abstractions such as area or Autonomous System.

Better security: Security objectives inherently are network-level goals. For example, the decision plane can secure the network perimeter by installing packet filters on all border routers. Managing network-level objectives, rather than the configuration of individual routers, reduces the likelihood of configuration mistakes that can compromise security.

Accommodating heterogeneity: The same 4D architecture can be applied to different networking environments but with customized solutions. For example, in an ISP backbone with many optimization criteria and high reliability requirements, the decision plane may consist of several high-end servers deployed in geographically distributed locations. A data-center environment with Ethernet switches may require only a few inexpensive PCs, yet still achieve far more sophisticated capabilities (e.g., traffic engineering with resilience) than spanning tree or static VLAN configuration can provide today.

Network evolution. A thin control plane could migrate much of the software responsible for controlling a network to common server platforms. This provides a unique opportunity to revisit the design of network control software with a clean slate, without requiring incremental changes in the installed base of existing routers.

Enabling of innovation and network evolution: Separating the network control from the routers/switches and protocols is a significant enabler for innovation and network evolution. The decision plane can incorporate new algorithms and abstractions for computing the data-plane state to satisfy a variety of network-level objectives, *without* requiring a change in either *data* packet formats or *control protocols* (dissemination

and discovery plane protocols in the case of 4D). In addition, moving the control functionality out of the router software enables new players (e.g., the research community and third-party software developers) to contribute to the creation of these algorithms.

2.2.3 Side-by-Side Comparison

Tesseract is one member of the family of centralized network control systems. The Routing Control Platform (RCP) [10, 11] and Secure Architecture for the Networked Enterprise (SANE) [24] are the most notable examples of control systems that share conceptual elements with Tesseract. We next distinguish Tesseract from its peers listed in Table 2.1.

In contrast to traditional telecommunications networks such as SS7, Tesseract focuses on packet-switching data networks that have more complex data-plane primitives (e.g., packet forwarding based on longest-prefix matching, access control, NAT, and tunnels) and higher network dynamics. Unlike SS7, which uses separate management links or channels, the Tesseract architecture explores a softer logical separation appropriate for links such as Ethernet.

Compared to the existing centralized network management tools, in Tesseract the decision elements use network-wide views to manage all network state—it explicitly establishes the decision plane as the place in the architecture for coordinating *all* of the data-plane mechanisms and provides the decision plane with the information it needs to operate.

RCP focuses on the control of inter-domain routing, while Tesseract focuses on control and management within a single network. While RCP is designed to be backward compatible with BGP, Tesseract is designed as a clean-slate control plane. RCP only considers BGP routes—a single part of the total state used by the data-plane to direct packets through the network. Tesseract controls multiple data-plane forwarding mechanisms including packet-filters. RCP assumes routers are already correctly configured with significant amounts of state, such as IP addresses and an Interior Gateway Protocol (IGP). Tesseract addresses how *zero* pre-configuration of routers/switches can be achieved. Beyond considering only IP networks, Tesseract also addresses how a single management architecture could control different types of networks such as Ethernet.

SANE is a solution for enforcing security policies in an enterprise network. In a SANE network, communications between hosts are disabled unless they are explicitly allowed by the domain controller. To permit a data flow, the domain controller issues a certified secure source route to the end host. Switches

only forward packets that have authentic secure source routes attached to them. The domain controller in the SANE architecture has a role similar to the decision plane in 4D. For communications between switches and the domain controller, SANE constructs a spanning tree rooted at the domain controller in a distributed fashion similar to the IEEE 802.1D spanning tree. This spanning tree has a role similar to the dissemination plane in Tesseract.

Tempest [25] proposes an alternate framework for network control, where each switch is divided into switchlets and the functionality of each switch is exposed through a common interface called Ariel. Tempest allows multiple control planes to operate independently, each controlling its own virtual network composed of the switchlets, and the framework has been used on both MPLS and ATM data planes. Tesseract's dissemination plane provides a complete bootstrap solution, where Tempest's implementation assumed a pre-existing IP-over-ATM network for communication with remote switches. While both projects abstract switch functionality, Tesseract does not assume that switches can be fully virtualized into independent switchlets, but leaves resource allocation to the decision logic.

FIRE [26] presents a framework to ease the implementation of distributed routing protocols by providing a secure flooding mechanism for link-state data, as well as hooks to which route computation algorithms can be attached, and a separate FIB used for downloading code into the router. Tesseract eases the implementation of centralized network control algorithms by assembling a network-wide view, enabling direct control via a robust and self-bootstrapping dissemination plane, and providing redundancy through the election of the central control servers.

CONMan [27] reduces the need to configure network elements and end hosts by augmenting them with protocols so that they can negotiate configuration details among themselves under high-level direction from a network manager. CONMan also provides mechanisms by which the network manager can automatically choose the best high-level directions to give network elements after discovering the features the elements support. CONMan depends on a separate management communication channel between routers and network managers.

Significant prior work attempted to define an open router interface analogous to OS interfaces at end-systems [28, 29, 30]. Whereas these systems provide the elegant framework and modules needed to create easily extensible routers, Tesseract attempts to provide an elegant framework for an easily extensible and robust network.

Most centralized network control systems share the design of separating the computation of routes from the individual switches, or creating a minimal kernel of functionality implemented on each switch to be invoked from another location [31]. Tesseract presents a complete design that realizes these goals, with the added difference that it can automatically bootstrap itself without requiring a pre-configured lower-layer system to route control messages between switches. Tesseract also describes how functionality beyond routing, such as packet filters, should be controlled, and how to do this for both IP and Ethernet networks.

The idea of direct control has continued to advance since the Tesseract work. In Chapter 3, we will discuss open standards [32] and commercial products [33] that embrace the direct control concept.

2.3 Network Control Dissemination

The Tesseract system adopts an external control model where network control decisions are made by a controller remotely connected to network elements. Such systems critically depend on a robust, secure, and low latency communication channel between the external controller and the network elements.

Unfortunately, computer networks today lack an autonomic mechanism to support management plane communications, and the stopgap solutions used in practice vary widely. Many commercial networks still rely on dial-up modems to access the serial console ports of routers for control; this method has poor performance and is clearly not self-healing or self-optimizing. Alternatively, many networks rely on an orthogonal Ethernet network to access the special management Ethernet ports of routers for control; however, Ethernet is insecure, and not self-protecting or self-optimizing. Other networks even rely on in-band connectivity to control routers (i.e., control communication is mixed with user data communication and relies on the same IP routing tables); this method is dangerous, as it risks losing remote access with no recourse if the router is accidentally misconfigured.

Technologies such as MPLS [34] and GMPLS [35] use IP protocols such as OSPF and IS-IS to establish logically out-of-band communications paths for management. Using IP routing protocols to carry management traffic re-introduces the circular dependency problem which we will elaborate in Section 4.1.

Much work has been done to address the scalability problem of distributing contents from a small number of sources to a large number of recipients. Among these solutions, the best-known are reliable multicast [36] and peer-to-peer protocols such as BitTorrent [37]. Both multicast and peer-to-peer aim to solve

the problem where all recipients receive the same data; thus they are not directly applicable to our problem, where different recipients may need different data.

A variation of BitTorrent called SET (Similarity Enhanced Transfer) [38] allows similar objects to be utilized to speed up transfer. Both BitTorrent and SET use the pull model in which the data receivers try to identify usable data sources from which to download data. We instead use a push model to globally optimize the total dissemination cost in terms of both traffic and time. We will further compare SET with our dissemination approach in Chapter 5.

Another category of related work provides redundancy elimination services to reduce network traffic. It includes early LBFS (low-bandwidth network file system) work [39] as well as very recent EndRE (end-system redundancy elimination service) projects [40]. LBFS and EndRE both treat network traffic as byte streams and seek compression schemes that work for generic network traffic, whereas we focus on solving the problem of distributing a specific type of payload - routing tables. Instead of attempting to develop a new generic redundancy elimination service with a smart fingerprinting scheme, we let the application, in this case the network decision element, decide how to compute the difference across data sent to different receivers. As we show in Chapter 5, domain knowledge helps improve compression rate when the bytes that carry the payload cannot be effectively fingerprinted by generic schemes such as Robin fingerprinting. Another difference between EndRE and our effort is that EndRE provides an “intra-host” solution to eliminate redundancy between a client and a server, while we aim to solve an “inter-host” problem by reducing traffic load between a server and a large number of clients. We want to compare the performance of our system with other “inter-host” solutions when they become publicly available.

Chapter 3

Tesseract

Previous position papers [7, 8] have laid down the conceptual framework of 4D. This chapter answers the question of how to make 4D a flexible system that greatly simplifies network control by providing the details of an implementation and applications of the first 4D prototype, Tesseract. The target of Tesseract is to enable the *direct control* of a computer network that is under a single administrative domain. The term direct control refers to a network control paradigm in which a *decision element* directly and explicitly creates the forwarding state at the network nodes, rather than indirectly configuring other processes that then compute the forwarding state. This paradigm can significantly simplify network control.

In a typical IP network today, the desired control policy of an administrative domain is implemented via the synthesis of several indirect control mechanisms. For example, load balanced best-effort forwarding may be implemented by carefully tuning OSPF link weights to indirectly control the paths used for forwarding. Inter-domain routing policy may be indirectly implemented by setting OSPF link weights to change the local cost metric used in BGP calculations. The combination of such indirect mechanisms creates subtle dependencies. For instance, when OSPF link weights are changed to load balance the traffic in the network, inter-domain routing policy may be impacted. The outcome of the synthesis of indirect control mechanisms can be difficult to predict, and exacerbates the complexity of network control [41].

The direct control paradigm avoids these problems because it forces the dependencies between control policies to become explicit. In direct control, a logically centralized entity, called the decision element, is responsible for creating all the state at every router. As a result, any conflicts between the policy objectives can be detected at the time of state creation. With today's multiple independent and distributed mechanisms,

these conflicts often only appear *in vivo* after some part of the configuration state has been changed by one of the mechanisms.

The direct control paradigm also simplifies the router functionality. Because algorithms making control decisions are no longer run at routers, the only distributed functions to be implemented by routers are those that discover the neighborhood status at each router and those that enable the control communications between the decision element and the routers. Thus, the router software can be very light-weight. Yet sophisticated control algorithms can be easily implemented with this minimal set of distributed functions.

This chapter presents the design, implementation, evaluation, and demonstration of the Tesseract system. To guide our design, we explicitly select a set of goals and devise solutions to address them. We deploy Tesseract on Emulab [6] to evaluate its performance. We show how Tesseract can rapidly react to link, node, and decision element failures and efficiently re-configure network routers in response. Also, micro-benchmark experiments show that the system can easily handle the intra-domain routing control for a thousand-node network.

We demonstrate Tesseract's flexibility by showing its applications in joint packet forwarding and policy-based filtering in IP networks, and in link-cost-driven Ethernet packet forwarding. Both applications are simple to implement in the decision element of Tesseract despite the fact that the applications operate in different data plane layers.

3.1 From Architecture to System

Tesseract is based on the general 4D architectural concepts, but these concepts admit a wide variety of design choices. We used the following goals to guide our decisions while designing Tesseract, and these goals can be roughly grouped into three categories. The first category concerns objectives for system performance and robustness:

Timely reaction to network changes: Planned and unplanned network changes, such as switch maintenance and link failures, can cause traffic disruption. Tesseract should be optimized to react to network changes quickly and minimize traffic disruption.

Resilient to decision plane failure: Tesseract should provide built-in support for decision plane redundancy so that it can survive the failure of a decision element.

Robust and secure control channels: The logical channels for control communications maintained by Tesseract should continue to function in the presence of compromised switches, decision elements, or failed links/nodes.

The next set of goals concerns making Tesseract easy to deploy:

Minimal switch configuration: The Tesseract software on each switch should require no manual configuration prior to deployment except for security keys that identify the switch. We do, however, assume that the underlying switch allows Tesseract to discover the switch's properties at run-time.

Backward compatibility: Tesseract should require no changes to the end host software, hardware, or protocols. Thus, Tesseract can be deployed as the network control system transparently to the end users.

The final set of goals concerns making Tesseract a flexible platform:

Support diverse decision algorithms: Tesseract should provide a friendly platform on which diverse algorithms can be easily implemented to control networks.

Support multiple data planes: Tesseract should support heterogeneous data plane protocols (e.g., IP or Ethernet). Thus, the system should not assume particular data plane protocols, and the dissemination service should be agnostic to the semantics of the control communications.

3.2 Design and Implementation of Tesseract

In this section, we present the design and implementation of Tesseract. We first provide an overview of the software architecture, and then discuss each component of the system in detail.

3.2.1 System Overview

The Tesseract system is composed of two applications implemented on Linux. These applications are called the `Switch` and the Decision Element (`DE`). Figure 3.1 illustrates the software organization of these applications.

The discovery plane implementation currently deals only with neighbor node discovery. It includes two modules, one for discovering hosts connected to the switch and the other for discovering other switches. The switch discovery module exchanges hello messages with neighbor switches to detect them, and creates

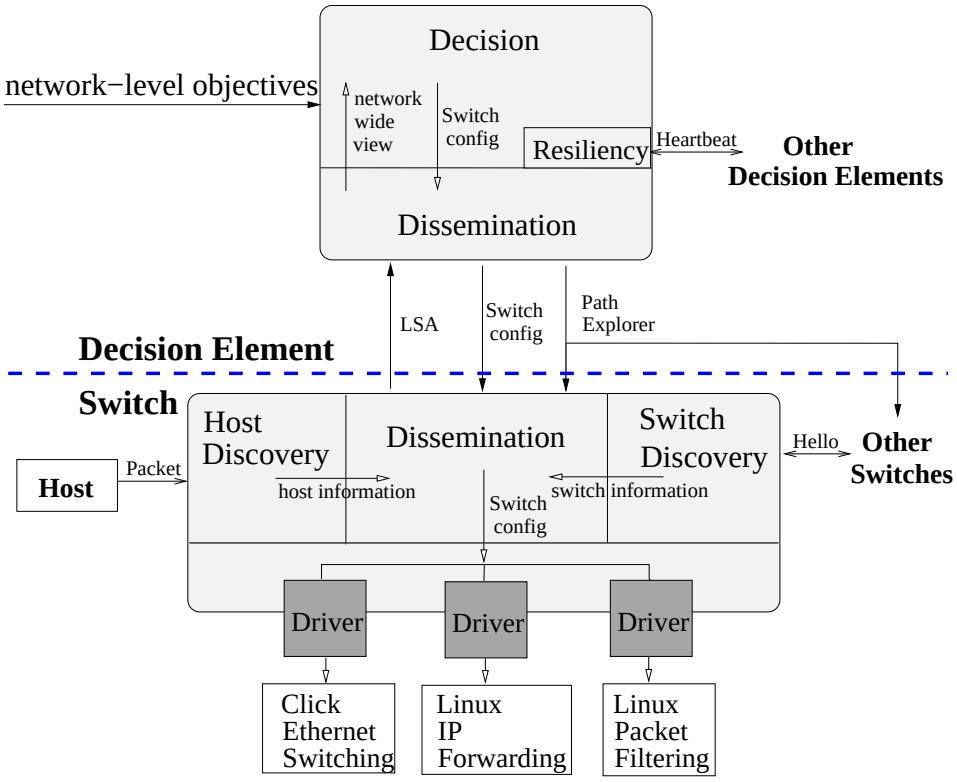

Figure 3.1: High-level overview of Tesseract.

Link State Advertisements (LSAs) that contain the status of its interfaces and the identities of the switches connected to the interfaces. The generated LSAs are reported to `DE` via the dissemination plane. To avoid requiring changes to hosts, the discovery plane identifies what hosts are connected to a switch by snooping the MAC and IP addresses on packets received on the interfaces that are not connected to another switch.

The dissemination plane is cooperatively implemented by both `Switch` and `DE`. The dissemination service is realized by a distributed protocol that maintains robust logical communication channels between the switches and decision elements.

`Switch` leverages existing packet forwarding and filtering components to implement the data plane. `Switch` interacts with `DE` in the decision plane through the node configuration service interface. The interface is implemented by data plane drivers, which translate generic configuration commands from `DE` into specific configurations for the packet forwarding and filtering components.

`DE` implements the discovery, dissemination and decision planes. The discovery and dissemination plane functions are as outlined above. The decision plane constructs an abstract network model from the information reported by the switches and computes switch configuration commands for all the switches

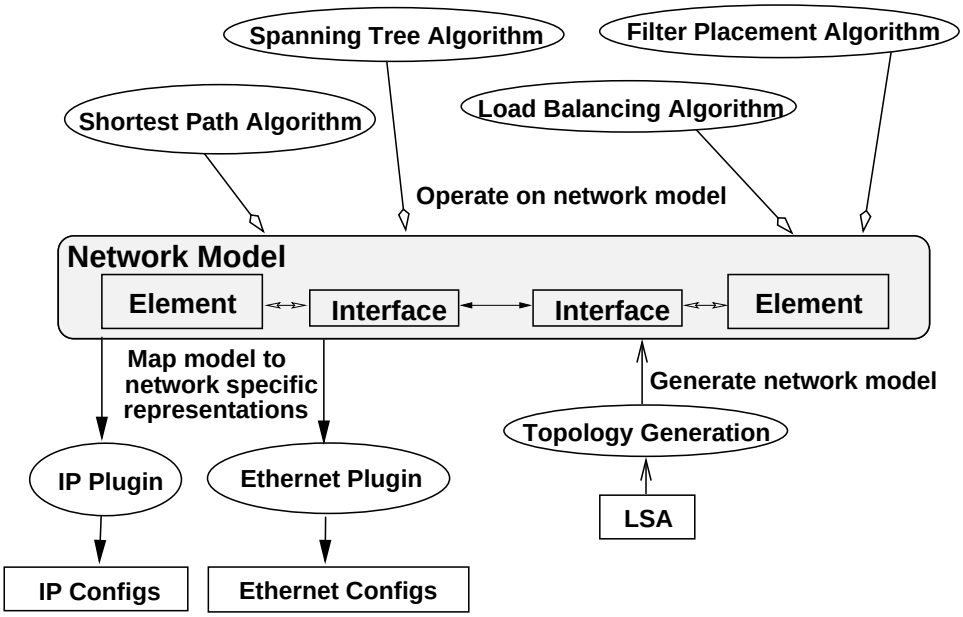

Figure 3.2: The network model separates general-purpose algorithms from network-specific mechanisms. based on the specific decision algorithm used. The computed switch configuration commands are sent to the switches via the dissemination service.

3.2.2 Decision Plane: Versatility, Efficiency and Survivability

The decision plane implements a platform for the deployment of network control algorithms. In addition, it implements mechanisms that enable the replication of the decision logic among multiple decision elements (DEs) so that DE failures can be tolerated.

Support diverse network control algorithms: In designing the decision plane, our focus is not to hard-wire sophisticated network decision logics into the system. Instead, our goal is to make the decision plane a friendly platform where any network control algorithm can be easily integrated and used to control any suitable network technology. Toward this end, we introduce an abstract network model to separate generic network control algorithms (e.g., shortest path computation, load balancing) from network-specific mechanisms (e.g., IP, Ethernet).

Figure 3.2 illustrates the abstract network model. The model consists of node element and link interface objects, and is constructed from information discovered and reported by switches (e.g., LSA) through the dissemination service. Operating on this model, Tesseract currently implements four generic algorithms: incremental shortest path, spanning tree, joint packet filter/routing (Section 3.4.1), and link-cost-based traffic

engineering (Section 3.4.2). Finally, technology-specific plug-ins translate the general control decisions into network-specific configuration commands that are sent to switches via the dissemination service. These commands are then processed by the node configuration service at individual switches.

As an example, we implement an incremental shortest path algorithm [42] on the abstract network model, and the same code can be used to generate either IP routing table in IP networks or Ethernet forwarding entries in Ethernet.

Efficient network event processing: The DE must efficiently handle multiple simultaneous network changes, which the DE will receive as events communicated over the dissemination plane. We chose a different event processing architecture than that used in typical implementation of OSPF, where a hold-down timer is used to delay the start of route recomputation after an event arrives to force the batching of whatever events arrive during the hold-down window.

Instead, the Tesseract DE uses a *push timer*. The DE runs a decision thread that processes all queued events to update the network-wide view, starts the push timer as a deadline for pushing out new switch configuration commands, and then enters its computation cycle. After the computation of new forwarding state finishes, the DE will immediately push out the new commands if the push timer has expired, if the event queue is empty, or if the queued events do not change the network-wide view used in the computation. Otherwise, the DE will dequeue all pending events and re-compute.

We use a push timer instead of a fixed hold-down timer for two reasons. In the common case where a single link fails, the push timer avoids unnecessary waiting. The first LSA announcing the failure starts the route recomputation, and subsequent LSAs announcing the same failure do not change the network-wide view and so are ignored. In the less common case of multiple failures, a push timer may result in recomputation running more than once for the same event. However, since recomputation has latency on the same order as typical hold-down timers, and DEs are unlikely to be CPU-limited, it is reasonable to trade extra computation for faster reconvergence.

The DE also records the state that has been pushed to each switch and uses delta-encoding techniques to reduce the bandwidth required for sending configuration commands to the switches. Acknowledgments between DE and the node configuration service on each switch ensure the delta-encoded commands are received.

Provide decision plane resiliency: Our decision plane copes with DE failures using hot-standbys. At any time a single master DE takes responsibility for configuring the network switches, but multiple DEs

can be connected to the network. Each standby DE receives the same information from the switches and performs the same computations as the master. However, the standby DEs do not send out the results of their computations.

The master DE is selected using a simple leader election protocol based on periodic DE heartbeats that carry totally ordered DE priorities. Each DE has a unique priority, and at boot time it begins flooding its priority with a heartbeat message every heartbeat period (e.g., 20 ms). Each DE listens for heartbeats from other DEs for at least five times the heartbeat period (we assume that 5 times heartbeat period will be greater than the maximum latency of a packet crossing the network). After this waiting period, the DE that has the highest priority among all received heartbeats decides to be the master and begins sending commands to switches. When the master DE receives a heartbeat from a DE with a higher priority than its own, it immediately changes into a standby DE and ceases sending commands to switches. A DE also periodically floods a path explorer message, which has the effect of triggering switches to reply with their current state. In this way, a new DE can gather the latest switch state. Switches simply process commands from any DE. Authentication is handled by the dissemination plane and is discussed next.

3.2.3 Dissemination Plane: Robustness and Security

The goal of the dissemination plane is to maintain robust and secure communication channels between each DE and the switches. With respect to robustness, the dissemination plane should remain operational under link and node failure scenarios. With respect to security, the network should remain operational when a switch or even a DE is compromised.

Observing that the traffic pattern in dissemination plane is few-to-many (switches communicate not with each other, but only with the DEs), we adopt an asymmetric design where the dissemination module at a DE node implements more functionality than the dissemination module at a switch.

Dissemination plane design overview: Tesseract's dissemination plane is implemented using source routes. Each control message is segmented into dissemination frames, and each frame carries in its header the identity of the source, destination, and the series of switches through which it must pass. We choose a source routing solution because: (1) It requires the minimal amount of routing state and functionality in each switch. Each switch needs only to maintain the routes to the DEs. (2) Source routes provide very flexible control over routing, as a different path can be specified for each destination, making it easy to take

advantage of preferred paths suggested by the decision plane. (3) Combining source routing with the few-to-many communication pattern enables us to design a dissemination plane with desirable security properties, as discussed below. To protect control communications from user data traffic, the queuing of dissemination frames is separate from user data traffic, and dissemination frames have higher transmission priority. To protect the source routes from being misused by adversaries inside the network, we encrypt them at each hop before they are forwarded.

Threat model: Tesseract is designed to cope with the following threats: (1) Adversaries can compromise a switch, gaining full control over It, including the ability to change the way dissemination packets are forwarded. (2) A compromised switch can piggyback data on packets to collude with other compromised switches downstream. (3) A compromised switch can peek into dissemination plane data to try to learn the network topology or location of critical resources. (4) Adversaries can compromise a DE and use it to install bad forwarding state on the switches.

Bootstrapping security: The Tesseract trust model is based on a *network certificate* (i.e., a signed public key for the network) — all the other keys and certificates are derived from the network certificate and can be replaced while the network continues operating. Switches will accept commands from any DE holding a DE certificate that is signed by the network certificate. The private key of the network certificate is secret-shared [43] among the DEs, so that any quorum of DEs can cooperate to generate a new DE certificate when needed.

When a switch is first deployed, the network certificate and a DE certificate are installed into it. This is done by plugging a USB key containing the certificates into each switch, or as part of the default factory configuration of the switch before it is deployed in the field. The switch then constructs a DeviceID, which can be as simple as a randomly generated 128-bit number, and a private/public key pair. The switch stores the network and DE certificates, its DeviceID, and its key pair into nonvolatile memory. The switch then encrypts the information with the public key of the DE, and writes it back onto the USB key. When the USB key is eventually inserted into a DE, the DE will have a secret channel to each device and a list of the valid DeviceIDs. As each switch communicates with a DE for the first time, it uses ISAKMP [44] and its private/public keys to establish a shared-secret key known only by that switch and the DE. All subsequent dissemination plane operations use symmetric cryptography.

Computing dissemination plane routes: Dissemination plane routes are computed by each decision element flooding a path explorer message through the network. To ensure fast recovery from link failures, the

path explorer is sent periodically every 20 ms in our prototype, and can be triggered by topology updates.

Onion-encryption (or encapsulated encryption) is used in path explorers to support dissemination security. The DE initiates the path explorer by embedding its DeviceID as the source route and flooding it over all its ports. When a switch receives the path explorer, it (1) optionally verifies the route to the DE contained in the path explorer; (2) records the source route; (3) encrypts the existing source route using the secret key it shares with the DE that sent the path explorer; (4) appends its own DeviceID to the path explorer in plain text; and (5) floods the path explorer out to its other interfaces. Path explorers carry sequence numbers so that switches can avoid unnecessary re-flooding.

To send data to a DE, a switch uses the encrypted source route it recorded from a path explorer sent by that DE. When an upstream switch receives the message, it decrypts the source route using its secret key. This reveals the ID of the next-hop switch along the path to the DE. By successive decryption of the source route by the on-route switches, dissemination plane packets are delivered to the DE. Since the DE knows the secret key of every switch, it can construct an onion-encrypted route to any switch it desires.

As part of the negotiation of its secret key over ISAKMP, each switch learns whether it is required to perform the optional source route verification in step (1) before forwarding a path explorer. If verification is required, the switch first checks a cache of source routes from that DE to see if the source route has already been verified. If the source route is not known to be valid, the switch forwards the source route to the DE in a signed VERIFY packet. Since the DE knows the secret keys of all the switches, it can iteratively decrypt the source route and verify that each hop corresponds to the link it has learned about in an LSA. Once verified, the DE sends a VERIFYOK message to the switch using the extracted source route, confirming the validity of the route. The DE confirmation is signed with an HMAC computed using the secret key of the destination switch to prevent it from being tampered or forged.

