Design of a Resilient and Customizable Routing Architecture

Karthik Kalambur Lakshminarayanan

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-30

http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-30.html

February 28, 2007

Design of a Resilient and Customizable Routing Architecture

by

Karthik Kalambur Lakshminarayanan

B.Tech. (Indian Institute of Technology, Madras) 2001
M.S. (University of California, Berkeley) 2004

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Ion Stoica, Chair
Professor Scott Shenker
Professor John Chuang

Spring 2007

The dissertation of Karthik Kalambur Lakshminarayanan is approved.

Chair Date

Date

Date

University of California, Berkeley

Spring 2007

Design of a Resilient and Customizable Routing Architecture

by

Karthik Kalambur Lakshminarayanan

Abstract

Design of a Resilient and Customizable Routing Architecture

by

Karthik Kalambur Lakshminarayanan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Ion Stoica, Chair

In today's Internet routing architecture, end-to-end paths are selected based on the complex interaction of several protocols. While these protocols operate at different levels, some within autonomous systems (ASes) and some across autonomous systems, all of them aim to provide customizability (e.g. traffic engineering within an AS or route choice across ASes), and resilience to failures of links and nodes. However, today's Internet's routing falls short of both these goals. Customization across ASes is at best indirect and is often achieved by tweaking knobs in BGP, the inter-domain routing protocol. Furthermore, the complexity of providing flexible routing makes failure recovery hard even with a single domain.

In this thesis, we propose an alternate routing architecture that decouples the goals of providing customizability and resilience. We propose a routing protocol based on a technique called Failure-Carrying Packets (FCP) that provides basic connectivity between hosts in the face of link failures. Sophisticated routing protocols, tuned to various specific needs, can be layered on top of FCP. We argue that such sophisticated routing should be provided as a service by third-party providers, and design a architecture in which such routing service provides can contract with ISPs and provide specialized routes to customers.

We present the design and implementation of both the underlying routing protocol (FCP), as well as the architecture for providing routing as a service.

Professor Ion Stoica
Dissertation Committee Chair

Contents

List of Figures

List of Tables

Acknowledgements

I am eternally grateful to Ion Stoica, my thesis advisor, for his guidance and support during my stay at Berkeley starting right from the day I arrived on campus. Ion has been a role-model and a tremendous source of inspiration. His belief that we can constantly better ourselves has been infectious and helpful, both during periods of ups and downs. I have learnt a lot from the many discussions we have had over the last six years; many a times, even when I failed to see his point due to my stubbornness, he has been patient and persistent. Ion's role in shaping me as a researcher goes much beyond teaching me how to solve problems, how to write papers or how to give effective talks; he taught me what good research is, and that research is about finding the right problem as much as solving it.

Over the years at Berkeley, Scott Shenker has been a great sounding board for my ideas. Scott's incredible clarity of thought and his ability to transform crude ideas into refined and polished statements within a matter of seconds has been as useful as it has been inspirational. The value of Scott's feedback and guidance on a range of topics—from high-level issues such as career advice and research directions to low-level issues such as how to organize a paper—cannot be understated.

I am grateful to John Chuang for not only graciously agreeing to be on my qualifying examination committee and thesis committee, but also giving useful feedback that improved the quality of my work.

Tom Anderson, along with Scott and Ion, has been instrumental in getting me to work on the problem of how to route around failures without propagating failure information, which eventually led to the design of the FCP protocol. Tom's critical and insightful feedback at various stages has been very useful in refining the problem and the solution. Many thanks to Matt Caesar and Murali Rangan for helping with the simulations and experiments for the FCP protocol, as well as providing much valuable feedback and ideas.

Dan Adkins and Adrian Perrig contributed a lot towards the chapter on securing forwarding infrastructures. Adrian's belief in the quality of the work, and his advice on

persisting with it despite several rejections at top conferences helped me tremendously in maintaining my focus when the chips were down.

I thank Jennifer Rexford for helping resurrect the idea of Routing as a Service after we had all but abandoned it. Her immensely insightful feedback and the new perspective she brought in helped in reformulating the problem and redesigning the architecture.

Venkat Padmanabhan, my mentor during my internship at Microsoft Research, taught me a lot about the value of being systematic and methodical when analyzing and solving problems. Venkat also gave valuable advice and feedback for the problems I worked later on.

I had the opportunity to work with Richard Karp during my first year at Berkeley; I learnt a lot not only from his deep insights but also his down-to-earth attitude.

I have also had the opportunity to work with and co-author publications with Tyson Condie, Michael Freedman, Brighten Godfrey, Dilip Joseph, Jayanth Kannan, Ayumu Kubota, Jitu Padhye, Anand Rangarajan, Ananth Rao, Sean Rhea, Sonesh Surana, Srinivasan Venkatachary, Michael Walfish, and Klaus Wehrle. I have learnt a lot from each of these experiences. I extend a special thanks to Michael Walfish for making me understand the value of writing and communicating well, and by setting an example for me to follow, he made it easier for me to appreciate his words to the fullest.

I thank all my office-mates and friends at Berkeley for their tremendous support and encouragement during my stay here. All of them gave great feedback for my papers any time I asked, despite the fact that I used to frequently bombard them with such requests.

My life at Berkeley was (in)appropriately spiced up by the company of Athulan Vijayaraghavan, Sriram Sankararaman and Sriram Balasubramanian (JJ), with whom I shared and learnt a thing or two about cooking. My house-mates at Berkeley—AP, Prashant, JJ, Adarsh and Kranthi—made my stay here memorable.

I am most deeply indebted to my parents and my brother for their love and support without which I wouldn't be what I am today. My mother instilled the perfectionist streak in me, and made me strive to achieve the best. Words can't express how thankful I am to her for the countless sacrifices she has made to make me a better person. My father taught me the value of pursuing higher goals in life, which has always stood by me in good stead.

I would like to express my gratitude to all who have given me musical company over the years, which has been a source of mental and spiritual uplift. My gurus, Smt. Suguna Purushothaman and Mullaivasal Sri. G. Chandramouli, have been of immense support in the last few years, as has been my brother Kaushik. I have enjoyed the musical sessions I have had with Krishna Parthasarathy, Ragavan Manian, Anil Narasimha, Prashant Subbarao, Divya Ramachandran, Sriram Raghavan, and Karthik Gopalaratnam. This list would be incomplete without mentioning the names of the stalwarts of yesteryears in whose music I have been immersed in for the last few years. To name a few doyens of Carnatic music that I have been inspired by: Semmangudi, GNB, Alathoor Brothers, Ramnad Krishnan, Musiri, TNK, and Lalgudi.

My sincere thanks to all those whose names I have missed by mistake. Perhaps I'll escape here by quoting Thyagarajaswami's catch-all acknowledgement: ”endarO mahAnubhAvulu andariki vandanamu”.

Chapter 1

Introduction

Routing in the Internet is based on a combination of several protocols, some operating across domains (such as BGP [90]) and some operating within a single domain (such as OSPF [3]). While these routing protocols differ significantly in their details, there are two important goals that they seek to achieve: (a) provide connectivity in the face of link and node failures and in the presence of malicious and misconfigured nodes, and (b) customizability of route selection. We argue that these goals are often opposing in nature; for example, providing route customizability makes recovering from link failures harder. Furthermore, we contend that customizability achieved by today's routing protocols is very limited in scope.

The most important goal of routing protocols is to provide *connectivity*, that is, two nodes should be able to communicate as long as there is a path in the underlying network. To ensure connectivity, routing protocols need to rapidly detect and recover from failures of links and nodes. BGP, the interdomain routing protocol used in the Internet today, could take on the order of a few minutes to converge to a new route after a failure is detected [67]. Even within a single autonomous system (AS), the fact that intradomain routing protocols (such as OSPF/IS-IS) do not converge fast enough has led to several works on using backup paths for fast failover [62, 40].

Fast failure recovery is only one aspect in ensuring connectivity; another important aspect is robustness to malicious or misconfigured nodes. Even a single misconfigured node can disrupt intra-domain connectivity to a large extent. Indeed, there are documented cases of a single misconfigured node causing disruption within an enterprise network [102]. Misconfigurations in BGP are much more globally visible—there have been several documented cases of BGP misconfigurations causing global connectivity issues [72].

Providing flexible routing—the ability to accommodate a wide range of policy and performance goals, and allow these goals to seamlessly change over time—makes the goal of rapid failure recovery harder. To allow flexibility, router vendors provide tens of complex knobs to configure routing protocols, such as OSPF and Cisco's EIGRP [2]. In fact, for OSPF, quite a few knobs need to be manipulated for just getting the basic protocol to work. This complexity not only makes these protocols prone to misconfigurations, but also complicates the failure recovery mechanisms. For inter-domain routing (such as BGP), the policy knobs are even more complicated and misconfigurations can even lead to persistent oscillations [90, 74, 72]. Since failure recovery is tied to the specific routing protocol, the recovery mechanisms are *constrained* by various policies and performance goals. For example, to prevent oscillations, link-state protocols are conservative in detecting link and node failures, thus increasing protocol convergence time after failures occur.

Customizability that current routing protocols achieve is limited, and at best indirect. In today's Internet, end-to-end path selection depends on the complex interaction between thousands of ASes, ranging from Internet Service Providers (ISPs) to enterprise networks [90]. Each AS controls traffic flowing through its infrastructure and cooperates with neighboring ASes to select paths to external destinations. Still, ISPs only have indirect control over the end-to-end path, typically by "tweaking" the routing-protocol configuration, making it difficult to offer meaningful service-level agreements (SLAs) to customers. An ISP's customers, such as end users, enterprise networks, and smaller ISPs, have even less control over the selection of end-to-end paths.

In this thesis, we ask the question of how one can design routing protocols that are customizable, yet resilient to failures of routers and links and misconfigurations of routers. To

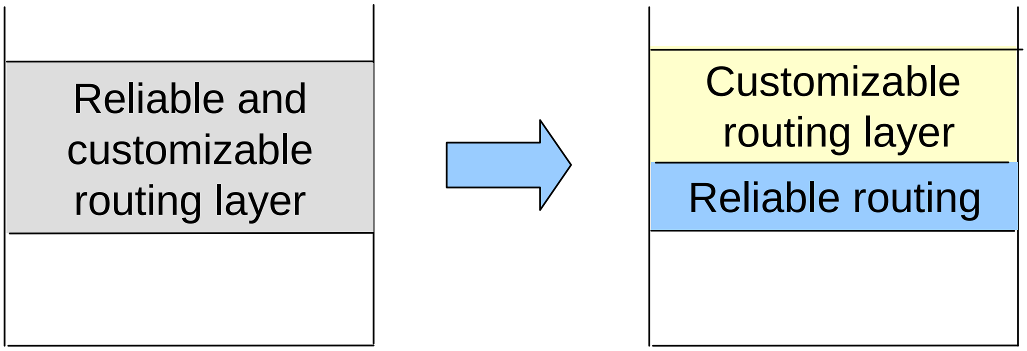

Reliable and customizable routing layer
Customizable routing layer
Reliable routing

Figure 1.1. Architecture for separating resilience and customizability goals: the lower layer provides resilience and customizability is provided at higher layers.

this end, we propose a different routing architecture: instead of designing a single protocol that provides both flexibility and connectivity, we *separate* these two goals. We propose a routing paradigm called Failure-Carrying Packets (FCP) that provides connectivity in the face of link failures, and prevents malicious or misconfigured nodes from arbitrarily subverting network traffic. Sophisticated routing protocols, tuned to specific needs, can be layered on top of FCP. In this way, designers of routing protocols can focus on the more complex demands while being assured that basic connectivity is still being provided by FCP.

1.1 Overview of a Layered Architecture

We propose a layered architecture (see Figure 1.1 that contrasts our approach with the current approach) that separates the resilience and customizability goals. Since providing connectivity in the face of failures and misconfigurations is the basic goal of routing, we believe that this property should be provided by the lower layer. Policy- or performance-based routing can be performed on top of FCP. Such a layering would imply that higher-layer customized routing would inherit the resilience properties of the lower layer.

Such an architecture raises three important questions:

- What are the technical underpinnings behind a scalable underlying routing layer that is resilient to failures of links and nodes?

- What is the architecture for providing customized routing? How do we resolve the tension between the local policies of multiple providers when constructing customized routes that span multiple providers?
- What is the interface provided by ISPs for creating such customized routes (that span domains) and how do we secure them?

We give a brief overview of our approach to solve these problems here; the subsequent chapters of this thesis are devoted to how we answer these questions in detail.

1.2 Part I: Providing Resilient Underlay Routing

We propose a different routing paradigm, called Failure-Carrying Packets (FCP) that guarantees that packets will be routed to their destination as long as a path to the destination exists in the network once a failure is detected locally.

FCP takes advantage of the fact that network topology in the Internet does not undergo arbitrary changes. In intradomain ISP networks and in the AS-level Internet graph there is a well-defined set of *potential* links (*i.e.,* those that are supposed to be operational) that does not change very often. Thus, one can use fairly standard techniques to give all routers a consistent view of the potential set of links. FCP hence adopts a link-state approach, in that every router has a consistent network map which is refreshed at coarse timescales.

To ensure routing resilience when links and nodes in this map fail, we propose a technique called FCP (Failure-Carrying Packets). In FCP, each packet carries in its header the failed links it encounters along the path. On receiving a packet, a node recomputes the path to the destination using its network map and the failed links contained in the packet. We present the detailed mechanisms in Chapter 2. Our main focus in the design of FCP is intra-domain ISP networks; however, we discuss how we can extend FCP to deal with BGP-like policy routing, as well as discuss how FCP can be leveraged to build robust managed distributed hash tables.

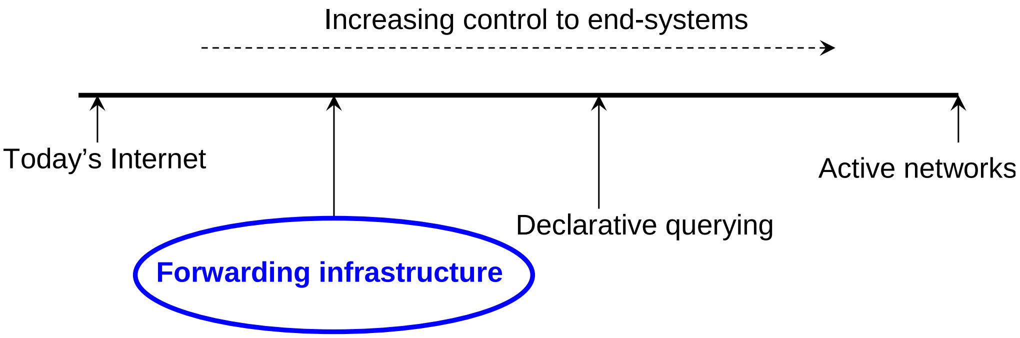


Figure 1.2. Spectrum of control given to end-hosts.

1.3 Part II: Providing Customized Routing

Rather than give control for providing customized routing to either the ISPs or the end users, we argue for exposing the tussle by allowing third-party providers to form business relationships with both users and ISPs, and select and install end-to-end forwarding paths on behalf of the users. Our proposal consists of three entities: the forwarding infrastructure (FI) that spans multiple underlying ASes, a collection of Routing Service Providers (RSPs), and the clients of the RSPs. We envision an RSP would buy *virtual links* (VLs) from various ASes with well-defined SLAs (something ISPs are quite willing to sell today), connecting some number of *virtual routers* (VRs). Our architecture meets the needs of both parties. The ASes still have sufficient control, in that they can limit the number and size of the VLs they sell and engineer the flow of traffic through their networks. The end-customers get the path performance they need, and do not have to deal with every AS along the path. We present the detailed architecture of RAS, and address the associated challenges in Chapter 3.

1.4 Part III: Providing a Secure Forwarding Interface for Customized Route Setup

One of the main challenges in the RAS architecture is the interface for allowing third-party providers to select routes across the underlying network. The interface must be flexible enough to allow a wide variety of routes to be selected, but at the same time ensure that this flexibility cannot be used to launch attacks on the network or the users. In the spectrum of providing increasing control to end-systems (shown in Figure 1.2), the current Internet lies at one extreme end that provides very little control, and active networks [123] lies at the other extreme that provides complete control. We propose a *forwarding infrastructure* model that lets third-party providers insert *forwarding entries* into the underlying infrastructure. Forwarding entries are similar to MPLS labels [97], but they are constructed in such a way that provably prevents several attacks on the network and its users. In Chapter 4, we present a detailed description of the forwarding infrastructure model with examples, and present the defense mechanisms, and show how they prevent a large class of attacks.

Chapter 2

Resilient Underlay Routing

Providing *connectivity* between nodes in the network is arguably the primary function of routing protocols; that is, two nodes should be able to communicate as long as there is a path in the underlying network. The two main issues that affect connectivity in networks today are failures (of routers and links) and misconfigurations. Even a single misconfigured node can disrupt intra-domain connectivity to a large extent. Indeed, there are documented cases of a single misconfigured node causing disruption within an enterprise network [102]. We contend that achieving fast-failure recovery is fundamentally hard in traditional routing paradigms that endure a convergence process to distribute failure state across the network.

Traditional routing paradigms—distance-vector, path-vector, and link-state—differ substantially in the nature of the state maintained by and exchanged between routers. However, all these paradigms rely on protocol messages to alert routers about changes in the network topology. It is only after the news of a topology change has reached all routers, directly in the case of link-state and indirectly in the case of distance-vector and path-vector, that the protocol can ensure that the forwarding tables define consistent routes between all pairs of nodes. Thus, all such routing protocols experience a *convergence* period—after the change has been detected and before all routers learn about the change—during which the routing state might be inconsistent.

While the convergence process is invoked whenever link costs change, link and router failures are the events that cause the most serious problems. They can cause losses [7] and, in some cases, trigger *LSA storms*, resulting in high CPU and memory utilization in routers and increased network instability [28]. Though the convergence period fundamentally depends on network properties such as the diameter of the network, it is exacerbated in practice due to system-level issues such as protocol timers.

The attempts to solve the failure-recovery problem in the literature can be roughly classified into three categories: (a) designing loop-free convergence protocols, (b) reducing the convergence period of protocols, and (c) using precomputed backup paths to route around failures. The first category of proposals involves protocol changes (such as ordering of LSAs [41]) to ensure that the convergence process does not cause transient loops. The second category involves reducing convergence period by tweaking protocol parameters (such as LSA propagation timers and periodicity of HELLO messages) [7]. These mechanisms often achieve lower convergence times but at the expense of additional overhead, and lower stability (as we show in some of our experiments). The third category deals specifically with link failures alone by precomputing backup paths for links which can be used when the link in question fails [60,86,62]. More recently, R-BGP [66] proposes using a simple precomputation-based backup for fast-failover during BGP convergence; R-BGP also provides provable guarantees such as loop-prevention. These backup mechanisms typically deal with the failure of single links gracefully; however, in order to provide guarantees for simultaneous failures of multiple arbitrary links, the number of precomputed paths needed is extremely high.

Using the state-of-the-art techniques, the convergence period can be eliminated for single failures, and more generally the duration and impact of the convergence period can be reduced. While these changes are quantitatively beneficial, they do not change the qualitative fact that these protocols could (due to multiple failures) endure a convergence period during when it is hard to provide routing guarantees.

Providing connectivity is further hampered by misconfigurations of routers. Router vendors provide several complex knobs to configure routing protocols, such as OSPF and

Cisco's EIGRP [2], to accommodate a wide range of policy and performance goals. This complexity not only makes these protocols prone to misconfigurations, but also complicates the failure recovery mechanisms. Since failure recovery is tied to the routing protocols, the recovery mechanisms are *constrained* by various policies and performance goals—for example, to prevent oscillations, link-state protocols are conservative in detecting link and node failures, thus increasing protocol convergence time after recovery.

In this chapter, we propose a different routing paradigm, called Failure-Carrying Packets (FCP) that *eliminates* the convergence period altogether. Once a failure is detected locally, packets are guaranteed to be routed to their destination as long as a path to the destination exists in the network.

FCP takes advantage of the fact that network topology in the Internet does not undergo arbitrary changes. In intradomain ISP networks and in the AS-level Internet graph there is a well-defined set of *potential* links (*i.e.,* those that are supposed to be operational) that does not change very often. The set of these potential links that are actually functioning at any particular time can fluctuate (depending on link failures and repairs), but the set of potential links is governed by much slower processes (*i.e.,* decommissioning a link, installing a link, negotiating a peering relationship). Thus, one can use fairly standard techniques to give all routers a consistent view of the potential set of links, which we will call the Network Map. FCP hence adopts a link-state approach, in that every router has a consistent network map.

Since all routers have the same network map, all that needs to be carried by the packets is information about which of these links have failed at the current instant. This *failure-carrying packets* approach ensures that when a packet arrives at a router, that router knows about any relevant failures on the packet's previous path. This eliminates the need for the routing protocol to immediately propagate failure information to all routers, yet allows packets to be routed around failed links in a consistent loop-free manner. We also present a variant called Source-Routing FCP (SR-FCP) that provides similar properties even if the network maps are inconsistent, at the expense of additional overhead in packet headers.

Though we primarily present FCP to introduce a new routing paradigm that is qualitatively different from previous approaches, we show through simulation that it has the potential to provide quantitative benefits as well. Using real-world ISP topologies and failure data, we show that the overhead of using FCP — in terms of computation, overhead in packet headers and stretch incurred — is very small. We also compare FCP with OSPF and show that, unlike OSPF, FCP can simultaneously achieve both low loss and low overhead. Finally, we show that compared to prior work in backup path precomputations, FCP provides much lower loss-rates while maintaining less state at the routers.

In the next section, we present an overview of the FCP algorithm. In Section 2.2, we present FCP in detail including several techniques that address efficiency issues in FCP. We present the map dissemination algorithm in Section 2.3. We present FCP primarily as a link-state protocol, and hence applies directly to intradomain and enterprise routing. However, we believe that the same idea can be used in the context of interdomain routing as well. To this end, we outline a strawman proposal for applying FCP to interdomain routing in Section 2.6; we leave a complete study of applying FCP to the interdomain context for future work. We finally present how the same idea can also be used to build robust managed distributed hash tables in Section 2.7.

2.1 Overview

In FCP, all nodes maintain globally consistent state about the system that gets periodically refreshed at very coarse timescales (*e.g.,* every few hours). However, such coarse-grained synchronization of state does *not* imply routing consistency since links and nodes might fail in between. The main challenge that FCP faces is maintaining this view consistent between refreshes, in the presence of link and node failures. In the remainder of this section, we provide an overview of how FCP works and how it addresses this challenge.

2.1.1 Providing Guaranteed Routing

The goal of this layer is to build a network that provides some routing guarantees based on the connectivity of the underlying network, *i.e.,* roughly speaking, if the network is connected during a time interval, then two nodes A and B can reach each other during that time interval. The main primitive implemented by this layer is sending a packet p to a node A, $send(A, p)$.

While FCP assumes that each node has a global consistent view of the network, the key aspect of our design that makes it scale better is *decoupling* periodic updates of global consistent views from routing. That is, global state is synchronized at coarse time-scales, and local repair algorithms ensure consistency of routing even when links and nodes fail. Hence, in a fairly stable managed environment, updates can be flooded once every few hours rather than every minute.

To achieve routing consistency when links and nodes[1] fail, we propose a technique called FCP (failure-carrying packets). The main idea is to have each packet carry in its header the failed links it encounters along the path. When a packet reaches a node that cannot forward the packet to the next hop because the link is down, the node will insert the failed link information in the header. On receiving a packet, a node recomputes the path to the destination using its network map and the failed links contained in the packet. While this basic description requires per-packet, per-node computation, we later present optimizations for reducing this overhead.

2.1.2 Distributing the Globally Consistent Map

An important problem we need to solve in our design is the dissemination of network map to all nodes. To address this problem, we assume the existence of a coordinator that builds a map of the entire network, and floods the map to all nodes. Assuming that the dissemination period is on the order of hours, this solution would easily scale to thousands

[1] We do not deal with nodes being added to the system; we assume that a new node can be added only when the map is updated.

of nodes. Our design of having a coordinator continues the recent trend of centralizing the route functionality as a way of simplifying the routing design. One example is RCP [22], that solves consistency issues arising from interaction of BGP with intra-domain protocols.

2.1.3 Challenges

In the rest of the chapter, we first present the design of FCP, and then address the following challenges.

- Computational overhead (Section 2.2.4): Whenever a packet carrying failure information arrives at a router, the router needs to compute new routes. We present mechanisms to reduce the computation overhead significantly.

- Map dissemination and updates (Section 2.3): FCP relies on all routers having a consistent view of the network map, which requires a map dissemination and update protocol.

- Quantitative performance (Section 2.4): While FCP's correctness properties might be theoretically appealing, to have any practical relevance FCP must be compared quantitatively to current OSPF performance, as well as backup path techniques that are commonly used in operational networks today.

- Deployment (Section 2.5): For FCP to have practical implications, the mechanisms should be deployable with minimal changes to current infrastructure. We discuss how we can leverage currently deployed mechanisms (such as MPLS), and several earlier proposals (such as RCP) to achieve our goals.

- FCP extensions (Section 2.6): Since we present FCP as a link-state routing protocol, it is directly applicable only in the intradomain context. We discuss how FCP can deal with incomplete maps and with policy constraints needed for interdomain routing.

2.2 Failure-Carrying Packets

In this section, we introduce the FCP algorithm and its properties using a simple network model, similar to many intradomain settings, where routers use link-state routing.

With FCP, all nodes (we will use the terms router and node interchangeably) in the network maintain a consistent *Network Map*, which represents the link-state of the entire network; we will relax the map consistency assumption in Section 2.2.2. In the absence of failures, FCP reduces to a link-state protocol; when there are failures, FCP behaves quite differently, as we now explain. For the purpose of our discussion, we assume that all nodes know the network map, and, unless otherwise specified, we assume that this map does not change. We discuss how the network map is disseminated and updated in Section 2.3.

Each node maintains liveness of adjacent nodes in the map using periodic keepalive messages[2]; however, this information is maintained locally, and not propagated into the routing tables of other nodes.

2.2.1 Basic FCP design

The main intuition behind FCP is that it is enough for a router to know the list of failed links in the network, in addition to the network map, to compute the path to a destination. FCP uses the packet header to gather and carry the list of failed links encountered while routing *that* packet. As we show later, the packet need only carry the failed links that the packet has so far encountered along its path, not all failed links in the network, in order for this to work. Thus, the number of failures carried in any packet header is typically very small.

Figure 2.1 shows the pseudocode of the basic FCP protocol. When a packet arrives at a router, its next-hop is computed using the network map minus the failed links in the header. If this next-hop would send the packet out an interface that has a failed link, then the router: (1) inserts the failed link into the packet header, (2) recomputes the route using

[2] When FCP operates over physical links, it can possibly get liveness information from lower physical layer (and not use keepalive messages).

```
Initialization: pkt.failed_links = NULL
Packet Forwarding:
    while (TRUE)
        path = ComputePath(M − pkt.failed_links)
        if (path == NULL)
            abort("No path to destination")
        else if (path.next_hop == FAILED)
            pkt.failed_links ∪= path.next_hop
        else
            Forward(pkt, path.next_hop)
            return
```

Figure 2.1. Basic FCP protocol.

this new failure information, and (3) returns to step one if the new next-hop also incurs a failure or, if not, forwards the packet to its next-hop. Note that each packet is treated separately, *the failure information contained in a packet is not incorporated into the routing tables.*

To understand FCP better, consider the example in Figure 2.2. Assume N_1 sends a message to N_d, and that links $N_3{-}N_d$ and $N_5{-}N_7$ are down. Since only nodes adjacent to the failed links know about the failure, the packet is forwarded along the shortest path in the original graph, (N_1, N_2, N_3, N_d), until it reaches the failed link $N_3{-}N_d$. At this point, N_3 computes a new shortest path to N_d based on the map minus link $N_3{-}N_d$, and includes the failed link $N_3{-}N_d$ in the header. Let us assume that this path is (N_4, N_5, N_7, N_d). When the packet reaches N_5, N_5 adds the failed link $N_5{-}N_7$ to the header, and computes a new shortest path that does not include the two failed links. Eventually the packet reaches the destination, N_d, along this path.

In general, there are two possibilities when a packet hits a failed link: either there is no path to the destination, in which case the packet is dropped, or there is some path to the destination in which case the graph on which routers compute the path becomes smaller (*i.e.,* because it does not include the failed link).[3] With every new failed link inserted in the packet header, the graph over which the packet is routed becomes monotonically smaller.

[3] There is a third possibility that arises due to congestion: if the router cannot hold on to the packet due to resource limitations, packets might be dropped.

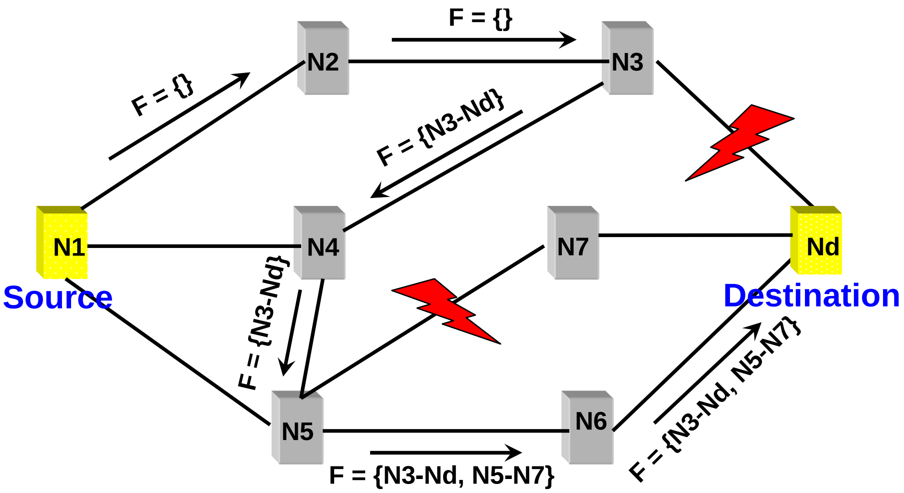


Figure 2.2. An example illustrating FCP routing.

2.2.2 Source-Routing FCP (SR-FCP)

Basic FCP assumes that all nodes have the same map. We now relax this assumption, by presenting an alternate design that employs source-routing. With source-routing FCP (SR-FCP), the first router on the packet path inserts the entire route to the destination in the packet header. Subsequent routers simply forward the packet based on the source route in the packet header until the packet either reaches the destination or encounters a failed link. In the latter case, the node adds the failed link to the packet header (exactly like basic FCP), and replaces the source route in the header with a newly computed route, if any exists, to the destination.

The main advantage of SR-FCP over FCP is that it works correctly even when not all nodes have the same network map. Thus, SR-FCP does not require that all nodes have the same map. A second advantage is that, unless there is a link failure, packet forwarding does not require a lookup operation, and thus can be implemented much faster in practice.

On the downside, SR-FCP increases the packet overhead, by requiring each packet to carry the source route. Furthermore, the inconsistency across maps can significantly increase the list of failed links, as any link that does not appear in all maps can be potentially marked as a failed link.

In order to reduce confusion, most of our discussion will focus on basic FCP, and then we will discuss briefly the properties of this alternate approach.

2.2.3 Properties

We present two key properties of FCP: *guaranteed reachability* and *path isolation*. Informally, the reachability property says that as long as the network is connected and there are no packet losses due to congestion, every packet is guaranteed to reach its destination despite any link failures. The path isolation property says that a malicious node cannot impact the path followed by a packet unless it is already on that path. Finally, we show that SR-FCP can provide these properties even when the node maps are inconsistent.

Since a failed node can be represented as node whose all links have failed, in the remainder of this section we consider only link failures. Furthermore, unless other specified, we consider only fail-stop failures, and assume that FCP employs shortest path routing. To state FCP's properties more precisely, we start with the following definition.

Definition 1. *Let M be the network graph (map). Define the liveness graph $LG(t_1, t_2)$ as the maximal graph consisting of only nodes and links of M that are alive at all times during the closed time interval $[t_1, t_2]$.*

Note that once a link goes down during $[t_1, t_2]$, it is removed from $LG(t_1, t_2)$ irrespective of whether the link comes back again or not. This aspect captures the fact that, in FCP, once a failed link is added to the packet header, it is never removed from the header.[4]

Next, we give sufficient conditions that guarantee packet delivery in FCP.

Lemma 1. Guaranteed Reachability: *Consider packet p entering network M at time t_1. Assume link failures are detected instantaneously, there are no packet losses due to network congestion, and the propagation delay over any link is one time unit. Let $d(G)$ denote the diameter of graph G.*

[4] FCP does not reuse failed links to avoid cycles.

Then, FCP guarantees that p will be delivered to the destination by time t_2, where t_2 is the smallest time, if any, such that the following two conditions hold:

1. *there are at most f failures during $[t_1, t_2]$, where $f \leq (t_2 - t_1)/d(LG(t_1, t_2)) - 1$*

2. *$LG(t_1, t_2)$ is connected and spans (all nodes of) M*

Proof. The main part of the proof is to show that packet p is delivered to its destination by *some* time t_2 that satisfies conditions (1) and (2). From here it follows trivially that packet p will be delivered by the smallest value of such t_2.

The proof is by contradiction. Assume p is *not* delivered to the destination by a time t_2 that satisfies both conditions (1) and (2).

We start with *two* observations. The first observation is that a packet will not encounter the same failed link twice. This follows from the fact that once a packet encounters a failed link l, this link is carried in the packet header, and each subsequent path computation will avoid l.

The second observation is that any packet forwarded during $[t_1, t_2]$ will take at most $d(LG(t_1, t_2))$ time to either reach the destination, or encounter a (new) failed link. This is because, unless a new failure is encountered, every node uses the same map and failed link list (*i.e.,* the one carried by the packet) to forward the packet along the shortest path. Furthermore, at any time $t \in [t_1, t_2)$, the shortest path is at most $d(LG(t_1, t))$. This is because, from condition (2) and definition 1, both $LG(t_1, t_2)$ and $LG(t_1, t)$ span M, and $LG(t_1, t)$ includes all links of $LG(t_1, t_2)$, which yields $d(LG(t_1, t_2)) \geq d(LG(t_1, t))$, $\forall t \in (t_1, t_2)$.

Let k be the number of link failures encountered by packet p during interval $[t_1, t_2]$, where

$k \leq f$ by hypothesis. After encountering the k^{th} failure, the packet is routed along shortest path to destination. Since there are no packet losses, the only reason p may not reach destination is either because (a) p encounters another link failure, or because (b) some node A, tries to forward p but does not have a route to D in the network map minus the list of failed links. However, (a) cannot be true, since by the second observation, p would encounter the $(k+1)^{th}$ failure by time $t_1 + (k+1)\times d(LG(t_1, t_2)) \leq t_1 + (f+1)\times d(LG(t_1, t_2)) \leq t_2$, which violates condition (1). Similarly, (b) cannot be true as it implies that the liveness graph is disconnected at some point during the interval $[t_1, t_2]$, which violates condition (2). This completes the proof. □

Note that FCP can fail even if there is a viable path in $LG(t_1, t_2)$, but this can only occur if the $LG(t_1, t_2)$ is disconnected and the packet-in-flight gets stranded on a disconnected component. As an example, consider Figure 2.1. As before, N_1 initially sends the packet to N_2. However, at this instant, let the links $N_1{-}N_2$, $N_3{-}N_d$ and $N_3{-}N_4$ all go down. Since N_2 and N_3 are disconnected from the destination they cannot route the packet to N_d despite the fact that the path $N_1{-}N_5{-}N_6{-}N_7$ was always active. Note that condition (2) in the above Lemma filters out this scenario, as it requires $LG(t_1, t_2)$ to span the entire graph.

In today's protocols, malicious routers can send fake route updates, and hence subvert a network to cause more packets to flow through them [111, 58]. In FCP, once the map is uploaded to each node, there are no dynamic link updates that nodes exchange to modify this map. Furthermore, each packet is treated independently of other packets—only failures that the packet encounters are taken into account for computing the paths. Hence, a node which is not on the packet's path, as computed by FCP, cannot affect the fate of the packet. The next lemma states this property.

Lemma 2. Path isolation*: Assuming the map distribution is secure, malicious nodes cannot perform off-path attacks.*

Proof. The proof follows directly from the fact that an off-path node has no way to contaminate the routing state of the nodes along packet's path, a these nodes compute the packet route *solely* based on the disseminated map and the list of failed links in the packet header. □

The main assumption we make here is that the map dissemination is much less frequent than route updates in today's routing protocols, and thus we can afford to improve the security of the map dissemination operation, even at the expense of an increased overhead.

Note that the path isolation property does not provide security guarantees against arbitrary attacks. For example, a malicious node can still mount denial of service attacks by sending spurious packets with large lists of fake failed links in the hope of overloading the CPUs of its neighbors. This attack is similar to a malicious node sending a large number of fake routing updates to its neighbors.

The next result shows that SR-FCP is able to provide these properties, even in the presence of inconsistent maps. In this case, the properties apply to the graph defined by the intersection of all maps in the system. Intuitively, this is because SR-FCP potentially treats any link that is not in all maps as a failed link. In particular, if a link l in a packet's source route is not in the map of a node A that forwards the packet, A simply adds l to the list of failed links.

Lemma 3. *Consider a network where the maps maintained by nodes are not necessarily consistent. Redefine the notion of link failure to include every link that does not belong to* all *node maps.*

Using the new definition of link failure, SR-FCP achieves both the guaranteed reachability and the path isolation properties, as stated by Lemmas 1 and 2, respectively.

Proof. The proof for guaranteed reachability is similar to the proof of Lemma 1. The only difference is that, in this case, nodes may have different maps. However, by using source routing, we ensure that the two observations in Lemma 1 are still true. Let A be the node that has computed and inserted the source route in a given packet p. Since, in the route computation, A eliminates the failed links encountered by p so far, this ensures that p will not encounter the same failed link twice. Furthermore, every subsequent node along p's path uses the source route inserted by A to route packet p until either p reaches its destination or encounters another failed link. Since the map used by A to compute the source route of p is a superset of $LG(t_1, t_2)$ (where times t_1 and t_2 are as defined in Lemma 1)[5], it follows that it takes p at most $d(LG(t_1, t_2))$ to reach the destination or the next failed link.

The proof of the path isolation property follows again from the fact that an off-path node has no way to contaminate the routing state of the nodes along packet's path. □

2.2.4 Reducing Computation Overhead of FCP

Basic FCP requires computation for every packet that encounters a failure at every node that the packet traverses. We present several mechanisms that reduce the overhead significantly by adding only a small overhead to router state.

Reducing per-node route computation

To avoid computation at *every* node in the packet's path, the nodes where failures are encountered insert a source route that they compute to the destination. Source routing between failures implies that only nodes where the packets encounter failures need to perform any computation; all other nodes merely forward packets based on the source route.

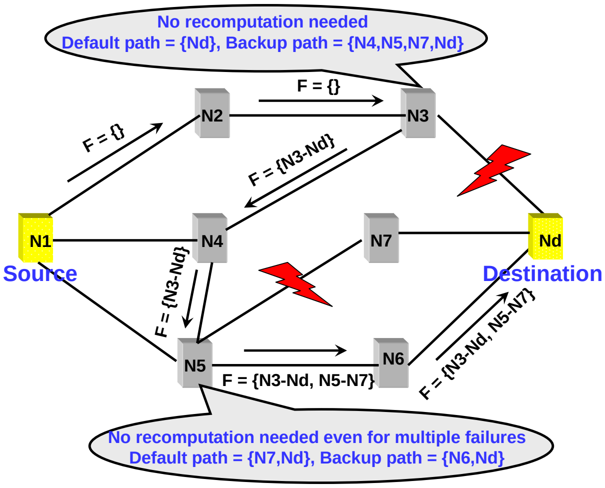

No recomputation needed
Default path = {Nd}, Backup path = {N4,N5,N7,Nd}
F = {}
F = {}
N2
N3
F = {N3-Nd}
N1
Source
N4
N7
Nd
Destination
F = {N3-Nd}
N5
N6
F = {N3-Nd, N5-N7}
F = {N3-Nd, N5-N7}
No recomputation needed even for multiple failures
Default path = {N7,Nd}, Backup path = {N6,Nd}

Figure 2.3. An example in which a packet experiences multiple link failures but recomputation is not necessary.

Reducing per-packet route computation

To reduce per-packet computation at nodes where failures are encountered, nodes perform some precomputation. Each node (in addition to the default forwarding table), for every adjacent link l, computes the forwarding table using the consistent map minus l; this table is used when l is failed. However, in terms of actual forwarding state, such a precomputation *only doubles* the memory requirement: for each destination, in addition to the default path P computed using the map, we need to store the precomputed path computed using map minus l_P, where l_P is first hop in P.

Lemma 4. *If a packet p encounters a failed link l at node N, and the precomputed path P_l to the destination (using the consistent map minus l) does not contain a link that belongs to the set of failed links that p carries, then P_l can be used to route p to the destination, and no recomputation is necessary.*

Proof. Proof follows from the fact that a shortest path is unaffected by removing a link not contained in that path. □

[5] By the definition of link failure in Lemma 3, $LG(t_1, t_2)$ does not contain any link unless the link is present in all maps, including the A's map

Figure 2.3 illustrates the intuition behind the above technique. When the packet reaches node N_5, multiple failures are encountered. But since the backup path at N_5 for the failed link $N_5{-}N_7$ does not traverse $N_3{-}N_d$, recomputation is not necessary. Hence, when the fraction of failed links is small, the chance that a recomputation is triggered is low.

Reducing computation time

Each node maintains a cache of the paths that it computes based on failures seen in packets. For each combination of failures, a node performs computation to find shortest paths only once. This is because performing a shortest path computation on $M \backslash F$, where M is the map, and F is the list of failed links, yields shortest paths to all destinations.

For performing recomputation, we borrow from the literature on incremental recomputation [82, 39]. Prior research has shown that incremental recomputation can be performed within the order of few milliseconds even for graphs with thousand nodes [7]. Performing recomputation within a few milliseconds is very reasonable—since failure detection itself would take that much time, recomputation does not substantially worsen the vulnerability period.

Furthermore, since many of the incremental algorithms construct shortest-path trees, the recomputation step yields paths to *all* destinations. Hence, by saving this information, the node can avoid recomputation for all future packets with the same set of failed links irrespective of the destination.

2.2.5 Reducing Packet Overhead of FCP

We now present a mechanism to reduce packet overhead further at the expense of local mapping state at the nodes. Consider a node N_1 sending a failure header (that includes a set of failed links) F to node N_2. With the failure header F the node N_1 associates a label l_f, and includes the mapping $l_f \rightarrow F$ when it sends the packet to N_2. After N_1 receives an acknowledgment from N_2 about the mapping, N_1 includes only the label l_f rather than the entire failure header F. Labels don't have global meaning but are specific to a pair of nodes. For robustness, a time-to-live value T can be associated with a label; if no packet with a particular label is seen for period T, the label is removed.

2.3 Dissemination of Network Maps

We now turn to the problem of disseminating the global network map to all nodes periodically. The purpose of the map is to provide all routers with a loosely-synchronized but globally consistent view of network state.

2.3.1 Network Map Information

To reduce overhead as well as react quickly to changes, the network map does not include transient changes to the network. For instance, if a link fails temporarily for a short duration, it is not removed from the map. Rather, only long-term updates such as planned outages and newly provisioned links are published in the map (short-term updates are handled by the FCP protocol described in previous sections). In order to reduce bandwidth consumption, only the difference from the previous version of the map can be disseminated.

2.3.2 Basic Map Dissemination

The map is disseminated by an RCP-like *coordinator* [22] using reliable flooding. The coordinator sends the map via TCP to a set of nodes, which in turn sends the map to their neighbors along the outgoing links in the map, and so on. To avoid receiving the map multiple times form its neighbors, a node can ask a neighbor to cancel the map transmission once the node gets the map from another neighbor. If a node is down during the map distribution, the node will get the map from its neighbors once it comes up.

To ensure path isolation property (see lemma 2), we use public key cryptography, where the coordinator signs each map with its private key. The coordinator's public key is distributed to all nodes in the network either out-of-band (*e.g.,* manually) or via a public-key infrastructure (PKI), if available. Since map dissemination is a relatively rare event, we believe that the overhead imposed by the signature operations will be acceptable. Furthermore, since the overhead on the coordinator is relatively independent on the network size, we expect the coordinator to scale to large network sizes.

While we have described how maps are disseminated, the following challenges remain, which we address next:

- How do we decide when the nodes switch to a new version of the map? How does routing happen during the window of switching maps when not all nodes route on the same version?
- How do we make the coordinator resilient to failures?

2.3.3 Transitioning to New Map

We first present a protocol that tries to ensure that all nodes route using the same map. Then, we describe a mechanism that ensures correct routing even when the protocol fails to transition all nodes to the new map simultaneously.

When each node receives a new map, it sets a local timer that expires after a period T_{lp}, where T_{lp} is a conservative estimate of the diameter of the network. A node switches to a new map either when: (a) the timer expires, or (b) it receives a packet that is routed using a new map. Intuitively, when the timer at the first node that receives the map expires, all nodes would have received the new network map completely. Hence, when it starts routing using the new map, all nodes that route packets would switch to the new map. This cascading process quickly resulting in the entire network switching to the new map depending on the amount of traffic flowing in the network. In the worst-case, all nodes would switch to the new map within T_{lp} of each other.

In practice, for an intradomain topology spanning an entire continent, we expect the that the longest shortest-path in the network would be no larger than a few hundreds of ms (for comparison, one-way delays for continental United States is roughly 50ms). Hence, T_{lp} can be conservatively chosen to be a few seconds to account for processing delays at each node as well.

Packet forwarding during map transitions

We now present a practical packet routing method using the map update process that works even in the case when the map-dissemination exceeds T_{lp}. Here, we assume that every node maintains not only the latest map, but the previous version of the map, as well. The basic idea is to downgrade a packet to using the previous version of the map if the new map is not completely disseminated.

Each forwarded packet contains a sequence number that indicates the sequence number of the map used to route that packet. When a node starts routing a packet, its current active network map (which is either the most recent map it has received or the previous version). Let a node n receive a packet from n'. Let the sequence number contained in the packet be s', the sequence number of the active map at the current node be s, and the largest received sequence number received at the current node be s_{max} ($s_{max} \geq s$).

Case (a) $s' < s$: Forward the packet using $M_{s'}$.

Case (b) $s \leq s'$: Set $s{=}min(s', s_{max})$, and forward the packet using M_s.

Using this protocol, originator starts forwarding the packet p using its active map. In the worst case, p is demoted to using the previous map if the latest map is not fully disseminated. Note that map demotion would happen only when the timer expires before all nodes get the map, and should not happen with conservatively chosen timeouts. Even if a packet is demoted to using the previous map, the protocol's basic correctness is not affected. During map transition, all nodes would not switch version numbers exactly at the same time. However, eventually all nodes would update to the new map, and eventually routing consistency is guaranteed. As noted earlier, if a node n receives a packet with a new sequence number s_{max} before its timer for s_{max} expires, then n updates s to s_{max}.

2.3.4 Coordinator Fault-tolerance

We now describe a mechanism for replicating the coordinator to improve fault tolerance. The main idea is to use replicated coordinators (much like how RCP does) with each replica having a replica number that denotes the ordering of the replica.[6] For example, replica 1 is more important than replica 2. Each replica independently chooses a map M (all of them execute the same algorithm, and hence should pick the same map, though they may not be synchronized), and disseminates M skewed in time such that replica 1 sends the map update at time t=0, replica 2 sends the update at time t=T/n, replica 3 at time t=$2T/n$ and so on, where T is the periodicity of updates that each replica sends, and n is the number of replicas. Each node floods only the map of the replica with the lowest sequence number such that the map is no older than time T+c, where $c > 0$ is the maximum time for the map to get disseminated across the network; the other maps it receives are suppressed to save bandwidth resources.

For routing, each node in the system would use the map from the lowest numbered replica with the highest sequence number such that that the map is no older than time T+c, where c is defined as above. Even if there are network partitions, within all connected components, the eventual routing consistency would hold. Partitions heal at the network layer during subsequent updates.

2.3.5 Periodicity of Map Updates

The map updation period would depend on how frequently links are decommissioned or added to the network, planned network outages, and frequency at which link costs change (say for traffic engineering reasons). In the extreme case, one could just disseminate the map as frequently as the default OSPF LSA generation period (which is typically 30 seconds). Unlike OSPF, the coordinator can send only the difference from the previous network map to save bandwidth resources. The only limitation is the overhead to sign the map.

[6] In practice, since the coordinator performs tasks only at coarse timescales, having a small number of replicas for the coordinator might suffice.

2.4 Evaluation of FCP

We first present our experimental methodology, the data sets we used, and protocol configurations we used for comparison purposes. We present results in two parts. In the first part, we present results that show that the overhead of FCP is very small. In the second part, we compare FCP with OSPF as well as backup computation techniques. Finally, we present the overhead involved in using SR-FCP as a function of degree of inconsistency in the maps at the routers. To summarize our results:

- The overhead of FCP is very low both in terms of computation overhead and packet header overhead.
- Unlike traditional link-state routing (such as OSPF), FCP can provide both low loss-rate as well as low control overhead,
- Compared to prior work in backup path pre-computations, FCP provides better routing guarantees under failures despite maintaining lesser state at the routers.