Security properties: The optional verification step exposes a classic trade-off between security and performance. In Tesseract, we provide a dissemination plane with two different levels of security. The network operator can choose the semantics desired.

The basic security property is that a compromised switch cannot order other switches to install invalid forwarding state or forge LSAs from other switches. This is achieved by each switch having a secret key shared only with the DE.

If path explorers are *not* verified before being forwarded, a compromised switch can forge path explorers that artificially shorten its distance to the DE and attract dissemination plane traffic from other switches (e.g.,

so the attacker can drop or delay the traffic). Compromised switches can also communicate with each other over the dissemination plane to coordinate attacks.

If path explorers *are* verified before being forwarded, a compromised switch cannot lie about its distance to the DE. Also, compromised switches are prevented from communicating arbitrarily over the dissemination plane unless they are directly connected. This is because the DE will not validate a source route that originates and ends at switches. A switch also cannot discover the identity or connectivity of another switch that is two or more hops away. This prevents attackers from identifying and targeting critical resources in the network.

The cost of the extra security benefits provided by verifying source routes is the extra latency during reconvergence of the dissemination plane. If a link breaks and a switch receives path explorers over a source route it has not previously verified, it must wait a round-trip time for the verification to succeed before the switches downstream can learn of the new route to the DE. One approach to minimize this penalty is for the DE to pre-populate the verified source route tables of switches with the routes that are most likely to be use in failure scenarios. A triggered path explorer flooded by the DE in response to link failure will then quickly inform each switch which preverified routes are currently working.

Surviving DE compromise: As a logically centralized system, if a DE were compromised, it could order switches to install bad forwarding state and wreak havoc on the data plane. However, recovery is still possible. Other DEs can query the forwarding state installed at each switch and compare it to the forwarding state they would have installed, allowing a compromised or misbehaving DE to be identified. Because the private key of the network certificate is secret-shared, as long as a quorum of DEs remain uncompromised they can generate a new DE certificate and use the dissemination plane to remotely re-key the switches with this new DE certificate.

Notice that while a compromised DE can Completely disrupt data plane traffic, it *cannot* disrupt the dissemination traffic between other DEs and the switches. This is one of the benefits of having control traffic traverse a secured dissemination plane that is logically separate from paths traversed by data packets. Once re-keyed, the switches will ignore the compromised DEs.

As a point of comparison, in today's data networks recovering from the compromise of a management station is difficult, as the compromised station can block the uncompromised ones from reaching the switches. At the level of the control plane, the security of OSPF today is based on a single secret key stored in plain text in the configuration file. If any switch is compromised, the key is compromised, and incorrect

LSAs can be flooded through the network. The attacker could then DoS all the switches by forcing them to continually rerun shortest path computation or draw traffic to itself by forging LSAs. Since a distributed link-state computation depends on all-to-all communications among the switches, one alternative to using a single shared key is for each switch to negotiate a secret key with every other switch. Establishing this $O(n^2)$ mesh of keys requires every switch to know the public key of every other switch. Both key establishment and revocation are more complex when compared to the direct control paradigm of Tesseract.

3.2.4 Discovery Plane: Minimizing Manual Configurations

The discovery plane supports three categories of activities: (1) providing the DE with information on the state of the network; (2) interacting with external networks and informing the DE of the external world; and (3) bootstrapping end hosts into the network.

Gathering local information: Since misconfiguration is the source of many network outages, the 4D architecture eliminates as much manually configured state as possible. In the long term vision, the switch hardware should self-describe its capabilities and provide run-time information such as traffic load to the discovery plane. The current Tesseract implementation supports the discovery of physical switch neighbors via periodic HELLO message exchanges. Switches are identified by the same DeviceID used in the dissemination plane.

Interacting with external networks: The DE directs the border switches that peer with neighbor networks to begin eBGP sessions with the neighbor switches. Through this peering, the DE discovers the destinations available via the external networks. Rather than processing the BGP updates at the switches, the switches simply report them to the DE via the dissemination service, and the DE implements the decision logic for external route selection. The DE sends the appropriate eBGP replies to the border switches, as well as configuring external routes directly into all the switches via the dissemination service. RCP [10] has already demonstrated that the overall approach of centralized BGP computation is feasible, although they continue to use iBGP for backward compatibility with existing routers.

It is important to note that an internal link or switch failure in a Tesseract network does not lead to massive updates of external routes being transmitted from the DE to the switches. The reason is that external routes identify only the egress points. External and internal routes are maintained in two separate tables and are combined locally at switches to generate the full routing table. This is identical to how OSPF and BGP

computed routes are combined today. In general, an internal link or switch failure does not change external routes and thus no update to them is necessary.

Bootstrapping end hosts: For backward compatibility, end hosts do not directly participate in Tesseract discovery plane.

In networks running IP, the discovery plane acts as a DHCP proxy. The DE configures each switch to tunnel DHCP requests to it via the dissemination service. Whenever a host transmits a DHCP request, the DE learns the MAC address and the connection point of the host in the network. The DE can then assign the appropriate IP address and other configuration to the host.

In networks operating as a switched Ethernet LAN, the discovery plane of a switch reports the MAC address and the connection point of a newly appeared end host to the DE. The DE then configures the network switches appropriately to support the new host. Section 3.4.2 describes how we use Tesseract to control a switched Ethernet LAN and provide enhancements.

3.2.5 Data Plane: Support Heterogeneity

The data plane is configured by the decision plane via the node configuration service exposed by the switches. Tesseract abstracts the state in the data plane of a switch as a lookup table. The lookup table abstraction is quite general and can support multiple technologies such as the forwarding of IPv4, IPv6, or Ethernet packets, or the tunneling and filtering of packets, etc.

Tesseract's data plane is implemented using existing Linux kernel and Click components. For each component, we provide a driver to interface the component with the Tesseract decision plane as shown in Figure 3.1. The drivers model the components as lookup tables and expose a simple `WriteTable` interface to provide the node configuration service to the DE. For example, when the DE decides to add or delete an IP routing or Ethernet forwarding table entry, it sends an `add_table_entry` or `delete_table_entry` command through the `WriteTable` interface, and the driver is responsible for translating the command into component-specific configurations. This allows the algorithms plugged into the DE to implement network control logic without dealing with the details of each data-plane component. We implemented three drivers and describe their details next.

Linux IP forwarding kernel: The Linux kernel can forward packets received from one network interface to another. To determine the outgoing network interface, the Linux kernel uses two data structures: a Forwarding Information Base (FIB) that stores all routes, and a routing cache that speeds up route search. As in

all Tesseract data plane drivers, the driver for the Linux IP forwarding kernel implements the `WriteTable` interface. The driver interprets commands from the DE, creates a `rtentry` structure with the route to add or delete, and invokes the `ioctl` system call to modify the FIB. To make sure that the routing cache is flushed immediately after the FIB is modified, we set `proc/sys/net/ipv4/route/min_delay` to zero.

Click router: We use Click for forwarding Ethernet frames. The driver for Click includes two parts: an implementation of the `WriteTable` interface, and a Click element package called the `4DSwitch` that is integrated into Click. The implementation of `WriteTable` parses commands and executes those commands by exchanging control messages with the 4DSwitch element in the Click process via a TCP channel. The `4DSwitch` element maintains an Ethernet forwarding table and updates the table according to the received control messages. To control the data forwarding behavior of Click, the `4DSwitch` element overrides the Click `Element::push` function and directs incoming traffic to the outgoing port(s) specified in the `4DSwitch` forwarding table.

netfilter/iptables: Tesseract uses netfilter/iptables to implement reachability control in IP networks. The driver for netfilter/iptables translates commands into iptables rules (e.g., `-A FORWARD -s 10.1.1.0/24 -d 10.1.2.0/24 -i eth0 -j DROP`) and forks an iptables process to install the rules.

3.2.6 Decision/Dissemination Interface

In designing the interface between the decision plane and the dissemination plane, there is a tension between the conflicting goals of creating a clean abstraction with rigid separation of functionality, and the goal of achieving high performance with the cooperation of the decision and dissemination planes.

The key consideration is that the dissemination plane must be able to function independently of the decision plane. Our solution is to build into the dissemination plane a completely self-contained mechanism for maintaining connectivity. This makes the dissemination plane API very simple, giving the basic decision plane only three interface functions: `Send(buf,dst)`, which sends control information to a specific switch, `Flood(buf)`, which floods control information to all switches, and `RegisterUpCall(*func())`, which identifies the decision plane function that handles incoming information.

However, to optimize the performance of the dissemination plane, we add two interface functions: `LinkFailure(link)`, which the DE uses to identify a known failed link to the dissemination plane

so the dissemination plane can avoid it immediately, and `PreferredRoute(dst,sourceRoute)`, which the DE uses to suggest a specific source route for carrying control information to switch `dst`. This solution enables a sophisticated DE to optimize the dissemination plane to its liking, but also allows the simplest DE to fully function.

3.3 Performance Evaluation

In this section, we evaluate Tesseract to answer the following questions: How fast does a Tesseract-controlled network converge upon various network failures? How large a network can Tesseract scale to, and what are the bottlenecks? How resilient is Tesseract in the presence of decision-element failures?

3.3.1 Methodology

We perform both emulation and simulation experiments. We use Emulab to conduct intra-domain routing experiments using two different topologies. The first topology is an ISP backbone network (AS 3967) from Rocketfuel [45] data that spans Japan, U.S., and Europe, with a maximum round-trip delay of 250 ms. The other is a typical enterprise network with negligible propagation delay from our earlier study [2].

that have more than 4 interfaces are modeled by chaining together PCs to create a "supernode" (e.g., a router with 8 interfaces will be represented by a string of 3 Emulab PCs). As a result, the backbone network is emulated by 114 PCs with 190 links, and the enterprise network is emulated by 40 PCs with 60 links. For each Tesseract experiment, there are 5 decision elements — these run on "pc3000" machines that have a 3GHZ CPU and 2GB of RAM. To inject a link failure, we bring down the interface with the `ifconfig down` command. To inject a switch failure, we abruptly terminate all the relevant software running on a switch.

To ensure that we evaluate the worst-case behavior of the control plane, we measure the time required for the *entire* network to reconverge after an event. We calculate this network convergence time as the elapsed time between the event occurring and the last forwarding state update being applied at the last switch to be updated. We use Emulab's NTP (Network Time Protocol) servers to synchronize the clocks of all the nodes to within 1 millisecond.

As a point for comparison, we present the performance of an *aggressively tuned* OSPF control plane called Fast OSPF. Fast OSPF's convergence time represents the best possible performance achievable by

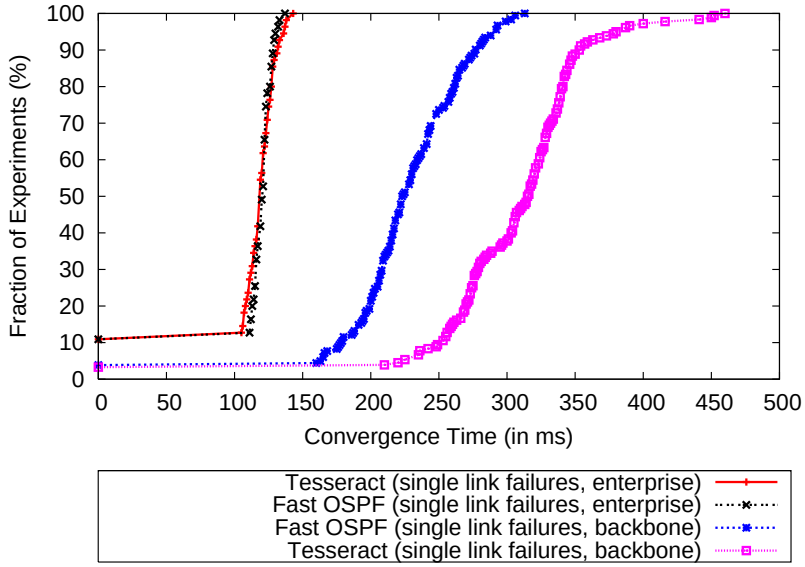

Figure 3.3: CDF of convergence times for single link failures in enterprise and backbone networks. We pick one link to fail at a time and we enumerate all the links to get the distribution of convergence times. The zero convergence times are caused by failures disconnecting switches at the edge of the network.

OSPF, and is determined by the time to detect a link failure and the one-way propagation delay required for the LSA flood. Such uniform and aggressive tuning might not be practical in a real network, as it could lead to CPU overload on older routers, but Fast OSPF serves as a useful benchmark.

We implemented Fast OSPF by modifying Quagga 0.99.4 [46] to support millisecond timer intervals. There are four relevant timers in Quagga: (1) the hello timer that sets the frequency of HELLO messages; (2) the dead timer that sets how long after the last HELLO is received the link is declared dead; (3) the delay timer that sets the minimum delay between receiving an LSA update and beginning routing computation; and (4) the hold-down timer that sets the minimum interval between successive routing computations. For Fast OSPF, we use hello timer = 20 ms, dead timer = 100 ms, delay timer = 10 ms (to ensure a received LSA is flooded before routing computation begins), and 0 ms for the hold-down timer. Tesseract uses the same hello and dead timer values to make direct comparison possible. There is no need for the delay timer or the hold-down timer in Tesseract.

3.3.2 Routing Convergence

Common concerns with using a logically centralized DE to provide direct control are that reconvergence time will suffer, or that the DE will attempt to control the network using an out-of-date network view. To

evaluate these issues, we measure intra-domain routing convergence after single link failures, single switch failures, regional failures (i.e., simultaneous multiple switch failures in a geographic region), and single link flapping.

Single link failures: Figure 3.3 shows the cumulative distribution of convergence times of Tesseract and Fast OSPF for all single link failures in both topologies (some convergence times are 0 because the link failure partitioned a stub switch and no forwarding state updates were required). Even though Tesseract uses a single DE machine to compute all the routes, its performance is nearly identical to that of Fast OSPF, thanks to the use of an efficient dynamic shortest path algorithm and the delta encoding of switch configurations. The only observable difference is that Tesseract's convergence time has a slightly larger variance due to the variability of the dynamic shortest path algorithm on different failed links.

In the backbone network scenario, propagation delay becomes an important factor as switch-to-switch RTT ranges from 1 ms to 250 ms. Tesseract's convergence requires the link state update to be transmitted to the DE, and the new switch configurations to be transmitted back to the switches. On the other hand, Fast OSPF only requires one-way flooding of the link state update. This is why Tesseract's convergence time is roughly a one-way delay slower than Fast OSPF. However, in return, the direct control paradigm enabled by Tesseract allows other control functions such as packet filtering to be implemented together with intra-domain routing in a simple and consistent manner.

Switch failures and regional failures: Next, we examine the convergence time under single switch failures and regional failures. To emulate regional failures, we divide the backbone topology into 27 geographic regions with each region containing a mean of 7 and a maximum of 26 switches, and we simultaneously fail all switches in a region.

Figure 3.4 compares the cumulative distributions of convergence times of Tesseract and Fast OSPF on switch and regional failures. In the enterprise network, again, the performance of Tesseract is very similar to that of Fast OSPF. In the backbone network, the difference between Tesseract and Fast OSPF is still dominated by network delay, and both are able to gracefully handle bursts of network state changes. There are two additional points to make. First, Fast OSPF has more cases where the convergence time is zero. This is because the 10 ms delay timer in Fast OSPF is acting as a hold-down timer. As a result, Fast OSPF does not react immediately to individual link state updates for a completely failed switch, and sometimes this can avoid unnecessary configuration changes. In Tesseract, there is no hold-down timer, so it reacts to some link state updates that are ultimately inconsequential. Second, in some cases Tesseract has faster convergence

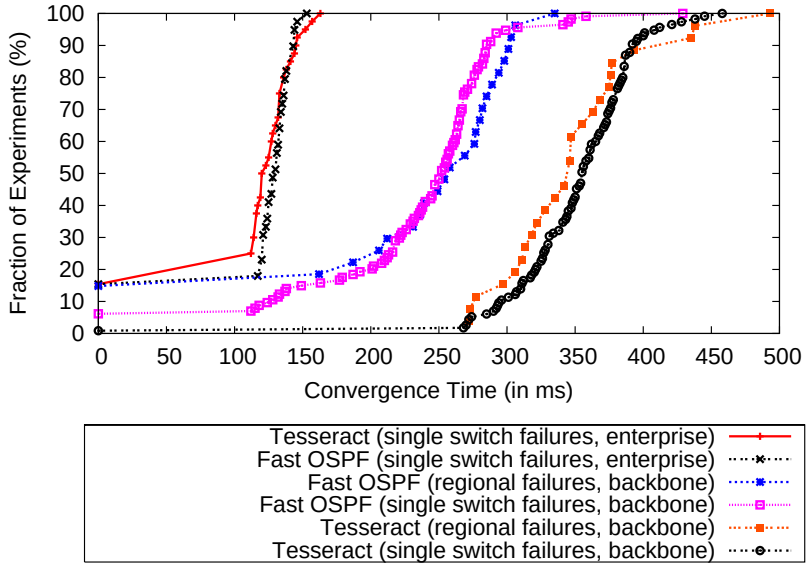

Figure 3.4: CDF of convergence times for single switch failures and regional failures.

time in regional failure than in single switch failure. The reason is that the large number of failed switches in regional failure reduces the number of configuration updates Tesseract needs to send.

Link flapping: From the earliest days of routing in the Internet there has been concern that a rapidly flapping link could overload the control plane and cause a widespread outage worse than the failure of that single link. Using Emulab we conduct an experiment to show the effects of link flapping on the end-to-end behavior of Tesseract. On the emulated backbone network, we `ping` the Tokyo node from the Amsterdam node at an interval of 10 ms and measure the RTT. We start to flap the link between Santa Clara and Herndon 2 seconds into the experiment. The flapping link is up for 100 ms and then down for 2 seconds. As the link flaps, the route from Tokyo to Amsterdam oscillates between a 10-hop path traversing Santa Clara, Herndon, Weehawken, and London with an average RTT of 240 ms, and a 12-hop path through San Jose and Oak Brook with an average RTT of 246 ms, as shown in Figure 3.5.

This experiment demonstrates that a logically centralized system such as Tesseract can handle continual network changes. It is also worth mentioning that the Tesseract decision plane makes it easy to plug-in damping algorithms in order to handle this situation in a more intelligent way.

3.3.3 Scaling Properties

An additional concern with a logically centralized system such as Tesseract is whether it can scale to the size of today's networks, which often contain more than 1,000 switches. Since Emulab experiments are limited

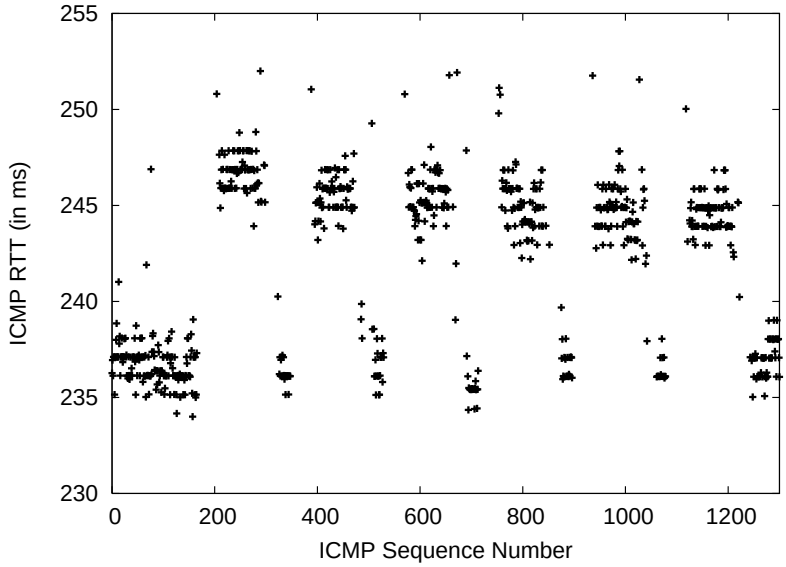

Figure 3.5: Effects of link flapping on ICMP packets sent at a rate of 100 packets/sec.

to at most a few hundred switches, we perform several simulation experiments to evaluate Tesseract's scaling properties. This evaluation uses a DE running the same code and hardware as the previous evaluations, but its dissemination plane is connected to another machine that simulates the control plane of the network.

We evaluate Tesseract's scalability on a set of Rocketfuel topologies of varying sizes. For each topology, we independently fail each link in the graph and measure the time for the DE to compute new forwarding state and the size of the state updates.

DE Computation Time: Every time a failure occurs in the network, the decision element needs to recompute the forwarding tables for the switches based on the new state of the network. Figure 3.6 shows the results of DE path computation time. As shown in the figure, even in the largest network of 1347 nodes and 6244 edges, the worst-case recomputation time is 151 ms and the 99th percentile is 40 ms.

Bandwidth Overhead of Control Packets: Each time the DE computes new forwarding state for a switch, it needs to push out the new state to the switch. Figure 3.7 plots the number of control bytes that the DE pushes out for independent link failures with different topologies. As shown in the figure, the worst-case bandwidth overhead is 4.4MB in the largest network of 1347 nodes and 6244 edges. This is a scenario where 90% of the switches must be updated with new state.

Notice that the bandwidth overhead reported here includes only intra-domain routes. Even when a Tesseract network carries external BGP routes, the amount of forwarding state expected to change in response to an internal link failure will be roughly the same. Switches use two-level routing tables, so even

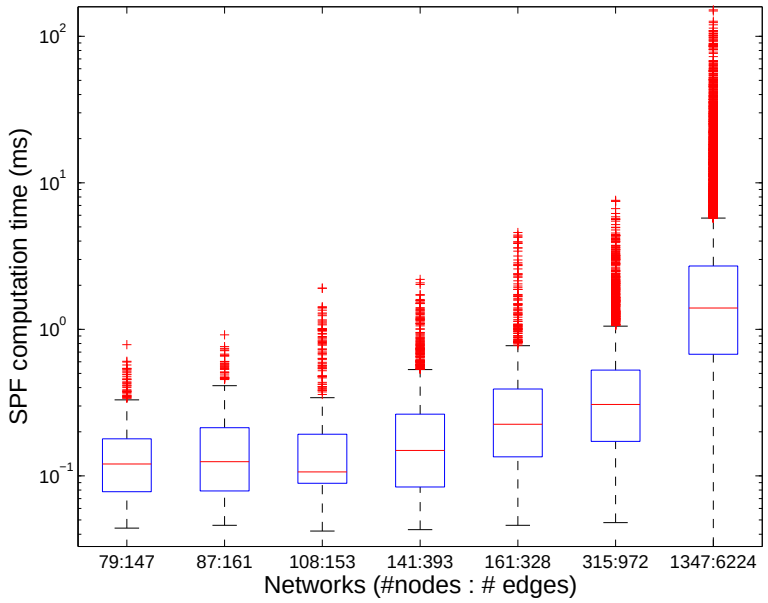

Figure 3.6: CPU time for computing incremental shortest paths for various Rocketfuel topologies in logarithmic scale. The box shows the lower quartile, upper quartile, and median. The whiskers show the min and max data values, out to 1.5 times the interquartile range, and outliers are plotted as '+'s.

if default-free BGP routing tables are in use, the BGP routes only change when the egress point for traffic changes — not when internal links fail. As has been pointed out by many [47, 10], Internet routing stability would improve if networks did not change egress points solely because the local cost changed, and Tesseract's framework for direct control makes it easier to implement this logic.

3.3.4 Response to DE Failure and Partition

This section evaluates decision plane resiliency by measuring the *DE failover time*, defined as the time from when the master DE is partitioned to when a standby DE takes over and becomes the new master DE. We use the backbone network topology and perform 10 experiments in which the master and stand-by DEs are 50 ms apart.

DE failure: Failure of any DE but the master DE is harmless, since in Tesseract the other DEs are hot standbys. To evaluate the effect of the failure of the master DE, we abruptly shut down the master DE. Table 3.1 shows the time required for a new DE to take control of the network after the master DE fails. As expected, the average failover time is approximately 140 ms, which can be derived from a simple equation that describes the expected failover time: ($DEDeadTime + PropagationDelay - HeartbeatInterval/2 = 100ms + 50ms - 10ms$).

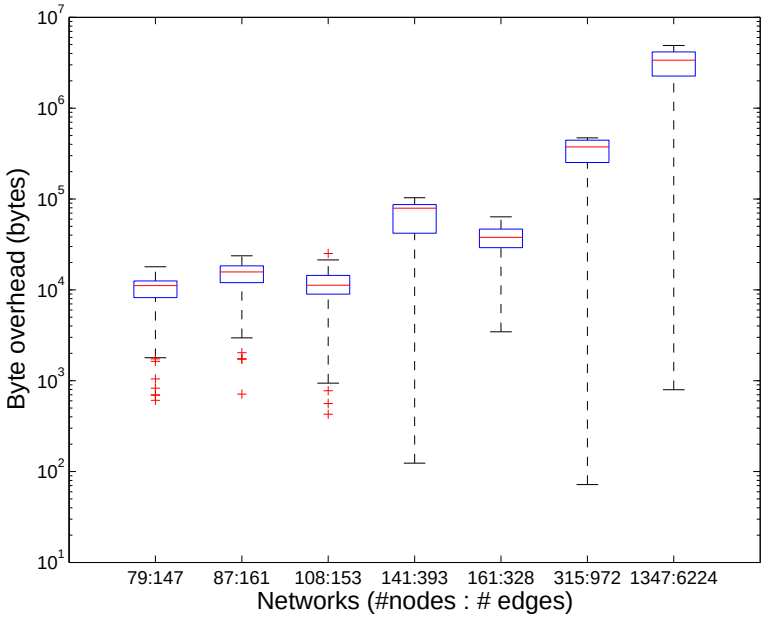

Figure 3.7: Switch configuration traffic sent out on a single link failure for various Rocketfuel topologies in logarithmic scale.

	Min	Mean	Max	SD
Backup DE takes over	130 ms	142 ms	155 ms	6 ms

Table 3.1: Minimum, mean, and maximum times, and standard deviation for DE failover in DE failure experiments on the backbone network.

Network partition: We inject a large number of link failures into the backbone topology to create scenarios with multiple network partitions. In the partition with the original master DE, Tesseract responds in essentially the same manner as in the regional-failure scenarios examined in Section 3.3.2, since the original master DE sees the partition as a large number of link failures. In the partitions that do not contain the original master, the convergence time is the same as when the master DE fails.