Table 2.1. Protocol configuration parameters

Parameter	**OSPFD**	**FCP/Backup computations**	**OSPFD default**
hello-interval	400 ms	50 ms	1 sec
dead-interval	2 sec	250 ms	5 sec
retransmit-interval	2 sec	n/a	5 sec
throttle-interval	2 sec	n/a	5 sec

2.4.1 Methodology

Protocols: We compared FCP with two alternate strategies: the *OSPF* [80] link-state protocol, and an MPLS-like protocol that precomputes *backup-paths*. To compare with OSPF, we leveraged the OSPFD software router developed by John Moy [81], which completely implements the OSPF protocol as specified by RFCs 2328 and 1765 [80, 79]. We configured OSPFD following the millisecond-convergence recommendations given in [7], including the incremental Dijkstra's algorithm described in [82].

OSPFD also contains a network emulation toolkit for evaluating deployments, which we extended to support FCP and backup-paths implementations. To compare with backup-path precomputation, in our results, we use the sample selection algorithm used by Juniper Networks [62]. Although algorithms exist to compute "optimal" backup paths we found that such algorithms involved substantial computation time, which led to poor results when applied to the dynamically changing networks we consider here.

Configuration: To configure OSPFD's timers, we conducted a simulation study to determine settings that performed well on our workloads. By default, we configured OSPFD to send one probe every 400ms, and to consider a link down if no probes are received after 2 seconds. To reduce sensitivity to flapping links, we configured OSPFD to "treat bad news differently from good news" by propagating link failures immediately but delaying propagation of link arrivals for five seconds. Given that FCP and the Backup-path scheme do not propagate failure information globally, we configured them with faster probing times (one hello every 50ms). Each router sends pings to all other routers, with one ping every 15 seconds[7]. Finally, to fully stress FCP routing, the network maps are never updated in the experiments we present.

Data: Link failures and arrivals were driven by ISIS traces collected on the Abilene Internet2 backbone [4]. These traces contain timestamped Link State Advertisements (LSAs). We modified the network emulator to drive link failures by playing back LSAs based on their timestamps. To evaluate larger networks and a wider range of parameters, we also used Rocketfuel [104] topologies and used a shifted Pareto distribution to drive the time-to-failure distribution for each link. Though we had six AS topologies from the Rocketfuel data, we report results from AS 1239 (Sprint) Rocketfuel topology as a representative sample. The Sprint AS topology has 283 nodes and 1882 links.

[7] We found that sending pings at a faster rate could overload the OSPFD implementation.

2.4.2 Overhead of FCP

FCP introduces extra overhead only for packets that encounter a failed link. The overhead can be classified into: (a) network overhead, *i.e.,* stretch of routing the packet, (b) packet header overhead, and (c) recomputation overhead.

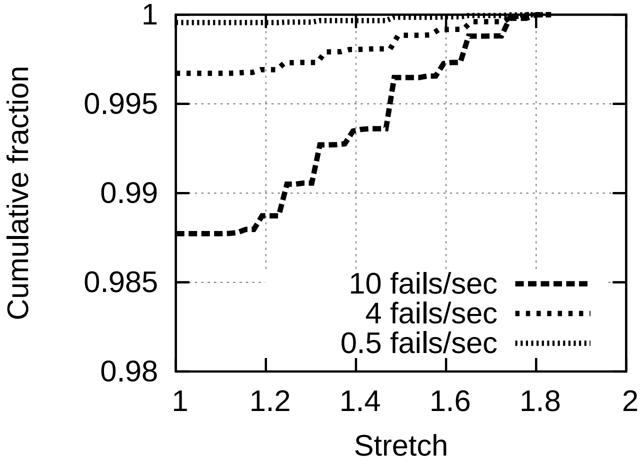


Figure 2.4. Stretch for varying mean failure inter-arrival times (in seconds). FCP incurs a stretch penalty, but this penalty is small even at high link failure rates.

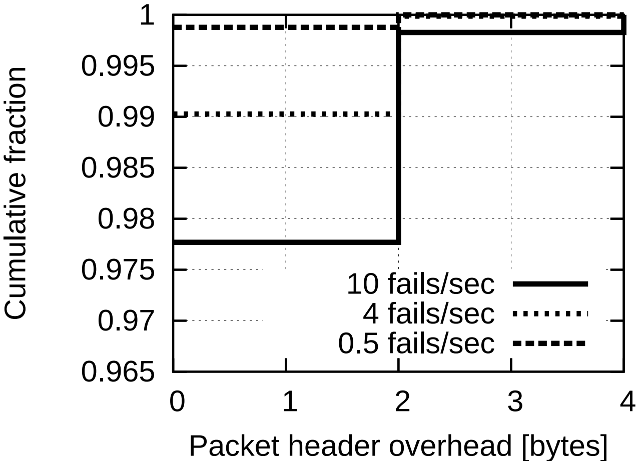
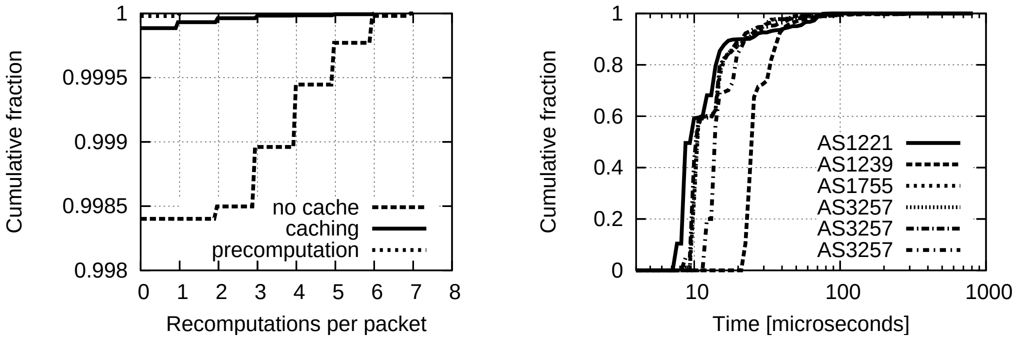


Figure 2.5. Packet overhead of FCP.

Stretch: After failure, FCP does not necessarily discover the next-shortest path, incurring a stretch penalty. Figure 2.4 shows the CDF of stretch over all pairs of sources and destinations for increasing failure rates (where failure rate is measured by the number of links that



(a) (b)

Figure 2.6. *Recomputation costs of FCP:* (a) Number of recomputations. (b) Recomputation times.

fail per second in the network). We found that the average stretch was very low (1.0012), and the worst-case stretch was under 1.8. However, for increasing failure rates, the average stretch increased to 1.0026, with a worst-case stretch of 1.83. This happens because the number of failed links a packet encounters increases with increasing failure rates. Hence, FCP is forced to reroute packets multiple times which reduces the chances that the optimal end-to-end path is taken. However, as shown later, we found this stretch was comparable to the backup-path selection strategy that we compared against.

Packet header overhead: Figure 2.5 shows the CDF of minimum packet overhead incurred by the failure header with FCP, using the OC48 trace from CAIDA [5] to generate realistic traffic workloads. For the Abilene failure trace [4], FCP inflates the average packet size by a negligible amount. The maximum header size during the run is 4 bytes per packet, assuming each failure header takes 2 bytes. Although header sizes can potentially be larger in networks with more simultaneous failures, we can reduce the byte overhead by using the label optimization described in 2.2.4. It is important to note that headers are only added on link failure, unlike MPLS, which appends a label (or stack of labels) to every data packet traversing a label-switched path.

Recomputation overhead: Figure 2.6(a) shows a CDF of the number of recomputations per packet in a network with 0.5 link failures per second. Roughly 0.2% of packets require recomputation to be performed under the vanilla FCP implementation. This value decreases to 0.01% when routes once computed are cached. Figure 2.6(b) shows the time to recompute paths after link failure as measured on a 3GHz Intel Pentium processor with 2GB of RAM. Recomputation time is below one millisecond on all topologies.

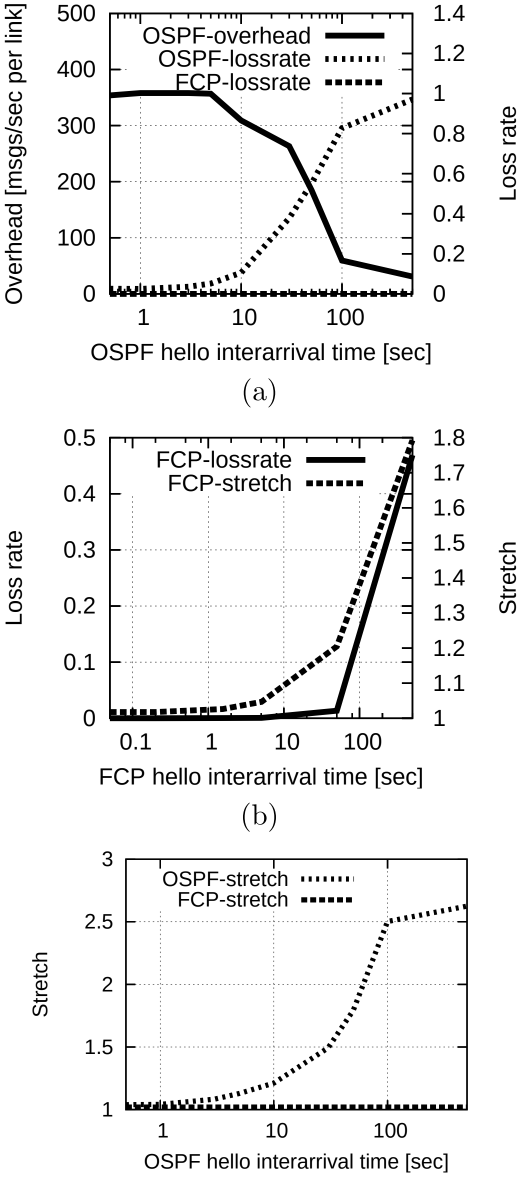
2.4.3 Benefits of FCP

Comparison with OSPF

In Figure 2.7(a), we vary the rate at which OSPFD sends HELLO packets to neighbors, and measure the resulting effect on control overhead and the fraction of data packets delivered. As mentioned previously, we tune FCP with a fast probing rate, since faster probing does not incur a penalty in control overhead in terms of LSAs disseminated since none of the link failures detected are announced. As the HELLO probing rate is increased, the number of data packets lost decreases, since failures are reacted to more quickly. However, probing at a faster rate also causes more short-term failures to be detected and propagated, increasing control overhead.

On the other hand, FCP undergoes only a very small (yet non-zero) loss-rate of less than 0.1%; the loss of FCP in our experiments is non-zero since link failure detection takes a finite time and during this period, all packets trying to use that link will be dropped. Although FCP exhibits similar behavior to OSPF in terms of stretch and loss rate while varying the probing rate (Figure 2.7(b)), its control overhead is not a function of link failure rate, and hence its probing rate can be increased without inflating control overhead. We found it was possible to tune OSPF to achieve this loss-rate, but only at the expense of a increasing its control traffic to above 300 messages per second per link.

Figure 2.7(c) which plots the average stretch shows a similar phenomenon as well; as the probing rate decreases, it takes longer to detect link repairs, and hence a larger fraction of working paths are not discovered by a packet. Since we can keep the FCP probing rate high without compromising overhead, FCP has much lower stretch than OSPF.

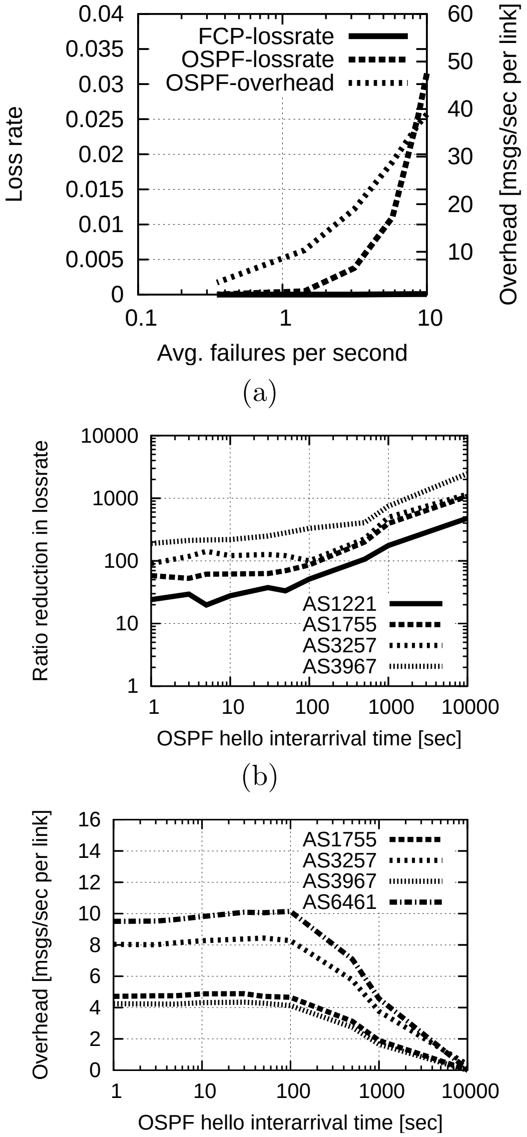


(a)

(b)

(c)

Figure 2.7. *Comparison with OSPF:* (a) Unlike FCP, OSPF cannot simultaneously provide low control overhead and high availability, (b) Reducing FCP's HELLO timer reduces stretch and loss without increasing control overhead, (c) OSPF's map becomes inconsistent with the topology at low probing rates, resulting in a stretch penalty.



(a)

(b)

(c)

Figure 2.8. *Effect of varying parameters:* (a) failure rate on control overhead and data packet loss rate. (b) topology on loss rate. (c) topology on control overhead.

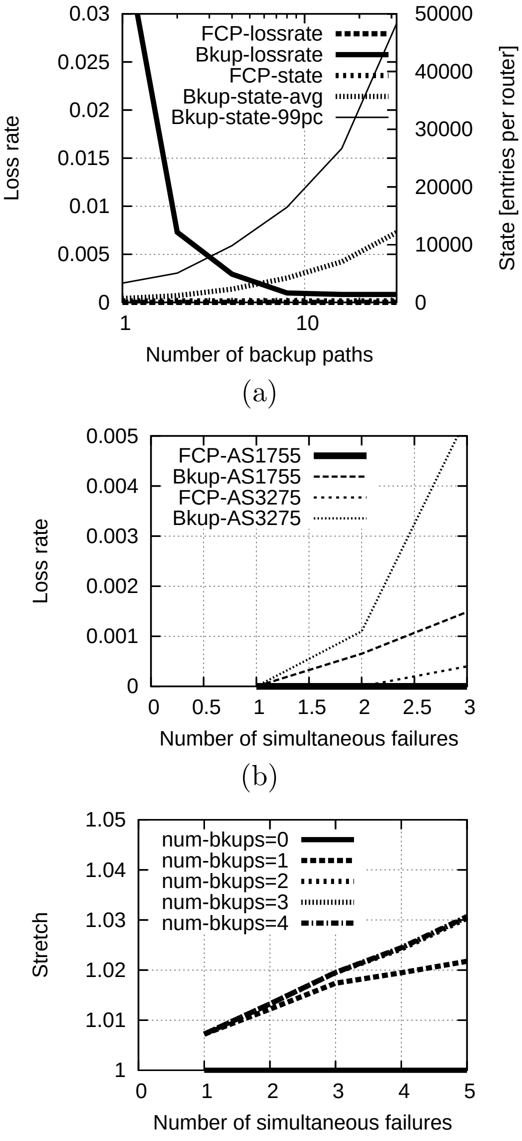
Effect of varying parameters

In Figure 2.8(a), we vary the mean interarrival time for link failures, but fix OSPF's probing interval at 400ms, and plot the loss rate and overhead. For a wide variety of failure rates, FCP outperforms OSPF by an order of magnitude, while simultaneously maintaining a lower control overhead. Note that OSPF's overhead begins decreasing as the failure rate increases past the ability of the probing protocol to keep up with link events.

In Figure 2.8(b), we vary the probing rate and plot the fraction increase in loss rate of OSPF over the loss rate of FCP for various topologies. Although the amount of improvement varies across topologies, FCP provides more than a order of magnitude lower loss rate than OSPF. As shown in Figure 2.8(c), FCP also reduces control overhead. In general, we found that denser topologies (e.g. AS 1221, with an average degree of 6.2) had less benefit from FCP than sparser topologies (e.g. AS 3257, with an average degree of 3.7). This happens because in denser topologies OSPF has a larger number of paths to choose from, and is hence more likely to discover a working path.

Comparison with backup-path selection

Unlike OSPF, the backup-path strategy we used can attain very fast failover times without a significant increase in control overhead. However, to minimize loss rates, the backup-path strategy needs to account for every failure contingency, and hence requires a substantial number of backup paths. Precomputing a large number of backup paths to account for different combinations of multiple link failures increases state per router. This tradeoff is shown in Figure 2.9a. For example, with 8 backup paths per link, the backup path strategy requires 4210 entries per router, and experiences a loss-rate of 0.05%. However on the same workload, FCP requires only 255 entries yet attains a loss rate of less than 0.002%. Moreover, unlike FCP, the distribution in state across routers is not uniform, and hence the top 1% of routers require more than $20,285$ entries. However, for switching between maps, FCP must temporarily maintain a second copy of its routing state. Although this state is only maintained for a short period, and can be stored as deltas (differences from



(a)

(b)

(c)

Figure 2.9. *Comparison with Backup-paths:* (a) Unlike FCP, Backup-paths cannot simultaneously provide low state and low loss-rate. (b) FCP maintains low loss for a variety of failure rates. (c) Backup-paths stretch penalty for varying numbers of backup paths.

the current map), in the worst case this could double FCP's state requirements (to 510 in this example).

Figure 2.9(b) shows the performance in the presence of simultaneous failures on two representative topologies. We fix the number of backup paths to two, and vary the number of randomly selected links to simultaneously fail. On both topologies, the backup strategy and FCP have roughly equal loss-rates during single failures. However, when more than one failure occurs, FCP has significantly lower losses. (Note that FCP has non-zero loss since the failure detection is not instantaneous.) This happens because as the failure rate increases, it becomes more likely the backup-path strategy will encounter a set of failures not covered by any one of the backup paths. Finally, Figure 2.9(c) shows that the backup-paths strategy incurs a stretch penalty during link failures. This happens because the backup-paths strategy attempts to find link-disjoint paths as backups, which tend to be longer than the path FCP finds around the failure.

Effect of inconsistent maps

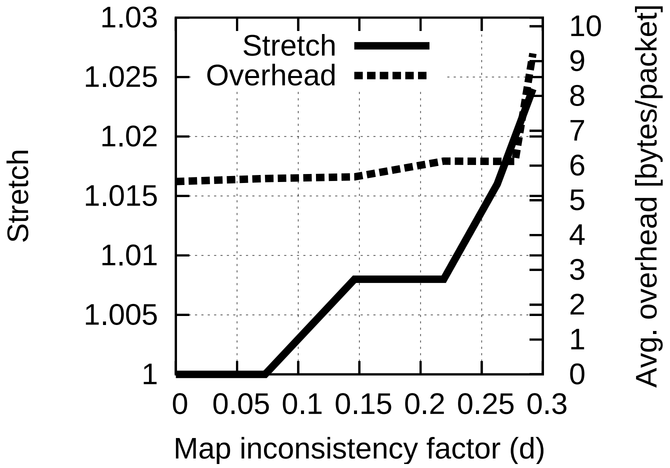


Figure 2.10. Effect of inconsistency in network maps on the overhead incurred by SR-FCP.

So far, we have assumed that all nodes have a consistent state of the network map. Here, we investigate the overhead incurred by SR-FCP—in terms of average routing stretch and per-packet overhead—as a function of map inconsistency factor (see Figure 2.10). Specifically, for a chosen map inconsistency factor d, we instantiate the network map M_n at each node n by picking links randomly from the actual network map M, such that the intersection of maps at all nodes forms a spanning and connected subgraph of M, which contains only a fraction $(1 - d)$ of links in M (with high probability). The x-axis is capped at 0.3 since that is the largest fraction of inconsistency for which the intersection of maps at all nodes forms a spanning subgraph.

The plot shows that the stretch and packet overhead are small even when maps are highly inconsistent. Even when the inconsistency factor is 0.3, the average stretch is less than 1.03, and the average size of a packet header is under 10 bytes (assuming 2 bytes per node for source routes and failures). The reason SR-FCP performs so well is because SR-FCP pays a penalty only if the source node performing a route computation misses some links that could have resulted in significantly shorter paths; an intermediate node just forwards a packet based on the packet's source route irrespective of whether downstream links in that source route (not adjacent to the intermediate node) are present in the node's map or not.

2.5 Deployment Issues

FCP represents a substantial departure from traditional routing mechanisms, and hence unsurprisingly requires several modifications to router design even for deployment at the intradomain level. Although these changes are by no means trivial, in this section we outline how they may be implemented as extensions to existing protocols and designs.

Map dissemination: The role of the coordinator is akin to the centralized node in the case of RCP [22]. Such a design is amenable in case of ISP networks where the centralized administrative node can act as the map coordinator and periodically disseminates the network map. In addition, to better handle packet forwarding during map transitions (see Section 2.3.3), routers must store both the current and the previous map, instead of only the current map as most of the existing protocols do.

FIB state: If dynamic failure-based path computations are not cached, the FIB state is doubled since at the minimum, the next hop information should be maintained for current map and previous map. Even if path computations are cached, the average additional state required is not high. With the precomputation optimization for each outgoing link (described in Section 2.2.4), the FIB state is again only doubled overall, which we believe is a modest requirement.

Forwarding: In the optimized version of FCP, routers add a label corresponding to a list of failed links to packet headers, and perform forwarding based on the label. Appending and forwarding based on labels is addressed by Multi-Protocol Label Switching (MPLS) [33]. FCP also needs to invoke recomputation for new failures encountered, by invoking special processing via the slow path.

Applicability of intradomain routing controls: Since the notion of having a link-state graph is retained, the key semantics of intradomain routing maps remain unaffected. FCP continues to provide cost-based shortest-path routing in the absence of link failures. Hence, assignment of addresses and access controls, traffic engineering, and other aspects of configuration/maintenance remain unchanged. Specifically for traffic engineering, long-term planning changes to the link costs can be introduced by the central coordinator. For short-term, reactive cost changes introduced by the routers themselves, there would be a short delay since the updates are not installed instantaneously, but have to go through the coordinator before the TE link-cost changes become active. Since the link-cost changes go through the coordinator, it can be ratified before it is incorporated into the network map in order to preserve the path isolation property.

2.6 Extensions to FCP for Interdomain Routing

We described FCP as a link-state routing protocol, and hence is directly applicable to intradomain networks. Here, we present extensions to FCP to broaden the scope of applicability. Specifically, we turn to how FCP can be used to improve interdomain routing, both in terms of iBGP and eBGP routing stability.

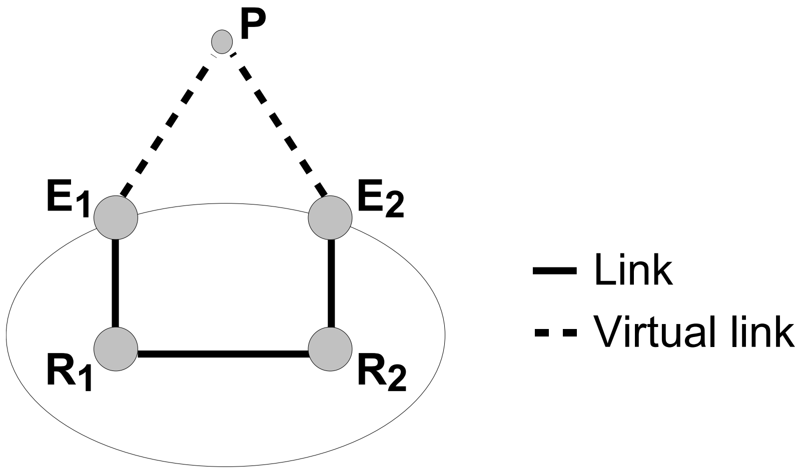


Figure 2.11. Mitigating iBGP disruptions using FCP.

2.6.1 Improving iBGP Stability under Link Failures

Hot potato routing is commonly used by ISPs to select the closest exit point amongst multiple equally-good interdomain routes. Failure of links within the network, failures of next-hop links going out of the network from the border routers, as well as small perturbations in intradomain costs can lead to *hot-potato disruptions*, where large amounts of traffic oscillate between egress points. Such disruptions can lead to routing loops, router overload, and externally visible BGP routing changes [114, 115]. Hence a scheme that can prevent such instability during change of egress points is desired [114, 113].