Just as network designers can choose to build a topology that has the right level of resistance against network partition (e.g., a ring versus a complete graph), the designers can intelligently select locations for placing redundant DEs to defend against network partition.

3.4 Tesseract Applications

In this section, we demonstrate two applications that take advantage of Tesseract's direct control paradigm.

3.4.1 Joint Control of Routing and Filtering

Today, many enterprise networks configure packet filters to control which hosts and services can reach each other [2]. Unfortunately, errors in creating network configurations are rampant. The majority of disruptions in network services can be traced to mis-configurations [48, 49]. The situation with packet filters is especially painful, as routes are automatically updated by routing protocols to accommodate topology changes, while there is no mechanism to automatically adapt packet filter configurations.

The Tesseract approach makes joint routing and filtering easy. The decision logic takes as input a specification of the desired security policy, which lists the pairs of source and destination subnets that should or should not be allowed to exchange packets. Then, in addition to computing routes, for each source-destination subnet pair that is prohibited from communicating, the DE initially places a packet filter to drop that traffic on the interface closest to the destination. The decision logic then further optimizes filter placement by pulling the filters toward the source of forbidden traffic and combining them until further pulling would require duplicating the filters.

As a concrete example, revisit the example network in Figure 1.1. This company's network is spread across two locations, A and B. Each location has a number of front office computers used by sales agents (AF1-2 and BF1-2) and a data center where servers are kept (AD1-2 and BD1-2). Initially, the two locations are connected by a link between the front office routers, R2 and R4, over which inter-office communications flow. The routing metric for each link is shown in italics. Later, a dedicated link between the data centers (shown as a dashed line between R1 and R3) is added so that the data centers can use each other as remote backup locations. The security policy is that front-office computers can communicate with the other location's front office computers and with the local data center's servers, but not the data center of the other location. Such policies are common in industries such as insurance, where the sales agents of each location are effectively competing against each other.

We experimentally compared the Tesseract-based solution with a conventional solution that uses OSPF and manually placed packet filters. During the experiments we generate data traffic from AF1 to BF1 (which should be permitted) and from AF1 to BD1 (which should be forbidden) at 240 packets per second and monitor for any leaked or lost packets. In the OSPF network, the filter is manually placed on interface i3.1 to prevent A's front office traffic from reaching BD. After allowing the routing to stabilize, we add a new link between the data centers (dotted line in Figure 1.1). In the OSPF network, OSPF responds to the additional link by recomputing routes and redirects traffic from AF to BD over the new link, bypassing the

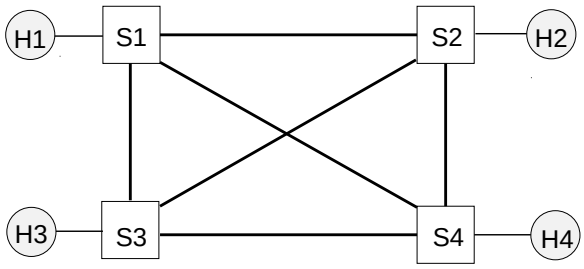

Figure 3.8: Full-mesh Ethernet topology.

packet filter on interface i3.1 and creating a security hole that will have to be patched by a human operator. In contrast, Tesseract computes both new routes *and new packet filter placements appropriate for those routes* and loads into the routers simultaneously, so no forbidden traffic is leaked. Most important, once the security policy is specified, it is automatically enforced with no human involvement required.

3.4.2 Link Cost Driven Ethernet Switching

Ethernet is a compelling layer-2 technology; large switched Ethernets are often used in enterprise, data center, and access networks. Its key features are: (1) a widely implemented frame format; (2) support for broadcasting frames, which makes writing LAN services such as ARP and DHCP significantly easier; and (3) its transparent address learning model, which means hosts can simply plug-and-play. Unfortunately, today's Ethernet control plane is primitive [50, 51, 52]. Based on routing frames along a spanning tree of the switches, it makes very inefficient use of the available links. Convergence time in response to failures can be long, as the IEEE 802.1D Rapid Spanning Tree Protocol (RSTP) is known to count to infinity in common topologies.

We have implemented a Tesseract control plane for Ethernet that preserves all three beneficial properties, avoids the pitfalls of a distributed spanning tree protocol, and improves performance. The DE first creates a spanning tree from the discovered network topology and generates default forwarding entries for the switches that follow the tree — this enables traditional tree-based broadcast. Additionally, when an end host sends its first frame to its first-hop switch, the switch notifies the DE of the newly discovered end host via the dissemination service. The DE then computes appropriate paths from all switches to that end host and adds the generated forwarding entries to the switches. From then on, all frames destined to the end host can be forwarded using the specific paths (e.g., shortest paths) instead of the spanning tree.

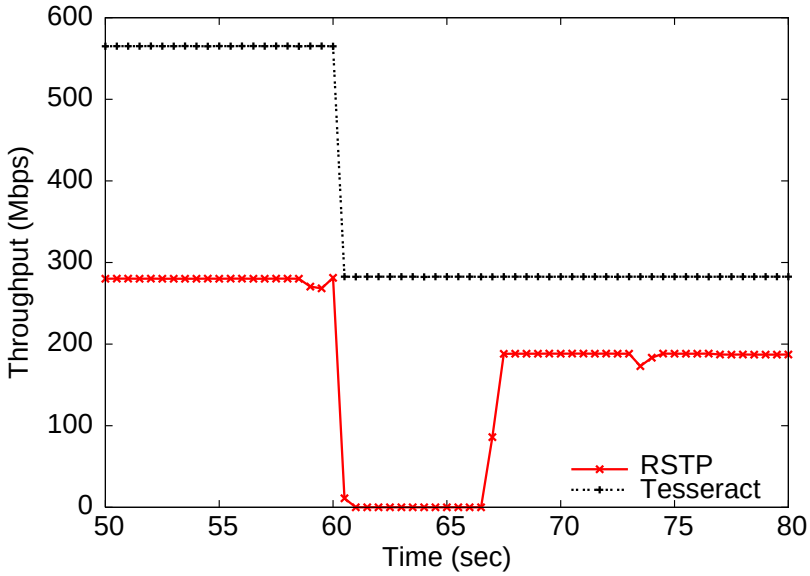

Figure 3.9: Aggregate network throughput, RSTP versus Tesseract. S1 fails at 60 second.

To experimentally illustrate the benefits of the Tesseract approach, we use the topology shown in Figure 3.8 on Emulab. The four switches are connected by 100 Mbps Ethernet links, and each end host is connected to one switch via a 1 Gbps Ethernet link. We run `iperf` [53] TCP servers on the four end hosts and simultaneously start six TCP flows. They are H1 to H2, H1 to H3, H1 to H4, H2 to H3, H2 to H4, and H3 to H4. In the first experiment, the network is controlled by Tesseract using shortest path as the routing policy. In the second experiment, the network is controlled by an implementation of IEEE 802.1D RSTP on Click.

Figure 3.9 shows the aggregated throughput of the network for both experiments. With the Tesseract control plane, all six TCP flows are routed along the shortest paths, and the aggregate throughput is 570 Mbps. At time 60 s, switch S1 fails and H1 is cut off. The Tesseract system reacts quickly and the aggregate throughput of the remaining 3 TCP flows stabilizes at 280 Mbps. In contrast, in a conventional RSTP Ethernet control plane, forwarding is performed over a spanning tree with S1 as the root. This means the capacities of the S2-S3, S2-S4, and S3-S4 links are totally unused. As a result, the aggregate throughput of the RSTP controlled network is only 280 Mbps, a factor of two less than Tesseract. When switch S1 fails at time 60 s, RSTP tries to reconfigure the spanning tree to use S2 as the root and begins a count-to-infinity. The combination of frame loss when ports oscillate between forwarding/blocking state and TCP congestion control back-off means the throughput does not recover for many seconds. When RSTP has finally reconverged, the aggregate throughput is again substantially less than the Tesseract network.

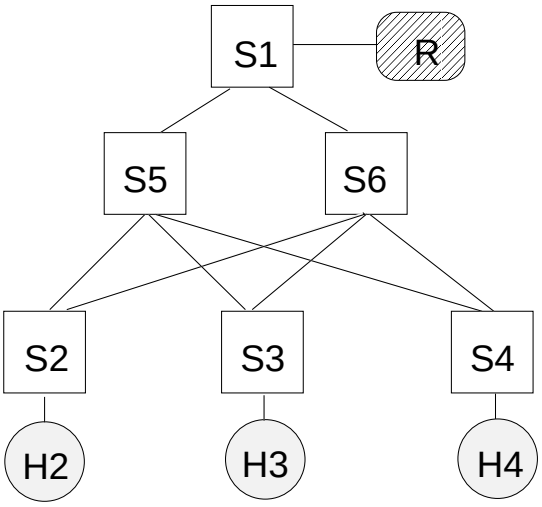

Figure 3.10: Typical Ethernet topology gadget.

As a second example of the value of being able to change the decision logic and the ease with which Tesseract makes this possible, consider Figure 3.10. This topology gadget is a typical building block found in Ethernet campus networks [54] that provides protection against any single link failure. Basic Ethernet cannot take advantage of the capacities of the redundant links since RSTP forms a spanning tree with S1 as the root, and the S2-S6, S3-S6, and S4-S6 links only provide backup paths and are not used for data forwarding. As a result, traffic flows from H2, H3, and H4 to R must share the capacity of link S1-S5. In contrast, when there exist two or more equal-cost paths from a source to a destination, the Tesseract decision logic breaks the tie by randomly picking a path. By centralizing the route computations and using even such simple load-balancing heuristics, Tesseract is able to take advantage of the multiple paths and achieve a substantial increase in performance. In our example, the capacities of both link S1-S5 and S1-S6 are fully utilized for a factor-of-two improvement in aggregate throughput over RSTP.

The examples in this section illustrate the benefit of the direct control paradigm, where the only distributed functions to be implemented by network switches are those that discover the neighborhood status at each switch and those that enable the control communications between the DE and the switches. As a result, it becomes easy to design and change the decision logics that control the network. There is no need to design distributed protocols that attempt to achieve the desired control policies.

3.5 Summary

This chapter presents the design and implementation of Tesseract, the initial 4D prototype that enables direct control. In designing Tesseract, we paid particular attention to the robustness of the decision plane and the dissemination plane. The security of Tesseract is enhanced by the mechanisms built into the dissemination service. The system is designed to be easily reusable, and we demonstrated how Tesseract can be used to control both Ethernet and IP services. Finally, good performance is achieved by adopting efficient algorithms such as incremental shortest path and delta encoding of switch configuration updates.

We find that Tesseract is sufficiently scalable to control intra-domain routing in networks of more than 1000 switches, and its reconvergence time after a failure is detected is on the order of one round-trip propagation delay across the network.

The most important benefit of Tesseract is that it enables direct control. Direct control means that sophisticated control policies can be implemented in a centralized fashion, which can be much easier to understand and deploy than a distributed protocol. Direct control also means that the software running on each switch is simplified, with potential benefits for operators and vendors. We strongly believe that the direct control paradigm is the right approach in the long run, as there is a clear trend toward ever more sophisticated network control policies.

Further developments in the philosophy of direct control have emerged since the Tesseract work. The OpenFlow Switching Consortium [32] was created in 2008 to support an open standard that allows researchers to control how an Ethernet switch forwards packets via a standardized interface. An OpenFlow switch is directly controlled by a remote control process which the consortium calls the Controller. The Controller is similar to the Tesseract decision element that makes network decisions and sends instructions to network elements through a robust and secure dissemination channel. Compared with Tesseract, OpenFlow takes the further step of making a network a programmable platform on which researchers can run experiments. While we try to use Tesseract to demonstrate that the direct control concept is practical on both backbone and enterprise networks, OpenFlow focuses on building real products for Ethernet switches; recently the OpenFlow consortium announced 48-port gigabit switches that support the open standard for direct network control. We hope that OpenFlow can be increasingly adopted by enterprise networks, and that it will eventually be interconnected by OpenFlow backbone networks. If Tesseract turns the direct control paradigm from vision to reality, we hope that the wide deployment of OpenFlow will make direct control the de facto standard for running tomorrow's networks.

2009 also saw startups such as Nicira [33] that develop real-world software to decouple network control from the underlying physical hardware. Nicira's goal is to make data center networks more flexible and economical. In this chapter, we show through a case study that decoupling network control from the data plane will allow Nicira to achieve sophisticated network goals (e.g., security) with relatively simple software.

Chapter 4

Making 4D Dissemination More Robust

Chapter 3 describes the 4D Tesseract system and answers the question of how to make 4D a flexible system that greatly simplifies network control. In Tesseract, a decision element in a single location computes network state (e.g., routing tables) and remotely configures geographically distributed network elements. Fundamental to this is the ability of the decision element to communicate securely and robustly with the network elements being managed. This chapter focuses on how we construct such a secure and robust communication channel to power Tesseract and other centralized network control systems.

In practice, creating the dissemination channels and maintaining them in the face of operational realities is a significant problem. Many network elements are in unattended locations, thus a loss of management connectivity means that a technician will have to travel to the network element before there can be any possibility of accessing the console to collect data or reconfigure the element. The resulting service outages are measured in hours — we report on several of these in our case studies. Denial of Service attacks and flash-crowds are well-known to cause management communications to fail at exactly the moment they are most needed to recover control over the network [55]. The frequency with which networks grow or evolve, coupled with the fact that today's management communication channels are as subject to outages caused by misconfiguration as the rest of the network, means that networks are in continual peril of losing the ability to remotely reconfigure network elements.

The contributions of this chapter include defining the challenges that face management communication channels, and presenting a system design that solves these challenges. We also demonstrate how our system solves several common problems that arise while remotely managing networks, and we evaluate the performance of the system to show that it is suitable for use even in large, fast networks.

4.1 Case for a Robust Meta-Management System

In 4D, the decision element must reach and configure a network element before the network element is configured with routing state. At the same time, the network element must communicate with the decision element before it is configured by the decision element. The initial 4D prototype uses source routing to break the circular dependency. In this chapter, we develop the source routing idea into a complete network solution called the Meta-Management System (MMS) [56] in order to not only address the circular dependency problems, but also to leverage all available network connectivity in order to achieve maximum robustness, while simultaneously protecting management traffic from regular data traffic. Management applications can access the MMS service via familiar socket APIs and transport protocols. Thus, both new 4D decision logic and existing IT management systems can run directly on the MMS.

To support remote management and configuration, we design the MMS to satisfy four key system requirements: (1) Automation: the MMS uses new protocols to build "plug-and-play" management channels — using no configuration beyond device identification keys; (2) Liveness assurance: the MMS both maintains management communication channels in the presence of failures, and provides APIs by which resources on network elements can be remotely managed to ensure the liveness of critical management tools; (3) Security assurance: The MMS automatically authenticates network elements, applying onion-encrypted source routing [57] in new ways that prevent memory exhaustion attacks and enable decision elements to detect and evict network elements that DoS-attack management communications; (4) Evolvability: The MMS allows multiple MMS versions to work concurrently, enabling the MMS to safely manage itself and to evolve seamlessly.

4.2 Requirements for a Management Communication Service

The two approaches used today for establishing management communications do a very poor job of handling these practical issues. The first common approach is *in-band* management of network elements, where the same IP stack and routing tables used to forward data through the network are also used to carry management traffic to network elements. This approach has two problems. First, a remote management process cannot reach and configure a network element before the network element can forward IP packets. At the same time, the network element cannot forward IP packets before it is configured by the remote management process. This circular dependency results in fragility. Second, the IP service in a network is frequently reconfigured

to satisfy customers' changing needs. If a service change causes a failure in the IP service, management communications will also fail as there is fate-sharing between data and management communications.

It might seem that building a physically separate network to carry management communications (often referred to as *out-of-band* management) solves the management communications problem. Unfortunately, building and maintaining a physically separate network for remote management all the way to devices at the edge of a large network is costly or is simply impossible. Beyond cost, a more fundamental problem is that the separate management network must still be configured and managed, and all existing software stacks to do this suffer from the problems of in-band management.

We believe the argument between in-band and out-of-management is misplaced. Remote management communications should be carried on *logically out-of-band* management links, where these links are constructed from both regular data traffic links as well as whatever separate dedicated management links exist. A logically out-of-band link is simply a logical partition (with performance guarantees) of an underlying physical link. We emphasize that realizing logically out-of-band links is *not* the primary challenge facing management communication. Circuit technologies such as SONET/SDH have long had the capability to create logically out-of-band links. Packet technologies such as MPLS [58], Frame Relay [59], and weighted fair queuing can also establish logically out-of-band links.

The primary challenge is to design a logically out-of-band management channel that is not circularly dependent on the data plane. In addition, the design must have the following features critical for management communications:

Automation: The system should require little or no configuration and create remote management communication channels automatically in a plug-and-play fashion. A system that requires complex manual setup only makes network management more difficult and error-prone. Modifying off-the-shelf protocol software on routers to support automation is not trivial. Protocols such as OSPF [60] require pre-configuration before they can discover neighbors and build routes. OSPF provides neighbor discovery between two preconfigured and adjacent interfaces, but physical device discovery and associations between entities require configuration commands and external inventory databases. The MMS software, on the other hand, expects no state other than a security certificate to be preconfigured in order to bootstrap a management communication channel to all network elements.

Liveness assurance: The system should ensure the liveness of the communication channels as well as the liveness of critical higher layer management software tools. The latter is somewhat subtle. It is important

for a system not only to ensure that messages can successfully cross a network, but also to ensure that those messages can reach important software (e.g., the secure shell daemon `sshd`) running on a remote device. No existing protocols help ensure the liveness of critical higher layer management software tools. An operator may indeed be able to "ping" a router through the management channel provided by an off-the-shelf routing protocol, but he might not be able to log in to the router if the command line interpreter or secure shell daemon is unresponsive. Further, conventional protocols might not be feasible to run on large networks with old or under-resourced network elements, or to scale to large numbers of very inexpensive devices. Off-the-shelf routing protocols typically require an amount of memory that scales with the network size. The MMS requires only a small constant amount of memory on routers that can be allocated statically, enabling it to maintain liveness to all routers in the network. The MMS also provides ways for operators to recover from many of the common scenarios that cause network elements to become unresponsive.

Security assurance: The system should be resistant to DoS attacks, automatically authenticate network devices, and be able to detect and evict a compromised device if it is behaving maliciously. Current routing protocols such as OSPF and IS-IS use a single shared secret key for message authentication. If any router is compromised, the compromised router could fake messages from arbitrary routers — announcing bogus connectivity to arbitrary routers to attract and intercept traffic. The MMS software is resilient to such attacks from single/multiple compromised switches. Moreover, even subtle collusions between compromised switches do not affect the delivery of management traffic.

Evolvability: The system should be evolvable, and it should enable itself to be managed remotely. That is, using the communication capability provided by the system itself, it should be possible to update the system remotely without the risk of crippling the system, even if the updates turn out to have bugs that require them to be rolled back. Updating the current off-the-shelf protocol software on routers remotely has inherent risk. If the new version is erroneous or misconfigured, the remote management communication channels can be lost. The MMS software can be seamlessly upgraded, and it can recover from erroneous updates.

4.3 Meta-Management System (MMS) Design and Implementation

In this section, we present the design and implementation of our remote management solution, the Meta-Management System (MMS). The MMS software runs on network elements (NE), by which we mean routers, switches, firewalls, and other middleboxes. The MMS also runs on decision elements (DEs), the

network-connected servers used to manage and configure the network. The design of the MMS leverages a key observation: most management traffic flows between a small number of DEs and the NEs. Once the management system provides robust and configuration-free communication channels between DEs and NEs, remote management is enabled. Any NE-to-NE communication that may be needed can be provided through forwarding via an overlay constructed on the DEs.

4.3.1 Key Features of the MMS

Automatic creation of management channels

When a DE with a valid security certificate is attached to a network, the MMS automatically establishes management channels between the DE and the NEs in the network. Likewise, when an NE with a valid security certificate is attached to a network, the MMS automatically establishes management channels between it and the DEs. The MMS logically separates management communication channels from data communication channels so that they no longer share the same fate. There is no manual configuration beyond exchanging security certificates at device installation time.[1] The MMS integrates, and thereby enforces, best practices — once the MMS solution is installed, the rest is automatic. The MMS exposes a datagram service to applications, so existing management applications can access the MMS management channels via a standard socket API.

Integrated security assurance The MMS assumes a hostile environment in which malicious end hosts attached to the network may launch a DoS attack at the MMS or try to compromise NEs. The MMS is robust to such attacks and NE compromise. First, regular end hosts have no way to address DEs in the network, thus launching a DoS attack at the DEs is not possible. The logically out-of-band MMS management channels have priority over data traffic, and thus DoS attacks against NEs in the data plane cannot disrupt management traffic. If an NE is compromised, it can drop MMS traffic or generate spurious messages in a DoS attack. However, due to the MMS's use of onion-encrypted source routing, such NEs can easily be detected and then quarantined by the MMS issuing new source routes that bypass the quarantined NE. Finally, the MMS provides a mechanism to revoke a DE certificate and replace it with a new one, which is useful, for example, when a DE laptop computer storing the certificate is lost.

Integrated liveness assurance MMS maintains the liveness of the management channels in an integrated fashion. It can dynamically re-route when a loss of network connectivity occurs. It is designed to protect

[1] Major vendors today already install security certificates onto their network elements before shipping them to customers.

itself against CPU resource starvation. Moreover, due to the use of source routing, MMS has very little runtime state, and all the memory it needs can be statically allocated at boot time, thereby defending against memory starvation. Furthermore, the MMS provides remote process management and packet filtering APIs to ensure the liveness of critical higher-layer management software tools.

Evolvable The MMS can be used to manage and evolve the MMS itself with zero down time. The design of MMS enables multiple parallel instances of the MMS to operate over the same network at the same time. This allows a new MMS instance to be brought up in order to manage or replace the old instance. For example, a new version of the MMS can be installed and brought up through the management channels provided by the old working version. The new version can be tested thoroughly before the old version is removed.

Handles large networks The protocols used in the MMS were specifically designed so that the amount of memory and CPU computation required of network elements is independent of the size of the network. This means that the network can grow without forcing the upgrade of all NEs. Instead, the computation and memory requirements are placed on the decision elements. The DEs can target their resources at reaching the specific NEs they wish to configure, and since DEs are just end hosts and comparatively few in number, they are easy to upgrade. Management stations can be connected to the network at any port, so service technicians in the field and operators in the network operation center can all access network elements using the MMS — there is no need to travel to special "network management ports" to connect. After a DE is plugged into a network, in about 30 seconds it can establish MMS secure channels to 1000 core devices (the size of many large enterprises and ISPs [2]).

4.3.2 Isolation of MMS Frames

Because the MMS leverages the same physical links used for regular data packet transmissions, the link-layer must logically partition the link so that the MMS module on one network element can send MMS frames to the MMS module on a neighboring network element. This partitioning should be done so that the logical link used by the MMS has a guaranteed minimum throughput — this prevents regular data traffic from interfering with the delivery or processing of MMS frames. This abstraction can be realized in a number of ways, and the exact solution may be slightly different for different link layers. For example, SONET links might use the supervisory channel to carry MMS frames, since that channel has been built into

SONET framing and has protected bandwidth. In a datagram network, weighted fair queueing or priority queueing might be used to create the required logical partition.

In our implementation, the network consists of point-to-point Ethernet links, and MMS frames are sent to a reserved multicast address and tagged with a specific protocol type. When the MMS module is activated, it configures the OS to hand it any MMS-tagged frames going to this multicast address. To prevent user traffic (e.g., DoS attacks) from interfering with management communication, we use the simple priority queueing system provided by the interface driver. MMS frames are put into the highest priority queue and thus served first by the scheduler.

4.3.3 Automatic Construction of Secure Channels

One of the most important and basic features of the MMS is the construction of a set of secure channels for management information to flow between a DE connected to the network and the NEs that make up the network. These channels must be authenticated, must survive DoS attacks and local link or NE failures, and must be able to recover from an NE compromise. This section explains our design for establishing and maintaining these management channels.

4.3.3.1 Threat Model

The MMS is designed to withstand the following threats:

Operator error: Mistakes made while altering the configuration of network elements.

Attack from an end-host: Hosts connected to the network may attempt to DoS or inject false commands into the management channel.

Compromise of a Network Element: Attackers may compromise any network element in the system, learn its secrets, sniff frames traversing it, and use it as a platform for launching DoS attacks against the DEs, the MMS, or the data plane.

4.3.3.2 Minimizing State Held by Network Elements

The first step in constructing a secure channel is defining and authenticating the endpoints of the channel.

Estimates show that configuration errors are responsible for 60 to 70 percent of network outages today [61]. Since the MMS must provide an always-available management channel, configuration errors that prevent communication between the DE and the NEs are intolerable. We argue that the best approach to eliminate configuration errors is to reduce the configuration state to the bare minimum needed.

In our design, each NE is configured with the following critical pieces of information prior to deployment. The first is a *network certificate* that identifies the public key with ultimate authority over the network. NEs will accept commands only from DEs that have a *DE certificate* signed by the network certificate's private key. The second is a private/public key pair that uniquely identifies the NE. The NE's public key must be made available to the DEs before the DEs can communicate with the NE. The network certificate and the private/public key pair should be preserved in non-volatile storage on the NE.

This basic configuration provides the toehold from which the DE will be able to authenticate and communicate securely with each NE. In addition to the basic configuration, each NE stores the following dynamically generated soft-state for each DE with which it communicates: (a) a secret key shared only between the NE and the DE, (b) an onion-encrypted source route by which the NE can communicate with the DE, (c) the version number of the DE's certificate, and (d) the time at which this per-DE state was last used. The exact definition of these fields and the means by which they are created will be explained next.

4.3.3.3 Secure Routing

The MMS is completely decoupled from the regular IP data plane services and therefore has its own routing subsystem. The forwarding of messages across the MMS is controlled by *onion-encrypted source routes* [62]. These are strict source routes placed in the headers of the MMS frames that list the series of NEs through which the frames must pass. A source route is built like an onion, with the list of hops remaining in the route encrypted in the secret key of the NE making the next forwarding operation. An NE without a valid onion-encrypted source route can only transmit MMS frames to its immediate neighbors. Since the DE knows the secret keys of all NEs, it can construct an onion-encrypted source route between any two NEs. As the frame is forwarded across the MMS, each hop re-encrypts the portion of the route over which the frame has already traveled.