We present a simple modification to FCP to allow it to operate over iBGP routes within a single domain for the case of failure of links. We augment the link-state network map maintained by internal routers to treat an egress route to a particular prefix as a *virtual link* directly connected to the destination prefix. FCP is agnostic to the the notion of virtual links, as it treats virtual and actual links equally; we use the term virtual link only for convenience. When either a normal link or a virtual link fails, a router can use FCP to forward the packet to an alternate egress connected to the same next-hop AS. This ensures that routing to external routes remains consistent even if failures are not immediately propagated.

A simple illustrative example is shown in Figure 2.11. The network map consists of actual links $E_1 - R_1$, $E_2 - R_2$ and $R_1 - R_2$, and virtual links $E_1 - P$ and $E_2 - P$ for prefix P. Initially, let both routers R_1 and R_2 use egress E_1 to reach the destination prefix P. When the link (R_1, E_1) (or the BGP next hop from E_1 towards P) fails, R_1 appends the (R_1, E_1) (or the virtual link (E_1, P)) to packet headers causing R_2 to forward the packets via E_2. Traditional iBGP/OSPF routing would undergo a routing loop between R_1 and R_2 lasting until R_2's scan process, i.e. visiting the BGP routing decision for each prefix, completes.

2.6.2 Interdomain Policy Routing

Interdomain routing today suffers from long outages arising from a slow convergence process that occurs after certain routing events [74]. In this section, we discuss how we can leverage FCP to avoid failures during the convergence process of BGP. In our proposal, we only consider changes on the data plane; we do not modify BGP's route announcement and propagation protocol. Next, we discuss two of the key challenges faced by our proposal: (a) how are the network maps defined and distributed? (b) how are policies respected when FCP is used?

FCP network map

Unlike the case of intradomain routing, there is no natural centralized authority to act as a coordinator for distributing AS-level network maps. Hence, we assume that nodes work with inconsistent maps and use SR-FCP (Section 2.2.2).

All routers in the network run BGP protocol for exchanging routes as they do today. Each router defines the FCP map using the latest set of BGP updates it has received from all its neighbors.

Using SR-FCP with policy routing

Naively implementing SR-FCP would have adverse policy implications. This is because, by using AS-level source routes, an AS can force downstream ASes to forward traffic at the expense of violating their own policies. We next present a solution to this challenge.

The main idea behind our solution is to treat any policy violation as a link failure. We assume that ASes only implement policies that are a function of the neighbor from whom they received the advertisement and local policy considerations (as opposed to, for example, policies dependent on the presence of an non-neighbor AS in the AS-path). Virtually all today's BGP policies fall into this category [121].

Consider a packet p routed from source AS S to destination AS D. When a router in the source AS S starts routing a packet, it adds the AS-level path (source route) in the packet header, just as proposed by SR-FCP. Let a router R belonging to an intermediate AS I receive the packet p with an AS-level source route $ASR(p)$. Let $ASR(R, D)$ represent the AS-level route computed by R to the destination AS D (based on its BGP route selection criteria). Finally, let $NextHop(Route)$ represent the AS-level next-hop for a route. The following cases are possible:

1. N=$NextHop(ASR(p))$=$NextHop(ASR(R, D))$ and next-hop to N is alive. In this case, R simply forwards the packet to N.
2. N=$NextHop(ASR(p))$=$NextHop(ASR(R, D))$ and that next-hop to N is dead. In this case, R invokes SR-FCP by adding the AS-path $I - N$ (recall that I is the intermediate AS that R belongs to) to the failure header. R then forwards the packet along the best AS-path that it knows which does not have any failures.
3. $NextHop(ASR(p))$≠$NextHop(ASR(R, D))$. We discuss this case next.

If the AS decides that the source route present in the packet is *not compatible* with its choice of routes, it does the following operations for preventing transient loops. First, it adds $NextHop(ASR(p))$ to the failure header of the packet. Second, it uses the best route it has to the destination that does not have any failed links, and adds that source route to the packet. We leave the degree to which an AS allows routes that are not the most-preferred to be used in the source route as a meta-policy decision of that AS. This meta-policy represents the tradeoff between the degree to which the AS allows FCP to recover from failures and the extent to which policies are obeyed. For instance, an AS can decide that it allows only the top two preferred routes.

Discussion

Since BGP propagates only routes that can be potentially used downstream, routers will not obtain the entire link-state of the network. However, after BGP converges, all the nodes should have consistent route selection information, *i.e.,* all nodes will pick the same AS-level path for each destination prefix. In other words, at all nodes, the AS-level source route in the packet header will be identical to its most preferred route to the destination (Case (1) above).

During the convergence process, routing using the above modified SR-FCP protocol ensures that packets will not enter into transient loops. This follows from the fact that packets are routed using SR-FCP, hence at every stage the packet either makes progress based on the source route or that a link is added to the list of failed links in the packet header. Eventually, the packet will reach the destination or will discover that there are no more paths (based on links in the failure header), and will be dropped. Also, as we mentioned above, the extent to which FCP will help route around failures depends on the flexibility of the policy at the ASes to use source routes that are not the most preferred.

One practical challenge of our scheme is deployability, as it requires one to insert the AS source route and the list of failed links in the network (IP) header. While one can use IP options to store this state, a comprehensive solution to this challenge is subject of future work.

2.7 Designing Robust Managed DHTs Using FCP

Distributed hash tables (DHTs) were proposed as a useful abstraction for peer-to-peer file sharing [107,89,98,57], where the most important goals are self-organization, scalability to millions of nodes, and robustness to node churn. One of the key design ingredients that make such systems scale to large sizes is having each node perform greedy routing over a logical identifier space using only its partial view of the network. Many of the existing DHT designs achieve a logarithmic (in the number of nodes) lookup cost, while requiring that nodes maintain only a logarithmic number of links.

However, the very routing technique that makes DHTs scale—greedy routing using a partial view of the network—makes it difficult for DHTs to achieve some desirable properties such as performance (viz. low network latency), security (viz. resistance to routing table pollution) and consistency of lookups. Though the first two issues have received tremendous attention [47,54,107,93,25], a more important issue is lookup consistency, a key correctness property of DHTs. In theory, greedy routing using an incomplete network view relies on complete connectivity of the underlying routing graph to ensure lookup consistency. However, in practice, the Internet connectivity graph is incomplete, and non-transitivity is common both on short (owing to routing problems) and long (owing to policy reasons) timescales [43]. Routing is said to be non-transitive if there are nodes A, B, and C such that B and C can both communicate with A but not with each other. Earlier work has shown that the degree of non-transitivity in PlanetLab can be as high as 9% [43,45]. Non-transitivity causes several problems such as inconsistent views of the ID mapping, broken return paths for lookups and invisible nodes. The authors of [43], who have implemented three DHTs independently, mention that they spent significant effort discovering and fixing problems caused by non-transitivity.

These problems manifest themselves not only in large-scale P2P systems, but even in moderate-scale, managed, deployments.[8] DHTs are increasingly used as a fundamental building block for Internet applications [99,119,106,120,42], and most of these systems have

[8] By a managed deployment, we mean a deployment that is run by an individual or an organization over an infrastructure consisting of fairly stable nodes.

been deployed and evaluated on PlanetLab, a world-wide infrastructure consisting of only a few hundred hosts. Even in such less-demanding environments, experience shows that many managed deployments including i3 [106] over Chord [107], OpenDHT [94] over Bamboo [93], and CoralCDN [42] over Kademlia [76] suffer from several DHT routing problems [43].

The situation we find ourselves in is unfortunate. On one hand, these systems, because of their moderate-scale and managed setting, do not reap the benefits that greedy routing offers. On the other hand, they inherit all the problems that greedy routing introduces. In other words, we have perhaps ended up with the worst of both worlds!

We present GVD (Global-View DHT), a DHT that addresses the shortcomings of greedy design, particularly *lookup consistency issues arising due to underlying network connectivity*,[9] for managed deployments. GVD assumes a distributed environment of moderate-scale (*e.g.,* thousands of nodes) consisting of a set of nodes that is fairly stable over time, *i.e.,* the lifetime of a node is on the order of days, and the addition of new nodes is planned.

The key ideas behind GVD are decoupling routing to nodes in the system from assigning ID responsibilities to the nodes, and routing based on a coarsely synchronized, globally consistent view of the topology. In GVD, each node starts with a consistent view of the overlay topology (called *consistent map*), and the ID responsibility of all nodes. To ensure routing consistency in the face of link and node failures, GVD uses the failure-carrying packets technique. FCP guarantees *eventual routing consistency*: If for a sufficient period of time, no node fails or rejoins, maps are not updated, and underlying paths are stable, and if a node N is able to route to a destination D using the map, then any node belonging to the connected component of N can route to D.

To ensure that all nodes start with a consistent map, a *DHT coordinator* (replicated for fault-tolerance) periodically, at coarse time-scales (on the order of hours or even days), determines the map and disseminates it to all nodes. New nodes are added to the system *only* when the coordinator updates the map. A coordinator that performs coarse-grained, light-weight, tasks is not only meaningful in managed deployments, but also makes routing

[9] On PlanetLab, other reasons, such as slow nodes, cause lookup inconsistency. We don't try to solve all these issues; our focus is addressing consistency issues arising from the interaction of greedy routing and non-transitivity alone.

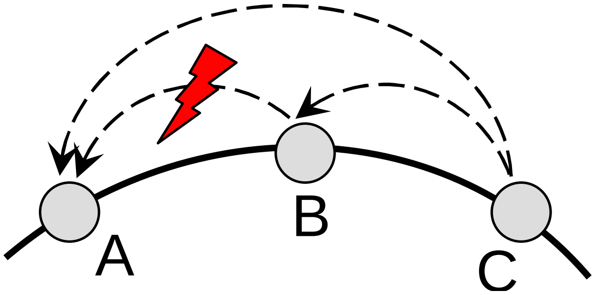


Figure 2.12. An example illustrating non-transitive routing. The links shown are predecessor links.

algorithms simple. The coordinator also disseminates the ID mapping, *i.e.,* the ID range that each node is responsible for. The ID-lookup layer is very simple: a node determines the owner of the ID from the mapping and determines if the owner is alive using FCP. Since ID-lookups are layered over FCP, they would reach the same node provided the map is connected, thus guaranteeing eventual lookup consistency.

Using both simulations as well as a deployment on PlanetLab, we evaluated GVD and compared it with Bamboo [93], a state-of-the-art DHT used in OpenDHT [94], a public DHT service. Our results show that GVD achieves over 99.7% lookup consistency in PlanetLab even when 10% of the links are failed; in contrast, Bamboo achieves only about 84% consistency at these link failure rates.

We provide a motivation for GVD in § 2.7.1, and present an overview of GVD in § 2.7.2. We describe how the consistent DHT maps are picked in § 2.7.3. § 2.7.4 presents the ID lookup protocol. We present an evaluation of GVD in § 2.7.6.

2.7.1 Motivation

All traditional DHT designs aim to provide eventual consistency, *i.e.,* if no node joins, leaves, or fails for a sufficient period of time, the system will reach consistent state. By consistent state, we mean that any two nodes in the system will have the same view of ID responsibilities, *i.e.,* if they issue lookups for an ID they will get back the same reply.

To achieve eventual consistency, DHTs typically assume that the underlying network is fully connected, *i.e.,* all nodes can communicate with each other using IP routing. However, this assumption does not hold in practice, since Internet routes fail on short and long time-scales. This non-transitivity in the Internet affects the eventual consistency property of DHTs. This issue has been recently reported for some deployed DHTs [43]. For illustration, we borrow one of the examples from [43]. Figure 2.12 shows a Chord network with IDs increasing from A to C. If the link $A{-}B$ is permanently down, then A would consider C as its successor. Hence, A would return C as the root for a lookup on a key k lying between A and B, which is incorrect.

Though it might be easy to fix the particular problem in Figure 2.12, we contend that this example exposes a fundamental limitation of current DHT designs. To achieve scalability, current designs use greedy routing over an incomplete graph, by making progress in the ID space at every hop towards the target. Since greedy routing is not guaranteed to explore all paths between an origin A and a target B, there can be cases where the only available paths between A and B are non-greedy paths. However, another node C may be able to reach B along a greedy path. Hence, lookups issued at nodes A and C for an ID owned by B will return different results. In other words, the non-transitivity of the underlying network translates to a lookup inconsistency at the DHT level.

Lookup consistency and end-to-end arguments. A natural question following from the end-to-end argument is why not deal with the inconsistencies introduced by non-transitivity at the higher layers. Most DHTs have a storage layer for storing key-value pairs. A common technique used for achieving consistency is using replication at the storage layer, and using techniques such as quorum-based protocols [91] and consensus protocols [98] among replicas.

We take the position that while ensuring lookup consistency will not relieve all applications from implementing their own data consistency mechanisms, ensuring consistency at the lookup layer has two advantages: (a) it decouples lookup consistency from data consistency, and (b) it can significantly improve the efficiency and reduce the complexity of applications. We elaborate these in turn.

Decoupling lookup consistency from data consistency makes the DHT design *independent* of application semantics. Consider systems in which data is updated or refreshed so often that maintaining even one additional replica is prohibitive. An example in this category is *i3* [106], a forwarding system, where the data items (*i.e.,* triggers) stored at the DHT nodes are refreshed every 30 sec. Since *i3* provides only best-effort service, the eventual consistency property that GVD provides would suffice.

GVD uses replication *only* for recovering data when nodes fail, and not for dealing with problems arising from underlying routing. Hence, the degree of replication is dependent only on the stability of the nodes. Since we are targeting a managed deployment, the stability of nodes is expected to be high, and hence the degree of replication can be smaller than in Bamboo. Using GVD also results in simpler algorithms for data consistency; *e.g.,* see Section 2.7.4 for how GVD can use a quorum read set size of just one. Compared to systems that use quorum-based protocols to mask lookup inconsistencies (they require quorum read sets larger than one), our protocol significantly reduces overhead, both in terms of communication and latency, for applications where reads are more common than writes, such as in CoDNS [87], CoralCDN [42], and OpenDHT [94].

2.7.2 Overview of GVD Design

The primary goal of GVD is to provide *eventual lookup consistency* in the face of non-transitivity of the underlying network. GVD also improves the lookup latency and security in the face of off-path attacks when compared to traditional DHTs. However, these benefits do not come for free. GVD can only scale to a few thousand nodes, instead of hundreds of thousands of nodes, and assumes a fairly static environment. GVD is hence targeted at managed DHT deployments.

While GVD can implement different DHT abstractions, to make the presentation concrete, we assume that GVD implements the abstraction provided by Chord [107]. In particular, we assume a circular ID space that is partitioned among the nodes in the system. Each node is assigned a unique ID, and node n with ID id_n is responsible for all IDs in

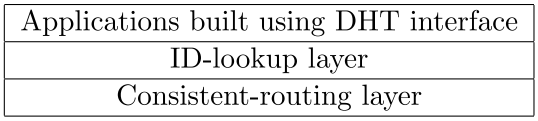


Figure 2.13. GVD's layered design.

the interval $(id_m, id_n]$, where id_m is the ID of the node that precedes n on the circle. Like Chord, GVD implements one main function, $n = lookup(id)$, *i.e.,* given an ID, GVD returns the node responsible for the ID.

We now provide an overview of GVD and how it addresses this challenge. GVD decomposes the functionality of DHTs into two layers (Figure 2.13: (a) consistent-routing layer, and (b) ID-lookup layer. We discuss these two layers in turn.

Note that the goal of the consistent-routing layer is different from the goal of overlays such as RON [9] that improve routing resilience. The main goal of GVD is to ensure that either a node can be reached by all other nodes in the system, or none at all. For instance, if there is only one node A that can reach another node B, RON will try its best to deliver the packets between A and B, while GVD could preclude the two nodes from communicating for routing consistency.

The ID-lookup layer implements the lookup operation on top of the consistent-routing layer. Each node maintains the mapping of IDs to node addresses for all nodes in the network. The implementation of lookup is straightforward. Suppose node A receives a lookup for ID id. Since node A knows the complete ID mapping, it selects the node B whose ID id_B succeeds id in the ID circle.

In the absence of failures, the lookup would return B. To deal with node failures, A has to check whether node B is alive. To check the liveness of B, A invokes the consistent-routing layer by sending a LOOKUP message to B. If B is alive, it will answer back to A, and A will return node B as the result of the lookup. If B is not alive, the consistent-routing layer will return an error to A, and A will then select the next successor of id. The procedure terminates when A finds a live successor. Though this basic lookup operation provides correctness, it could be inefficient when nodes fail; we present optimizations that address this issue in § 2.7.4.

Other Benefits of GVD Design

While our main focus is on achieving better consistency, GVD design has several other desirable properties.

Routing Performance. DHT routing typically takes logarithmic hops (in the network size), and some of these hops might be long-haul links causing the route to be much more expensive than the direct IP route. Even with proximity-based route selection (refer [52]), the choices reduce exponentially as we near the target because of the greedy routing. In contrast, FCP does optimal routing over a link-state graph.

Protection from malicious nodes. Traditional DHT routing with latency optimizations is prone to off-path attacks, *i.e.,* malicious nodes can pollute the routing tables of benign nodes [25]. In GVD, the coordinator synchronizes the graph over which nodes route, and there are no dynamic updates that nodes exchange. Hence a node can only affect the fate of packets that are anyway routed through the node, and nodes *cannot* launch off-path attacks.

Decoupling ID-lookups from node routing. Since ID-lookups are decoupled from node routing in GVD, any node in the system can be chosen for replication, *e.g.,* replicas can be placed on geo-diverse nodes to prevent correlated failures.

Load-balancing and Traffic-engineering. Mapping IDs to nodes using a DHT coordinator implies that performing coarse-grained traffic engineering and load balancing across nodes of the network is simple. Furthermore, the extent of ID space each node is responsible for (which would decide the bandwidth and storage requirements) can be decided based on the capacity of the nodes, to accommodate heterogeneity. Contrast this with several complex load-balancing algorithms that have been designed for DHTs [46, 21, 6, 63].

Ease of maintenance. Running and managing DHTs is sometimes a nightmarish task. For instance, nodes joining close to one another in time (during startup or partition healing) can cause inconsistency of DHTs that last several minutes, and even pathological routing state (*e.g.,* loopy graphs in Chord [107]). On the other hand, running GVD is extremely simple. There are no complicated topology construction protocols, and nodes can be started, and updated simultaneously.

2.7.3 Picking Links for Consistent Maps

An important question to be addressed in determining the consistent map for DHTs is how links in the map are chosen. We are not going to advocate one particular algorithm because we believe that the coordinator gives a lot of flexibility; hence, we are going to present various possibilities for choosing the consistent map. For our deployment and evaluation, we chose a couple of strategies for fair comparison with other DHTs, and we present those in the experiments section.

To start with, we note that the link picking algorithm is completely independent of the routing protocol—given a map, the node-routing layer tries to route using the shortest path, avoiding the currently failed links. Hence, to be extremely naive, one can use any distributed overlay construction and use the links that the overlay constructs as the graph to be used in the network—after all, we are doing shortest path routing over the graph, so FCP would perform better on that graph. For instance, we can run a Bamboo network, crawl the network and get the graph that Bamboo constructs, and use that graph as a starting point. Clearly, we are not using the routing layer of Bamboo; we are only using the part that Bamboo uses to choose links.

A logical question that arises is why the map is not the complete graph of all IP paths. There are two main reasons. First, since our main goal is to improve consistency, we should explore all the paths in the network to a destination before declaring that the destination cannot be reached. (Otherwise, we'll get inconsistent routing in cases where some nodes can reach a destination and others can't.) If we pick a complete graph, and a node fails, then FCP would try to reach the node through every node in the system in the worst case. The network latency of such an operation would be disastrous. This problem is not particular to FCP; no matter what routing protocol is used, the link status has to be dynamically maintained or probed on-demand. Even for vanilla link-state or distance-vector protocols, the problem manifests as several control messages when links or nodes go up and down. The second reason is that we need to maintain liveness of neighbor links, and if we have a network with thousand nodes, aggressively maintaining neighbor status might not be feasible. Despite these reasons, we aren't suggesting that one should *not* use a complete graph, only that there are strong reasons for not using a complete graph.

The fact that links are picked by a central node gives a lot of flexibility in being smart about link choice. For instance, the traffic that is served by each node can be monitored and the coordinator can pick links for coarse-grained *traffic engineering*—if a certain link l is overloaded, then other links that offload the traffic off l can be chosen.[10] Alternatively, links can be chosen to incorporate policies between nodes that run the network. For federated services, individual organizations might have policies on how it prefers traffic to flow depending on the costs. Also, some of the underlying IP paths might not be usable due to policy reasons; rather than let the DHT algorithms figure this information out, we can pass this information by removing the links in the map.

2.7.4 ID-Lookup Layer

Implementing the basic lookup operation over the consistent-routing layer is straightforward (see Figure 2.14). Since each node knows the IDs and the addresses of all the other nodes in the system, the lookup operation reduces to checking if the node whose ID succeeds the queried ID is alive. Checking the liveness of the node through the consistent-routing layer guarantees that if the originator of the query cannot reach the target, no other node in the system reachable from the originator can reach the target. In such cases, the originator tries the following successors of the queried ID.

We now discuss three aspects of ID lookup layer. First, we describe how nodes themselves maintain the current up-to-date state of what IDs they are responsible for. Second, how do we improve the performance of lookups when nodes fail. Third, how it is possi-

[10] For instance, ISPs monitor all their routers' traffic today, and state-of-the-art techniques for such monitoring easily scale to thousands of nodes.

```
Lookup of $id$ at node $n$
    $i \leftarrow 1$
    while (TRUE)
        $s \leftarrow i^{th}$ successor of $id$ using local info
        $result \leftarrow$ send(s, LOOKUP)
        if ($result$ = SUCCESS)
            return $s$
        else if ($result$ = FAILURE)
            $i \leftarrow i + 1$
```

Figure 2.14. Vanilla ID-lookup protocol.

```
Weak-stabilization at node $n$ with ID $id_n$
        $s \leftarrow$ successor of $id_n$ using local info
        $x \leftarrow$ send($s$, PREDECESSOR_REQUEST)
        if ($id_x \in (id_n, id_s)$)
            update local successor, $s$, to $x$
        send($s$, NOTIFY_SUCCESSOR)

Notify successor received at $n$ from $n'$
        $p \leftarrow$ predecessor of $id_n$ using local info
        if ($n' \in (p, id_n)$)
            update local predecessor $p$ to $n'$
```

Figure 2.15. Weak-stabilization protocol from [109].

ble for a node to add new links to its map to improve the lookup performance, without compromising the consistency provided by the routing layer.

Maintain ID Responsibilities

For any meaningful storage layer to be built on top of the lookup layer, each node should have an up-to-date knowledge of what ID range it is responsible for. Though the DHT coordinator initializes this ID mapping at all nodes, nodes failing or re-joining can cause this mapping to change.

Since we offer the Chord abstraction, for maintaining ID responsibility dynamically, we leverage the stabilization protocol from Chord (refer Figure 2.15) which has provable eventual correctness of ID responsibilities [109]. However, the implementation of the Chord

```
Lookup of $id$ at node $n$
    $i \leftarrow 1$
    while (TRUE)
        $s \leftarrow i^{th}$ successor of $id$ using local info
        if ($s.status$ is DOWN)
            $i \leftarrow i + 1$
            continue
        $result \leftarrow$ send(s, id, LOOKUP)
        if ($result$ = SUCCESS)
            return $s$
        else if ($result$ = NOT_RESPONSIBLE)
            reset status of $i$ successors of $id$
            $i \leftarrow 1$
        else if ($result$ = FAILURE)
            $s.status \leftarrow$ DOWN
            $i \leftarrow i + 1$
```

Figure 2.16. Optimized ID-lookup protocol.

abstraction for GVD is much simpler as it can take advantage of the fact that nodes know the complete set of possible nodes in the systems and their assigned IDs. Hence, all the messages are sent directly to the destination node's IP address using FCP. In addition, we do not need to maintain any finger tables or successor lists.

Improving Performance when Nodes Fail

When nodes fail, FCP explores each link in the consistent map adjacent to the node before finally declaring the node as failed. The task of exploring the map could be expensive since all paths are explored. We now present a simple mechanism that reduces this overhead.

Consider a lookup for *id* originating at node n. Let s_1 be the successor of *id*. If routing to s_1 using FCP fails, then n could set the *local status* corresponding to s as DOWN. For future lookups, n uses only nodes that are not DOWN. However, even if s_1 comes up later, n would continue to skip s_1 even though s_1 is the correct answer for the lookup. To address this issue, n includes the *id* of the lookup in the LOOKUP packet it sends during the lookup process. The node that receives the lookup performs an up-call to the ID-lookup layer to determine if it is indeed the owner of the *id*. If it is, the lookup succeeds. Otherwise, the node returns a NOT_RESPONSIBLE message back to the source, in which case the source falls back to the vanilla ID-routing protocol.