We use onion-routing for two main reasons. First, it creates in each MMS frame a secure log of the frame's path that the DE can read — as described in Section 4.3.3.5, this property will be used to detect and evict misbehaving NEs. Second, source routing ensures that the MMS on each NE does not need to

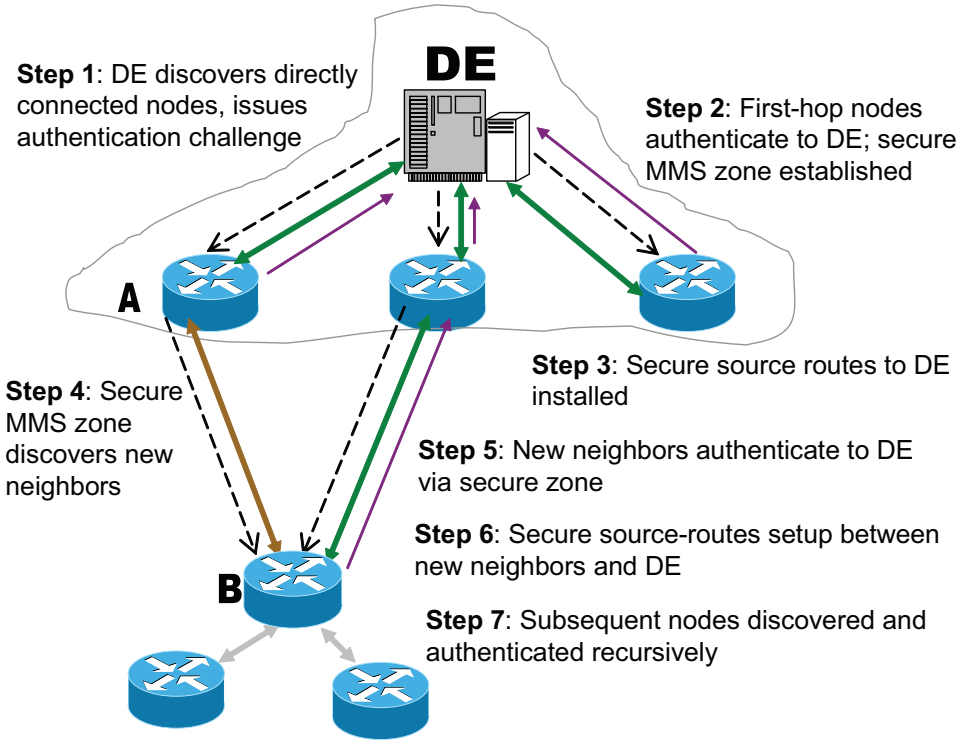

Figure 4.1: Recursive MMS Authentication.

maintain a dynamic routing table that grows with the network size. Thus, the MMS on an NE only needs a small static amount of memory and will not run into memory allocation failures.

To establish the MMS onion-encrypted source routes, a DE first recursively authenticates and establishes secret keys with the NEs in the network. During this process, the DE computes an onion-encrypted source route for each NE to communicate with the DE, and the DE installs this route on the NE. Subsequently, the DE learns changes in the topology of the network by collecting encrypted link state advertisements (LSAs) from NEs. The DE reacts to topology changes by recomputing and pushing out new onion-encrypted source routes as needed. There can be multiple DEs in the network, but each DE performs these tasks independently. The details of the authentication process are explained next.

Recursive Authentication

The DE is responsible for authenticating the NEs and sending them an encrypted source route they can use to communicate with the DE. The NEs prove their identity to the DE using a challenge-response protocol, and the DE proves itself to the NE by sending it a verifiable signed source route.

Figure 4.1 gives an overview of the process by which a DE establishes communication to and through the NEs in the network. The DE initiates and drives this process, enabling it to limit the set of NEs it contacts

to the ones of interest. This will be important in very large networks with many edge NEs. The DE begins by initiating the authentication process with the directly connected NEs (e.g., A).

The DE authenticates an NE by sending it a challenge via an onion-encrypted source route. For an NE directly connected to the DE, this source route is trivial. This challenge contains several things. The first is a 128-bit session key that serves as a shared secret between the DE and that NE. The shared secret is encrypted by the NE's public key and signed by the private key from the DE certificate. The second is the public key from the DE certificate signed by the private key from the network certificate. This signed public key is preconfigured on a DE by the administrator. It is important to note that a DE does not know the private key of the network certificate. Thus, even if a DE is compromised, the network certificate is still safe. The third component is an onion-encrypted source route from the NE to the DE signed by the private key from the DE certificate.

By verifying the certificates and decrypting the session key, the NE proves its identity, verifies it is communicating with a valid DE, and obtains an onion-encrypted source route it can use to communicate with the DE (since it can decrypt the first layer of the route using the session key). The NE then encrypts its current LSA by the session key, and sends it to the DE using the onion-encrypted source route. If the LSA informs the DE of new NEs it should communicate with, the DE recursively authenticates those NEs (e.g., B) by sending them challenges via onion-encrypted source routes over authenticated NEs.

A DE certificate contains a version number, and NEs will only accept a DE certificate with the highest version number they have seen. This means if a DE certificate is compromised, it can be cheaply "revoked" by creating a new DE certificate with a higher version number and using it to authenticate all the NEs.

Authentication in Large Networks

Since the DE drives the recursive authentication process, it can easily target the authentication toward the NEs it wants to control. This is important in large networks, even those with millions of edge NEs. As a simple example, the DE can authenticate with all the core NEs (as identified by an inventory database), obtaining LSAs that list the edge NEs and their attachment points. Even the largest networks have no more than a few thousand core NEs, which the MMS can easily handle (see Section 4.5). Subsequently, the DE can initiate authentication with only the desired edge NEs.

LSA Creation

The MMS implements a simple HELLO protocol by which an NE discovers the identities of its neighbors. As part of this HELLO protocol, neighbors exchange lists of the DEs with which they were previously

authenticated. This information need not be verified by an NE; it is advisory only. From this information, an NE creates an encrypted LSA to send to each DE it has authenticated with. In each LSA, for each neighbor there is a bit that indicates whether that neighbor claims to be authenticated to that DE.

When the link state changes, the NE that detects the change sends a new LSA to each DE for which it knows an onion-encrypted source route. Each NE limits the rate at which it sends LSAs, so that a compromised NE attempting to DoS the MMS by flooding LSAs can only flood its immediate neighbors (which is unavoidable), but not the rest of the NEs.

New LSAs are retransmitted periodically until acknowledged by the DE (the implementation uses a period of 500 ms). If a new LSA is generated, it replaces the one currently being sent. To make the system as simple as possible, an LSA is acknowledged by the DE by sending a hash of the LSA back to the NE. There is no need to use sequence numbers, as there can be only one outstanding LSA at a time, and the hash provides protection against bit-corruption in the LSA.

4.3.3.4 Response to Failures

If the connectivity between NEs changes, new LSAs are sent to the DE, and the DE re-calculates onion-encrypted source routes for affected NEs and sends the new routes to the NEs. Should an NE reboot or otherwise lose soft-state for a DE, LSAs sent by this NE's neighbors will show that this NE is unauthenticated to the DE, and the DE can re-authenticate the NE if needed. Should a DE fail, all NEs will eventually purge their soft-state for it.

The MMS is designed to survive even simultaneous failures of multiple links. In addition to the experimental results presented in Section 4.5, we are able to prove this formally.

Convergence Property: If each NE knows the shortest path to a DE, and the DE has the initial network topology, the above LSA propagation scheme ensures that the DE will eventually re-discover the shortest paths to all NEs in its network partition after any period of link failure events followed by a period without failures.

Proof. Let G_{DE} be the network topology, including the DE itself, perceived by the DE, G_{real} be the topology after the link failure event(s), $p(x)$ be the shortest path in G_{DE} from any NE x to the DE, S be the set

of NEs that have different link state in G_{DE} and G_{real}. We define a path $p(x)$ as a *working path* if it is a path in both G_{real} and G_{DE}.

After the failure event(s), at least one NE in S has a *working path* to the DE. This follows since there is always at least one NE $a \in S$ such that $p(a)$ is the shortest. Since no other NE in S is between a and the DE, there is no failed link along $p(a)$. It follows that the LSA from at least one NE in S can reach the DE, and that NE will continue to send that LSA until it is acknowledged. After the DE receives and processes the LSA, G_{DE} and p are updated and a is removed from S. The DE repeats the above procedure until S is empty. When S is empty, G_{DE} is identical to G_{real}. Thus, it takes at most $|S|$ steps to make S empty, at which point the network has converged. □

Therefore, as long as the DE assigns each NE the shortest onion-encrypted source route, the network is guaranteed to converge even when multiple failures occur simultaneously. In addition to the shortest route, the DE can optionally give an NE a *preferred* route which is not necessarily the shortest. An NE can use the preferred route to send management traffic to the DE, and only uses the shortest route to send LSAs. The flexibility of assigning preferred routes allows more advanced features to be implemented on the DE. One such example is described next.

4.3.3.5 Resilience to Attacks

Under this security framework, only authenticated NEs can communicate with DEs via the MMS. When used with traffic isolation techniques (see Section 4.3.2), data plane DoS attacks cannot disrupt management communication. Even if an NE is compromised, the attacker cannot modify the MMS frames in transit because they are all encrypted with the secret key of another NE. The compromised NE also cannot announce bogus connectivity to non-compromised NEs in order to attract traffic to it because the DE can detect the inconsistency in the LSAs.

A compromised NE can, however, launch a DoS attack on the MMS by dropping frames in transit or by sending useless frames to the DE. A simple attacker that sends traffic using its own source route would be trivially caught, since the source route identifies the sender. A sophisticated attacker could attempt to

hide its identity by reusing a source route extracted from a frame it has forwarded, thereby making its attack traffic appear to come from the origin of the source route.[2]

The use of onion-encrypted source routes, however, offers both a mechanism to identify the origin of the DoS and a mechanism to isolate the offender once identified. Onion-encrypted routes give us strong assurance that any malicious packet received by the DE must have been sent by an NE listed in the packet's source route. The techniques of Zhang et al. [63] can then be used to identify the malicious NE. Summarizing that work, we assume that a DE can determine it is receiving malicious packets if they are sent at a rate above some detection threshold. Over time, the DE orders NEs to change the source routes they use. This allows identification of the attacker by forcing it to move its malicious traffic among different source routes, and the attacking node will eventually be the only node in common among the source routes along which malicious traffic arrived. The attacker's only strategy is to limit the number of malicious packets it sends to stay below the detection threshold, but this bounds the impact of its DoS attack. If an attacker is identified, the DE can issue new onion routes that avoid it.

4.3.4 Assuring Liveness

To achieve liveness, beyond the ability to react to link or NE failures as explained in Section 4.3.3.4, there are three additional challenges.

Protecting Against CPU Starvation

A common issue on NEs is CPU starvation caused by a runaway process or a data-plane DoS attack. However, the MMS must maintain management communication channels during these events so that an operator can remotely diagnose the problem.

The MMS relies on the NE's kernel scheduler to remain sufficiently live so that the MMS can send and receive frames.[3] The MMS can use any mechanism the kernel provides to protect a sufficient share of CPU cycles. To minimize the share of CPU cycles needed to run the MMS on NEs, the MMS has a centralized design in which the most compute-intensive work, i.e., route computation, is carried out on the DE.

However, even when the core kernel services of an NE remain live, it is possible for a process running on the NE (e.g., the OSPF or the BGP process) to consume so many CPU cycles that critical processes (e.g.,

[2]Including nonce or timestamps in the source route could prevent this replay attack, but would require that NEs share keys with all downstream NEs, rather than just the DE. We rejected that approach for scalability reasons.

[3]The problem of surviving arbitrary failures of the NE's kernel or operating system is intractable.

the command shell) become unresponsive. For instance, this could happen when a misconfiguration causes hundreds of thousands of inter-domain routes to be mistakenly injected into an intra-domain routing process. If the command shell remains unresponsive, operators have no way to remotely resolve the problem.

To enable recovery from this type of situation, the MMS provides a process management API and a packet filtering API. Using these APIs, a DE can command the MMS to return a list of the processes running on an NE, kill a particular process, change a process priority, install an IP data plane packet filter, or reboot the NE. We elaborate on the features of these APIs in Section 4.3.6.1. Together, these mechanisms allow an operator to remotely restore liveness to an NE's command shell via the MMS, investigate the cause of the problem, and reconfigure the NE as needed to prevent a recurrence of the problem. In the extreme case, an operator can remotely reboot an NE via the MMS.

Protecting Against Memory Outages

The MMS is designed to avoid "out of memory" errors by using static rather than dynamic memory allocation. In this way, as long as the MMS is successfully loaded at system startup time, it is unlikely to be impaired by memory allocation problems caused by misbehaving processes. This design requires the MMS to limit runtime state. In particular, this led to our use of source routing in the MMS, ensuring that only DEs need to build the complete network topology, which requires memory proportional to the network size. The state stored by each NE scales only with the number of ports on the NE, which is known at boot time, and with the number of simultaneously active DEs communicating with the NE. In our implementation the soft state maintained by an NE for each DE takes up approximately 500 B of memory. A static array of 500 KB can, for example, support 1,000 simultaneously active DEs.

4.3.5 Evolving the MMS

Networks are constantly evolving in ways difficult to anticipate. No matter how well the MMS has been designed and engineered, one cannot rule out the need for updating the MMS running in the field. Thus, the MMS must provide a robust means by which the MMS itself can be remotely managed and evolved.

Our approach to robustly evolving the MMS is to allow multiple versions of the system to operate over the same network at the same time. This allows the new version to be brought up and thoroughly tested before the old version is removed. Each version of the MMS operates independently and in parallel. Copies of all MMS frames are delivered to each version. An MMS frame contains a version number in the header, and an MMS version skips over frames marked for other versions.

In our design, management applications can specify through the use of a socket option which version of the MMS should carry its traffic. Packets sent by applications that do not specify an MMS version are handed to every version of the MMS running on that DE or that NE. Each copy is independently routed by its respective version of the MMS to the destination.

Management applications built on top of TCP will not see duplicated packets, as they will be discarded by TCP. For non-TCP applications, we leave it up to the application itself to ensure that these duplicated packets do not cause a problem. Robust UDP- and ICMP-based applications already cope with duplicated packets, and in our experiments we did not find duplicated packets to be a problem. We choose this design so that management applications will work unmodified over the MMS without additional configuration, and we accept the performance cost of handling duplicated packets as a reasonable trade-off.

4.3.6 MMS APIs

The MMS provides two key APIs: one for remote recovery to address liveness issues, and another to support existing network management applications that use TCP/IP protocols for transport.[4]

4.3.6.1 MMS API for Remote Recovery

After reviewing common attacks, configuration mistakes, and management failure scenarios, we designed the APIs described in Section 4.3.4 to balance simplicity against the wide range of possible capabilities. These two APIs should enable recovery in many situations where remote NEs are overloaded and unresponsive. Via the process management API, a DE can command the MMS to return a list of the processes running on an NE, kill a particular process, change a process priority, start a process, or reboot the NE. When the process management API is invoked on a DE for an NE, a special MMS frame that carries the parameterized process management command is sent to the NE and interpreted by the MMS running on that NE. For example, when the destination NE receives a kill command with a process id parameter, the MMS kernel module running on the destination NE iterates through the kernel process table and sends a kill

[4]Our MMS prototype provides two additional APIs: (a) domain-name resolution and dynamic registration; and (b) a proxy service that enables management applications on NEs to communicate with each other and external networks. Implementing this functionality requires no changes to the network elements, merely code on the decision elements where it can be easily modified or upgraded. We have found that these APIs increase the utility of the MMS in practice.

signal to the intended process. While extremely simple, in practice these capabilities are the primitives that operators and IT staff commonly use to mitigate problems and restore service.

The MMS IP data plane packet filtering API allows packet filters to be installed directly via the MMS, without first obtaining a shell to run a user space application (in contrast to iptables [64] invocation, for example). Similar to the process management API, when the packet filtering API is remotely invoked, a packet filter rule is sent from a DE to an NE. The MMS on the target NE directly communicates the rule to the packet filtering kernel module, for example netfilter [64], without competing with any user space applications for resources. Together with the process management API, the packet filtering API can be used to isolate a misbehaving NE and download a new patch or image.

The security provisions of the MMS ensure that these APIs can only be invoked by a valid DE, and the DE software itself can validate that the DE operators have the rights to perform the tasks.

In our case studies in Section 4.4, we demonstrate the use of these APIs to restore liveness under resource exhaustion conditions.

4.3.6.2 MMS API to Applications

A large number of existing network management tools use the Internet Protocol for transport, for example, SNMP pollers (e.g., MRTG, Cricket), remote scripting tools (e.g., rancid, expect), and PlanetLab administration tools. To maximize backward compatibility with existing management tools, the MMS creates a virtual "management LAN" on top.

Inside the kernel, the MMS intercepts any packets sent over the management LAN, encapsulates these packets into MMS frames, and transports the packets via MMS routes.

4.3.7 MMS Implementation

Our MMS implementation is a Linux loadable kernel module that is introduced into the kernel network stack of the DEs and NEs as shown in Figure 4.2. The MMS traffic is captured by a trap in the network stack and bypasses layer 3 IP processing completely. On the DE, traffic sent by a management application is injected into the MMS, and the traffic is forwarded by the MMS on intermediate NEs and delivered via the MMS to the application running on the receiver NE. Other implementation variants of the MMS are possible; our Linux implementation serves as a proof of concept.

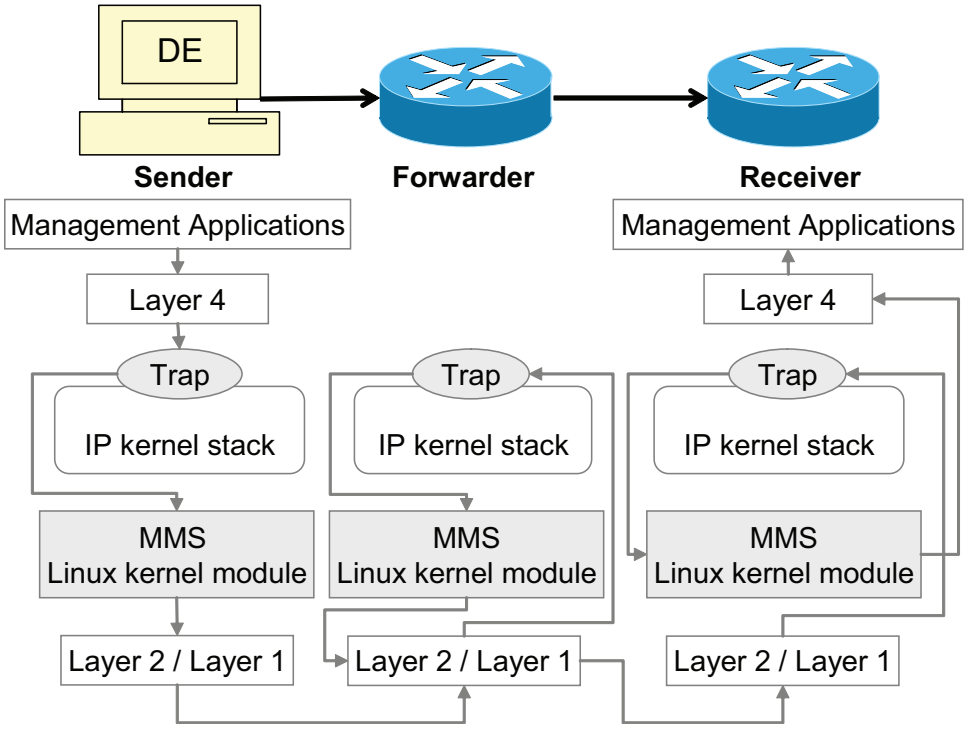

Figure 4.2: High-level overview of the MMS implementation.

The system consists of 21K lines of C code. Almost 17K lines of code are from the GNU MultiPrecision (GMP) library used to support cryptographic mechanisms. With additional engineering work, we could strip out the many unneeded functions from the library and reduce the code size.

4.4 Case Studies

To demonstrate the effectiveness of the MMS mechanisms, we give examples of how the MMS solves concrete problems that arise in network management. We implement these scenarios on Emulab [6] to illustrate these examples. In the first example, we show how the MMS enables operators to quickly respond to and recover from critical conditions where network devices become unresponsive due to a flood of malicious traffic or misbehaving management applications. Next, we show how the MMS enables the remote management and configuration of a virtualizable network infrastructure. Finally, we demonstrate the suitability of the MMS for managing a wireless mesh network.

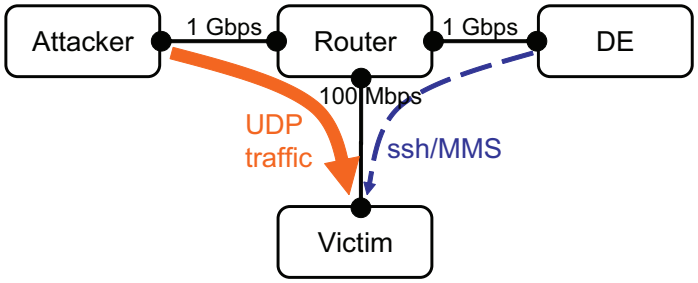

Figure 4.3: Topology of the experiments in case study A. *Attacker* sends UDP packets to *Victim*; *DE* tries to establish an `ssh` session with *Victim* over MMS.

4.4.1 Relieve Control and Management Plane Stress

A router's control and management planes run a variety of applications: routing daemons, traffic monitors, intrusion detection/prevention systems, and SNMP agents. Software bugs, network operation errors, and network attacks (e.g., DoS, worms) can cause applications to consume excessive computing resources and can even render a router unreachable or unable to respond to remote management commands. For example, during the breakout of the Slammer worm [65], many routers and switches became unresponsive. This was because the Slammer worm generated an enormous amount of packets with class D IP multicast addresses, and many routers and switches processed such multicast packets using their control plane CPUs [55]. As a result, routers' CPUs and memories were overwhelmed, forcing operators to physically visit the affected devices to install packet filters to block the worm traffic. This dramatically increased the time required to get the network back under control.

In situations where the control and management planes are threatened by resource starvation, the MMS mitigates the threat through its `packet filtering` and `process management` APIs.

Using packet filtering API

Typically, an operator installs packet filters by changing router configuration files or issuing shell commands such as iptables [64]. Ironically, under situations when the control/management planes are overloaded due to abnormal traffic and the deployment of packet filters is most desperately needed, it can be difficult to secure enough computing resources to change and commit the configuration or to launch the shell commands. The MMS APIs, however, provide a solution.

We demonstrate the benefits of the MMS using a real-world example based on `Snort`. `Snort` is an open-source network intrusion detection/prevention system widely used in enterprise networks. When run

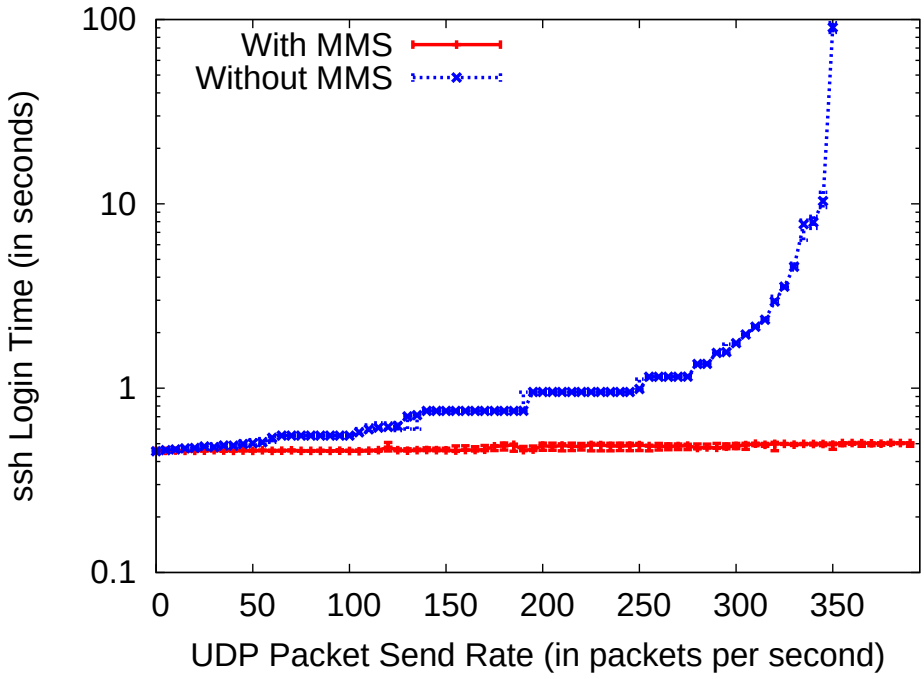

Figure 4.4: `ssh` login time versus UDP packet rate in the `Snort` experiment. Each login is attempted 100 times and the first quartile, median, and third quartile of the login times are plotted. The Without MMS plot ends at 350 packets per second, because beyond that rate all of the attempted logins fail. The With MMS plot represents the sum of the time for invoking the packet filtering API and completing the ssh login.

in the inline mode, it holds packets in a user space queue and inspects them to make accept/drop decisions based on a set of rules. Unfortunately, when `Snort` (run in the inline debug mode) encounters bursty UDP packets, it can consume excessive resources and starve other applications[5]. We conduct an Emulab experiment to measure the impact of such starvation. We create a network of star topology as shown in Figure 4.3, where we run `Snort` version 2.4.3 on the *Victim* node with a 600 MHz CPU and the Linux 2.6.12 kernel and we send UDP packets from the *Attacker* node to the *Victim* node at increasing rate. Figure 4.4 shows the time it takes to `ssh` log in to the *Victim* from the *DE*. We can see that without the MMS `ssh`, login becomes impossible when the UDP packet rate is merely above 350 pps because `ssh` is starved and times out. In contrast, with the MMS, `Snort` barely impacts `ssh` login. This is because the MMS provides a live communication channel through which the MMS packet filtering API can be remotely invoked to block UDP packets, and then an `ssh` login via the MMS channel can be successfully completed. In such critical situations, the MMS can mean the difference between maintaining remote manageability or losing it completely.

[5]The problem exists on Linux kernels older than version 2.6.14.

Using process management API

Even when there is no malicious traffic, application software bugs can cause resource exhaustion. Anecdotally, it is known that certain bugs in the SNMP agents running on a tier-one provider's Alcatel 1630 switches caused severe CPU overload on the switches when they received bursty SNMP queries. The problem persisted for minutes and the switches eventually shut down. The consequence was that thousands of customers lost their local telephone services for half an hour, and the provider had to report the incident to the Federal Communications Commission.

We conduct an experiment to emulate the scenario in the above example. We use the same network topology as in the previous experiment (Figure 4.3). We inject a bug into an SNMP agent (`snmpd`) so that it enters an infinite loop when it receives a certain SNMP query. We run this buggy SNMP agent with a high priority on the *Victim* node. When the SNMP bug is triggered, we find that `ssh` login to the PC from the *DE* becomes impossible because it times out.