Caching

DHTs perform caching of nodes to improve routing performance. The tradeoff between performance and correctness that caching introduces is well-known. But we explicitly walk through this tradeoff since GVD's goal is to improve lookup consistency. Before doing so, we make an observation about the need for caching in GVD. In traditional DHTs, caching is important since routing links are mostly chosen based on IDs (with perhaps some optimizations such as PNS [52]). Caching is less important in GVD since routing over the consistent map is optimal, and the links in the map are chosen by the coordinator.

If the ID-routing layer performs caching, then from a node-routing perspective it is equivalent to adding *local links* that other nodes are not aware of. Since the guarantees about node reachability that FCP provides depends on consistent routing information, such link additions might cause inconsistencies. Consider the following example: Node A adds a cache link to node B. At a later time, the network gets partitioned (with respect to the consistent map) with A and B on different partitions. Now, only node A in its partition can reach node B. To summarize, cache links can be added or refreshed only if there is a path to the destination using the underlying consistent map alone, and the TTL of the cache determines the maximum period of inconsistency.

Replica Data Consistency

GVD can provide eventual data consistency without requiring the client to read from more than one replica. This can be achieved if we never allow a failed node or link to come up. Let a data item have k replicas that are ordered $r_1, \ldots, r_k$. A *put* tries the replicas in order, and succeeds if for each of the replicas the data was either successfully stored or FCP returned a failure (i.e. there was no path in the map). Similarly, a *get* tries the replicas in the same order, and succeeds when it is able to read the data from some replica. This achieves consistency because the only way a write fails is when the client concludes that the replica is unreachable using FCP, in which case the replica will never become reachable again.

If we allow node (and link) resurrections, there is a subtle race condition that we need to deal with. If the links adjacent to a node A are flapping, then it is possible for a write to falsely conclude that node A is down though at all times A is connected by at least one link. The main problem is that there is no mechanism to recover the lost read, so any subsequent reads that reach A will return old data forever.

To address this problem, we force node A to fail if a write could have potentially failed (*i.e.,* FCP concludes that A cannot be reached). We mandate that links that go down are not brought up for a certain time T_{flap}. We choose T_{flap} to be large enough such that the node would eventually find that it has no adjacent links and it will bring itself down. Later node A may rejoin the system, and its state will be updated, as explained next.

To deal with node rejoins, we do not allow concurrent joins of new nodes at the same node; this is acceptable since we are dealing with managed systems where node joins is not common. Let node A which is the primary replica of the key join, and let B be the secondary replica. When A rejoins, B starts to update data items at A. In addition, we handle reads and writes as follows: (a) A will start acting as a root for writes (*i.e.,* updates to our data item will be done first at A and then at B), and (b) all reads will start at B.

When B finishes updating the state at A (at this point A will have the most current state among all replicas, since writes are performed at A first), A will start to serve the reads as well. At this point, the joining of A is deemed complete. In effect, we ensure that read always starts at the node which has the most up-to-date copy of data.

2.7.5 Implementation

For evaluation, we performed stand-alone simulations and experiments of GVD, as well as compared GVD with Bamboo DHT. In this section, we briefly explain the GVD implementation and the modifications that we made to Bamboo for performing head-on comparison.

GVD Implementation

We wrote an implementation as well as a simulator of GVD in C++ with code shared between them, consisting totally of about 3700 source lines of code (SLOC). We implemented the incremental recomputation algorithm based on the balls-and-springs model in [82]. All nodes maintain link liveness information to their neighbors by periodically performing *pings*. We did not implement the map update protocol; instead we deployed GVD with precomputed consistent maps.

Modifications to Bamboo

We took the December 2005 release of Bamboo code and made the following changes:

- We changed the *Dht* stage and the ONC-RPC API for *put* so that it returns the IP address of the destination which accepted a key-value pair. This in effect makes the *put* API as a *lookup*, which returns the root node for a given key. We will refer this variation of the *put* API as *lookup* API. Such a *lookup* API was implemented because our goal is only to measure the lookup consistency and performance of the basic DHT algorithms, and factor out the storage-layer techniques used by Bamboo to mask DHT inconsistencies.

- We changed the semantics of the value field of *put* to be a placeholder for sending *commands* to a node. These commands were used for controlling our experiments.

- In order to artificially introduce IP path failures, we changed *Network* stage to drop both incoming and outgoing packets from or to a specified set of nodes, specified during the course of the experiments.

- We modified the Bamboo simulator to initialize Bamboo router objects with routing

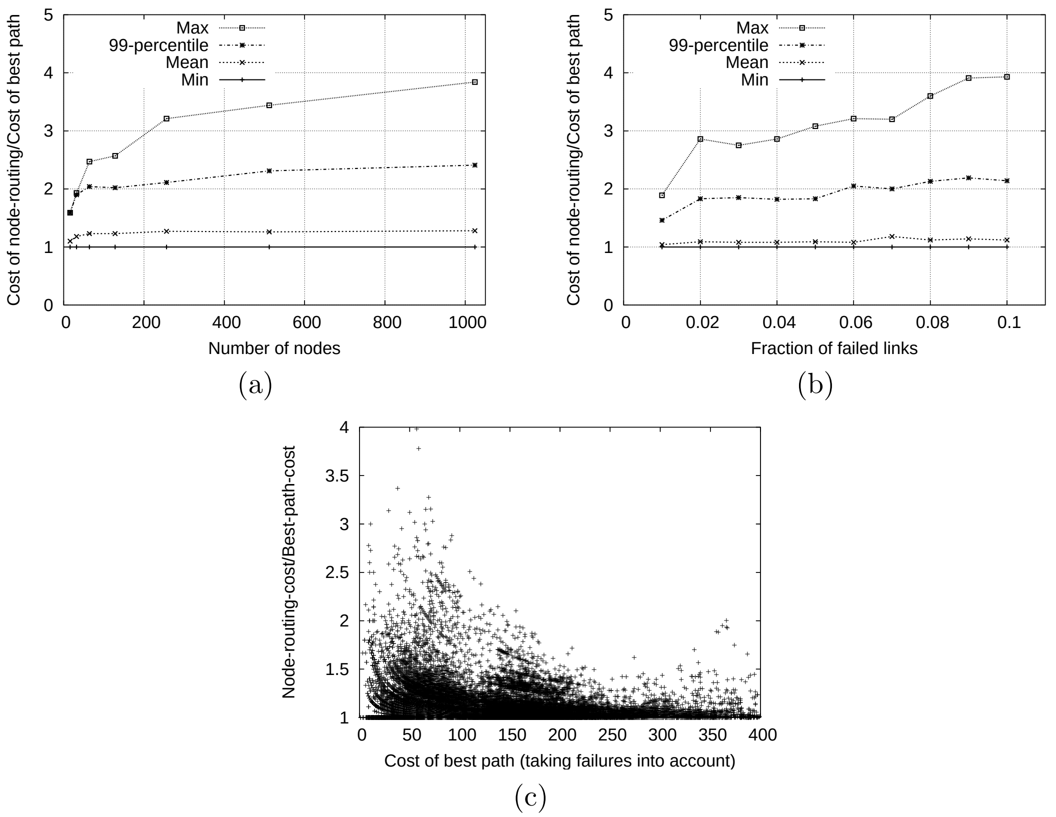


Figure 2.17. (a) Min, mean, 99-percentile, and max stretch (ratio of cost of GVD to the cost of the best path in the liveness graph) as a function of number of nodes, (b) stretch as a function of number of failed links using crawled Bamboo topology, (c) scatterplot of GVD stretch as a function of cost of best path when 10% of links are failed.

tables and leaf sets obtained by recursively crawling a real Bamboo deployment. This setup is used to statically compute routes that Bamboo would have taken.

2.7.6 Evaluation

In our evaluation we answer the following questions: (a) How much overhead does GVD incur? (b) How effective is GVD in providing lookup consistency? (iii) How do GVD lookups perform in terms of latency?

Methodology

We conducted two kinds of experiments: (a) using a custom simulator, and (b) using a PlanetLab deployment, and compared GVD with Bamboo in both cases. We used simulations primarily to systematically study the behavior of GVD in a controlled environment and to conduct more scalable experiments that are not practical in a real testbed. We used PlanetLab experiments mainly for comparing lookup consistency with Bamboo when links fail in the underlying IP network. All experiments are performed over all pairs node routing. All the simulation results report median over 20 runs.

PlanetLab experiments. Experiments for both GVD and Bamboo were performed on the same set of about 200 nodes.[11] We formed the Bamboo network by starting nodes, one at a time, and letting them immediately join the network using a pre-selected list of gateways. We waited for 15 minutes after the last Bamboo node was started before running the experiments. Lookups are issued reliably from a control center node using the modified *put* API. We conducted GVD experiments similarly from a control center node which reliably communicated with the GVD nodes.

Picking consistent maps. For picking consistent maps, we used two methods. First, we deployed Bamboo on PlanetLab, took a snapshot of Bamboo overlay topology by crawling router information from its web interface and used the crawled data to construct the consistent map. Such a topology would exactly mirror the routing links used by Bamboo. With a coordinator, we can presumably do better, but we use this method to factor out the map selection process in our comparisons. Further, though Bamboo dynamically updates its tables, we stick to the fixed snapshot for GVD's consistent map. Second, for obtaining larger topologies for simulation, we used uniform-degree random graphs (where degree was chosen to be log of number of nodes) with latencies of edges chosen from the distribution of latencies in PlanetLab all pairs ping data.

Introducing failures. Since IP routing anomalies are unpredictable, we artificially emulated failures to systematically study the effect of link failures on both systems. We varied

[11] Though the PlanetLab testbed is large, we could reliably conduct our experiments only on about 200 nodes

the fraction of links (varied from 0 to 10%) uniformly chosen at random from all pairs of IP paths. If an IP path that we failed was used by a Bamboo router in any its routing entries, then Bamboo was allowed to dynamically find replacement entries and fix its routing tables. The GVD routers however do not get any replacements.

Feasibility

We performed microbenchmarks of the forwarding rates that GVD router supports between two commodity 2.4 GHz Pentium IV PCs with 512 MB RAM running Linux 2.6.12 kernel connected by a 100Mbps Ethernet. We varied the number of failed links and the length of the source route in the header from 0 to 50. By blasting 1400-byte packets between the hosts and recording the successfully received packets, we compared the throughput of GVD with a raw UDP transfer. The difference (using a median over 20 runs) was negligible in all cases with both transfers receiving approximately 95.5Mbps.

Using simulations, we now quantify the inefficiencies that GVD introduces when links fail, specifically: (a) routing overhead in terms of extra latency of FCP, (b) the amount of packet header space consumed by FCP, and (c) the recomputation overhead introduced by FCP.

Latency overhead under link failures. When links fail, FCP could take suboptimal paths relative to the liveness graph $LG(t_1, t_2)$, i.e., the graph of live links over the time the packet is forwarded. We quantify the latency overhead of such routes as a function of: (a) network size, and (b) fraction of failed links. We compute the stretch for all pairs of sources and destinations that encounter at least one failed link. We use the latency stretch relative to the liveness graph to isolate the effects of the link picking algorithm.

Figure 2.17 (a) shows the scaling effects of stretch over the random topologies with 5% of the links failed. Mean stretch is almost a constant; even for 1024 nodes, stretch is just 1.28. The intuition is as follows. Let the mean path length for a graph be l, and the mean cost of routing around a single failure be f. The number of failures experienced in a path is proportional to l; hence, expected extra cost of an FCP path is proportional to fl. Stretch experienced is then proportional to $(1+f)$. Since FCP performs local repairs, the mean cost of routing around a failure would be almost independent of the network size. The maximum stretch seen by any node increases faster, but is still under 4 for 1024-node network.

Figure 2.17 (b) shows the variation of stretch with link failures for the Bamboo topology. We see that even with 10% link failures, the 99-percentile stretch is just about two. In addition, Figure 2.17 (c) shows that high stretch (more than 2) is experienced only for short paths with latency $<$ 50ms.

Packet overhead. FCP introduces packet overhead in the form of failed links and source routes (if FCP optimizations are used), which are functions of both the choice of map as well as the fraction of failed links. We determine the overhead, computed over all possible paths that encounter at least one failed link, as a function of: (a) network size, and (b) fraction of failed links. Due to space constraints, we plot only the results for number of failed links.

As the network size grows, we would expect the maximum number of failed links experienced in a path to be proportional to the network diameter. Figure 2.18 (a), which plots this metric by failing 5% of the links, confirms this intuition: the maximum number failed links grows roughly logarithmically, and the average is almost a constant.

As the fraction of failed links increases, the maximum number of failed links in a path would grow linearly, which is consistent with Figure 2.18 (b) which plots this variation for Bamboo topology. Even with 10% link failures, the 99-percentile stretch is only 7 for a 200 node network, which translates to an overhead of just 14 bytes, assuming a two-byte node ID. The variation of the length of source routes was also similar. To detect node failures, in the worst-case, we need to add all its adjacent links to the header, which would cost $2d$ bytes, assuming a maximum degree of d; however, only the first packet would suffer this overhead.

Recomputation overhead. On-the-fly recomputation is needed in cases when precomputed backup route traverses a failed link contained in the header. To quantify the extent of recomputation, we failed a certain set of links, and measured the number of re-

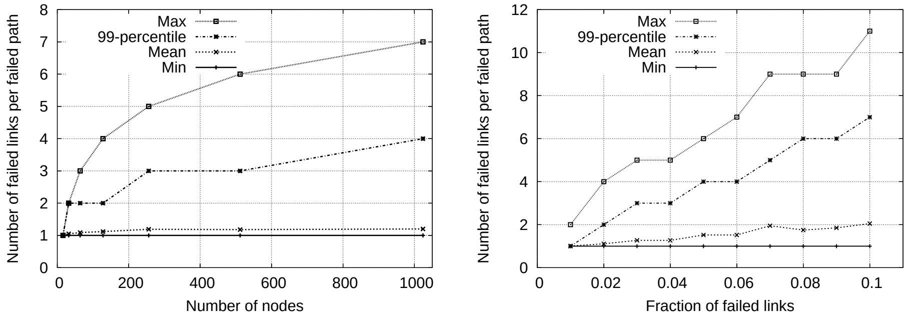


Figure 2.18. Average/Maximum number of failed links encountered by a packet, restricted to packets that encounter at least one failure, as a function of (a) number of nodes, and (b) fraction of failed links.

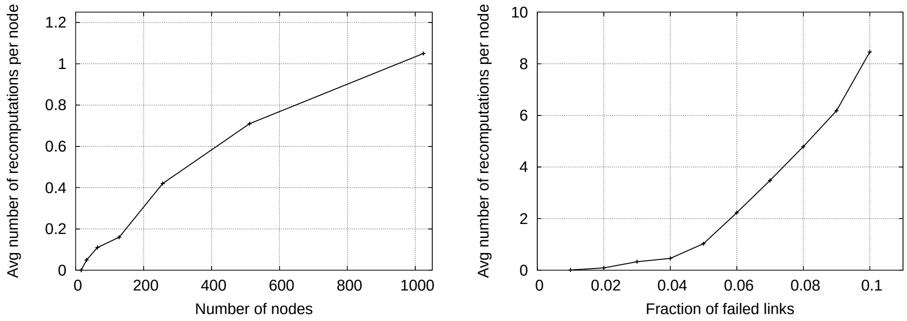


Figure 2.19. Mean number of recomputations needed per node (for all possible paths in the graph) as a function of (a) number of nodes, and (b)fraction of failed links.

computations needed at all nodes over all pairs of sources and destinations. Since the result of a recomputation is a single-source-shortest-path tree, we can reuse the result for all packets with same list of failures but different destinations.

Figure 2.19 plots the average number of recomputations needed per node as a function of (a) network size, and (b) fraction of failed links. The average number of recomputations increases roughly linearly with network size. The reason for this is that while the number of failed links per path increases logarithmically, there are a combinatorial number of possible choices of failures in a path. Also, the average number of recomputations needed for low link failure rates is very small; even when 10% of links are failed, the average number needed over *all* pairs of nodes is just 8. Using incremental algorithms, we can keep the cost of recomputation very low; even for thousand node topologies, the average cost of a recomputation is just over ten milliseconds [7, 82, 39].

Lookup Consistency

Lookup inconsistencies arise due to several factors; GVD addresses those arising due to underlying routing. We evaluate lookup consistency using PlanetLab experiments. On a shared testbed like PlanetLab, several parallel experiments loading the hosts cause nodes to become excruciatingly slow. For GVD, this could translate to several nodes being unreachable using the consistent map. To isolate such issues, we performed simulations also, since it exactly points out the benefit GVD offers when inconsistencies are caused by underlying routing. In simulations, GVD, as expected, achieves 100% consistency. We now present our results on PlanetLab.

We measure lookup consistency as follows: for a pair of nodes (A, B), we issue a *lookup* at A for the node ID of B. If either of the nodes is dead or unreachable from the command center, then both the data points were ignored. Otherwise, we deem the lookup from A to B as inconsistent if the lookup returns an answer that is not the IP address of B and vice versa. This experiment is repeated for all pairs of nodes.

Figure 2.20 compares the lookup consistency of GVD and Bamboo over PlanetLab.

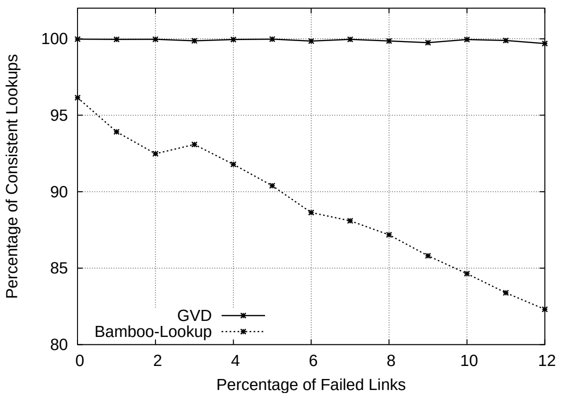


Figure 2.20. Consistency of lookups with number of failed links. y-axis is scaled to show GVD variation better.

For Bamboo, the fraction of consistent lookups drops linearly with the increase in the fraction of dropped links (this was consistent with Bamboo simulations as well). Even on PlanetLab, GVD consistency stays close to one. However, GVD suffered a small degree of inconsistency (up to 0.3%). On investigation, we found that the reasons were as we expected: (a) consistent map getting temporarily disconnected due to several nodes not responding to pings (presumably because they were slow), and (b) transient problems of links flapping combined with the fact that the lookups that were compared were not exactly synchronized. We emphasize that these inconsistencies do not contradict the eventual consistency property of GVD.

We made a few interesting observations in the behavior of Bamboo during our experiments. During an all-pairs lookup test with 120 nodes, even when no links were dropped, about 40 lookups ended up at an incorrect node. It turned out that particular node incorrectly considered itself as the root of the key. The greedy routing path for that key from a third of the nodes in the network happened to pass through that node, which wrongly declared itself as the root.

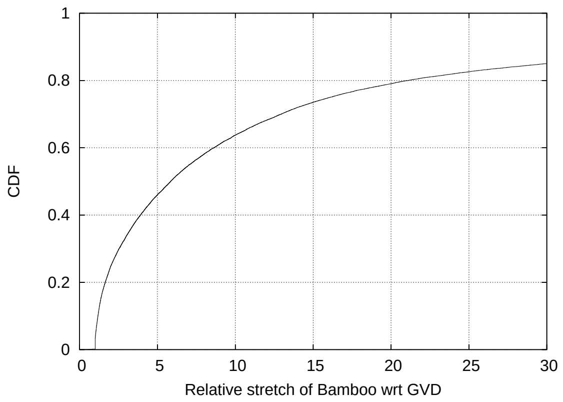


Figure 2.21. CDF of Bamboo stretch relative to GVD. Only GVD paths with cost greater than 10ms are included.

Performance

When comparing Bamboo and GVD latencies using PlanetLab, the variation in latencies were so large for both that we could not get any meaningful head-on comparison results. Hence, we present results based on simulations alone.

Since GVD performs optimal routing on the consistent map, performance is expected to be better which is reflected in Figure 2.21 which plots the relative stretch of Bamboo with respect to GVD for all pairs of sources and destinations for which GVD latency is $>$ 10ms. We investigated the unexpectedly high stretch Bamboo had with optimal routing over the entire topology, and found two reasons: (a) some of the Bamboo greedy paths took very long links, and (b) the latency estimates that Bamboo topology gave could have been biased by the slow node problem [92] on PlanetLab, and GVD avoids these high-cost paths. A caveat to note is that we are using a snapshot of the Bamboo routing tables after they stabilize. However, since Bamboo can potentially change the link set when the link quality changes, Bamboo can potentially do better over time than what the graph suggests.

2.8 Related Work

We separate related work into three parts: those that address the routing convergence issue in routing protocols, and those that provide guarantees on secure routing, and those that seek to cope with non-transitivity issues in DHTs.

2.8.1 Improving Routing Convergence

Prior work to address the problem of routing convergence in the literature at the protocol level can be roughly classified into three categories: (a) designing loop-free convergence protocols, (b) reducing the convergence period of protocols, and (c) using precomputed backup paths to route around failures. We don't discuss attempts at addressing such issues at the higher layers, for instance using overlays to get around underlying routing problems.

The idea of carrying information to route a packet in the header of the packet itself is inspired by Stoica's work on dynamic packet state [105]. FCP is similar in spirit to LOLS [84] used for routing in wireless adhoc networks. However, FCP separates network map information and transient failures by not introducing transient failures into the map, and hence can provide better routing guarantees.

Loop-free convergence. Several approaches ensure that convergence takes place in a way that it obeys certain correctness constraints. For example, link-state vector routing [14] advertises a subset of links, and uses a termination-detection algorithm to break loops. Diffusing computations [44] achieves theoretical loop-free routing convergence using a distance-vector paradigm. However, as the authors of that paper note, the performance after node failures and network partitions is a concern because all network nodes have to be involved in the same diffusing computation.

Reordering LSAs during propagation has been proposed to ensure that transient loops are avoided, for the specific cases of protected and planned link failures and cost changes [41] alone. Not-via addresses [20] uses a mechanism very similar to the precomputation optimization presented in Section 2.2.4. Consider a router R that performs the following computation: For dealing with node failures, by iterating over each other router R', R precomputes backup paths to all destinations assuming that R' is down. For dealing with link failures, it performs a similar precomputation by iterating over neighboring links. However, the draft states that they do not aim to handle multiple simultaneous link/router failures. Nearside tunneling [19] dynamically constructs tunnels to the closest router adjacent to the failure, and forward traffic via the tunnel during the convergence process. A simpler scheme is to simply forward via a loop-free alternate path in the presence of failure, or to forward to a U-turn alternate when no loop-free alternate exists [12]. While these approaches improve properties of the convergence process, they still require routing updates at the control plane, and are hence subject to the control overhead versus availability tradeoff we discussed in our results.

Reducing convergence times Some efforts have addressed failure recovery directly at the level of routing protocol. For instance, Alaettinoglu *et al.* [7] propose to modify IGP implementations to reduce convergence time to a few milliseconds even when links fail, by modifying timers, and improving run-time of the route computation algorithm. However, reducing timers can increase the control overhead and worsen network stability, as shown by our experimental results. In general, there has been substantial debate over what parameters to use, and it is not clear that there is a single correct choice of these timers or if they can be eliminated completely.

Such protocol tweaks are restricted by protocol constraints; for example, arbitrarily reducing the timer values for detecting change in link status could potentially make routes oscillate due to false positives in detecting failed links. Furthermore, adjusting link weights in OSPF can temporarily destabilize the network, even with fast convergence, because often multiple weights need to be adjusted simultaneously.

Using precomputed backup-paths Several works, have proposed using precomputed backup routes when primary paths in the network fail, for example IP restoration [60], and MPLS Fast-Reroute [86], and several others [12, 29, 18, 62]. A short evaluation of the fast reroute techniques is presented in [40]. More recently, R-BGP [66] proposes using a simple precomputation-based backup method for fast-failover during BGP convergence process that has some provable guarantees such as loop-prevention and valley-free routing.

Backup routes are practical only when there are small numbers of simultaneous failures; to achieve the guaranteed reachability property of FCP with multiple failures, several backup paths would be needed. In fact, from our experiments, we see that even with low failures rates, multiple failures can simultaneously occur in real networks. In contrast to precomputed backup paths, FCP not only provides correctness guarantees in the face of multiple link failures, but does so by requiring much lesser state at the routers than backup path computations typically do.

2.8.2 Achieving Secure Routing

There has been lot of theoretical research on achieving reliable communication in the presence of byzantine faults (such as [88, 111, 35]). However, these protocols have high overhead, and are not practical at high link speeds. Much of the recent research on secure routing has been directed at BGP in particular. In contrast to these works, FCP leverages a practical assumption—that a centralized node can transmit the consistent map to all nodes—to achieve its security property. FCP's data path involves no cryptographic operations; the reliable transmission of the consistent map requires some cryptographic operations, but the process is infrequent. While the security property of FCP is less strict than those offered by some prior theoretical works, we believe that FCP is useful in practice, as it restricts the damage caused by a malicious node to only the traffic that the node itself forwards.

2.8.3 Non-transitivity Issues and DHTs

Prevalence of non-transitivity. Non-transitivity in the Internet can be both transient (due to routing problems) as well as permanent (due to policies). Earlier work has reported the degree of non-transitivity to be as high as 9%, and that about half the cases exhibit transient non-transitivity [43, 45]. Li et. al. [70] have studied, using simulations, the effect of non-transitivity on the robustness of different DHTs. Several efforts such as RON [9], Detour [100] and one-hop source routing [53] have improved the resilience of Internet routing by eliminating transient non-transitivity by using detour paths.

How DHTs cope with non-transitivity. DHT deployments have implemented many workarounds to get around the problems (such as routing loops and broken return paths) that non-transitivity introduces; these are discussed in [43]. To the best of our knowledge, we know of two DHT deployments—Bamboo and FreePastry—that have used special mechanisms to improve consistency. (Some of these techniques are very recent and largely unpublished work.)

Bamboo [93] masks the non-transitivity problem by using quorum-based replication protocols at the storage layer. Let the root of a key k be N, then the key is replicated on a subset of nodes in the leaf-set of N; write/read succeeds only if a certain subset of the leaf-set successfully store/return the data. Even if the lookup doesn't successfully reach N, quorum establishment ensures that a read R_2 occurring after R_1 will not receive a value older than what R_1 receives. Bamboo thus uses replication not only for reliability (dealing with node failures) but also for consistency (dealing with link failures). We have discussed the advantages of GVD (*i.e.,* ensuring consistency at the lookup layer) in § 2.7.1.

FreePastry, the open-source implementation of Pastry from Rice University, has incorporated some techniques to mask non-transitivity (refer technical report [55]). Rather than treat IP paths as the underlying links of the graph for performing greedy routing, they use *virtual links* which are overlay paths. That is, even if node A cannot reach its leaf-set member B directly, if it finds a path through C, then A can route to B over the virtual link $A–C–B$ (by using overlay source routing). They also use a stabilization protocol among members belonging to a leaf-set so that each member can reach the other using some other node if not directly—the hope here is that the leaf-set is large enough to mask routing problems among members of the leaf-set.

Conceptually, the mechanism in FreePastry layers DHT routing on top of a RON-like routing restricted to the leaf-set. The experiments in a technical report [55] show that such a link-state stabilization protocol eliminates inconsistencies observed in their deployment. However, the correctness of lookups depends on the size of the leaf-set and the degree of non-transitivity. For example, if the leaf-set members are partitioned into two components, but are connected to the rest of the graph using routing links, then each partition would individually converge to a stable state; however lookups originating from different parts of the network might reach different components. GVD extends this idea of consistency across leaf-sets systematically to all nodes of the system to provide eventual consistency guarantees irrespective of underlying routing.

Very recently, Chen and Liu have proposed consistency semantics for key-based routing systems [26]; using their definitions, GVD is a weakly-consistent key-based routing system. For maintaining consistency, they use group membership protocols and divide nodes into zones for scalably maintaining groups. In contrast, since we address issues in managed DHTs, we simplify the design by having the coordinator disseminate coarse-grained consistent topology.

Global Views for Consistency. Using a DHT coordinator for improving consistency guarantees, continues a recent trend in networking of centralizing certain aspects route computation. RCP [22] solves consistency issues arising from interaction of BGP with intra-domain protocols. For network routing, the 4D architecture [51] presents a clean-slate approach for network control and management, and advocates explicitly separating the dissemination and decision planes from the data plane. The functionality of DHT coordinator is analogous to the decision and dissemination planes in 4D.

2.9 Summary

In this chapter, we introduced a new routing technique, Failure-Carrying Packets (FCP), that eliminates the convergence period endured by traditional routing protocols. The basic idea behind FCP is simple both conceptually and practically: once all routers have a loosely-synchronized, consistent view of the network, it is enough for a router to know the list of failed links to correctly compute the path to a destination.

Though we primarily present FCP to introduce a new routing paradigm that is qualitatively different from previous approaches, we also present simple optimizations that makes FCP feasible in practice. Using real-world ISP topologies and failure traces, we show that both the computation overhead as well as packet overhead incurred by FCP is very small. We also present comparisons with both OSPF (with timers carefully chosen) as well as a commercially-used backup path technique. In the former case, we show that unlike OSPF, FCP can provide both low loss-rate, as well as low control overhead. In the latter case, we shows that FCP provides better routing guarantees under failures despite maintaining less state at the routers. Though the basic model of FCP as a link-state routing paradigm is directly applicable only to intradomain networks, we discuss how FCP can be applied to interdomain policy routing as well. Studying the applicability of FCP in different routing networks (such as interdomain routing, wireless networks, sensor networks) more deeply is topic of future work.

Chapter 3

Customized Routing as a Service

3.1 Introduction

Interdomain routing has long been based on three pillars:

- *Local control*: ASes have complete control over routing and forwarding decisions within their domain.
- *Bilateral agreements*: An AS has pairwise contracts with neighboring ASes to collaborate in providing service.
- *Distributed algorithms*: End-to-end paths are computed using a distributed routing protocol, where each AS applies local policies to routes learned from neighbors.

The distributed routing protocols produce default paths that satisfy most customers. However, the default paths are not sufficient for some customers with special performance or policy requirements. We propose that third-party Routing Service Providers (RSPs) satisfy the needs of these customers through (i) end-to-end control over the forwarding infrastructure, (ii) business agreements with the various ISPs along the paths, and (iii) logically-centralized computation of the paths based on a global view of the topology.

3.1.1 Conflict Between Users and ISPs

In today's routing architecture, end-to-end path selection depends on the complex interaction between thousands of ASes, ranging from Internet Service Providers (ISPs) to enterprise networks. Each AS has control over the flow of traffic through its part of the infrastructure and cooperates with neighboring ASes to select paths to external destinations. The operators of these ASes configure the routing protocols running on their routers to make efficient use of network resources, maximize revenue in sending traffic to customers, and control which neighbors can transit traffic through their infrastructure. Still, each ISP has at best indirect control over the end-to-end path, typically by "tweaking" the routing-protocol configuration, making it difficult to offer meaningful service-level agreements (SLAs) to customers or to identify the AS responsible for end-to-end performance problems.

An ISP's customers, such as end users, enterprise networks, and smaller ISPs, have even less control over the selection of end-to-end paths. By connecting to more than one ISPs, an enterprise can select from multiple paths [13]; however, the customer controls only the first hop for outbound traffic and has (at best) crude influence on incoming traffic. Yet, some customers need more control over the end-to-end path, or at least its properties, to satisfy performance and policy goals. For example, a customer might not want his Web traffic forwarded through an AS that filters packets based on their contents. Alternatively, a customer might need to discard traffic from certain sources to block denial-of-service attacks or protect access to a server storing sensitive data. Another customer might want low end-to-end delay for Voice-over-IP traffic, or high throughput for a large data transfer.

The conflict between ISPs and their customers for control over path selection is fundamental [30]. Unfortunately, existing proposals skew the control to one stakeholder at the expense of the other. On the one hand, ubiquitous deployment of a QoS-routing protocol would enable ISPs to select end-to-end paths that satisfy user requirements. However, QoS routing between ASes would require deploying a complex protocol that is needed for only a small fraction of requests, and even then is unlikely to meet all the specialized needs. On the other hand, source routing would give customers complete end-to-end control over

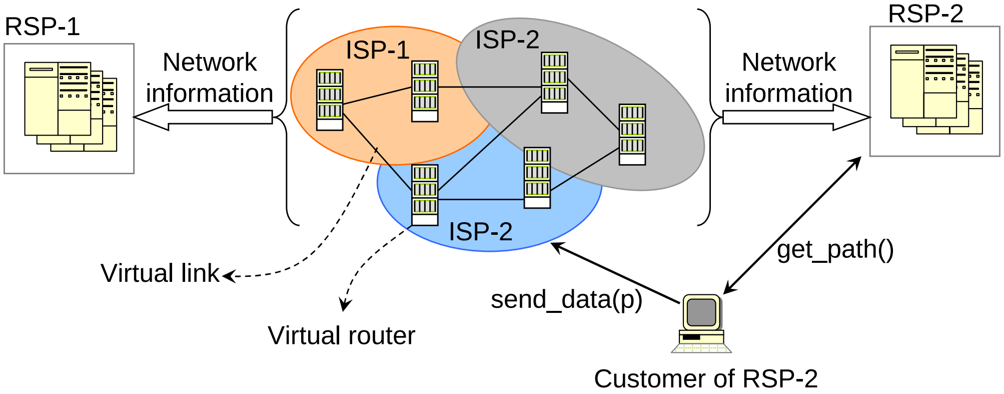


Figure 3.1. The main components of the RAS architecture: the forwarding infrastructure (FI), one or more routing service providers (RSPs), and RAS clients.

the forwarding paths. However, ISPs do not have an economic incentive to cede control over routing decisions, due to the lack of business relationships with end users; in addition, source routing introduces difficult scalability and security challenges. Instead, we argue for pulling the tussle out of the infrastructure by allowing third-party providers to form business relationships with both users and ISPs, and select and install end-to-end forwarding paths on behalf of the users.

3.1.2 Routing as a Service (RAS)

Our proposal consists of three entities: the forwarding infrastructure (FI) that spans multiple underlying ASes, a collection of Routing Service Providers (RSPs), and the clients of the RSPs, as illustrated in Figure 3.1. The RSPs contract with both ASes and end-customers, so they do not have to negotiate directly. More specifically, we envision an RSP would buy *virtual links* (VLs) from various ASes with well-defined SLAs (something ISPs are quite willing to sell today), connecting some number of *virtual routers* (VRs). The RSP sets the forwarding state of these VRs, though the underlying ASes control how traffic flows between VRs. Customers desiring customized routes contract with an RSP, which would then set up an appropriate end-to-end path along its virtual links. The fact that there are a limited number of these VRs allows the RSP to compute these routes in a centralized fashion, so that the path characteristics can be carefully tuned. Multiple

RSPs may coexist, forming a competitive market-place for offering a value-added service to customers at a reasonable price.

Our architecture meets the needs of both parties. The ASes still have sufficient control, in that they can limit the number and size of the VLs they sell and engineer the flow of traffic through their networks. The end-customers get the path performance they need, and do not have to deal with every AS along the path. Our approach has precedence, in that Content Distribution Networks (CDNs) perform a similar role. Rather than having content providers contract with every ISP for caching, and having some complicated inter-ISP protocol for deciding who serves which requests, a CDN acts as a middleman in the process. The CDN has contracts with the ISPs, and then is able to offer a comprehensive content delivery service for content providers. Most Web sites do not need a CDN so, rather than complicate the basic HTTP protocol with sophisticated content-delivery mechanisms, only those customers with specialized needs contract with a CDN. As with CDNs, we envision that a few competing RSPs would coexist and provide flexible service for different customers.

In the next section, we present three examples of customer requirements that RSPs could satisfy, overcoming fundamental limitations with today's routing architecture. Section 3.3 discusses the interaction between ISPs and RSPs, and Section 3.4 explores how customers interface to the RSP. We present related work in Section 3.6. Section 3.7 concludes with a discussion of future research directions.

3.2 Case for End-to-End Route Control

Although default routes are sufficient for most traffic, some traffic needs to follow paths that satisfy high-level policy goals. In this section, we present three examples that illustrate why flexible route control is important, how today's routing architecture is insufficient, and how Routing Service Providers (RSPs) can direct traffic on the appropriate paths.

3.2.1 Example 1: Avoiding Undesirable ASes

Avoiding paths through certain ASes: Some users may want their traffic to avoid traversing certain intermediate ASes. For example, suppose an AS is known to perform content-based filtering of data packets, or to redirect Web traffic to alternate Web servers that return sanitized content. Alternatively, consider two government agencies that would not want their traffic to traverse networks run by another country. Or, suppose that a user wants its traffic to avoid ASes that do not apply best common practices for securing the router infrastructure or preventing DDoS attacks. In each of these cases, the end user wants the packets to and from certain destinations to avoid forwarding paths that traverse particular ASes.

Avoiding selected ASes is easier with RAS: Today, path selection depends on the composition of the BGP routing policies implemented in multiple ASes. Selecting paths that avoid certain ASes is difficult, if not impossible; RAS can overcome these limitations:

- Because BGP is a path-vector protocol, an AS only learns the paths advertised by its immediate neighbors. If all of these paths traverse an undesirable domain, the AS has no way to select a suitable path, even if such paths exist in the AS graph. In RAS, an RSP that has complete information about the virtual topology can easily compute paths that avoid selected ASes.

- BGP routers select a single best path for each destination prefix. This precludes an ISP from allowing one customer to avoid a downstream AS (for policy reasons) while allowing other ASes to use paths that traverse the AS (e.g., for performance reasons). In RAS, an RSP can selectively direct some traffic to a special path that avoids the selected AS, while allowing the remaining traffic to use the default path.

- Today, ASes use BGP routing policy to implement business relationships with neighboring domains. The ISP does not have an economic incentive to direct traffic through a peer or provider, even if the path avoids the AS in question, if a path exists through one of its customers. In RAS, the RSP has its own business relationship with the

ISP, which provides the necessary incentive for the ISP to direct selected traffic on the chosen path.

- BGP is destination-based, making it extremely difficult to ensure that the reverse path from the destination back to the source avoids the AS in question. In RAS, the RSP can install forwarding state along both directions of the path between the two hosts.

That said, the RSP (or set of RSPs) must resolve potential conflicts between the desires of the sending and receiving hosts. For example, the sender might want to avoid a particular AS, whereas the receiver might prefer paths that traverse this AS. We argue that the RSPs are the natural place to resolve these inherent tensions, based on full knowledge of the virtual topology and the routing policies of each party.

3.2.2 Example 2: Blocking Unwanted Traffic

Discarding traffic from unwanted senders: End hosts may want control over which sources can send traffic to them, and which links they can use. For example, the victim of a denial-of-service attack may want to block the offending traffic, based on the source IP address and where the traffic enters the network. To prevent future DoS attacks, an enterprise might block traffic from source prefixes in geographic regions known for being the source of attacks. Similarly, to prevent spam, an enterprise may want to block traffic from the IP addresses of mail servers known for sending spam. A university campus may want to block incoming traffic with a source port number corresponding to certain application. Sites in a virtual private network (VPN) might want to receive traffic only from addresses belonging to other sites in the VPN. In each of these cases, the end host wants the network to selectively discard incoming traffic.

Blocking unwanted traffic is easier with RAS: Today, blocking unwanted traffic depends on configuring access control lists (ACLs) or null routes at various points inside the network. Ideally, unwanted packets should be discarded close to the sender, to reduce the bandwidth consumed by the traffic and to amortize the overhead of applying the filtering rules [11]. Achieving this goal in today's routing system is difficult, but RAS can overcome these challenges:

- Today, blocking unwanted traffic depends on the joint configuration of the routing protocols and access-control lists [126]. In RAS, an RSP can forward traffic to "null" based on a wide variety of policies, such as a five-tuple of source and destination prefix, source and destination port numbers, and protocol.
- Pushing the filters further away from the target network requires cooperation between many pairs of ASes; with n ISPs, this may require $O(n^2)$ business relationships. In RAS, an RSP forms the relationships with the n ISPs to install the needed forwarding state, obviating the need for pairwise relationships.
- Knowing which traffic to block requires keeping track of the IP addresses that often originate spam (and other unwanted traffic), and knowing the local filtering policies of each destination. Having each AS maintain this information is inefficient. In RAS, an RSP can keep track of the filtering rules and install them at the relevant locations in the forwarding infrastructure.

That said, the RSP must resolve conflicts between sources who want to send packets and destinations who do not want to receive them, and balance the trade-off between dropping the traffic close to the senders and installing a large amount of state in the forwarding infrastructure. Again, we argue that RSPs are a natural place to address these trade-offs.

3.2.3 Example 3: Guaranteeing Quality of Service

Providing performance guarantees for traffic: Communicating hosts may want guarantees on the quality-of-service for certain traffic. For example, a user may want strict delay guarantees for interactive phone calls; a remote user listening to an audiocast may have much looser performance requirements. Another user may want a bandwidth guarantee for downloads from a video-on-demand server. Two scientific organizations may want a bandwidth guarantee for bulk transfer of a large data-set. In each of these cases, the end hosts have a particular “flow” that requires an end-to-end guarantee on one or more performance metrics.

Guaranteeing QoS is easier with RAS: Today, ISPs provide coarse-grained service-level agreements, only for traffic that stays inside a single AS. Providing fine-grained quality-of-service over an end-to-end path is extremely difficult, but RAS can address this challenge:

- Today’s Internet does not provide end-to-end signaling to reserving resources along a path that traverse multiple institutions, making it difficult to offer performance guarantees. In RAS, RSPs negotiate strict QoS guarantees with individual ISPs, and then stitch virtual links together to provide end-to-end QoS to customers.
- Although an ISP can provide guaranteed QoS for highly-aggregated traffic, offering performance guarantees for individual flows is extremely challenging, in terms of the signaling overhead and the need for fine-grained packet scheduling. In RAS, RSPs reserve bandwidth across an ISP for aggregated traffic and manage the division of these resources across individual flows.
- Today’s ISPs can determine which traffic should receive priority service based on bits in the packet headers. However, an ISP cannot easily classify packets based on finer-grained information, or direct packets on different paths based on their performance requirements. In RAS, RSPs can classify packets based on diverse customer policies and assign a sequence of virtual links with the necessary performance properties for each flow.

In the absence of a standard signaling protocol for specifying requirements and reserving end-to-end resources, each RSP can decide what performance guarantees to offer, and how. This provides an opportunity for an RSP to differentiate itself by offering special QoS services to customers.

3.3 Virtual Links: ISP-RSP Interaction

Rather than controlling the entire forwarding infrastructure, an RSP contracts with ISPs for *virtual links* with Service-Level Agreements (SLAs). Virtual links allow ISPs to retain control over the flow of traffic within their networks, while reducing the overhead for RSPs to compute end-to-end paths.

3.3.1 Virtual Links With Service-Level Agreements

RSPs do not need control over packet forwarding at the level of individual routers and links, and ISPs may not be willing to cede such fine-grained control anyway. Instead, we envision that ISPs offer *virtual links* as a service that RSPs can purchase; then, the RSP constructs an end-to-end path by stitching together a collection of *virtual links* from the source to the destination. The virtual link is unidirectional, and connects two virtual routers that the RSP controls. More specifically, the virtual router could be inside the ISP network as a RSP-specific context located on the ISP's own router, or a separate network element outside the ISP but connected directly to the ISP. Either way, at each virtual router, we envision complete isolation between the forwarding state and virtual links controlled by different RSPs—hence, a misconfigured RSP cannot affect other RSPs or the ISP itself.

An ISP can use existing technologies, such as MPLS [97], to create virtual links and provide the necessary bandwidth isolation. The ISP can offer an SLA for the virtual link. On one extreme, a virtual link could be a constant-bit-rate pipe with a maximum propagation delay, allowing the RSP to construct end-to-end paths with hard QoS guarantees. On the other extreme, a virtual link could offer best-effort service, allowing the RSP to construct low-cost paths that obey the end-user's policy requirements, such as avoiding certain intermediate ASes. In some sense, virtual links are not much different than the services ISPs can offer today to direct customers; the main power lies in the ability of an RSP to stitch together virtual links across different providers.

When buying a virtual link with a particular SLA, the RSP guarantees not to exceed the maximum traffic load, though the ISP could install traffic shapers in the data plane to enforce these limits. The predictability of the offered load should greatly simplify how the ISP does traffic engineering. Whereas ISPs today must measure or infer the "traffic matrix," the virtual links can provide an upper bound on the traffic between two virtual routers. This aids the ISP in configuring the intradomain routing protocols to make efficient use of network resources and to ensure that the SLA is met even if internal failures occur. For example, the ISP would have an incentive to over-provision the network or provide explicit back-up paths, to avoid incurring a penalty when the SLA is violated. By controlling both ends of the virtual link, the RSP is in a good position to identify when these SLAs are violated, and to identify which "hop" in the end-to-end path is responsible for a performance problem.

To further improve the robustness of virtual links, neighboring ASes could cooperate to offer a virtual link that spans their networks. For example, two ISPs that peer in multiple locations could coordinate to balance load across these links, perhaps using the negotiation scheme discussed in [73]. Working together, these ISPs could mask the effects of a failure of one of the links or routers between them, allowing them to offer stronger SLAs for these "long haul" virtual links. The RSPs benefit directly from the reduced overhead of managing the virtual routers, and the stronger service guarantees they can offer to end users. In fact, the presence of RSPs provide meaningful incentives for neighboring ISPs to cooperate in this manner, by paying a higher price for virtual links that span two or more ISPs.

Still, an RSP must be able to react when virtual links fail. In some cases, virtual links may fail due to planned maintenance in the ISP network. Because of their direct business relationship, the ISP can notify the RSP in advance of planned maintenance, to allow the RSP to migrate the end-user traffic to an alternate path, perhaps using a virtual link through another ISP, without violating the SLA offered to the end-user. This kind of graceful end-to-end rerouting is extremely difficult today, due to the lack of business relationships between end users and intermediate ASes. In other cases, an unexpected failure occurs. The virtual router connected to the failed virtual link must direct traffic to an alternate virtual link, or notify the RSP to compute a new end-to-end path.

3.3.2 Scalable Computation of End-to-End Paths

The abstraction of a virtual link plays an important role in reducing the path-selection overhead and how often RSPs must recompute the paths (e.g., due to virtual-link failures). An RSP computes paths on a virtual topology consisting of virtual routers and the virtual links between them. For an initial estimate, suppose an RSP has a virtual router for every border router in every AS, and a virtual link for each pair of virtual routers connected to the same AS and for each connection between neighboring ASes. Although the Internet consists of around $20,000$ ASes, around 80% are stub ASes [110] that have just one or two border routers. The Internet has around twenty tier-1 ISPs [110] that have around 500 border routers. Focusing on these ISPs alone, the RSP would need to manage around ten thousand virtual routers (20×500) and five million virtual links ($20 \times 500 \times 500$). The actual number, when including the other ASes, might easily grow to a few hundred thousand virtual routers and several million virtual links.

As with any large network, we adopt the conventional technique to achieve scalability—hierarchical organization. Several natural scaling techniques can reduce the number of virtual routers and virtual links substantially. For example, most ISP backbones consist of a relatively small number of Points-of-Presence (PoPs) in key cities. A large ISP with 500 border routers might have just 30 PoPs. Having a single virtual router per PoP would reduce the number of virtual routers and virtual links substantially. Considering 20 large ISPs with 30 PoPs each, the number of virtual routers drops to 600 (20×30), and the number of virtual links drops to $18,000$ ($20 \times 30 \times 30$), a much more manageable number. If some ASes cooperate to provide virtual links that span multiple networks, these numbers would be even smaller. With the high-end PCs available today, computing (say) shortest paths on a graph of this size is in the realm of possibility, especially with the use of incremental algorithms for path computation.

In addition, an RSP can divide the responsibility for path selection into multiple servers, each representing a particular region of the Internet. We envision hierarchical routing in an RSP to be simpler than today's Internet routing hierarchy because our division of computation is driven *only* by scalability concerns, not differing routing policies or regions of administrative control. Geographic boundaries offer a natural way to distribute the computation, for two main reasons. First, these boundaries have relatively small bisections, reducing the impact of subdividing the computation. For example, a relatively small number of (high-bandwidth) links interconnect the U.S. and Europe. The failure of a link inside the U.S. is not likely to affect how traffic flows to Europe. Second, having separate RSP servers in each major geographic regions is important for rapid responses to virtual-link failures. In fact, we envision that the local RSP servers can often react to a failure by performing a "local reroute" over just a portion of the path, rather than requiring a complete change in the end-to-end path.

3.4 Gateways: RSP-Customer Interaction

Customers need a way to subscribe to a particular RSP and specify the policy requirements that should drive end-to-end path selection. In addition, the data packets need a way to use the customized paths, without giving arbitrary end users direct control over the forwarding infrastructure. To address these issues, we introduce the concept of *RSP gateways*. An RSP gateway is software controlled by the RSPs which *bridges* the customers to the forwarding infrastructure. Both control-plane messages (policy specifications), as well as data-plane traffic (data packets) go through the gateway.