The MMS process management API solves this problem. Through the live MMS communication channel, the MMS process management API can be remotely invoked to lower the priority of the misbehaving `snmpd` process, and a `ssh` login can then be completed normally within one second. Again, in this situation the MMS can mean the difference between maintaining remote manageability or losing it completely.

4.4.2 Management and Configuration of Virtual Networks

Network virtualization [66] is being pursued by many groups, from operators to device vendors, as a way to make more flexible and efficient use of network resources by segmenting a physical network into multiple logical ones. For example, a single physical router can serve as multiple virtual routers by hosting virtual machine monitors [67]. Each virtual router operates independently of the others, and can route packets using routing protocols of its own choice.

However, one still needs a way to remotely manage the creation, deletion, configuration, and peering of virtual routers. There is again a circular dependency: how can the virtual routers be remotely configured before the virtual network is functioning? The MMS solves this problem through its ability to bootstrap itself and automatically establish secure management communication channels between a DE and the NEs. Through the secure channels, the DE can remotely install the virtualization software on NEs and issue commands to set up and configure virtual routers.

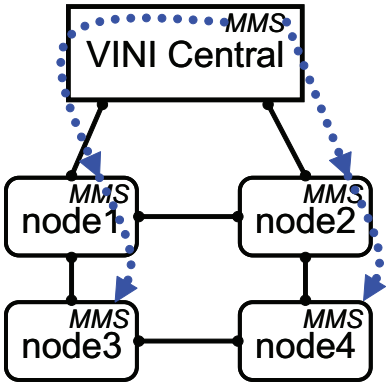

Figure 4.5: VINI virtualization system with the MMS. The solid lines indicate data links between VINI nodes within the network, and the dotted lines show the MMS channels between the VINI Central server and the VINI nodes.

To demonstrate the feasibility of our solution, we have experimented with a virtualizable network infrastructure on Emulab using VINI [68] as the virtualization software. Figure 4.5 shows the Emulab network with 4 physical nodes connected with 100Mbps links. VINI operates by creating a virtual *slice* on some or all of the VINI nodes, and by provisioning UDP tunnels between virtual slices to form a virtual network on which experimental protocols can be run and tested. The VINI Central server is designed to use `ssh` to remotely create and destroy virtual slices on the physical nodes. The MMS provides a standard socket interface, so the VINI Central server management code runs without modification.

At deployment time, the MMS software is installed on all the VINI nodes and the VINI Central server (which also serves as the DE). The MMS automatically establishes management channels between the VINI Central server and each VINI node (shown as dotted lines in the figure). Subsequently, from the VINI Central server, we are able to upload the VINI software image to the nodes and remotely bring up and configure the VINI software via the MMS. We find that it takes the VINI Central server 200 milliseconds to discover and authenticate the four VINI nodes, and another 4 seconds to upload the 16 MB VINI software image to the VINI nodes via the MMS. The MMS provides the needed management communication service in an integrated, secure, and automatic way.

4.4.3 Manage Wireless Mesh Networks

The MMS is well suited for managing wireless mesh networks. Nodes in a wireless mesh network often do not have any wired network connectivity, and the only way to remotely manage them is via the wireless mesh itself. The MMS provides precisely this capability. However, wireless mesh networks also pose interesting

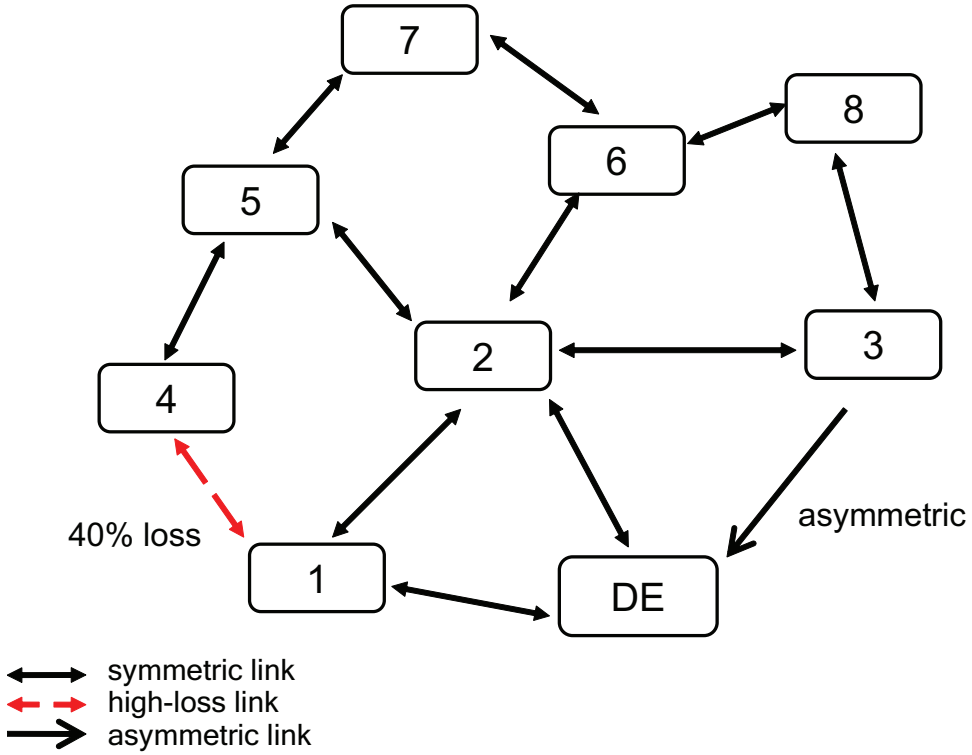

Figure 4.6: MMS management of an emulated wireless mesh network. The experiment shows MMS operating over lossy and asymmetric links that are common in wireless mesh networks.

challenges for the MMS due to the nature of the wireless medium. Wireless links can be asymmetric, and links can have unpredictable packet loss rates. To show the MMS's effectiveness in managing wireless mesh networks, we conduct an emulated wireless mesh network experiment.

Figure 4.6 shows an emulation of a wireless mesh network using Emulab. The DE and the NEs run on PCs with a 3GHz CPU, running the Linux 2.6.12 kernel. They are richly connected to each other to emulate a mesh topology. The Emulab traffic-shaping nodes are employed to induce 40 percent packet loss between NE-1↔NE-4, and an asymmetric simplex-link is set up between DE←NE-3.

When the DE and the NEs are first brought up, the DE detects its immediate neighbor NE-3 and tries to authenticate it using the asymmetric link, but fails. Meanwhile, the surrounding nodes of NE-3 are authenticated to the DE and provide alternate paths that the DE can use to reach and authenticate NE-3. Once all the NEs are authenticated, the DE has the full topology of the network and computes a source-route for NE-3 that avoids the asymmetric link. It takes 300ms to install a route on NE-3 that avoids the asymmetric link.

MMS handles lossy links in a similar way. It uses link quality estimates to detect links with high packet loss. In our experiment, the link between NE-1 and NE-4 is induced with a 40-percent packet loss. When the DE and the NEs are brought up, the DE may authenticate NE-4 via the lossy link or through its other neighbors. Meanwhile, the NEs use periodic HELLO messages to track the packet loss between their neighbors by measuring the time gap between individual HELLO messages. Once all the NEs have been authenticated to the DE, the NEs start reporting the packet loss estimates to the DE via the LSAs. The DE uses these estimates as link weights in its network topology. When the DE receives the LSAs from NE-1 and NE-4, it detects the poor quality of the NE-1↔NE-4 link and re-computes source routes for NE-4 to avoid using the lossy link. In the experiment, it takes 500ms to detect the lossy link and route around it.

4.5 Performance Evaluation

In this section, we evaluate the delay and throughput overhead introduced by the secure forwarding mechanisms in MMS, the convergence speed of MMS routing in response to failures, and the speed of the recursive authentication mechanism used to authenticate NEs during initial network bootstrap. The results show that the MMS has excellent performance, and it is practical to deploy the system.

Low forwarding overhead - This experiment measures the end-to-end delay and throughput overhead introduced by the MMS.

To measure the delay overhead, we connect nodes with 1 Gbps Ethernet links to form a linear chain topology. The sender and receiver exchange ICMP packets. We vary the hop count between the sender and receiver and compare round-trip delays for ICMP packets carried by the MMS and by the regular IP data channel. Figure 4.7 shows that the round-trip delays increase linearly with hop count, and the latency added by the MMS is less than 0.1 milliseconds per hop.

We measure the throughput overhead of MMS using a three-node chain topology, with a DE as the sender, one NE as the forwarder, and a second NE as the receiver. We use iperf [53] to measure the TCP throughput between the DE and the receiver. Using Emulab's configuration ability, we vary the bandwidth of the links connecting the three nodes. Figure 4.8 shows that the throughput difference between the MMS and the regular IP data channel becomes noticeable only after link bandwidth increases to 400 Mbps, and the best TCP throughput the MMS achieves is 800 Mbps. Investigating further, the performance degradation is

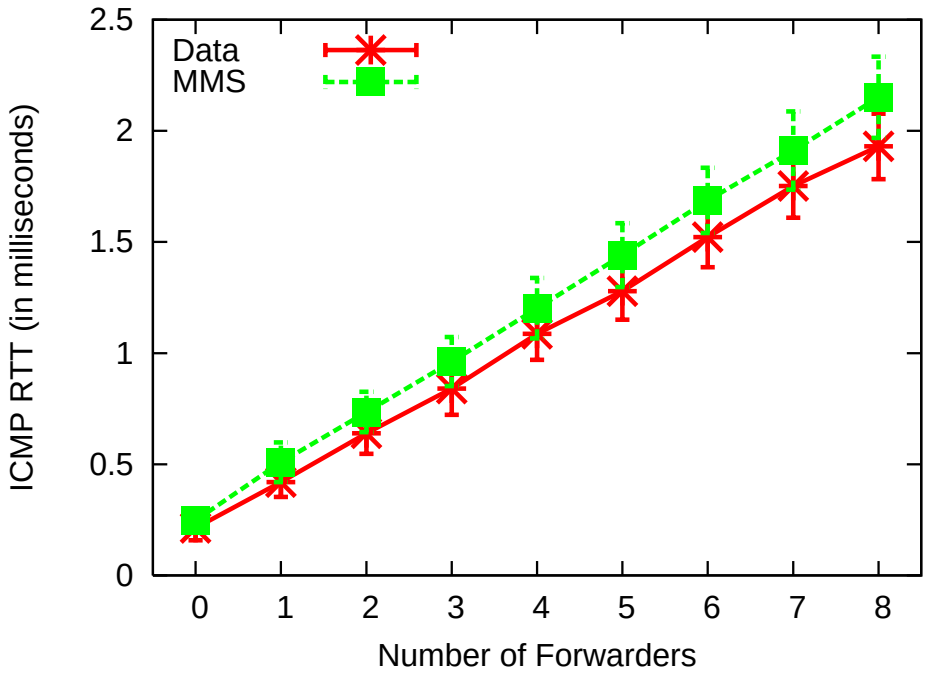

Figure 4.7: Comparison of round-trip delay of ICMP packets using MMS channel versus using regular IP data channel.

due to the encryption and decryption operations involved in using onion-encrypted source routes[6]. If cryptography is disabled, the MMS's TCP throughput is the same as the regular IP data channel. Nevertheless, the overhead imposed by the encryption is not large, and we believe security assurances made possible by onion-encrypted source routes outweigh the overhead.

Resilient routing - During network failures, NEs send LSAs to DEs, and DEs re-compute and push out updated onion-encrypted source routes to NEs. When multiple failures occur simultaneously, some LSAs may fail to reach the DE. To address this issue, the MMS requires NEs to keep sending LSAs until an acknowledgement from the DE is received or the DE's soft-state is timed out. To evaluate the ability of the MMS to maintain working communications in the presence of network failures, we construct the scenario as shown in Figure 4.9(a). In this scenario, two links fail at the same time, and the failure of link R1-R3 cannot be immediately propagated to the DE since neither end of the failed link has a working route to the DE. Table 4.1 shows a timeline of the steps taken during re-convergence. The DE first detects the failure of link R1-DE and commands R1 to re-route using R2. When R1's LSA reaches the DE and notifies it of the failure of link R1-R3, the DE obtains an accurate view of the network and repairs R3's route.

[6]Our implementation uses the "twofish" cipher with 128-bit keys.

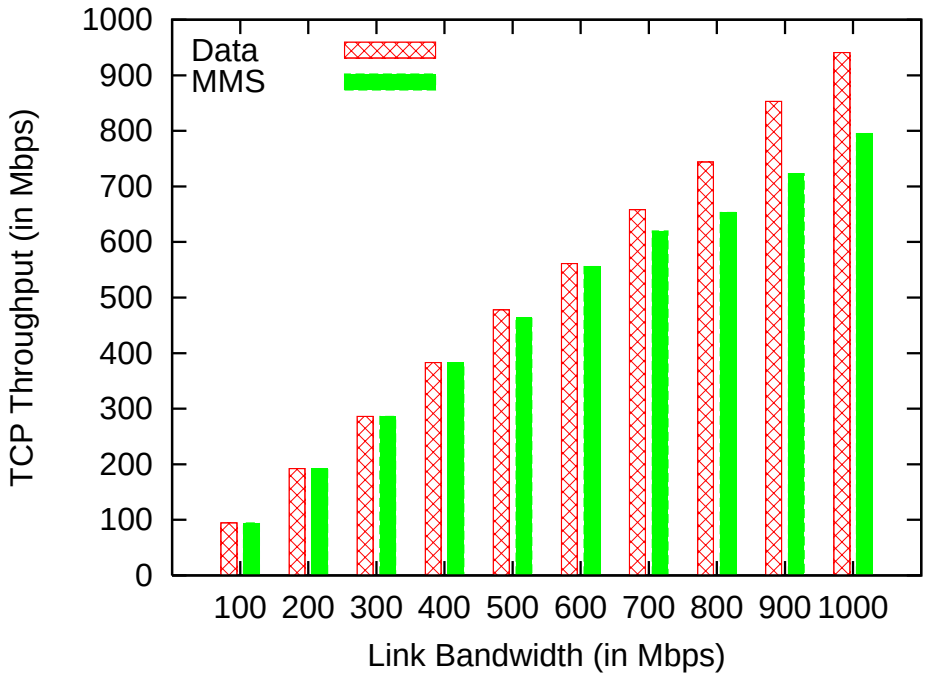

Figure 4.8: Comparison of TCP throughput using MMS channel versus using regular IP data channel.

In this case, one LSA retransmission is needed to update the DE with the accurate view of the network. Since the LSA retransmission timeout is 500 ms, it takes about 500 ms for the MMS routes to re-converge. We can recursively construct scenarios where more LSA retransmissions are needed. For example, two rounds of retransmissions are needed for the scenario in Figure 4.9(b) to re-converge. In Section 4.3.3.4, we proved that the MMS does eventually re-converge even after multiple failures.

Fast secure-channel setup - When a new DE is brought up, it first authenticates its direct neighbors and then recursively authenticates the network as described in Section 4.3.3.3. To estimate how long this process will take in networks of different sizes, we first develop a simple model of the authentication process and

Time line	Event
0 ms	R1 detects link failure & sends LSA to DE via route R1-DE, but LSA is lost
2 ms	DE detects the failure of link R1-DE, re-computes paths, and instructs R1 to use route R1-R2-DE
500 ms	R1 resends its LSA using the new route R1-R2-DE
505 ms	DE ACKs R1's LSA, re-computes paths, and instructs R3 to use route R3-R4-R2-DE

Table 4.1: Timeline of events triggered by concurrent link failures in Figure 4.9(a).

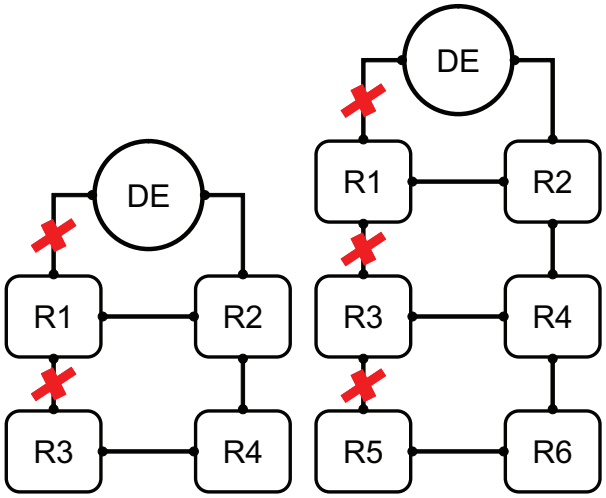

Figure 4.9: Topology of the resiliency experiment. In (a), R1 simultaneously loses two links, and its initial LSAs to the DE are lost; DE detects failure of the link to R1 and informs R1 to re-route through R2; LSA from R1 gets through, allowing DE to re-compute and push a new route to R3. In (b), three links fail at the same time. The DE restores R1's route, receives LSA from R1, restores R3's route, receives LSA from R3, and finally restores R5's route.

validate the model using experimental data. We then use our model to predict the time required to establish secure channels in large networks.

Consider Figure 4.10. Given a network of n nodes, we divide the nodes into groups based on their hop-count distance to the DE. We define the nodes in group d to be the nodes d hops away from the DE and the number of nodes in this group to be $s(d)$. We define D as the maximum d; H as the hop latency; C_{node} as the time for a node to answer a challenge from the DE; and C_{DE} is the time for the DE to verify an answer. In our model, nodes in group d are challenged after all nodes in group $d-1$ have been verified, and the time cost for authenticating nodes in group d includes the DE sending challenges to the nodes, the nodes answering the challenges, and the DE verifying the answers. Let the time when the DE is brought up be time 0 and $t(d)$ be the time when nodes in group d have been verified, we have

$$t(d) = t(d-1) + d \times H + C_{node} + d \times H + s(d) \times C_{DE}$$

Solving for $t(D)$

$$t(D) = D \times (D+1) \times H + D \times C_{node} + n \times C_{DE}$$

The importance of this equation is that it highlights that nodes with the same hop-count distance to the DE can compute in parallel, resulting in the term $D \times C_{node}$ and implying that the time to authenticate will

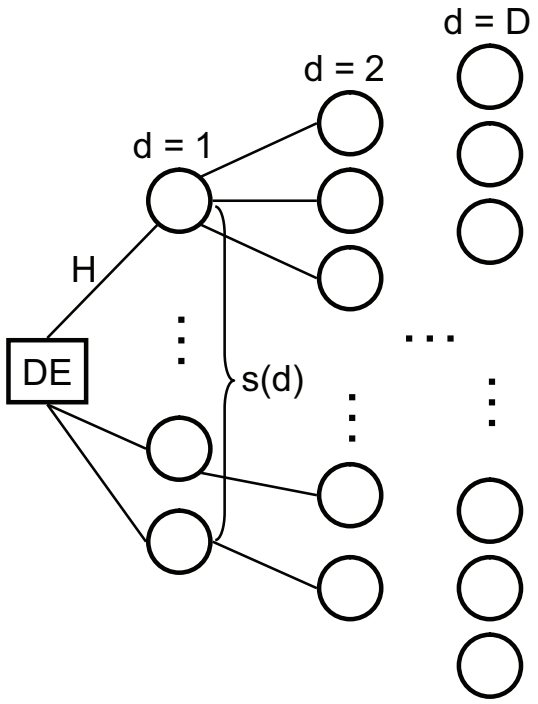

Figure 4.10: Model for computing secure channels setup time. NEs are grouped by their hop-count distances from the DE. d stands for the hop-count distance of a group, H is hop latency, $s(d)$ is the number of NEs in the group d hops away from the DE.

not be significantly affected even if network elements have slower CPUs than the DE and $C_{node} >> C_{DE}$. As shown in the equation, $t(D)$, the time the DE finishes authenticating and establishing secure channels to all nodes, is dominated by the term $n \times C_{DE}$ which grows linearly with the number of network nodes owing to the fact that the single DE has to verify answers from all nodes. And $t(D)$ is subjected to an offset bounded by the network diameter and average round-trip delay.

We conduct experiments to measure MMS channel setup time using three different types of topologies. The first is the Abilene backbone topology [69]; the second is an ISP backbone topology (AS 3967) derived from Rocketfuel [45] data; the third is a set of production enterprise network topologies used in [2]. Our measurements show that on the 3 GHz PC acting as the DE, C_{DE} is 27 milliseconds, and on the 800 MHz PCs serving as NEs, C_{node} is 45 milliseconds. Figure 4.11 plots the predicted and measured channel setup time for each topology. As shown, the measured times fit our analytical result.

According to the equation which we deduced and experimentally validated, a new DE plugged into a network with 1000 NEs will take only about 30 seconds to build secure channels to all NEs.

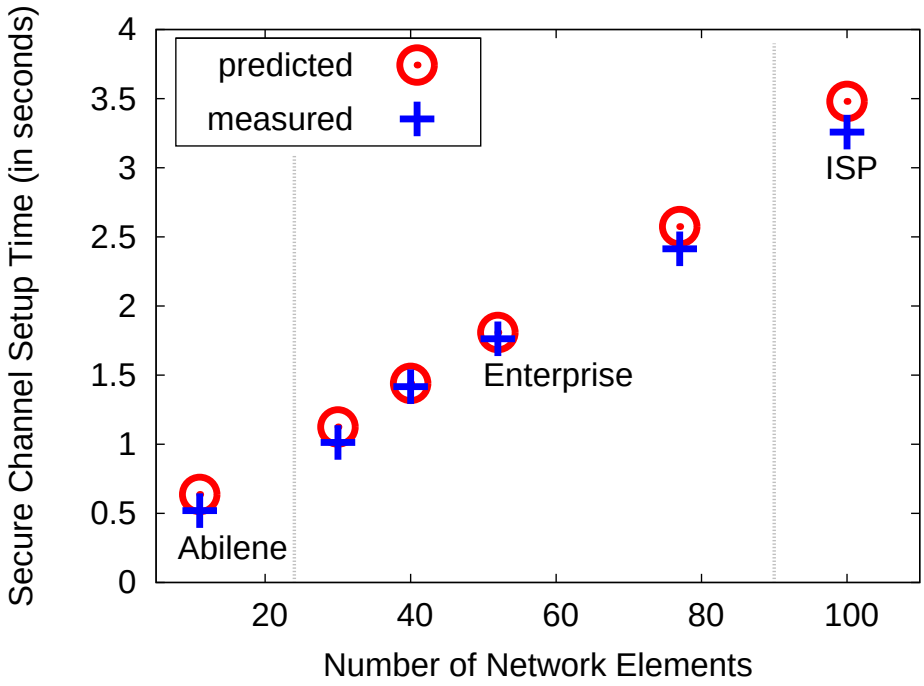

Figure 4.11: Predicted secure channel setup times (plotted as circles), and measured setup times for real topologies (plotted as crosses).

4.6 Summary

Providing robust remote connectivity to geographically distributed devices is a practical problem that begs for an economical and highly reliable solution. In-band solutions, while economical, are fraught with circular dependencies. Dealing with these dependencies requires carefully constructed operation procedures, where any small glitch or misconfiguration breaks the solution. Out-of-band solutions are also in wide use today. For example, IP network operators often dial in over the telephone network to terminal servers connected to routers for remote configuration. Similarly, frame relay switches, on losing frame relay connectivity, can "dial around the cloud," placing an ISDN call to a pre-configured number to replace lost connectivity. In essence, the out-of-band solutions fall back on older technologies, which are assumed never to fail. Therefore, these solutions bring several challenges: cost; the assumption that the telephone network is separate from the data network; and the fragility of depending on pre-configured information such as phone numbers.

In this chapter, we introduce the Meta Management System (MMS). The MMS takes the best from the two existing approaches today — leveraging whatever connectivity is available, in-band or out-of-band,

and unifying it into management communication channels with strong reliability, performance, and security assurances. At its heart, the MMS relies on source routing. Source routing provides scalability – the state required on each network element scales independently of the size of the network. Source routing also provides flexibility – management traffic can be directed to avoid existing problems when the network is overloaded or degraded. However, source routing introduces security vulnerabilities. The MMS overcomes these by using onion routing to detect and evict bad actors. Onion routing introduces potential overhead from cryptographic operations, but the MMS uses standard mechanisms to control these overheads. The performance evaluation of the MMS implementation reveals these overheads are insignificant.

Finally, the MMS includes APIs that allow the network operator or automated operations systems to recover devices and/or the network as a whole when in distress; in particular, through lightweight, general mechanisms for remote configuration and management of routing. The MMS implementation demonstrates the power and robustness of the solution. We argue that the MMS should be deployed ubiquitously – on every network element.

Chapter 5

Making 4D Dissemination More Scalable

Chapter 4 describes the management communication system — MMS. Although MMS solves the robustness problem of 4D dissemination, a big challenge remains: how to scale dissemination to up to thousands of routers and switches without slowing it down. This chapter describes a *SimCast* technique that we have developed to address this challenge. With *SimCast*, the dissemination time cost barely increases when the number of target routers and switches grows from hundreds to thousands.

5.1 Dissemination Bottleneck

Since the birth of the 4D architecture, there has been an emerging trend to move decision logic from network elements to a central server in order to make networks more manageable. Among the proposed implementations are Tesseract [9], RCP [11], Ethane [70], CONMan [27], and Maestro [71]. Despite their diversity in application domains and ways of computing network state, centrally controlled networks share the need for a dissemination service to distribute computed data-plane state (e.g., routing tables) from a logically centralized server to a large number of network nodes.

Existing centrally controlled networks employ different mechanisms to disseminate data-plane state. For example, in MMS a DE floods "path explorers" to explore paths between the DE and all the routers in a 4D control domain and uses source routing to reach the routers. Route Control Platform (RCP) relies on intra-domain routing protocols (e.g., OSPF) to set up routes between any two nodes within a network, and a Route Control Server (RCS) maintains an iBGP session with every single router and uses the sessions

for dissemination. Ethane creates a spanning tree rooted on the Domain Controller (DC), and the DC uses the tree paths to push flow tables to switches. CONMan is sufficiently flexible to accommodate any dissemination service with an open API. Although existing dissemination systems differ in mechanisms to establish dissemination routes, once the routes are obtained the server uses reliable unicast to push network state to every node. This unicast-based approach does not scale well with respect to network size because the dissemination cost in terms of both total traffic and transmission time increases in proportion to the number of network nodes. Previous work on centrally controlled networks has not addressed this issue of dissemination scaling.

In this chapter, we present a solution to the problem of dissemination scalability and efficiency. We leverage the key insight that data pushed to nodes across the network shares similarity when the data is network state. Three contributions are made around the following points.