3.4.1 Control Plane: Route-setup Based on Customer Preferences

A customer communicates its preferences to the RSP gateway, and the RSP gateway contacts the appropriate RSP nodes that perform the path computation and obtain a path that the customer's traffic can use. The RSP gateway explicitly performs the route setup on behalf of the customer to install the forwarding state in the virtual routers, or add a source route to the customer's data packets. By enforcing that virtual routers accept control messages only from RSP gateways, the RSP can ensure that malicious users cannot cause any damage (by inserting arbitrary paths or spoofing RSP packets). We now discuss various issues relating to how customers express their preferences to RSPs.

Customer configuration: When a customer signs up with an RSP, the RSP allows the customers to connect to a particular gateway (or a small set of gateways). The location of the RSP gateways could differ depending on the customers. For example, an enterprise that obtains service from an RSP can have an RSP gateway at the edge of its network so that its traffic is routed through the RSP. Configuration is easy, since individual users need not request paths separately. Also, policies might be decided by the enterprise and not the individual users. An alternate category of customers include end-users who want to use a particular RSP from, say, their laptop. The end-user could be an employee of the government and might want to avoid certain ASes in the path. In this case, the policy should stay with the laptop. Alternatively, end-user might want to leverage the fact that the laptop has multiple interfaces. In such cases, the gateway software running on the end-user's machine can monitor the performance of the last hop and use the link that does a better job satisfying the requirements.

Customer policy specification: Customers have to specify to the RSPs what their policies and preferences are for their specialized paths. Defining a flexible API that captures the different forms of customer preferences is required. Furthermore, in order to allow customers to flexibly use multiple RSPs, and for the different RSPs to coordinate the requests of their customers, a uniform interface across RSPs is also needed. Addressing this issue is an interesting avenue for future work; a possible area of exploration is using logical predicates [96].

Resolving conflicts. Different RSP customers might have conflicting policies when requesting for paths between one another. Since RSPs have global information, they are the natural point to resolve conflicts between senders and receivers. In order to combine the policies from different customers, the RSPs might need more information from customers as to what are the fallback policies when the primary policy cannot be satisfied.

A more general, and perhaps more challenging, form of conflicts arises from the fact that

RAS architecture allows an open competitive market for multiple RSPs to coexist. Since customers getting service from different RSPs need to talk to each other, the RSPs must coordinate to satisfy the customer requests. Though this problem is hard, it is important to note that this problem is fundamental. Unlike the Internet today, RAS creates a playing field for different customers to interact through their RSPs and resolve the conflicts.

3.4.2 Data Plane: Forwarding Customer Traffic and Bookkeeping

Based on the paths computed using customer policies, the RSP forwards the customer traffic. The gateways keep track of the amount of traffic sent as well as the number of specialized route requests made by the customers. The gateway rate-limits the customer traffic so that the RSP does not exceed the traffic contracts on the virtual links with the ISPs. Also, since the gateways know the complete customer statistics, billing the customers is also easy. We expect that for large-scale RSP deployments, the cost of RSP gateways can be amortized across different RSPs. We can leverage the virtual router architecture [118] to isolate the RSPs from one another within the gateway.

In turn, a customer can monitor the service it receives from the path that the RSP provides. If the guarantees do not conform to what the RSP promises, it can inform the RSP, based on which the RSP can investigate the problem. (Of course, we believe that if the service is popular, competition would lead to multiple RSPs, and hence users would switch to better providers if they receive prolonged bad service from an RSP.) In addition, the RSP can refund the customer for the lower service, and reclaim the cost from the appropriate virtual link provider. In today's Internet, such diagnosis is almost impossible—if a host receives bad service, it is extremely difficult to pinpoint the problem to a particular AS, even if the problem is just a few hops away. RSP gateways can themselves monitor the status of its customers and detect whether the ISPs do provide the SLAs they promised.

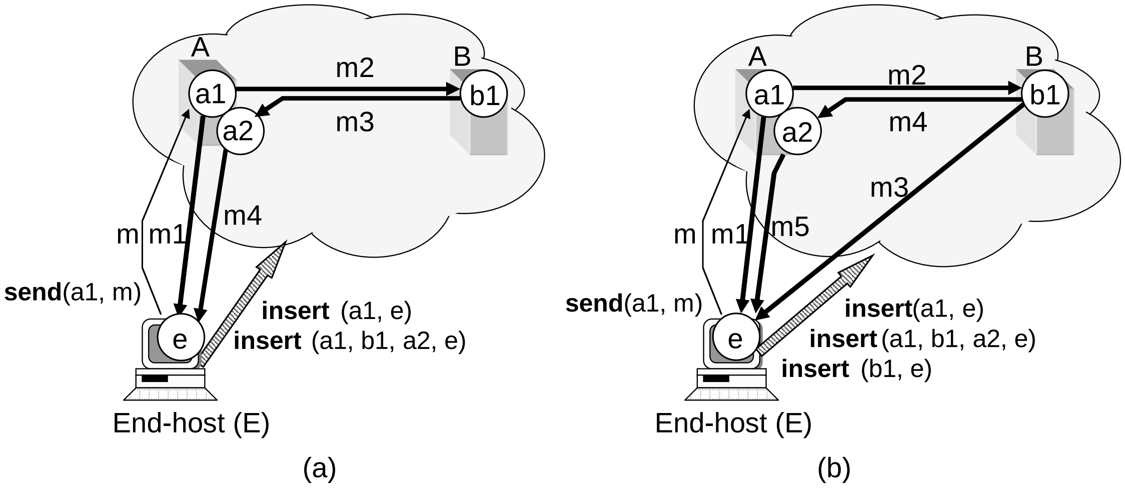


Figure 3.2. Communication pattern used to measure the (a) round-trip time (RTT) between two nodes A and B; E sends a probe to a_1; E computes the RTT as the difference between receiving the probe back from a_2 and a_1, (b) loss rates along virtual links ($A \rightarrow B$), and ($B \rightarrow A$).

3.4.3 Verifying Virtual Link Performance

In this section, we present several simple techniques that can be used by end-hosts and RSPs to measure the performance characteristics between any two nodes in the infrastructure. We assume that the infrastructure provides no support other than the ability to setup paths. The reason we make this assumption is two-fold. First, it allows us to illustrate the power and flexibility of the FI model. Second, this assumption leads to techniques that are more extensible and deployable than techniques that require additional support from the infrastructure.

Next we describe three simple techniques to measure the round-trip time, loss rate, and available bandwidth between any two infrastructure nodes.

Round-Trip Time. Consider the problem of estimating the RTT between two arbitrary infrastructure nodes A and B. [1] Figure 3.2(a) illustrates the technique used by a host E to perform this measurement. Let a_1 and a_2 be two IDs associated with node A, and b_1 be a ID associated to node B. E first inserts paths (a_1, e) and (a_1, b_1, a_2, e), and then sends

[1] Measuring the one way delay between A and B requires the clocks of the two nodes to be synchronized. Thus, measuring the one way delay is hard even assuming full control on the two nodes.

periodic probes to ID a_1. Upon reaching a_1, the probe packet is sent back to E, while a replica of the probe packet is sent along path (a_1, b_1, a_2) and finally back to e. The RTT between A and B is then computed as the difference in the time between receiving the copy and the original probe, *i.e.,* from a_2 and a_1 respectively.

Loss Rate. To measure the *unidirectional* loss rate between two nodes A and B, we use a setup similar to the one used to measure the RTT. The only difference is that E also inserts path (b_1, e) (see Figure 3.2(b)).

Let m be a probe sent by host E to ID a_1. For ease of explanation, in Figure 3.2(b), we label the copy of the probe along each virtual link of the path. Then E concludes that there was a loss from A to B (*i.e.,* m_2 was lost) if E receives m_1 but does not receive m_3 and m_5.

We now show that under the assumption that (i) the loss probability on each virtual link is $p \ll 1$ and (ii) the loss probabilities on virtual links are not correlated, the probability of false positives, *i.e.,* the probability that E incorrectly decides that m_2 was lost, is $O(p^2)$. Let $P(m_i)$ denote the probability that m_i is lost. Then the probability of false positive, P, is equal to the product of the probabilities of the following events: m_2 is not lost, either m_4 or m_5 are lost, and m_3 is lost.

$$\begin{aligned} P &= (1 - P(m_2))(1 - [1 - P(m_4))(1 - P(m_5)])P(m_3) \\ &\simeq 2p^2 \end{aligned} \tag{3.1}$$

Thus, the probability of false positive, $2p^2$, is considerably smaller than the measured loss rate p.

Though E can compute the loss rate on the reverse link, (B→A), by inverting the communication pattern shown in Figure 3.2(b), E can estimate this loss rate without any additional measurements. In particular, while measuring the loss rate for m_2, E also records the following two events:

1. the receipt of m_3 but not of m_5, and

2. the receipt of m_3 or m_5 but not of m_1.

Let f_1 be the frequency of occurrence of event 1, and f_2 be the frequency of occurrence of event 2. Then f_1 estimates the loss rate on virtual link $(B \rightarrow E)$ while f_2 estimates the loss rate on virtual link $(A \rightarrow E)$. Finally, E estimates the loss rate on $(B \rightarrow A)$ as $f_1 - f_2$. Note that this estimation procedure assumes that the losses on links $(B \rightarrow A)$ and $(A \rightarrow E)$ are not correlated. Finally, it can be shown that if the loss probability on each virtual link is $O(p)$, the probability of false positives for both f_1 and f_2 is $O(p^2)$.

Available Bandwidth. To measure the available bandwidth, we use a TCP-Vegas like algorithm [17]. Such an algorithm reacts to congestion when the RTT increases rather than waiting for a packet loss. This helps to minimize the impact of the measurement algorithm on the background traffic.

To estimate the available bandwidth on virtual link $(A \rightarrow B)$, host E inserts path (a_1, b_1, e), and sends traffic to ID, a_1. As a result the traffic will follow the path $(E \rightarrow A \rightarrow B \rightarrow E)$ as shown in Figure 3.3(a). E uses a slow start algorithm which exponentially increases the congestion window size, and consequently the sending rate, until the RTT exceeds the minimum RTT observed so far by a threshold (as in [17]). When this happens, E concludes that there is congestion on path $(E \rightarrow A \rightarrow B \rightarrow E)$. In order to verify whether the congestion is on link $(A \rightarrow B)$, E performs two operations:

1. Set-up a new path, (a_1, b_2), which has the effect of multiplying the traffic on $(A \rightarrow B)$ link alone.

2. Reduce the sending rate by half.

The net result of these operations is that while the rate of the traffic on both $(E \rightarrow A)$ and $(B \rightarrow E)$ is halved, the rate on $(A \rightarrow B)$ remains unchanged, as each packet is now sent twice on this link. Now, if RTT increases on increasing the sending rate, E concludes that $(A \rightarrow B)$ is congested and that the available bandwidth on the link is twice E's current sending rate.

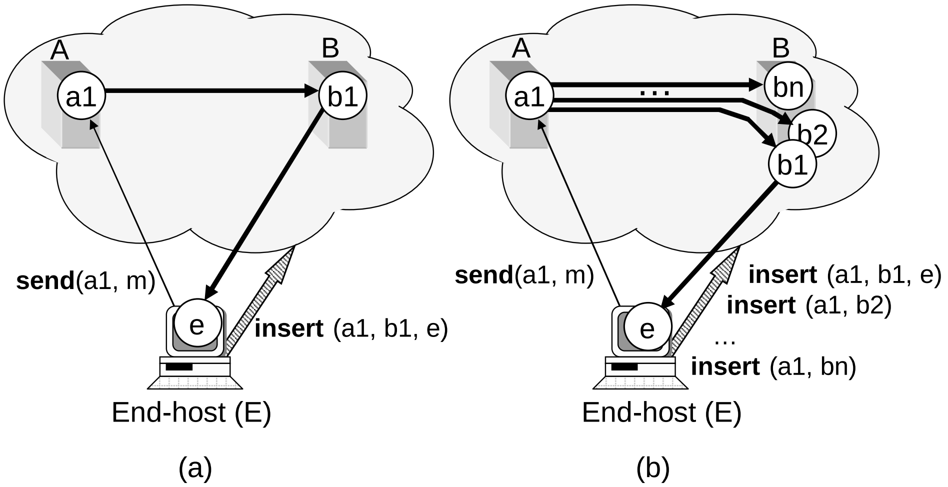


Figure 3.3. Communication pattern used to estimate the available bandwidth along virtual links ($A \rightarrow B$).

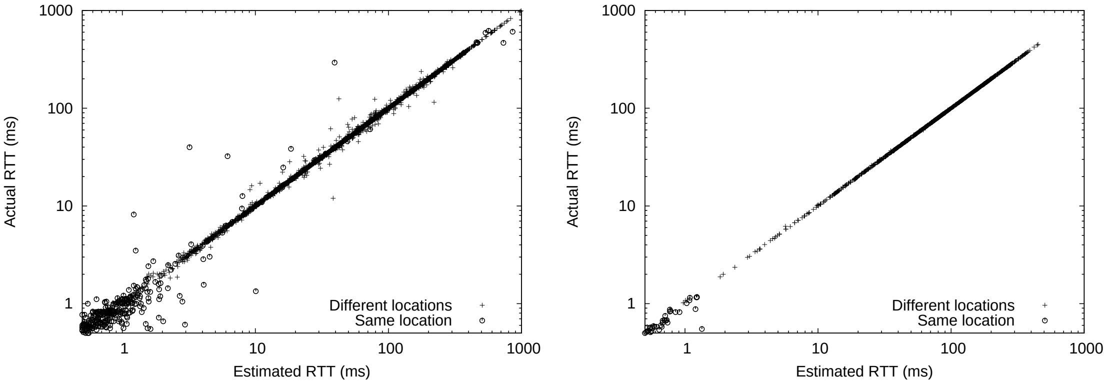


Figure 3.4. The scatter plot of the actual and the measured RTT between two arbitrary nodes.

If not, E increases the degree of replication on $(A \rightarrow B)$ further and repeats the process. In general, the degree of replication can be set to i by inserting $(a_1, b_1), \ldots, (a_1, b_i)$.[2]

3.5 Experiments

Our implementation of the routing substrate (forwarding infrastructure) was based on *i3*. We benchmarked our implementations on a Pentium 2.2 GHz machine running Linux

[2] In the implementation of our abstraction, to avoid the problem of an end-host performing a DoS attack on the infrastructure node, we require that all paths terminate at end-hosts.

2.4.20. Each routing entry takes 120 bytes of memory. To evaluate the cost of inserting an entry into the table, we first populated the routing table with $100,000$ entries. Then the cost of insertion, computed as the average of next $10,000$ insertions, was $2.5\mu s$. Not including the cost of routing table lookup, the cost of packet forwarding averaged over $10,000$ packets of size 1 KB was $18.5\mu s$. This corresponds to a forwarding rate of up to 47.6 thousand packets per second.

In the rest of the section, we present the experimental evaluation of our design. Our experiments are conducted on the PlanetLab testbed, on 108 machines located at over 50 locations in US, Europe, Asia and Australia. The fact that PlanetLab machines are relatively well-connected is consistent with our design that assumes an overlay *infrastructure* rather than an end-host based overlay. The goal of the following experiments is two-fold: (i) evaluate the accuracy of the algorithms that estimate the virtual link characteristics (described in Section 3.4.3), and (ii) present a brief evaluation of how using the system along with RAS would impact applications.

3.5.1 Virtual Link Experiments

In this section, we evaluate the estimation algorithms described in Section 3.4.3. For each performance metric (*i.e.,* RTT, loss rate and available bandwidth) we compare the *actual* values to the values estimated by our algorithms. To compute the actual values we instrument infrastructure nodes to log packets they receive (for delay and loss rate) and perform pairwise measurements between them (for bandwidth). Our experiments show that our estimation algorithms are accurate in the case of RTT and loss rate, and reasonable in the case of bandwidth estimation. To understand how RTT and loss rate estimation algorithms performs in different cases, we divided the pairs of infrastructure nodes into two bins:

- Pairs of nodes in the same location (*e.g.,* both in MIT)
- Pairs of nodes in different locations (*e.g.,* one in Stanford, other in Rice).

For the former, we considered 44 pairs (44 locations during our experiment had two or more alive nodes), and for the latter we considered about 2450 pairs, *i.e.,* two data sets, each set comprising all pairs of nodes, one in each of the 50 PlanetLab locations.

Round Trip Time (RTT). Figure 3.4(a) shows the scatter plot of the estimated RTT versus the actual RTT for the samples over a 100 sec interval. Every virtual link is sampled every 10 sec [3] From Figure 3.2(a), recall that the estimated RTT is computed as the difference between the arrival time of copy m_4 and the arrival time of packet m_1 at node E, while the actual RTT is computed as the time interval between sending m_2 and receiving m_3 at node A. Figure 3.4(b) shows the scatter plot of the median RTTs for the samples plotted in Figure 3.4(a). These results show that our RTT estimation algorithm is very accurate, even when both nodes are in the same location. In Figure 3.4(a), less than 2.7% of samples have an error >10%. If we take the median among 10 consecutive samples (as in Figure 3.4(b)), only 0.7% of the samples have a relative error >10%, most of which are due to nodes in the same location.

Loss Rate. Figure 3.5 shows the scatter plots of the actual versus the measured loss rates for the aforementioned data sets. To estimate the loss rate between two nodes we use the scheme described in Section 3.4.3. Each data point is the result of sending 1000 probes. In most cases, the measured loss rates were quite small; only in about 8% of cases we measured a loss rate larger than 2%. Figures 3.5(a) and (b) show the loss rates for all links in the forward, and reverse directions. The points below the line $x = y$ are mainly due to false positives, *i.e.,* the node incorrectly decided that there was a loss on the monitored link. The points above $x = y$ are due to the fact that the measurement node ignores the probes for which it does not receive any response.

From the description of the technique, it follows that inaccuracies could occur when, (i) the loss between the measurement server and the measured link ($A{\to}E, B{\to}E$ links in Figure 3.2(b)) is considerably larger than the loss rate on the measured virtual links ($A{\to}B$ and $B{\to}A$ in Figure 3.2(b)) and (ii) losses on links are not independent. To verify if our

[3] We restricted our sampling rate as we monitored all virtual links for the sake of validating our techniques.

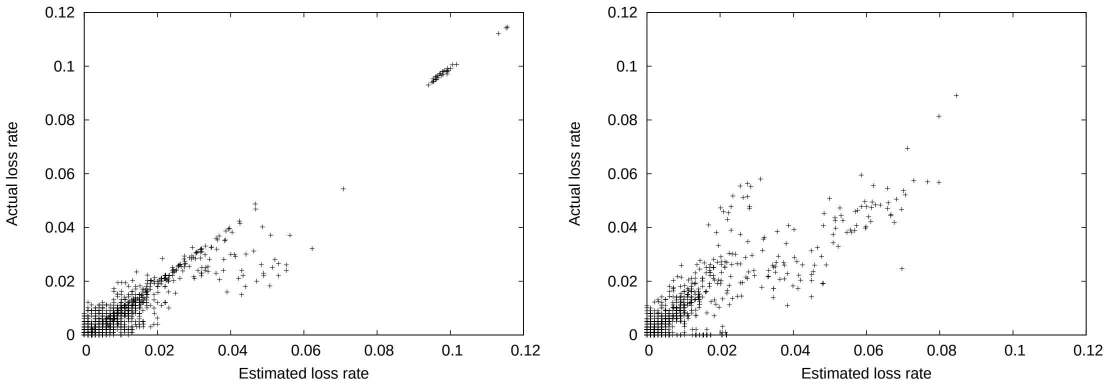


Figure 3.5. Scatter plots of the actual versus the measured loss rates for: (a) forward path, (b) reverse path.

data bears this out, we identify the nodes that are responsible for the highest loss rates, and eliminate them from the measurements.[4] As expected, the estimation accuracy improves considerably, especially on the reverse path. The loss rate estimates were within 90% of the actual loss rate in 70% of the cases in the forward direction and in 74% of the cases in the reverse direction. Finally, if we consider only virtual links with loss of two or more packets out of 1000 packets, then the estimates were within 90% of the actual for 81% of cases in forward direction and for 83% of the cases in the reverse direction. In summary, the algorithm performs very well in identifying moderately/highly lossy links, and does reasonably well for links with very low loss rate.

We make two further observations. First, in the forward direction (Figure 3.5(a)), we are more likely to overestimate than underestimate the loss rates (all points below $x = y$ represent overestimations). We do not expect over-estimations to be a serious problem in practice. If the RSP over-estimates the loss rate on a particular link, the worst it can do is to not to return the best path to an application. However, in a network with rich connectivity we expect that the effects of such occasional sub-optimal paths to be minimal. Second, one can easily obtain better results for the reverse path by simply reversing the measurement setting for that links. However, this results in doubling the overhead since now we have to send probes in both directions of the measured link.

[4] During the experiments reported here we identified five such nodes: two at `cs.unibo.it`, one at `Intel Pittsburgh`, one at `CMU`, and one at `diku.dk`.

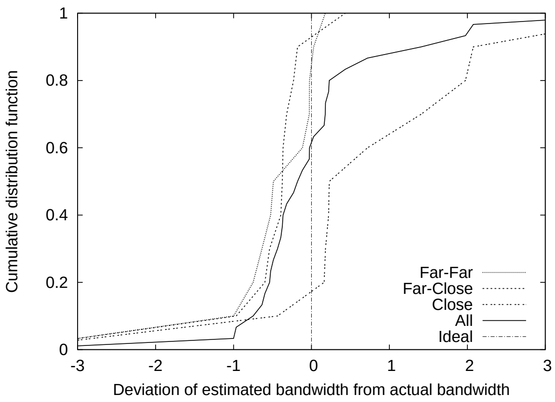


Figure 3.6. CDF of deviation in estimated bandwidth from measured bandwidth.

Available Bandwidth. To evaluate the technique for determining the available bandwidth (avail-bw), we chose one node at each PlanetLab site.[5] Though there are multiple definitions of avail-bw, to get a better application-centric picture, we use the one that defines it as the bandwidth that a long term TCP flow would get. Thus, we compute the actual avail-bw between any two nodes by transferring 1 MB of data between the two nodes, and recording the bandwidth over the transmission of the last 500 KB.[6] To understand the performance of our algorithm clearly, we divided the pairs of nodes into the following three vastly different cases, primarily motivated by the fact that longer RTTs might affect the estimation process: (i) measurement server is close to one (or both) of the two nodes (labeled Close in Figure 3.6) (ii) measurement server is far away from the two nodes which are themselves far apart (labeled Far-Far), and (iii) measurement server is far away from the two nodes which are close to each other (labeled Far-Close).

Figure 3.6 shows the CDF of the relative deviation of the estimated bandwidth from the stable TCP bandwidth. We make the following observations; a detailed analysis about the timing effects of the measurement is beyond the scope of this paper: (i) Overall, in 60% of the cases, the relative error in estimated bandwidth is less than 0.5. For comparison, in 43% of the pairs, the relative deviation in two values measured using TCP transfers was itself 0.5 or more, (ii) For Far-Far and Far-Close, in the cases when our estimation is

[5] Only one node per site was used to economize on the bandwidth that we use for our experiments.

[6] This is to allow the flow to get into congestion avoidance phase, so that we see the stable TCP bandwidth.

off by more than 30%, we only under-estimate (not over-estimate) the bandwidth. Also, serious under-estimation happens only when the actual bandwidth is itself very large. Much like over-estimation of loss rate, we do not expect under-estimation of bandwidth to be a serious problem in practice, (iii) The technique did not work well when we estimated high-bandwidth links that were very far from the measurement server – a node in USA could not estimate the high bandwidth between `uu.se` and `diku.dk` well. This was because the two Europe nodes were separated by only 11ms, whereas the measurement server was over 200ms away from each of them. In a deployed system, we can alleviate the problem by having a distributed RSP service at multiple locations, and (iv) In the case when one of the nodes is close to the measurement node, the available bandwidth technique over-estimates in some cases. We believe that this is because the overlay path and the IP path are the same, and a delay-based technique sees fewer losses. Finally, we recognize that this particular technique would work only for drop-tail queues; studying techniques for various AQM techniques (such as RED) is a topic of future work.

3.5.2 Evaluating Application Metrics

In this section, we briefly evaluate the effectiveness of RAS as seen by applications by comparing the path returned by RSP to the underlying IP path; we plan to conduct a more comprehensive evaluation in the future. While we are not the first to show that applications can get better performance by choosing alternate paths (this point has already been made in [9,100]), we show that one can achieve similar results by monitoring only a small fraction of the overlay links.