First, we conduct a similarity analysis of routing tables downloaded from routers in a tier-1 ISP backbone network. Our analysis shows that there is significant cross-router similarity, and the similarity can be exploited to reduce dissemination traffic volume. An interesting outcome of the analysis is that the similarity pattern discovered from our dataset is different from those identified in multimedia files with changed header bytes [38], or documents/software across multiple versions [39]. This leads to the second contribution of this chapter: a method of leveraging the unique similarity pattern to "compress" data. The basic delta encoding method is explored and then extended with a variety of clustering algorithms. New methods of exploiting similarity are applied to our dataset, and substantial benefit is observed.

Third, a system called *SimCast* that incorporates our new similarity exploitation techniques is designed and implemented. When applied to a large backbone network topology, the SimCast system achieves a two-orders-of-magnitude improvement in scalability and efficiency over existing unicast-based solutions.

5.2 Related Work and Case for a Better Dissemination Service

Consider a hypothetical ISP backbone network with 1000 routers[1] where each router has 300K routing table entries[2]. Assume further that each routing table entry translates into approximately 40 bytes of data transferred over the wire. With unicast, the server must send each router its routing table. Thus the total

[1]The number of routers is based on the Rocketfuel ISP topologies study [45].

[2]The number of routing table entries is based on the CIDR BGP routing table report.

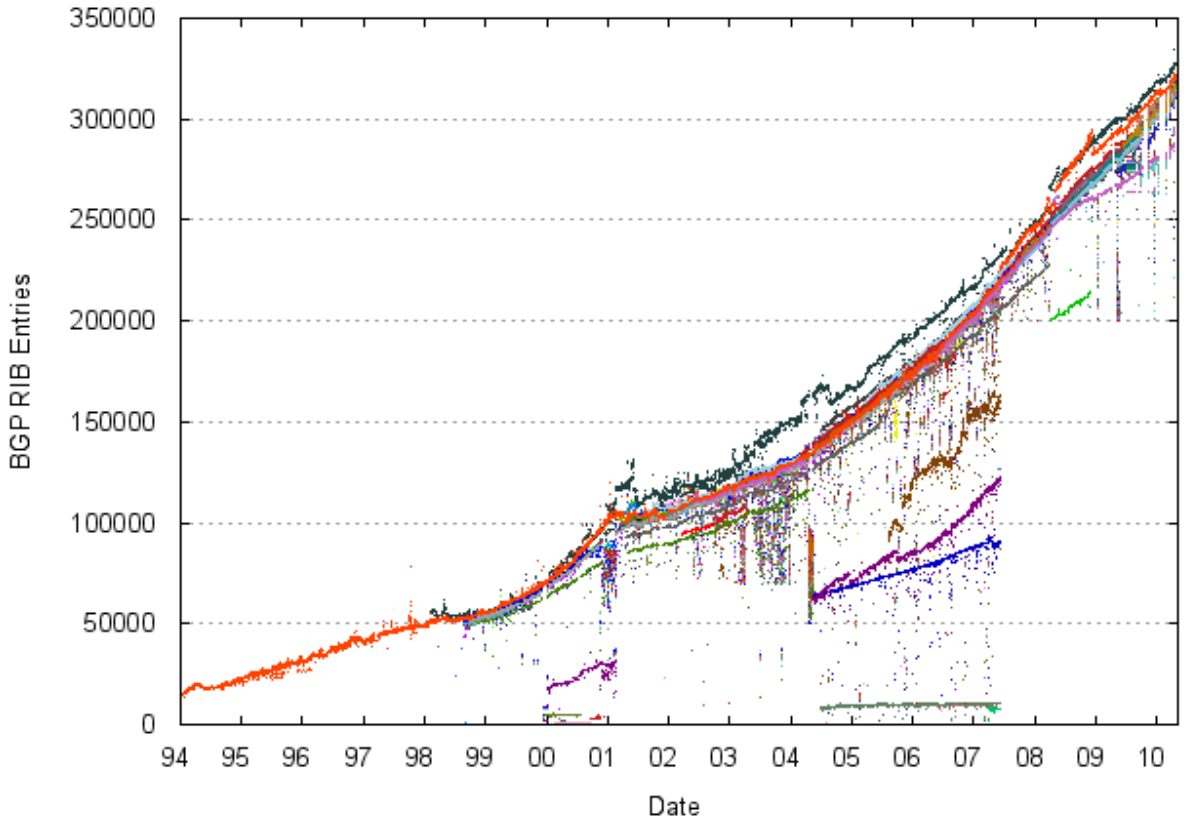

Figure 5.1: Growth of BGP routing table size. Data source: CIDR Report [72].

amount of data to disseminate would be $300K \times 1000 \times 40 = 12000$ million bytes. Assuming that we reserve 200 Mbps bandwidth for routing table dissemination, transferring this data would take 8 minutes. Another routing table distribution scenario is described in the RCP technical report [73], where the estimated time to transfer the complete route updates is about 3 minutes. The problem becomes more acute as the routing table size continues to increase, as shown in Figure 5.1 and studied by Bu et al. [74]. In order to achieve sub-second network convergence, we must dramatically cut the time for distributing the network state.

A similar dissemination problem was observed in network operation and brought up at a NANOG (North American Network Operators' Group) presentation [75]. Figure 5.2 shows the example used in the NANOG presentation to illustrate the problem. The example describes a typical PoP (Point of Presence) with core routers forming a full iBGP mesh and acting as route reflectors for the PoP's access routers. Suppose one access router loses a peering connection with UUnet (a tier-1 network) and sends withdraws to its local core routers. Since other UUnet peerings are in different PoPs, the core routers relay the withdraws to all the other core routers. As iBGP uses unicast, the number of withdraws that need to be pushed out is multiplied

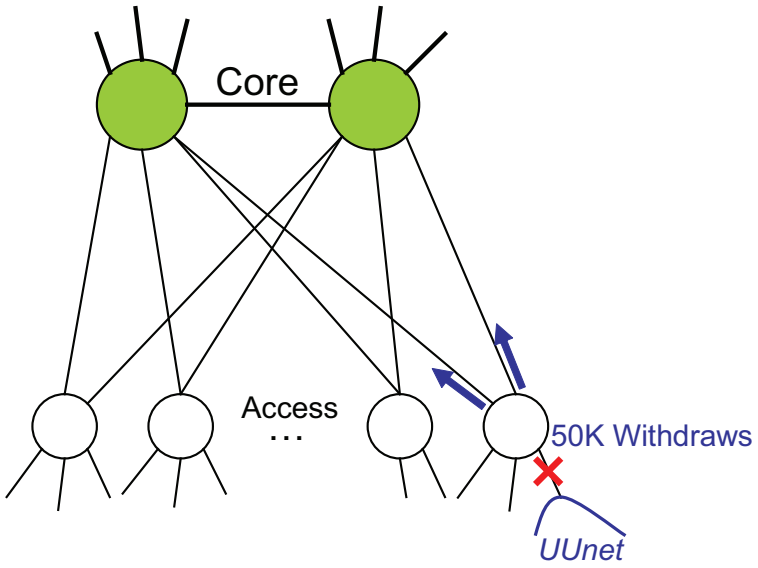

Figure 5.2: When one of the access routers loses a peering with UUnet, that router sends withdraws for 50K prefixes to its local core routers (2 million bytes of data). The core routers must send those withdraws to all the other core routers (100-200 of them). Unicast inflates 2 million bytes into 400 million bytes. Source: NANOG Presentation [75].

by 200. According to the presentation, propagation of the withdraws takes at least 1 minute in practice. Using route reflectors can reduce this effect, but the route reflector still needs to unicast multiple copies of the withdraws to its clients and/or peers.

Dissemination of either routing tables in centrally controlled networks, or route updates in traditional networks, can be very slow as it scales linearly with the number of routers. In this chapter, we propose a technique that exploits the characteristics of disseminated data to significantly improve scalability and efficiency. We focus on addressing the dissemination scalability problem for centrally controlled networks.

5.2.1 Off-the-shelf Solutions

Previous approaches for delivering content from one or few source(s) to a large number of recipients include:

Reliable unicast: The source establishes a reliable connection (e.g., TCP) with each individual recipient and sends data over the connection. The reliable connection can be set up on any routing infrastructure. For example, Tesseract uses source routing, and iBGP relies on IGP to set up intra-routing tables on routers. This approach suffers the inflation problem discussed above.

Reliable multicast: Multicast was invented to send a group of recipients the same data, so it is not directly applicable to our problem where different recipients might need different data.

P2P: P2P protocols such as BitTorrent are used to distribute large files to downloaders who desire the same content. In our case, data for recipients is not identical. SET (Similarity Enhanced Transfer) [38] is a variation of BitTorrent that allows similar objects to be utilized to speed up transfer. Unfortunately, as shown in Section 5.3 the similarity pattern of routing table data is different from that measured in SET, and thus a new solution is needed to achieve the desired effectiveness.

BST: Poduri et al. propose replacing TCP unicast with a new BGP transport called BST (BGP Scalable Transport) [75]. They chose to use application-level replication and flooding for multipoint transport. Their goal was to deliver one copy of the same BGP messages to all interested routers, whereas ours is to deliver similar routing tables.

Having demonstrated the critical role of a dissemination infrastructure, and the limitations of existing solutions, we believe it is both an intellectual and a practical contribution to conceive a technique to provide scalable and efficient data dissemination for centrally controlled networks.

5.3 Analyzing Similarity

We analyze and leverage similarity across BGP routing tables to explore the potential for reducing dissemination load. The reason for focusing on BGP routing tables is that they dominate the data-plane state on a router in terms of data size, and their sizes keep growing.

5.3.1 Dataset

Our dataset is a collection of BGP routing table snapshots captured from 71 routers in a tier-1 backbone network. The network contains more than 800 routers and crosses multiple continents. The 71 routers from which we collect data are distributed in different regions of the network, and most are route reflectors. Thus they adequately represent the diversity of the BGP tables across the whole network.

For each prefix in the BGP routing table, we only retrieve the best route selected by the router. The reason being that when a server in a centrally controlled network pushes out routing tables, it only disseminates the best route for each router. We end up collecting about 276K prefix next-hop pairs from each sampled router.

We use 8 snapshots of the routing table state in our study. Each snapshot is obtained by simultaneously downloading the BGP tables from the 71 routers at midnight on the first day of every month from January

to August. We primarily show the result of our similarity analysis on the latest (August) snapshot, but a comparison study is also conducted on all the 8 snapshots to show how the result changes over time.

At the same time that we collected the BGP tables, we also took a snapshot of the physical topology of the network. The use of the topology is described later in Section 5.4 and Section 5.5.

5.3.2 Metrics

The basic question we want to answer is to what degree the similarity exploitation can reduce the data a server has to push out to the network. To that end, we define a *compression ratio* metric to quantify the benefit of exploiting similarity.

Let:

S = The size of a routing table entry

C = The number of routing tables

N_i = The number of entries in routing table i

U = The number of unique routing table entries

The *compression ratio* is then the ratio of the sum of all routing table sizes over the size of the unique routing table entries:

$$Compression\ Ratio = \frac{\sum_{i=1}^{C} N_i \times S}{U \times S} = \frac{\sum_{i=1}^{C} N_i}{U}$$

The compression ratio quantifies server dissemination load reduction due to similarity exploitation. For unicast, the ratio is always 1. If the data sent to every receiver were identical, the ratio would be the number of the receivers. The higher the ratio, the more benefit similarity provides.

Although we use the compression ratio as the major metric in our analysis, we also apply another metric to our dataset called *parallelism gain*. *Parallelism gain* is proposed in SET [38] to quantify the number of different sources from which a receiver can download a particular object. The metric is defined as:

$$Parallelism = \frac{C}{\sum_{i=1}^{C} \frac{1}{S_i}} \tag{5.1}$$

where:

C = The number of chunks in a file

S_i = The number of sources for chunk i

The more similar the objects are, the higher the parallelism gain can be. High parallelism gain implies that a receiver can potentially download data from many sources and is thus more likely to saturate the download bandwidth. We are less concerned about parallelism gain than compression ratio because the former does not directly tell how much server bandwidth we can save by exploiting the similarity. The purpose of computing the parallelism gain on our dataset is to compare the result with that presented in the SET chapter to see whether our dataset has unique similarity patterns and whether existing similarity exploitation methodologies are effective on our dataset.

5.3.3 Similarity Patterns

Data similarity exhibits different patterns. For example, two revisions of a document or two releases of a software program can be very similar if not many modifications to the original copy are made, in which case the new copy can be obtained by applying the change logs to the old one. Two different MP3 music files can share a large fraction of common data if they have the same sound content but different meta information such as album and title. By replacing the header bytes that describe the meta information, we can transform one MP3 file into the other. Now consider two files containing exactly the same set of data records in different orders. If we sort the records and reorganize each file into two parts – a list of sorted records and its original ordering of the records – the difference between the two files only exists in the second part.

Similarity exploitation techniques that ignore the causes and characteristics of the resemblance have the advantage of being universally applicable. On the other hand, understanding from where the similarity arises and what the similarity patterns are can help us much more effectively exploit the similarity under certain circumstance. An analogy to this is audio/video compression, which is widely used and actively researched despite the existence of many mature general-purpose compression algorithms.

In the rest of this section, we analyze the similarity pattern of routing tables. We apply two similarity exploitation techniques to our dataset: one is the general chunk-based method, and the other incorporates

the knowledge about the structure of the data. We compute the metrics described in Section 5.3.2 and show the gain obtained from knowing the similarity pattern.

5.3.3.1 Chunk-based Similarity

Existing systems that exploit data similarity to speed up data transfer employ a chunking strategy. Such systems are exemplified by SET [38], CDF [76], and Shark [77]. They split files into chunks, and if files share identical chunks, downloaders can draw chunks from multiple sources simultaneously to increase parallelism and thus speed up data transfer. There are two ways of chunking a file: fixed block size or variable block size. The latter is almost always preferred because it allows similarities between mis-aligned objects to be detected. A common technique for discovering block boundaries is Rabin fingerprinting [78] which is used in systems such as LBFS [39] and SET [38]. Rather than using a static block size, Rabin fingerprinting walks through a file byte-by-byte and computes the hash of the file data covered by a sliding window with a fixed size. After the walk, N hash codes are generated for a file of size N bytes. Then, for a given expected block size, say B, $\log B$ bits of each hash are compared to a pre-selected value, and the place in the file where a match is found becomes a block boundary.

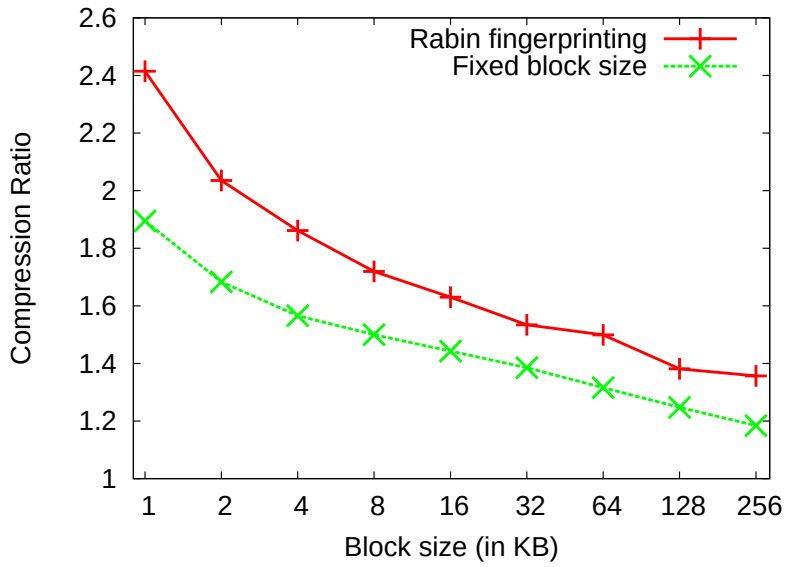

Figure 5.3: Ratio of the total number of blocks to the number of unique blocks after removing duplicates. Two chunking methods with varied chunk sizes are compared: Rabin fingerprinting and chunking with fixed block size.

With fixed-size chunking, adding one byte to the first block of a file can cause the new file to have blocks completely different from the original ones. Rabin fingerprinting solves this problem by dynamically adjusting boundaries and preventing changes from propagating from one block to the others.

We first examine how effectively chunking routing tables can produce identical blocks. Two existing chunking strategies are compared: chunk with fixed size, and chunk with Rabin fingerprinting.

We first preprocess the dataset. We save routing tables as binary files, each corresponding to a router's BGP table with about 276K active BGP route entries. The size of each file is about 3 megabytes, and we have 71 in total. We chunk the files into blocks and then write the blocks into a hash set so that duplicate blocks are removed. The *compression ratio* metric is thus calculated as the total number of blocks (including duplicates) divided by the size of the hash set that contains all the unique blocks.

Figure 5.3 shows the measured compression ratio as the chunk size increases from 1 kilobyte to 256 kilobytes. As expected, smaller chunk sizes yield better compression ratios. The chunk size used in SET implementation is 64 kilobytes, which corresponds to a 1.5 compression ratio. Rabin fingerprinting produces a compression ratio 30% to 50% higher than fixed-size chunking.

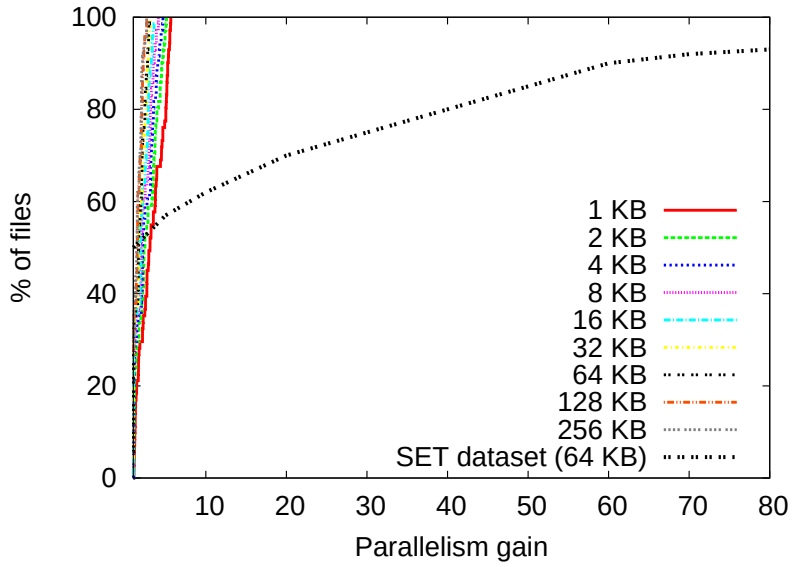

Figure 5.4: Parallelism gain of chunking with Rabin fingerprinting as the expected block size is varied. The gains over our routing table dataset are compared with the gain over the SET dataset which contains 6208 audio and video files downloaded from P2P networks.

To compare the benefit of data chunking over our dataset and other datasets, we also compute the *parallelism gain* metric specified in the SET chapter. We chunk the same 71 files into blocks and write the blocks

into a hash table to count how many times a unique block appears in different files. The occurrence count for chunk i in a file is denoted as S_i, and then we use Equation 5.1 to calculate the *parallelism gain* metric. Figure 5.4 shows the parallelism gain on our dataset with Rabin fingerprinting under various block sizes [3]. The parallelism gain reported in SET over their multimedia dataset is also incorporated into Figure 5.4 for comparison. As shown, Rabin fingerprinting works about 10 times better on the multimedia dataset than on our routing table dataset.

5.3.3.2 Structure-aware Similarity

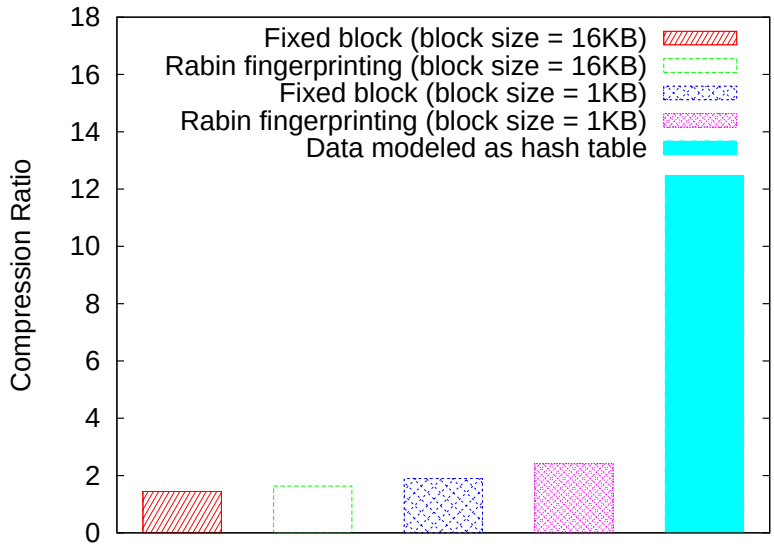

Figure 5.5: Ratio of the total number of pieces to the number of unique pieces after removing duplicates. Two block-based chunking methods (Rabin fingerprinting and chunking with fixed block size) are compared with a structure-based approach where data is modeled as a hash table of prefix next hop pairs. When routing tables are modeled as hash tables, the compression ratio is 12.4.

The low compression ratio and parallelism gain from Rabin fingerprinting on our dataset are not unexpected, if we consider the dataset collected in the SET project. When two MP3 files are similar, their differences concentrate on a small portion of the files (e.g., the header bytes). So when using Rabin fingerprinting to chunk the files, many large (multi-kilobytes) continuous common blocks can be found. However, when the differences between two files are spread across the files, Rabin fingerprinting becomes much less effective. Consider the following experiment. Write 300K IP addresses (in text format, e.g., 10.3.2.1) into a file, then randomly shuffle the IP addresses and write them into a second file. We know that if we do not

[3] Those are actually expected block sizes as Rabin fingerprinting generates blocks with variable sizes.

care about ordering, the two files carry the same information. But if we chunk both files with Rabin fingerprinting, we cannot find any identical blocks shared by the two files and we thus consider them two distinct copies of data. This experiment shows that not knowing the structure of the data can limit the effectiveness of block-based chunking techniques.

We thus seek a new chunking technique that can take the data structure as a hint and produce a better compression ratio than the de facto block-based techniques. Consider again the 300K IP address experiment described above. The new chunking algorithm takes as input not only the data file but also a function that splits the records (in this case the IP addresses). Now instead of treating the data as a block of bytes and trying to identify the shared chunks, the algorithm first extracts records from the data and then runs a set_difference function to find the difference between the record sets the two copies of data represent. Because the new chunking method leverages the structure information to exploit the data similarity, we call the new method "structure-aware" chunking.

We apply "structure-aware" chunking to routing table compression. We give the chunking algorithm the hint that the bytes representing a routing table can be deserialized into a key-value map where the key is the IP address prefix and the value is the next hop for that prefix. With the hint, the chunking algorithm can compare two serialized routing tables as two maps instead of two byte streams. As a result, the difference between the routing tables becomes map entries with the same key but different values or entries that exist in one map but not the other, and the compression ratio is now simply the sum of the map sizes divided by the size of the union of the maps. Figure 5.3 shows the compression ratio obtained by applying the new chunking method. Comparing with the block-based methods, the "structure-aware" approach enjoys a five- to six-fold improvement in compression ratio.

The similarity analysis in this section implies the potential benefit of leveraging "structure-aware" similarity across routing tables to lower server dissemination load. We conclude that "structure-aware" similarity exploitation brings out the best possible server load reduction we can achieve, and the result over the 71-router dataset provides a valuable guideline for evaluating solutions we will arrive at in Section 5.4 to cut dissemination load.

5.4 Leveraging Similarity

In this section, we present algorithms that leverage similarity to reduce the dissemination stress upon the server. We use the analysis result from Section 5.3 to help us evaluate how closely our solutions approximate

the optimum.

Number of Routing Tables	Average Table Size	Original Server Load	Size of the core	Average Size of a delta	New Server Load
71	276K	20M	276K	100K	7.3M

Table 5.1: The effect of reorganizing routing tables into a $core$ and a set of $delta$s. The table size and server load are both in the number table entries. Server load is reduced from 20M to 7.3M prefix next-hop pairs after the reorganization.

5.4.1 Basic Delta Encoding

The first solution we arrive at is to use delta encoding. It works as follows: find in all the routing tables the one, denoted as the $core$, that most resembles the others; for each routing table, iterate entries to find those that differ from the $core$ and mark those entries as $delta$; as the $core$ is shared by all routers, the server only needs to send out one copy of the $core$.[4] Assuming $delta$ is smaller than the original table, the server has less to send. An optimization to further shrink the $delta$s is to use majority voting to find the $core$: for each prefix, find the most popular next-hop, and put it in the $core$.

Table 5.1 shows the result of reorganizing routing tables into a $core$ and a set of $delta$s. We notice that the achieved compression ratio ($\frac{20}{7.3} = 2.7$) is far below the optimal ratio 12.4 shown in Figure 5.3. Thus we continue to seek more sophisticated algorithms to attain higher compression ratio.

5.4.2 Delta Encoding with Clustering

We define the *similarity* between two routing tables as the percentage of prefix next-hop pairs they have in common, and we define the *similarity distance* as $1.0 - similarity$. We make the observation that if we model routing tables as vertices of a graph and connect them with edges of *similarity distance*, those vertices tend to form clusters. This is unsurprising because routers located in the same area or configured with the same routing policies tend to select the same egress points for given prefixes, causing the *similarity distance* between their routing tables to be very short.

[4] This can be powered by existing one-to-many distribution technologies such as multicast and BitTorrent [37].

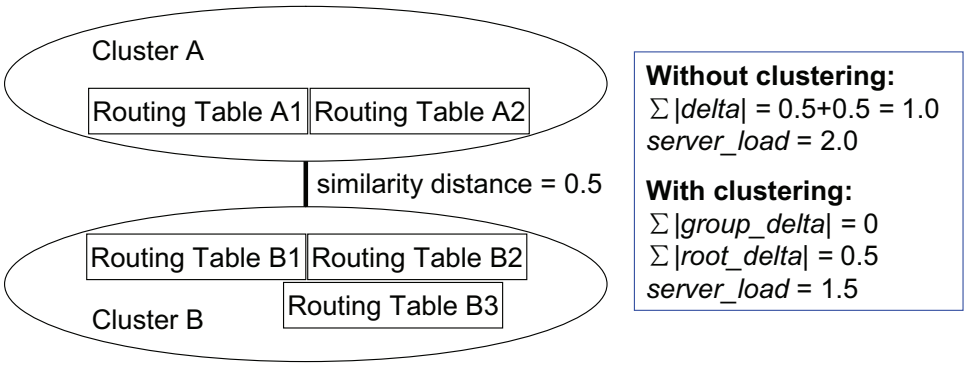

Figure 5.6: Example illustrating the benefit of clustering. The similarity distances between routing table A1 and A2 are negligible, and the same for B1, B2, and B3. The distances between A and B routing tables, however, are as large as 0.5. With basic delta encoding, the server generates the $core$ to contain entries in B routing table entries, and thus the size of $delta$ (denoted in the figure as |delta|) for every A router is 0.5, so the total server load is 2.0. With clustering, the server generates two $group\ core$s, one for each cluster; as no routing table in each cluster differs from its $group\ core$, the size of $group\ delta$ is zero, and thus the total server load amounts to the size of $root\ core$ plus the size of $root\ delta$ which is no larger than 1.5.

The observation intrigues us to extend the basic delta encoding with a single $core$ to a more complicated scheme with multiple $core$s. The extended delta encoding has three stages. First, routing tables are clustered into non-overlapping groups using clustering algorithms such as K-Means [79]. Second, within each group a $group\ core$ is generated using majority voting and then a $group\ delta$ is computed from the $group\ core$ for each group member. Finally, a $root\ core$ is produced by majority voting on all the $group\ core$s, and then a $root\ delta$ is deduced from the $root\ core$ and $group\ core$ for each group.

The intuition behind the above algorithm is illustrated by Figure 5.6. Suppose we have 5 routing tables perfectly clustered into two groups as shown in Figure 5.6. The clustering mechanism saves the server from sending two identical $delta$s to routers in Cluster A and thus reduces the server load from 2.0 to 1.5. The savings can be more remarkable when the cluster size is larger.

5.4.2.1 Clustering Algorithms

Two clustering algorithms are implemented to classify routing tables into non-overlapping groups.

K-Means K-Means [79] is one of the most widely used non-overlapping clustering algorithms. The input of K-Means includes: a set of points, a distance function that computes the distance between any two points

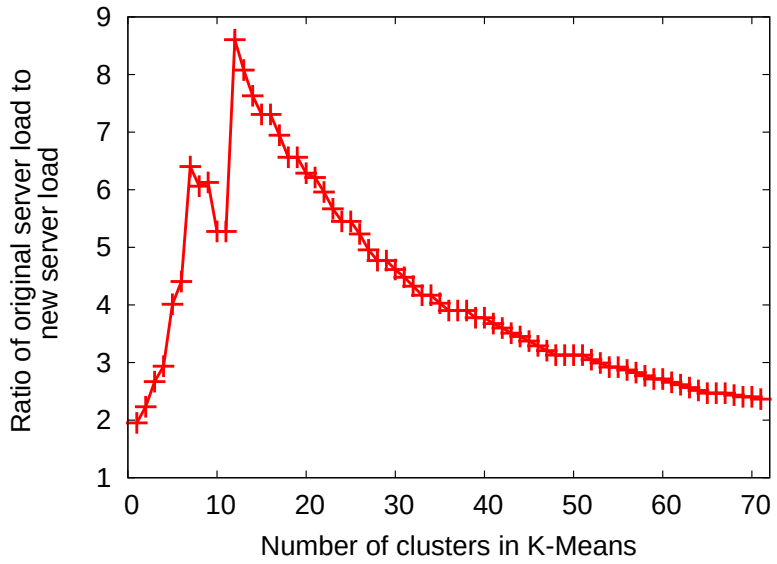

Figure 5.7: Compression ratio versus the number of clusters. The figure shows the load reduction benefit of applying the K-Means algorithm.

in the set, and an integer K that specifies how many groups the points are to be classified into. The output of K-Means is the group membership of every point.

When running K-Means over our dataset, each routing table is treated as a point, and the distance between them is simply the *similarity distance* we defined earlier. Given a K, K-Means initially randomly picks K routing tables as group centers. The algorithm then walks through every routing table and assigns it to the group to whose center it has the shortest *similarity distance*. After all routing tables have been assigned to groups, the group centers are re-computed by majority voting of all group members. Then the algorithm iterates all the routing tables again to re-assign routing tables according to their distances to the new centers. The procedure repeats over and over until no routing table switches groups.

We run K-Means with different K values and record the compression ratio (the ratio of the original server load to the server load after delta encoding). As shown in Figure 5.7, the compression ratio reaches a local maximum when K increases to 8, and it performs the best when the routing tables are classified into 12 groups. The highest achieved ratio is 8.8.

Randomized Greedy Heuristics We also use a randomized greedy algorithm to cluster routing tables. The idea is simple. We first generate a randomly ordered list of the routers. In order of its position in the list, a router creates a group and elects itself as the group head unless the router has been selected into a

group by a preceding router. The new group head then uses certain heuristics to drag qualified routers into its group. After every router has either become a group head or has joined a group, we compute the resulting compression ratio. This completes one iteration. We then generate another random ordering and repeat the above procedure. We keep track of the best compression ratio and its corresponding group assignment. The algorithm terminates and returns the best clustering solution when the compression ratio has reached a satisfactory value or when an iteration limit has been reached.

The following two greedy heuristics have been experimented.

Nearest first: Intuitively, we want routers geographically close to each other to be in the same group. The nearest first heuristic works as follows: given a distance threshold D, when a router is randomly picked to form a group, it drags routers within D hops away into its group. An experiment is conducted to test this heuristic. We vary D from 1 to 7. The program performs 100 iterations and reports the best compression ratio. The resulting compression ratio is shown in Figure 5.8. The heuristic produces a fairly good result only when it selects direct neighbors and neighbors 2 hops away. Figure 5.9 shows the number of groups generated by given distance thresholds, and it gives some clue about what happens when a particular threshold parameter is used. For example, when D is 1, too many groups are created, causing the compression rate to drop sharply. When D is larger than 2, it appears that routing tables clustered together are not really similar. In summary, as shown in Figure 5.8 the best achieved compression ratio by the greedy algorithm with the nearest first heuristic is 6.8 when up to second-degree neighbors are selected; while K-Means, as shown in Figure 5.7, achieves 8.8 compression ratio when it classifies routing tables into 12 groups.

Algorithm 1: The *most similar first with distance constraints* algorithm for clustering routers (and thus their routing tables).

Input: set of routers and their routing tables, similarity threshold s, distance threshold d

Output: list of groups G

```
1  b is the currently best compression ratio;
2  b = 0;
3  R is the set of routers not belonging to any groups;
4  while b not good enough and still have iteration quota do
5      Gi is grouping solution in this iteration;
6      Gi = ∅;
7      R = all routers;
8      L = list of randomly ordered routers;
9      foreach router r ∈ L do
10         if r ∈ R then
11             remove r from R;
12             t = the routing table of r;
13             group g = {r};
14             foreach router r2 ∈ R do
15                 t2 = the routing table of r2;
16                 if similarity(t, t2) > s and distance(r, r2) < d then
17                     append r2 to g;
18                 end
19             end
20             add g to Gi;
21         end
22     end
23     update b;
24     update G;
25 end
```

Most similar first with hop constraints: The second heuristic is more complicated. Two threshold parameters are given: one is hop distance D and the other is routing table similarity metric S. When a randomly picked router is not yet in any group, it walks through routers within distance D and for each calculates the similarity metric. If the calculated similarity metric is smaller than S, it pulls the router into its own group. The complete algorithm is shown in Algorithm 1.

As above, 100 iterations are performed for different similarity thresholds combined with hop constraints. Figure 5.10 shows the compression ratio this heuristic yields under different parameters. Again Figure 5.11 shows the corresponding number of groups. We find that the greedy algorithm has excellent performance when the "most similar first" heuristic is applied. It achieves the compression ratio of 10.4 when the similarity threshold ranges from 60% to 80%. The number of clustering groups corresponding to the best compression ratio is always 8.

Although the hop distance constraints, when set to be larger than 2, do not affect the compression ratio, it bounds the hop distance between the group head, and we describe the benefit of this in Section 5.5.4.

As the "most similar first" algorithm gives the best compression performance, we conduct two additional experiments to try to understand it better. We examine the distribution of the results over a larger number of iterations. We also run the algorithm on a dataset across months to find out how results vary over time.

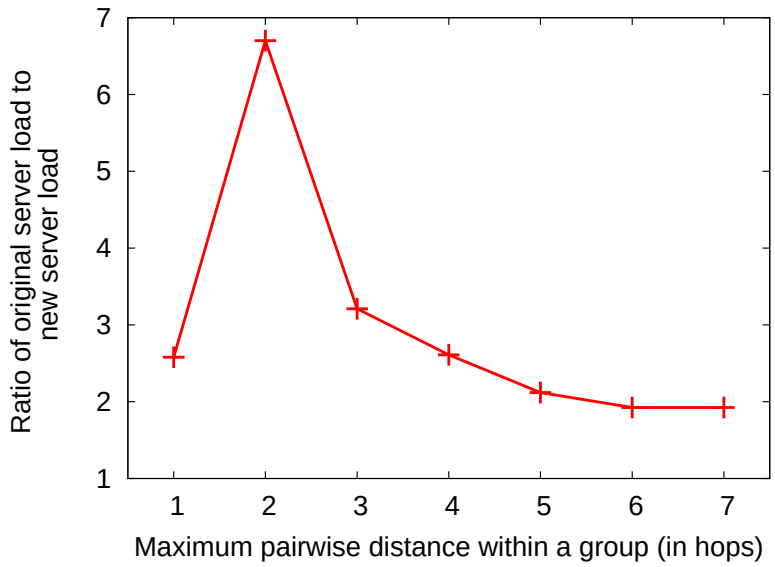

Figure 5.8: Compression ratio versus the clustering threshold (the maximal allowed distance). It shows the load reduction benefit of applying the randomized greedy algorithm with nearest-first heuristic.

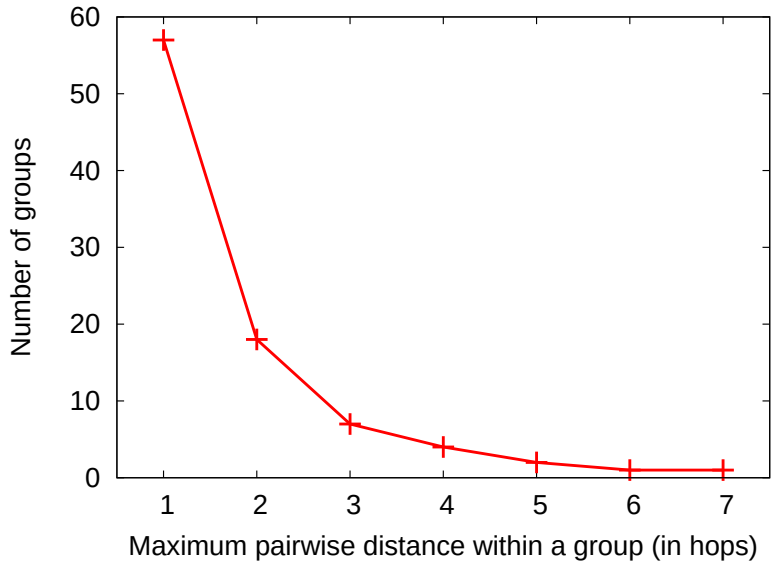

Figure 5.9: Number of groups generated by the nearest-first algorithm as the clustering threshold (the maximal allowed distance) is varied.

In the first experiment, we select 3 similarity thresholds and run the algorithm over 10,000 randomly ordered router lists. Figure 5.12 shows the cumulative distribution of compression ratio for the selected 3 thresholds. The threshold 70% is the apparent winner. Between compression ratio 8 and 10 the distribution is approximately uniform. Also ratio 9 corresponds to 50%. This implies that after 100 iterations have been performed, the chance that we have not seen a compression ratio higher than 9 is $(\frac{1}{2})^{100}$ which is extremely small.

In the second experiment, we run the randomized greedy algorithm with most-similar-first heuristic over the routing table data collected over 8 months. Again 3 similarity thresholds are compared. For each routing table snapshot captured in a given month, 100 iterations are performed, and the best result is shown in Figure 5.13. The 70% similarity threshold consistently yields the best results, and they are above 10.

5.4.2.2 Random Sampling

Running clustering algorithms is time-consuming, especially when the distance computation involves high dimensional vectors. Although Figure 5.13 shows that the clustering result is stable over time, and thus the clustering algorithm can run offline, a speed-up of clustering might be desirable and can be easily achieved. The solution we adopt is random sampling. Rather than running the clustering algorithms over the full routing tables, we first randomly sample a certain percentage of the route entries and use them for clustering.

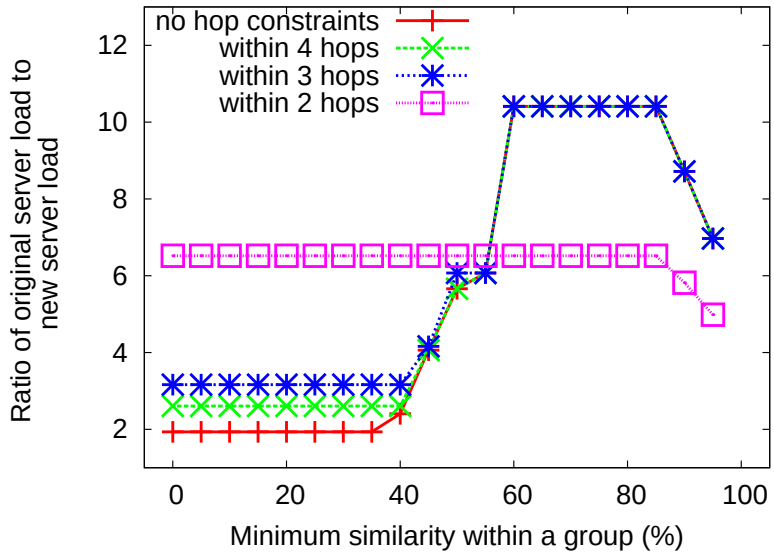

Figure 5.10: Compression ratio versus the clustering threshold (the minimal allowed similarity) under different hop distance constraints. It shows the load reduction benefit of applying the randomized greedy algorithm with most-similar-first heuristic.

Figure 5.14 shows the sampling error which represents the difference between the true similarity distance and the similarity distances computed using a randomly sampled portion of the routing tables. We find through experiments that at 10% random sampling rate, all the clustering algorithms discussed above output the same results as using the full routing tables.

Summarizing, we start with basic delta encoding and then extend it with various clustering algorithms. Experiments show that delta encoding with clustering outperforms the basic delta encoding. Among the clustering algorithms we have experimented, the randomized greedy algorithm with the "most similar first" heuristic produces the best compression ratio over our dataset. The running times of the above clustering algorithms are all within the range from 5 to 10 minutes depending on the sampling rate and number of iterations. Since we run the clustering algorithms offline and the running times of the three clustering algorithms in comparison are in the same scale, we use the produced compression ratio as the major winning criterion and pick the randomized greedy algorithm with the "most similar first" heuristic to implement in the system that we are to discuss in the next section.

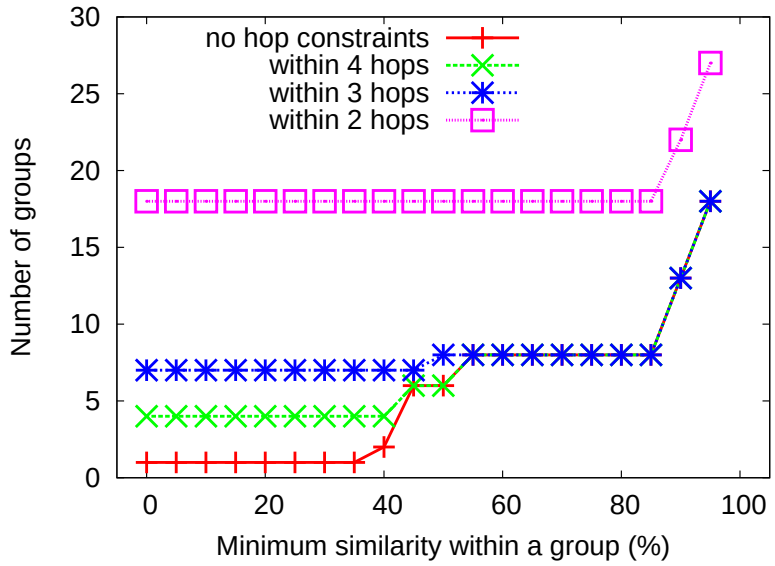

Figure 5.11: Number of groups generated by the most-similar-first algorithm as the clustering threshold (the minimal allowed similarity) is varied.

5.5 SimCast System

Since our similarity exploitation deviates from existing chunk-based ones, we are unable to directly use those off-the-shelf swarming protocols as transport. We thus design *SimCast* – a new dissemination system that builds on top of the similarity exploitation technique described in Section 5.4 to provide a scalable and efficient dissemination service for centrally controlled networks.

5.5.1 System Overview

The SimCast system is composed of a Client and a Server. Figure 5.15 illustrates the software organization of the SimCast system.

The Server has an offline Clustering Engine which from time to time runs the "most similar first" clustering algorithm described in Section 5.4. The Server interacts with the network decision server, such as 4D Decision Element or RCP Route Control Server, through a dissemination API. When the API is invoked to send data out, the Server encodes the data and distributes it via a mix of flooding and gossiping protocols. The next three sections describe those protocols.

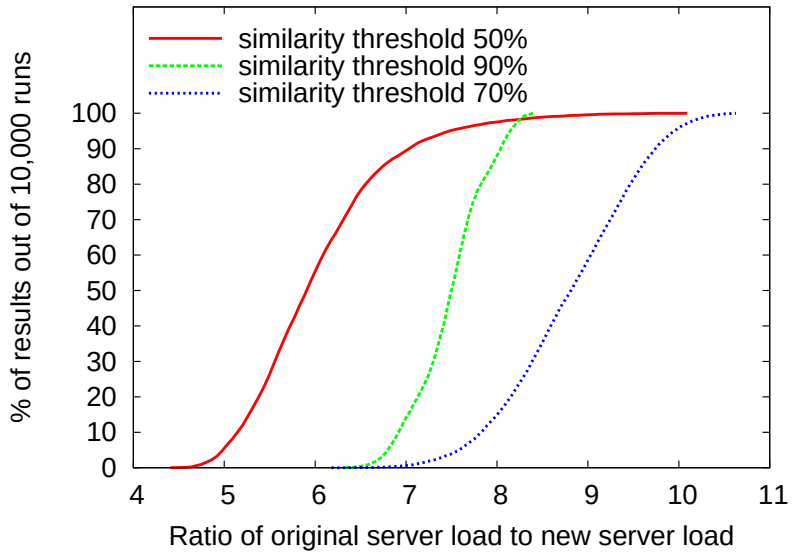

Figure 5.12: Cumulative distribution of compression ratio. 10,000 iterations of the randomized greedy algorithm with most-similar-first heuristic are performed under three different clustering thresholds.

5.5.2 Flooding

As described in Section 5.4, after "delta encoding with clustering", a server has four types of data to distribute: $root\ core$, $root\ delta$, $group\ core$, and $group\ delta$. The server first uses a variation of reliable flooding to spread the $root\ core$ together with meta information regarding each router's group membership. In addition to the normal flooding procedure, when a packet is flooded across the network, it keeps track of the number of routers it traverses, and when the packet reaches the receiver it reports the hop count to the receiver. We explain why this is useful in the next section.

5.5.3 Distance Guided Gossiping

After a group head, say h, receives the flooded $root\ core$, it sends a request to all its neighbors for more data. The requests carry the information of the hop distance between h and the server. When that request reaches a neighbor, the neighbor checks if itself is closer to the server than h, and if so it forwards the requests to its neighbors excluding h. When the request reaches the server, the server makes an offer to answer the request, and the offer is propagated back all the way to h. Finally, h accepts the offer, and the server sends data to h along the path that the offer and acceptance have traversed.

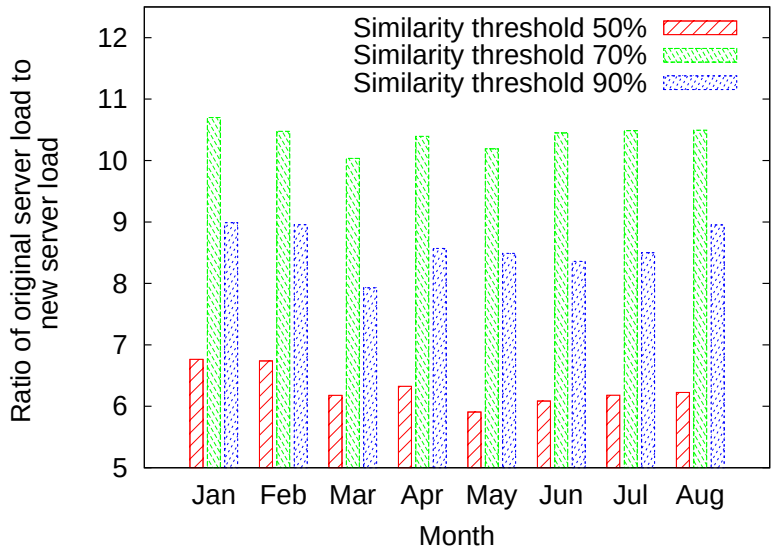

Figure 5.13: Comparison of compression ratios over 8 months. 100 iterations of the randomized greedy algorithm with most-similar-first heuristic are performed under three different clustering thresholds.

5.5.4 Bounded Flooding

Via gossiping, the group head gets the following data from the server: $root\ delta$, $group\ core$, and $group\ delta$s of all members in its group. The group head floods $group\ core$ within its group. The distance between the group head and the group members is bounded due to the "most similar first with hop constraints" algorithm, so the flooding stops whenever reaching the distance constraint, hence "bounded flooding". Finally, a group member gets $group\ core$ and uses the same distance guided gossiping protocol to get its $group\ delta$ from the group head.

5.6 Evaluation

In this section, we experimentally evaluate the scalability and efficiency properties of SimCast.

5.6.1 Methodology

We conduct experiments on our SimCast system designed and implemented for real deployment. We use Emulab [6] as the testbed. Due to resource constraints, we are neither able to reserve enough PCs or create an Emulab topology to match the physical topology of the tier-1 backbone network in our dataset. Instead,

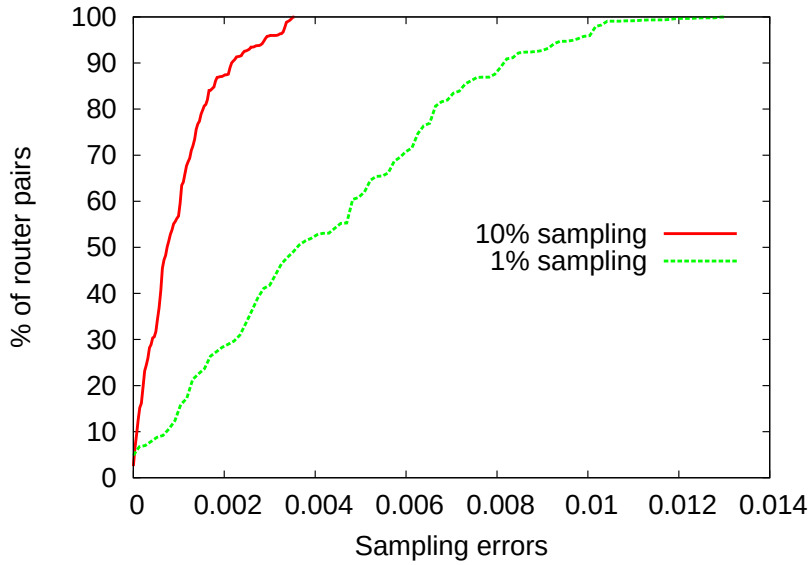

Figure 5.14: Sampling error is the difference between the true similarity distance and the similarity distances computed using a sampling of the routing tables.

we set up experiments on Emulab with the number of Linux PCs ranging from 30 to 100, and we create virtual interfaces on those PCs and run multiple instances of SimCast clients on each node. We use a mix of LANs and point-to-point links to connect the Emulab nodes to ensure our implementation works with both types of links. Application layer throttling is used to approximate communications between SimCast instances located on the same PC.

Since we need more routing table data to evaluate scalability, we synthesize routing tables based on our existing dataset. As our evaluation results are most affected by the similarity characteristics of routing tables, we try to ensure that the synthesized data resembles the real data in that perspective. For that, we

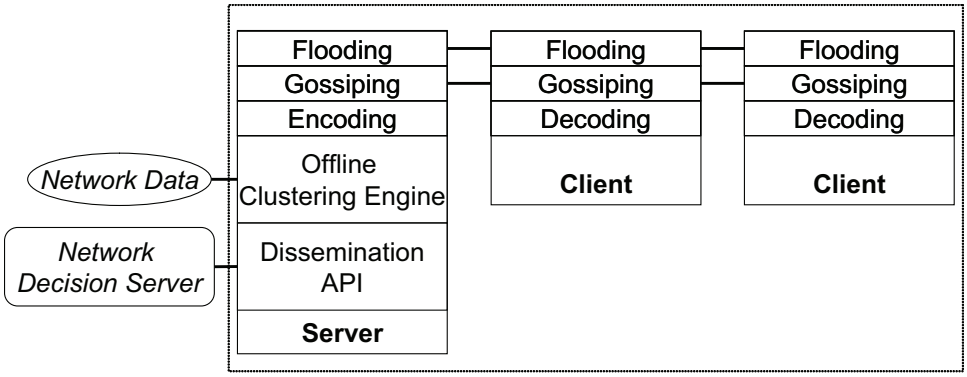

Figure 5.15: High-level overview of the SimCast system.

first discover the distribution

5.6.2 Scalability

One important goal of SimCast is to improve scalability: the size of router data-plane state a server has to push out should increase much slower than linearly with the number of routers in the network.

We take the existing 800-router topology and trim nodes and their incident edges to generate 17 smaller networks, with the number of routers ranging from 271 to 808. We run experiments on the 18 network topologies (including the original one).

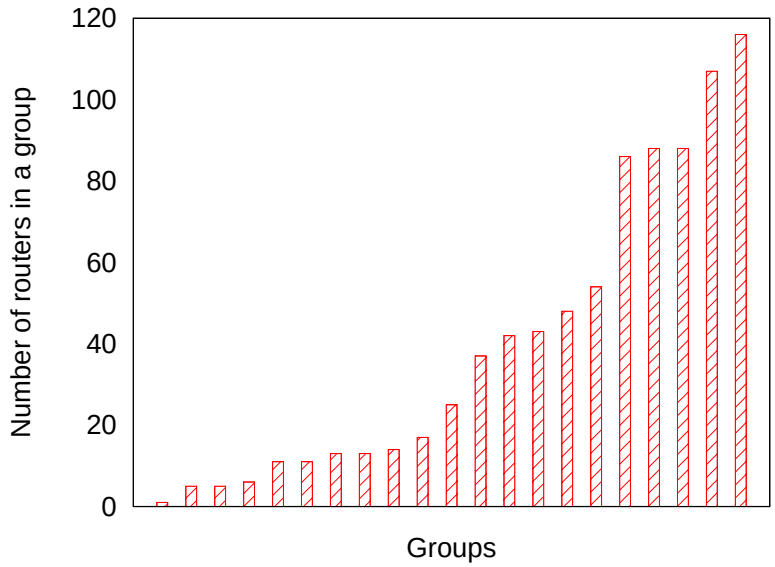

Figure 5.16: Distribution of group sizes. The routers are clustered into 22 groups. The number of routers within each group ranges from 1 to 116.

For each network topology, we first feed the routing table data (a mix of synthetic and real data) and network topology to the SimCast Clustering Engine to compute the clustering scheme. Figure 5.16 and Figure 5.17 show the characteristics of the clustering scheme. 22 groups in total are generated. The number of routers within each group ranges from 1 to 116. Within each group, the distance between routers and the group head is bound by 3. The cross-group distances roughly follow a normal distribution with a mean of 5.

We then measure the traffic volume coming out of the SimCast server, and compare it with unicast. As shown in Figure 5.18, unicast traffic volume increases linearly with the size of the network, while the increase rate for SimCast is much lower. To be more accurate, the slope of the former versus the latter is 6.8

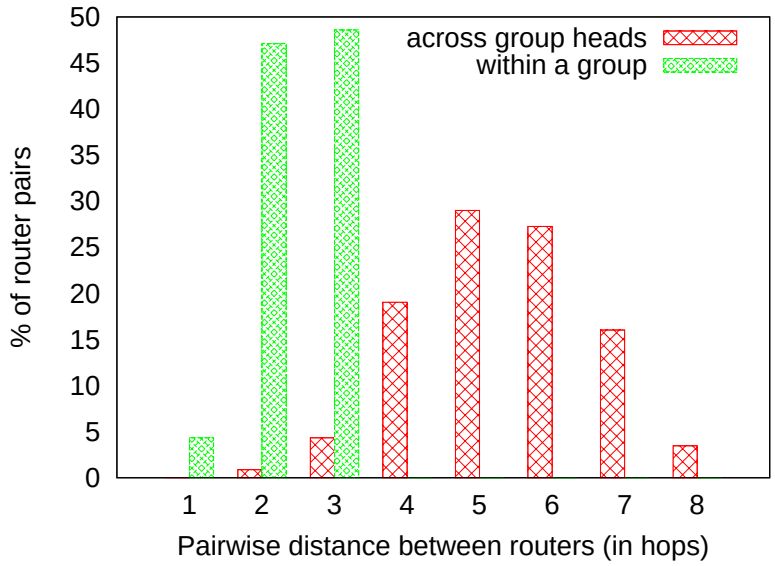

Figure 5.17: Distribution of router-to-router distances within and across clustered groups. For "across groups" the distance is the hop count between group heads; for "within groups" the distance is the hop count between a router and its group head.

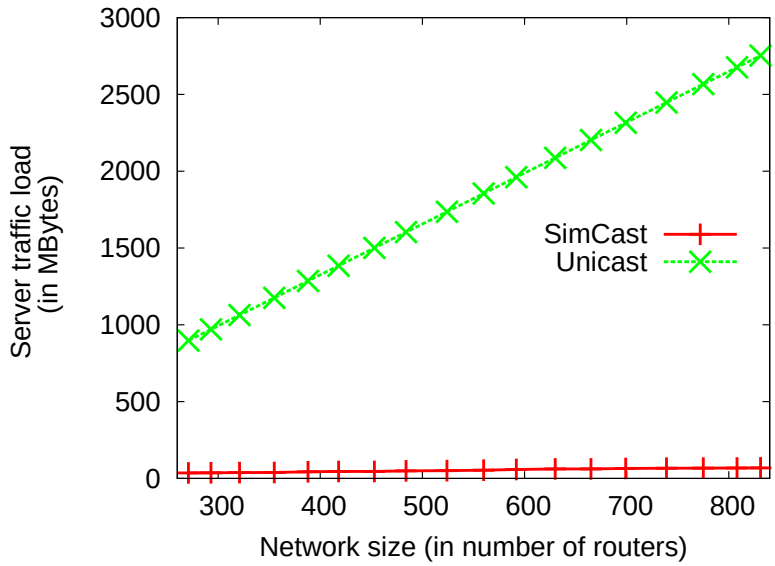

Figure 5.18: Comparison of server traffic load with SimCast and unicast as the network size increases.

versus 0.06 (megabytes/router). For the network with 831 routers, SimCast reduces server traffic load from 2.8 gigabytes to 69 megabytes.

5.6.3 Efficiency

For centrally controlled systems such as Tesseract and RCP, for the sake of resiliency it is crucial for the server to swiftly disseminate network state.

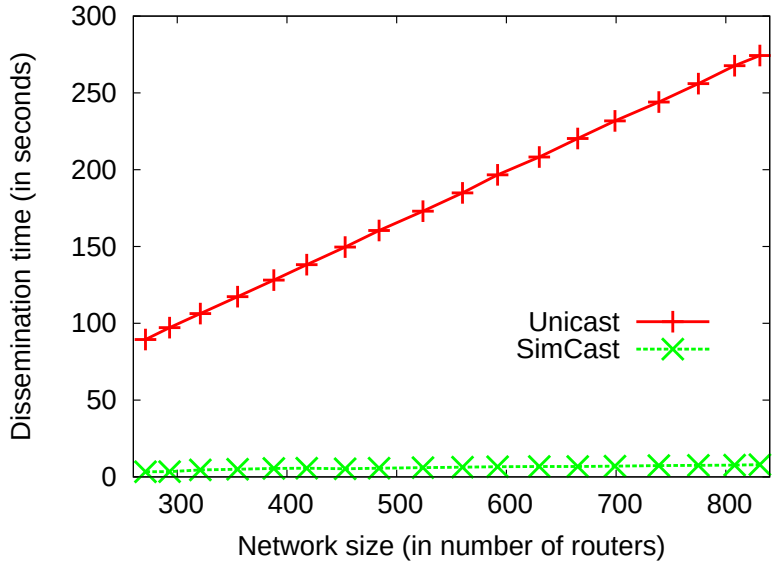

Figure 5.19: Comparison of dissemination speed of SimCast and unicast as the network size increases.

As above, we use networks with 18 different sizes to test the dissemination speed of SimCast. We run hundreds of SimCast instances on tens of Emulab nodes. A single server pushes the $core$s and $delta$s to all SimCast instances, and we measure the time from the push starts until all SimCast instances receive the data. Figure 5.19 compares the speed of SimCast with the speed of unicast in which a server simply `scp` the routing table files to all receivers. As expected, SimCast outperforms unicast significantly.

As mentioned in Section 5.6.1, due to resource constraints we are unable to reproduce the real network topology on Emulab. To compensate, we run a simulation on that topology to estimate the node traffic load during dissemination. For $core$ distribution, we use the flooding model; for $delta$ gossip, we assume the traffic traverses the shortest path from the source to the destination. The simulation is also done for unicast with a simple server-to-router shortest-path model. Figure 5.20 shows the node traffic load for unicast is much more unevenly distributed, and a small number of nodes have sky-high load, while the node load for SimCast is on average much lower and none of the nodes undergo traffic volume higher than 70 megabytes.

Experiments throughout this section demonstrate that SimCast greatly improves dissemination scalability and efficiency in centrally controlled networks.

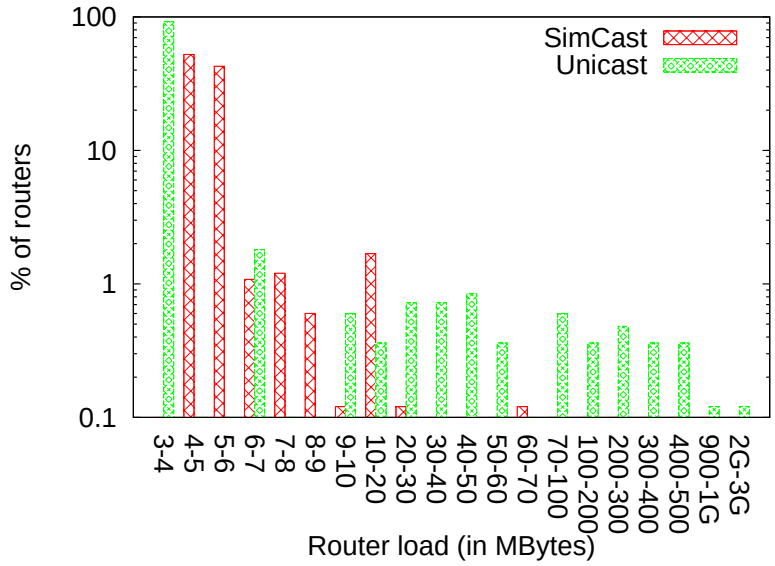

Figure 5.20: Comparison of router traffic load with SimCast and unicast as the network size increases.

5.7 Summary

Dissemination scalability and efficiency are among the major hurdles to the deployment of emerging centrally controlled networks such as RCP and Tesseract. This chapter addresses the dissemination scalability and efficiency issues by thorough analysis and effective exploitation of similarity across the dominant-sized data-plane state component – BGP tables. We argue for exploiting structure-based similarity, and we develop an effective delta encoding and data clustering scheme to leverage routing table similarity and achieve deep reduction of server load. We design protocols and build a prototype called SimCast to turn our findings and algorithms into a real dissemination system. Evaluation shows that SimCast enjoys remarkable scalability and efficiency improvements over existing solutions. In a tier-1 backbone topology with more than 800 routers, and in the case where routing tables need to be pushed to all routers, SimCast reduces server traffic load from 2.8 gigabytes to 69 megabytes and dissemination time from 274 seconds to 8 seconds. As the number of routers in the network increases, SimCast reduces the server load increase rate from 6.8 to 0.06 megabytes per router.

Chapter 6

Conclusion

In this chapter, we conclude the dissertation by (1) summarizing our contributions, (2) exposing limitations of current solutions, and (3) proposing directions for future work.

6.1 Contributions

Today's Internet must support objectives and capabilities far more sophisticated than best-effort packet delivery. Retro-fitting these network objectives on today's control-plane architecture has led to bewildering complexity, with diverse states and logic distributed across numerous network elements and management systems. The resulting complexity is responsible for the increasing fragility of IP networks and the tremendous difficulties that confront people who are tasked with understanding and managing their networks.

The research community is attempting to address fundamental questions central to improving IP control and management: How to transition from networks that blend decision logic with specific protocols and mechanisms, to an architecture that abstracts and isolates the decision logic and admits a range of efficient implementations. How to bridge from networks consisting of numerous uncoordinated, error-prone mechanisms, to networks where the low-level mechanisms are driven in a consistent manner by network-level objectives. How to advance from networks where people set parameters (twist knobs), hoping to coax the system to reach a desired state, to networks where designers can directly express controls that automatically steer the system toward the desired state. How to evolve from networks where human administrators leverage network-wide views and box-level capabilities at slow timescales in decision-support systems, to where the network itself leverages this information in real time.

In the field of data networking at this point, our community has a strong understanding of what is broken and how to create incremental solutions that only partially address the problems, at the expense of adding more complexity to an already unwieldy system. The contribution of this thesis work is to *explore breakthroughs in removing many of these problems altogether by providing a flexible, robust, and scalable platform that optimizes network-wide objectives in a more systematic way*.

This dissertation describes the basic centralized network control framework and presents solutions to the following design challenges that are central to achieving flexibility, robustness, and scalability:

Practical protocol decomposition (Chapter 3) A major drawback of today's architecture is that it has complicated decision logic distributed horizontally across the network elements and vertically across many layers. Our platform must achieve the same functionality as today's systems, while having the flexibility to introduce new capabilities through centralized decision computation. Using Tesseract, we have shown that it is practical and scalable to decompose a monolithic network protocol into decision logic and dissemination/discovery primitives. And we have also demonstrated that the decomposition makes network control more flexible.

Reachability paradox (Chapter 4) How can we provide reliable connectivity to remotely manage distributed network elements without relying on the communication services that are being managed? We have designed a robust *Meta-Management System* to break this circular dependency.

Dissemination scalability (Chapter 5) The largest networks today have thousands of routers/switches and tens of thousands of devices, and the default-free zone of today's Internet handles routes with hundreds of thousands of destination prefixes. Will the amount of management information being moved by the dissemination plane overwhelm the network's ability to carry data? We have presented a *scalable dissemination* technique that addresses this challenge.

Our centralized network control solution has already made an impact in the research and industrial communities. Since we published the first papers [8, 9], several new and interesting similar systems have been built. They include extensions to the decision server architecture [27, 71] and centralized network control solutions on specific networks such as enterprise networks [70]. In addition, there have been industrial efforts to build routers controlled by centralized decision planes.

6.2 Limitations and Future Work

While in this dissertation we have shown that the centralized network control platform is sufficiently flexible to implement different types of network functionality, and is reasonably robust and scalable, one important question remains: What are the limitations of our solution? In the next sections, we informally answer this question with regard to flexibility, robustness, and efficiency. We also identify several research directions for future work.

6.2.1 Programmable Decision Plane

A flexible decision plane is highly desirable in a heterogeneous network environment with a variety of requirements for security and quality of service policies. We have designed Tesseract so that different algorithms can run over an abstract logical network view composed of nodes and links; however, the algorithms are still tightly coupled with the Tesseract system. Adding or changing these algorithms requires non-trivial knowledge of the system itself.

We have begun to explore a programming interface for the decision plane [80]. Rather than embedding the control algorithms in the Tesseract system, we make them plug-ins that are interpreted by Tesseract. As network control applications are treated as plug-ins running on the decision platform, we need a language to define how the components are connected via their data and control interfaces, and how their network control state outputs should be composed. The composition language needs to be flexible enough to allow reasonably sophisticated network control compositions. The language should be capable of specifying composition ordering, prioritization of actions, and conflict resolution rules. For better performance, we might need to pre-compile the plug-ins into machine code rather than interpreting them at runtime.

Different plug-ins might have very different time complexity and thus might require different scales of running times. For example, finding the shortest paths in a network with hundreds of routers takes less than one second, while optimizing paths to balance load can take hours. Plug-ins might also differ in priorities: when a link fails, it is critical to change routes to bypass the failed link; but when traffic pattern changes, it is not as urgent to update routes, as long as the traffic pattern change does not cause network congestion. The decision platform that composes these plug-ins needs to provide interfaces for them to express their desired runtime support and schedule the executions accordingly.

6.2.2 Dissemination Error Handling

In MMS, if packets carrying management information through the dissemination plane get lost, they are retransmitted. However, retransmission of lost packets may not be the best policy. Instead, it may be better to notify the decision elements which packets were lost, and have them compute additional state updates, obviating the need for the information in the lost packets. In addition to the existing MMS system we will evaluate this possible alternative.

Most state change ordered by decision elements involves updating state on multiple routers. We currently use weak session layer semantics where each router independently applies an update as soon as it is received. To handle the error condition where not all routers receive the update, we will design and evaluate stronger semantics such as network-wide commit semantics that apply all received updates at a particular time, and full ACID distributed-commit semantics. We will also explore means of grouping related state updates into a single session "transaction" and methods for allowing multiple decision elements to send updates to overlapping sets of routers. We will evaluate the benefits of good time synchronization (e.g., through NTP or a GPS receiver at each router or PoP) to instruct the routers to change from one configuration to another at a specific time, resulting in infinitesimal convergence delay.

6.2.3 Redundancy Elimination in Dissemination

The SimCast algorithm scales dissemination by clustering similar state updates into groups to reduce traffic concentration around the decision servers. Recently, a class of packet-level redundancy elimination techniques has been proposed to remove redundant packets from traffic flow [81]. The basic idea of packet-level redundancy elimination is to make the upstream router cache packets over a period of time so that when it receives a similar packet it can send a delta to its downstream router to decode the new packet using the delta and its local cache.

The advantage of the packet-level redundancy elimination scheme over SimCast is that it is agnostic to semantics of any specific applications and can be built as generic network services. Although we believe that the knowledge of the application semantic helps us to further reduce the overlapping dissemination content and optimize the dissemination routes, we will quantitatively compare SimCast with the generic packet-level redundancy elimination techniques.

6.3 Final Remarks

In this dissertation, we have presented a flexible, robust and scalable solution that simplifies the control of networks and makes this important infrastructure more dependable. While it is difficult to predict the future course of research in this area, we believe that our pioneer work has created a new landscape of opportunities for networking researchers to deploy their ideas on real networks. Previously closed and proprietary control plane protocols will be replaced by software running on conventional servers. New algorithms and logic for network control can be developed and deployed.

Bibliography

[1] LAN/MAN Standards Committee of the IEEE Computer Society, *IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Common specifications Part 3: Media Access Control (MAC) Bridges*, 1998.

[2] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, August 2004.

[3] LAN/MAN Standards Committee of the IEEE Computer Society, *802.1Q IEEE Standards for Local and metropolitan area networks Virtual Bridged Local Area Networks*, 2003.

[4] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh, "Viking: A multi-spanning-tree Ethernet architecture for metropolitan area and cluster networks," in *Proc. IEEE INFOCOM*, March 2004.

[5] "Yipes." http://www.yipes.com.

[6] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. Operating Systems Design and Implementation*, pp. 255–270, December 2002.

[7] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, pp. 59–64, November 2004.

[8] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM Computer Communication Review*, October 2005.

[9] H. Yan, D. A. Maltz, H. Gogineni, Z. Cai, T. S. E. Ng, and H. Zhang, "Tesseract: A 4D network control plane," in *Proc. Networked Systems Design and Implementation*, April 2007.

[10] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2004.

[11] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and Jacobus van der Merwe, "Design and implementation of a Routing Control Platform," in *Proc. Networked Systems Design and Implementation*, May 2005.

[12] A. Chiu and J. Strand, "Control plane considerations for all-optical and multi-domain optical networks and their status in OIF and IETF," *Optical Networks Magazine*, vol. 4, no. 1, pp. 26–35, 2003.

[13] T. Russell, *Signaling System #7*. McGraw-Hill, 2nd ed., 1998.

[14] "Introduction to CCITT signalling system no. 7." ITU-T Standard Q.700.

[15] "Introduction to intelligent network (IN) capability set 1." ITU-T Standard Q.1211.

[16] "Cariden MATE framework." http://www.cariden.com/products/. Last visited 9/2005.

[17] "OpNet SP Guru." http://www.opnet.com/products/spguru/home.html. Last visited 9/2005.

[18] "Arbor Networks Peakflow." http://www.arbornetworks.com/products_sp.php. Last visited 9/2005.

[19] R. Chadha, G. Lapiotis, and S. Wright, "Policy-based networking," *IEEE Network Magazine*, vol. 16, pp. 8–9, 2002.

[20] A. Feldmann and J. Rexford, "IP network configuration for intradomain traffic engineering," *IEEE Network Magazine*, pp. 46–57, September/October 2001.

[21] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, "The cutting EDGE of IP router configuration," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2003.

[22] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. Networked Systems Design and Implementation*, May 2005.

[23] G. Varghese and C. Estan, "The measurement manifesto," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2003.

[24] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, and N. M. adn Scott Shenker, "SANE: A protection architecture for enterprise networks," in *Usenix Security*, August 2006.

[25] S. Rooney, J. van der Merwe, S. Crosby, and I. Leslie, "The Tempest: a framework for safe, resource assured, programmable networks," *IEEE Communication Magazine*, vol. 36, pp. 42–53, Oct 1998.

[26] C. Partridge, A. C. Snoeren, T. Strayer, B. Schwartz, M. Condell, and I. Castineyra, "FIRE: flexible intra-AS routing environment," in *Proc. ACM SIGCOMM*, 2000.

[27] H. Ballani and P. Francis, "CONMan: A step towards network manageability," in *Proc. ACM SIGCOMM*, (Kyoto, Japan), August 2007.

[28] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullmann, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, and J. Hartman, "A NodeOS interface for active networks," *IEEE J. Selected Areas in Communications*, March 2001.

[29] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Computer Systems*, August 2000.

[30] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter architecture," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[31] J. M. Smith and S. M. Nettles, "Active networking: One view of the past, present and future," *IEEE Transactions On Systems, Man and Cybernetics*, vol. 34, pp. 4–18, Feb 2004.

[32] "The openflow switching consortium." http://www.openflowswitch.org, 2008.

[33] "Nicira." http://www.nicira.com, 2009.

[34] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture." RFC 3031 (Proposed Standard), January 2001.

[35] E. Mannie, "Generalized Multi-Protocol Label Switching (GMPLS) Architecture." RFC 3945 (Proposed Standard), October 2004.

[36] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the Internet using an overlay multicast architecture," in *Proc. ACM SIGCOMM*, August 2001.

[37] B. Cohen, "Incentives build robustness in BitTorrent," in *1st Internation Workshop on Economics of P2P Systems*, June 2003.

[38] H. Pucha, D. G. Andersen, and M. Kaminsky, "Exploiting similarity for multi-source downloads using file handprints," in *Proc. Networked Systems Design and Implementation*, April 2007.

[39] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, (Banff, Alberta, Canada), October 2001.

[40] B. Aggarwal, A. Akella, A. Anand, P. Chitnis, C. Muthukrishnan, A. Nair, R. Ramjee, and G. Varghese, "EndRE: An end-system redundancy elimination service for enterprises," in *Proc. Networked Systems Design and Implementation*, (San Jose, CA), 2010.

[41] A. Shaikh and A. Greenberg, "Operations and management of IP networks: What researchers should know," August 2005. ACM SIGCOMM Tutorial.

[42] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," *J. ACM*, vol. 51, no. 6, pp. 968–992, 2004.

[43] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 1, pp. 612–613, 1979.

[44] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP) ." RFC 2048, November 1998.

[45] N. Spring, R. Mahajan, and D. Wetheral, "Measuring ISP topologies with RocketFuel," in *Proc. ACM SIGCOMM*, August 2002.

[46] "Quagga Software Routing Suite." http://www.quagga.net.

[47] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in IP networks," in *Proc. ACM SIGMETRICS*, June 2004.

[48] Z. Kerravala, "Configuration management delivers business resiliency." The Yankee Group, Nov 2002.

[49] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do internet services fail, and what can be done about it," in *Proc. USENIX Symposium on Internet Technologies and Systems*, 2003.

[50] A. Myers, T. S. E. Ng, and H. Zhang, "Rethinking the service model: Scaling Ethernet to a million nodes," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[51] R. Perlman, "Rbridges: Transparent routing," in *Proc. IEEE INFOCOM*, March 2004.

[52] K. Elmeleegy, A. L. Cox, and T. S. E. Ng, "On count-to-infinity induced forwarding loops in ethernet networks," in *Proc. IEEE INFOCOM*, 2006.

[53] "Iperf – The TCP/UDP Bandwidth Measurement Tool." http://dast.nlanr.net/Projects/Iperf.

[54] "Gigabit campus network design – principles and architecture." Cisco White Paper.

[55] G. Travis, E. Balas, D. Ripley, and S. Wallace, "Analysis of the "SQL Slammer" worm and its effects on Indiana University and related institutions," *Technical report, Advanced Network Management Lab, Indiana University*, February 2003.

[56] H. Gogineni, A. Greenberg, D. A. Maltz, T. S. E. Ng, H. Yan, and H. Zhang, "MMS: An autonomic network-layer foundation for network management," *IEEE JSAC Special Issue on Recent Advances in Autonomic Communications*, January 2010.

[57] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security*, 2004.

[58] E. Rosen and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)." RFC 4364 (Proposed Standard), February 2006. Updated by RFCs 4577, 4684.

[59] J. T. Buckwalter, *Frame Relay: Technology and Practice*. Addison-Wesley, 1999.

[60] J. Moy, "OSPF Version 2." RFC 2328 (Standard), April 1998.

[61] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do Internet services fail, and what can be done about it?," in *Proc. USENIX Symposium on Internet Technologies and Systems*, 2003.

[62] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Information Hiding*, pp. 137–150, 1996.

[63] X. Zhang, H. Chan, A. Jain, and A. Perrig, "Bounding packet dropping and injection attacks in sensor networks," Tech. Rep. CMU-Cylab-07-019, Carnegie Mellon, 2007.

[64] "netfilter/iptables." http://www.netfilter.org/projects/iptables.

[65] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer worm," *IEEE Security and Privacy*, vol. 1, pp. 33–39, July–August 2003.

[66] V. Moreno and K. Reddy, *Network Virtualization*. Cisco Press, 2006.

[67] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on operating systems principles*, (New York, NY, USA), pp. 164–177, ACM Press, 2003.

[68] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: realistic and controlled network experimentation," in *Proc. ACM SIGCOMM*, September 2006.

[69] "Abilene Backbone Network." http://abilene.internet2.edu/.

[70] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proc. ACM SIGCOMM*, (Kyoto, Japan), August 2007.

[71] T. S. E. Ng and H. Yan, "Towards a framework for network control composition," in *Proc. ACM SIGCOMM workshop on Internet network management*, (Pisa, Italy), September 2006.

[72] "CIDR Report." http://www.cidr-report.org.

[73] N. Feamster, J. Rexford, A. Shaikh, and Jacobus van der Merwe, "Route control platform architecture and practical concerns," *Technical report*.

[74] T. Bu, L. Gao, and D. Towsley, "On characterizing bgp routing table growth," *Comput. Networks*, vol. 45, no. 1, pp. 45–54, 2004.

[75] K. Poduri, C. Alaettinoglu, and V. Jacobson, "BST – BGP scalable transport." NANOG27 http://www.nanog.org/mtg-0302/ppt/van.pdf, February 2003.

[76] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, (Banff, Alberta, Canada), pp. 202–215, October 2001.

[77] S. Annapureddy, M. J. Freedman, and D. Mazières, "Shark: Scaling file servers via cooperative caching," in *Proc. Networked Systems Design and Implementation*, (Boston, MA), May 2005.

[78] M. O. Rabin, "Fingerprinting by random polynomials," Tech. Rep. TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, 1981.

[79] J. B. Macqueen, "Some methods of classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability*, pp. 281–297, 1967.

[80] Z. Cai, F. Dinu, J. Zheng, A. L. Cox, and T. S. E. Ng, "The preliminary design and implementation of the maestro network control platform," *Rice University Technical Report TR08-13*, OCT 2008.

[81] A. Anand, V. Sekar, and A. Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," in *Proc. ACM SIGCOMM*, August 2009.