Delay CDF. Figure 3.7 plots the CDF of the relative delay penalty (RDP) between two arbitrary nodes in PlanetLab. The RDP between two nodes n_1 and n_2 is computed as the ratio of (1) the lowest RTT path between n_1 and n_2 in the graph maintained by RSP to (2) the RTT of the direct IP path between n_1 and n_2, respectively. By maintaining a random graph (RG heuristic) by picking a random subset of the links, the RDP is quite large. However, by adding proximity links (PRG heuristic) with a net average degree of

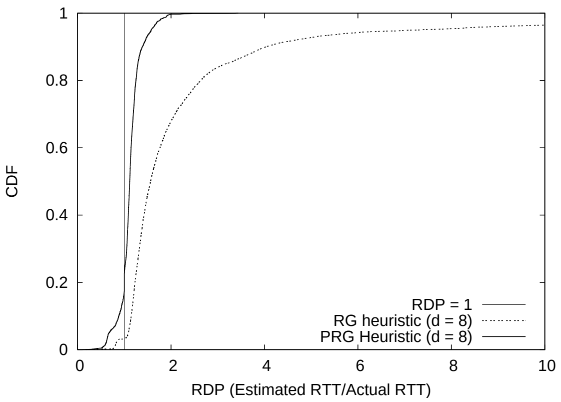


Figure 3.7. The cumulative distribution function (CDF) of the relative delay penalty (RDP) for all pairs of a 108 node network.

just 8, more than 99.7% pairs have RDPs smaller than 2, and no pair has an RDP larger than 4. Furthermore, 13% of the pairs have RDPs smaller than one, *i.e.,* the latency of the path returned by RSP is *smaller* than the latency of the direct IP path.

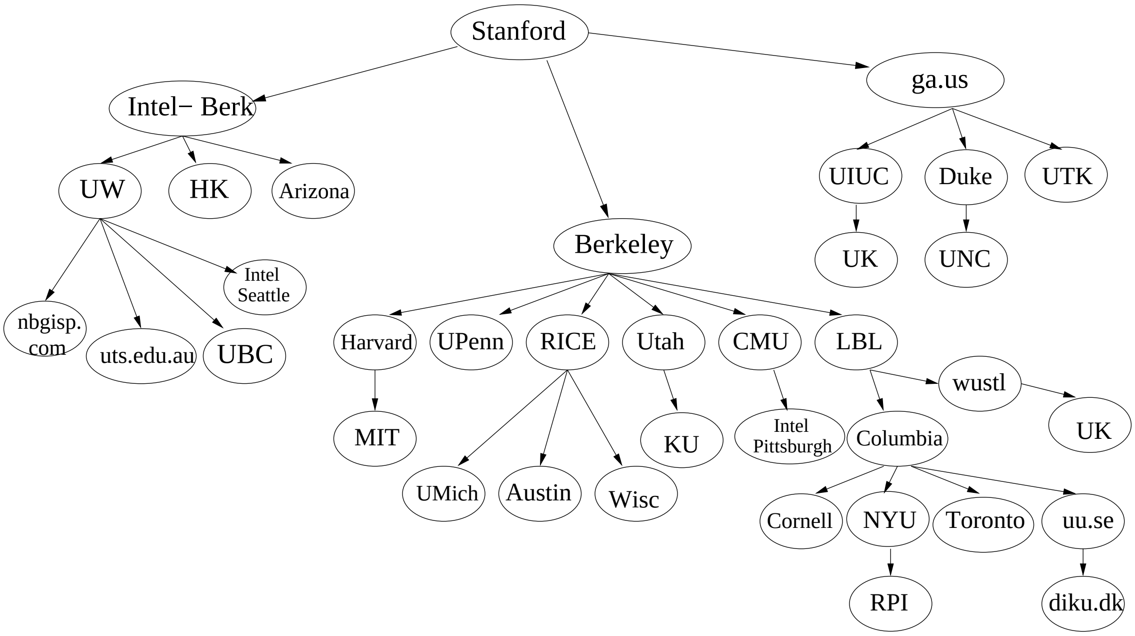


Figure 3.8. Delay-based multicast tree, with source at Stanford

Multicast study. A single source multicast tree is merely a union of the unicast paths that the RSP would return for each receiver. Using nodes on 37 PlanetLab sites and having one receiver at each node, we built a single source delay-optimized multicast tree using the unicast paths that the RSP returned. Figure 3.8 shows the tree with the source at planetlab-2.stanford.edu. Since in most cases, the machines on the same site were adjacent in the multicast tree, we represent each site by one node in the figure. We make a couple of observations: (i) Link stress, i.e. the number of times a packet is replicated on a link, is small (atmost 6 in this case), (ii) The tree reflects the underlying geography to a good extent, for e.g., nodes in the vicinity of NY, *i.e.,* Columbia, NYU, RPI, Cornell and Toronto are close together in the tree.

3.6 Related Work

Host control over routing. Though several proposals give end-hosts more control over routing, they do not directly address the tension between ISPs and customers. Architecturally, the design of RAS is closest to that of service overlay networks (SON) [36]. SONs purchase bandwidth from individual network domains via bilateral agreements, and use them to construct end-to-end paths that provide certain guarantees. However, they focus on QoS routing alone, and deal with the related problem of bandwidth-provisioning; in contrast, RAS provides end-to-end paths based on performance and policies, and includes ISPs' preferences in computing such paths.

To promote competition among providers, Yang [127] proposes a solution that allows both senders and receivers to choose routes at the AS level. The Nimrod [24] architecture proposed computation of routes by the clients of the network, and introduced mechanisms for distribution of network maps. Broker [16] is a centralized entity that computes routes based on QoS requirements within a domain, and across domains. TRIAD [27] has proposed a name-based routing scheme where end-hosts specify the path across multiple address space domains. To provide resilience, feedback-based routing [128] presents an architecture where edge networks perform measurements to find efficient routes in the network. None of these approaches provide flexible, yet scalable, mechanisms for selecting *end-to-end* paths that RAS aims to provide.

Commercial solutions. Several companies have created Intelligent Route Control products [13, 1] that allow a multi-homed organization to select which upstream provider to use to reach each destination. However, these solutions only provide control over the choice of the next-hop AS, not the end-to-end path, and operate at the granularity of destination prefixes, not individual applications.

Pushing routing control out of the network. The RCP and 4D proposals [38, 51] advocate moving control of routing from the individual routers to dedicated servers in each AS, but does not consider giving third-party providers control over the end-to-end path.

3.7 Summary

In order to resolve the fundamental tussle of routing control between ISPs and customers, we propose that customized route computation should be offered as a *service* by third-party providers. Outsourcing specialized route computation allows different path-selection mechanisms to coexist, and evolve over time.

While the overall approach of RAS is promising, there are particular issues that merit further research. For instance, in our basic design, RSPs have to sign SLAs with ISPs for individual virtual links. This might introduce scalability concerns when RSPs deal with tens of thousands of virtual links. Perhaps, combining various virtual link SLA into a topology-level SLA across all virtual links within a domain is a possible strategy for RSPs. Finally, we plan to investigate incremental deployment strategies for RSPs to deploy the forwarding infrastructure.

Chapter 4

Securing Forwarding Infrastructures

Several recent proposals have argued for giving third-parties and end-users control over routing in the network infrastructure. They range from various source-routing architectures [27, 106, 127] to more radical architectures that allow users to setup forwarding state in the infrastructure [106, 116, 117]. Exposing control over routing to third-parties significantly increases the flexibility and extensibility of these network infrastructures. Previous proposals have shown that, using such control, hosts can achieve many functions that are difficult to achieve in the Internet today. Examples of such functions include support for mobility, multicast, content routing, service composition, and denial-of-service (DoS) protection.

We propose a generic network model for such route-control-based architectures, which we call *forwarding infrastructure* (FI), that allow untrusted third-parties to install forwarding state at the infrastructure nodes. Examples of FIs include *i3* [106], DataRouter [116] and Network Pointers [117].

FIs provide numerous benefits, but unfortunately their flexibility also introduces many security vulnerabilities. Their flexibility allows malicious entities to attack both the FI as well as hosts connected to the FI. For instance, we show that an attacker can use an FI to amplify a flooding attack, or eavesdrop on the traffic of another end-host even when the attacker does not have access to the physical links carrying the victim's traffic. These vulnerabilities should come as no surprise; in general, the greater the flexibility of the infrastructure, the harder it is to make it secure [8, 123].

Our main goal is to show that the FIs are no more vulnerable than traditional communication networks such as IP, which do no export control on forwarding. To this end, we present several general mechanisms that make these FIs secure against all attacks we are aware of, yet retain the essential features and efficiency of the original design. Our main defense technique, which is based on light-weight cryptographic constraints on forwarding entries, provably prevents several attacks including eavesdropping, loops, and traffic amplification attacks. From earlier work, we leverage some techniques, such as challenge-responses and erasure coding techniques, to thwart other attacks.

In this chapter, we make the following contributions.

- We propose a model for FIs (Section 4.1), and analyze security vulnerabilities of these FIs, such as creating loops and amplifying attacker traffic (Section 4.2). While the attacks by themselves are rather unsurprising, they range of attacks we present help us in motivating why the problem is hard, and the properties needed for a solution.

- We derive the following properties which, if guaranteed, address all the vulnerabilities we are aware of: (a) an attacker should not be able to eavesdrop on the traffic to an arbitrary host, (b) an attacker should not be able to worsen its attack on end-hosts using the FI, and (c) an attacker can only cause a small bounded attack on the FI (Section 4.3).

- We describe a set of security mechanisms that achieve these three properties for a secure FI (Section 4.4). The most important contribution, light-weight cryptographic constraints on forwarding entries, *provably* prevents malicious topologies. For example, to prevent loops, we leverage the difficulty in finding short loops in the mapping defined by cryptographic hash functions [77]. To the best of our knowledge, this is the first system that exploits the difficulty in finding short loops in cryptographic hash functions for designing a secure routing system.

Our main high-level goal is to push the envelope of the extent of security that truly flexible communication infrastructures, that provide a diverse set of operations including packet replication, allow.

4.1 Forwarding Infrastructure Model

Since the designs of various FIs proposals (such as [31, 97, 106, 116, 117]) vary greatly, we present a simple model that abstracts the functions of these proposals. At its core, the FI model we present is similar to MPLS [97]. For generality, the model allows both source routing and insertion of forwarding entries. In contrast to source routing, allowing forwarding entries enables some functions like mobility and multicast [106, 117]. The three steps in routing a packet are: (1) matching the packet header with forwarding entries, (2) modifying the packet header based on the forwarding entry it matches, and (3) forwarding the packet to the next hop. Figure 4.1 illustrates the packet processing at an FI node.

Packets are replicated if multiple forwarding entries are matched, thus allowing the FI to provide multicast. While precluding replication would eliminate several of the attacks that we discuss here, we believe that multicast is a key functionality that future FIs will provide.

Forwarding Entries. Each FI node maintains a table of forwarding entries. A forwarding entry is a pair *(key, finfo)* in which *key* is used to match packet headers and *finfo* is used to modify the packet header and forward the packet. Keys are not necessarily unique; so, there may be multiple entries with the same key. The scope of a *key* is *local* to an FI node. To distinguish between entries with the same *key* stored at different FI nodes, we denote

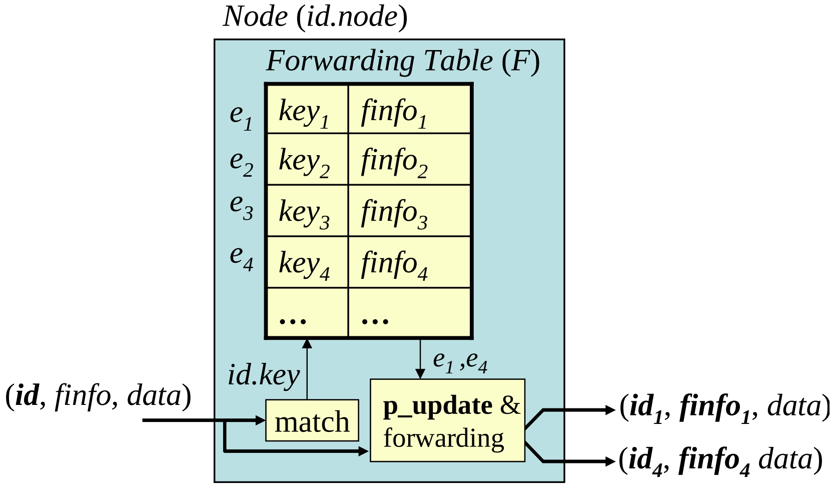


Figure 4.1. The operations performed by an FI node upon the arrival of a packet with identifier *id*.

a forwarding entry by *(id, finfo)* where *id* is an identifier with two fields: *id.node* which represents the node where the forwarding entry is stored, and *id.key* the key.

Packet Matching. A packet is a tuple *(id, finfo, payload)* in which *id.key* identifies one or more forwarding entries stored at the next-hop *id.node*, *finfo* contains additional forwarding information such as a source route or a stack of identifiers, and *payload* is the packet's payload. When a packet arrives at node *id.node*, its *id* is matched against keys in the forwarding table by some general matching function:

$$\mathbf{match}(id, F) \rightarrow \{e_1, e_2, \ldots, e_k\}, \tag{4.1}$$

which takes as input a packet's *id* and a forwarding table F (stored at node *id.node*), and outputs a set of entries. Examples of matching operations are exact, longest prefix, and proximity matching.

Packet Header Update. The header and destination of a packet are based only on the incoming packet's header and the matching entry. If multiple entries are matched, the packet is replicated. The update function:

$$\mathbf{p\_update}(p, e) \rightarrow p' \tag{4.2}$$

takes a packet p and an entry e, and produces a modified packet p'. The update function does *not* modify or inspect the payload; it only updates *p.id* and *p.finfo*. The semantics of **p_update** is protocol-specific and not relevant to our discussion here. The only important aspect is that **p_update** *determines* the next hop by updating *p.id*. We assume that hosts know the **match** and **p_update** operations at an FI node.

4.1.1 An Example: Internet Indirection Infrastructure (*i3*)

For concreteness, we give an example of an FI to illustrate how an existing FIs can be instantiated in our model. *i3* is an indirection overlay that allows hosts to specify which packets they want to receive by inserting forwarding entries with the appropriate identifiers [106]. In the simplest case, at an *i3* node, the incoming packet p contains the identifier id_a (*i.e., p.id* $= id_a$). The identifier, id_a determines the matching entry in the forwarding table, as well as the next hop of the packet. Let us say that id_a matches an entry $id_a \rightarrow (id_b, id_c)$; here (id_b, id_c) denotes a stack of IDs. Then, the ID id_a is replaced by the ID id_b, id_c is added to *p.finfo*, and the packet is forwarded to id_b.

Both *p.id.node* and *p.id.key* are encoded in the *i3* ID of the packet. The **match** operation is longest prefix matching. The **p_update** operation swaps the identifier at the top of the stack with the stack in the matching entry *e.finfo*. Note that host addresses can be encoded in *e.finfo* for packets sent to end-hosts.

4.1.2 User Control over Forwarding Entries

We assume that FI nodes allow insertion and removal of entries to and from their own forwarding tables.

insert(*n, e*); *// insert entry e into n's forwarding table*

remove(*n, e*); *// remove entry e from n's forwarding table*

In the rest of the chapter, we denote an entry that changes the ID of a packet from id_1 to id_2 by $[id_1 \rightarrow id_2]$, *i.e.,*

$$\mathbf{p\_update}(p, [id_1 \rightarrow id_2]) \rightarrow p',$$

where $p.id = id_1$ and $p'.id = id_2$. We use this notation *only* when a forwarding entry completely determines the new value of *id*. Note that in some cases (such as source routing), it is not possible to determine the next ID using the forwarding entry alone.

Following *i3* terminology [106], we assume that there are two types of IDs: *public* and *private*. These IDs differ in their level of "visibility" to end-users: a public ID is publicly known, while a private ID is known only to a trusted set of users. Similarly, we call a forwarding entry whose ID is public/private, a public/private forwarding entry. Public forwarding entries might be used by servers that arbitrary users can contact—all packets delivered to such servers will be relayed through their public forwarding entries.

4.2 Threat Model

We describe our assumptions and the attacker threat model, and then derive the attacks that can be launched.

4.2.1 Security Assumptions

Our main goal is to show that the FIs are no more vulnerable than traditional communication networks such as IP, which do no export control on forwarding. To achieve this goal, we rely on several assumptions about the underlying routing layer. We assume that the virtual links between FI nodes and between FI nodes and end-hosts provide secrecy, authenticity, and replay protection—i.e., we do not consider link-level adversaries that can eavesdrop on arbitrary network links. These virtual links represent customer-ISP and ISP-ISP relationships, which can be readily secured through standard security protocols (*e.g.,* IPsec [64]), and do not need a public-key infrastructure.

FI proposals rely on an underlying routing protocol that routes packets between FI nodes. For example, DataRouter uses IP routing, and *i3* uses the Chord lookup protocol [108]. We do not address the security issues of these underlying protocols. We note that there are several ongoing research efforts to address security issues both in the context of IP routing [49, 59, 65, 101, 112, 125] and DHT-routing [25, 103]. Finally, we do not consider state-based attacks such as insertion of many forwarding entries at an FI node since these attacks are well-studied in the literature and can be solved using cryptographic puzzles [34, 37, 78].

4.2.2 Attacker Threat Model

We consider two attacker types: internal and external attackers. An *external attacker* does not control any compromised FI node but misuses the flexibility given by the FI to mount attacks. An external attacker can perform only the operations that a legitimate host can: insert a forwarding entry and send a packet. As we will see next, the attacker can use these operations to amplify flooding attacks[1] on the infrastructure or end-hosts, and eavesdrop on end-host communication. An *internal attacker* is an adversary who controls some compromised FI nodes. Ideally, we want to ensure that an external attacker cannot misuse an FI network to amplify the magnitude of attack. In the case of an internal attack, we want to ensure that an attacker who compromises an FI node cannot affect other traffic that is not forwarded through that compromised FI node. Our main focus in this paper is design of security mechanisms to protect the hosts and FI against external attackers; we discuss the issue of internal attackers in Section 4.5.

4.2.3 External Attacks

The two types of attacks that an external attacker can mount are those that: (a) involve only packets and forwarding entries inserted by the attacker, and (b) involve packets or forwarding entries of other end-hosts.

[1] By flooding attack, we refer to a DoS attack in which the attacker floods the victim's network link by sending data at a large rate.

While an attacker can insert arbitrary forwarding entries, we enforce that an attacker cannot update or remove entries inserted by *others.* To remove or update an entry, users need to specify both fields of the entry: the *key* and the *finfo* fields. Since our model does not provide a primitive to read an entry, an attacker can modify an entry only by guessing the fields. By allowing the owner of an entry to include a sufficiently long random nonce (80 bits long suffice [69]) in the *finfo* field, we can ensure that guessing the *finfo* field is highly improbable. We also assume that it is infeasible for an attacker to guess the ID of a *private* forwarding entry. As before, we enforce this by including a nonce in the ID of a private entry.

Creating New Topologies: Attacks on FI

An attacker can insert forwarding entries to amplify a flooding attack on the FI. We present examples of such undesirable topologies.

Cycles. An attacker can form a loop by inserting forwarding entries $[id_1 \rightarrow id_2]$, $\ldots$, $[id_{n-1} \rightarrow id_n], [id_n \rightarrow id_1]$ (see Figure 4.2(a)). A packet with identifier id_i $(1 \leq i \leq n)$ would indefinitely cycle around the loop and consume FI resources.

Dead-ends. An attacker can construct a chain of forwarding entries, or even a multicast tree, which do not point to a valid end-host (see Figure 4.2(b)). Data packets sent on such a topology would be forwarded and replicated only to be dropped at the dead ends.

Confluence. An attacker can refine a dead-ends attack by constructing a multicast tree with m leaves, all pointing to a victim FI node. For every packet sent by the attacker, the destination will receive m duplicates.

Attacking Existing Topologies: Attacks on Hosts

Though an end-host cannot update or remove the forwarding entries inserted by another end-host, it can affect the traffic forwarded from/to these entries in the following ways.

Eavesdropping. Consider an end-host R that inserts a public forwarding entry[2] $[id{\rightarrow}R]$ (see Figure 4.2(d)). An attacker X can eavesdrop on packets sent to R by inserting a forwarding entry $[id{\rightarrow}X]$. All packets that are forwarded via $[id{\rightarrow}R]$ will be replicated and forwarded via $[id{\rightarrow}X]$ to X as well.

Impersonation. A variant of eavesdropping involves an attacker X making an end-host R drop its public entry by flooding it.[3] Then, if attacker X inserts $[id{\rightarrow}X]$, X can not only eavesdrop on R's traffic but also actively respond to it, thus impersonating R.

Malicious linking. Consider a forwarding entry $[id_1{\rightarrow}{*}]$ that receives a large number of packets. An attacker can sign up an end-host R, with an existing public forwarding entry $[id{\rightarrow}R]$, to the high bandwidth traffic stream of the popular entry by inserting the entry $[id_1{\rightarrow}id]$.

Cycles involving end-hosts. Consider two benign hosts R_1 and R_2 inserting entries $[id_1{\rightarrow}R_1]$ and $[id_2{\rightarrow}R_2]$ respectively. An attacker can create a cycle by inserting entries $[id_1{\rightarrow}id_2]$ and $[id_2{\rightarrow}id_1]$. Packets sent to id_1 and id_2 would be indefinitely replicated, thus overwhelming R_1 and R_2.

End-host confluence. This is a variant of the confluence attack where the target is an end-host rather than an FI node. By making the leaves of the tree point to the public entry of an end-host (see Figure 4.2(c)), an attacker can overwhelm the host.

Why TTL alone does not suffice

Using a time-to-live field is a common technique to prevent routing cycles in networks, with IPv4 networks being the prime example. We briefly explain why TTLs alone aren't sufficient for preventing cycles and confluences in FIs.

If we use TTLs alone, then with a TTL of l, an attacker can replicate a packet l times by

[2] To improve readability, we simplify the notation: we write $[id{\rightarrow}R]$ to mean $[id{\rightarrow}id_R]$, where $id_R.node = R$.

[3] We assume that R maintains its entry using soft state (since forwarding tables are usually managed using soft-state). We also assume that a host under flooding attack cannot refresh its entries.

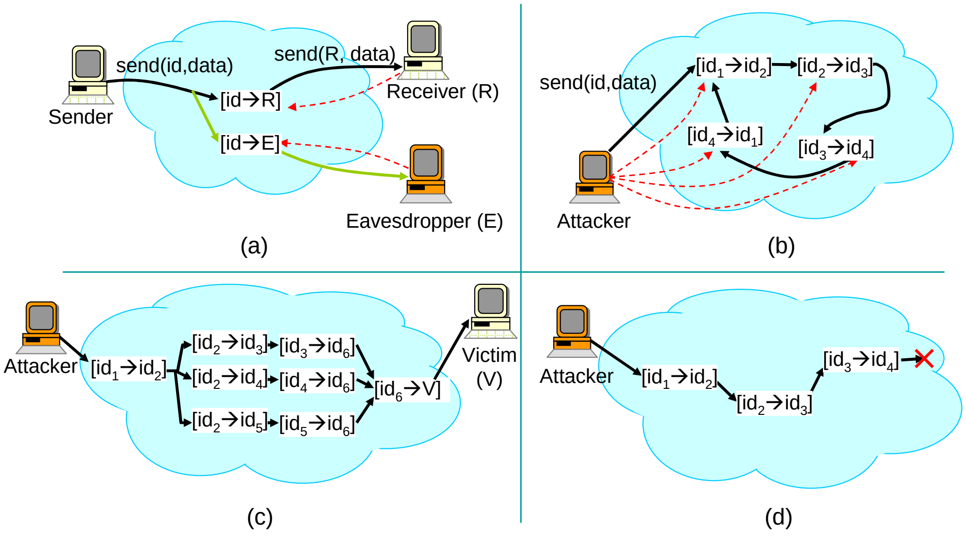


Figure 4.2. Attack examples: (a) eavesdropping, (b) cycle, (c) end-host confluence, and (d) dead-end.

inserting just two entries (id_1, id_2) and (id_2, id_1). However, the constrained IDs technique makes the construction of short cycles infeasible. Hence an attacker has to insert several forwarding entries to replicate the traffic. By bounding the rate of insertion of new entries, we show that we can alleviate attacks effectively.

4.3 Properties of a Secure FI

To abstract out the details of the different attacks, we present three properties of a secure FI that, if guaranteed, ensures that these attacks are thwarted. In particular, the properties ensure that: (a) an attacker *cannot* launch attacks on end-hosts (Properties 1 and 2 address attacks in Section 4.2.3), and (b) an attacker can cause only limited packet replication to attack the FI nodes (Property 3 addresses attacks in Section 4.2.3). In the next section, we present defenses and show how the defenses guarantee these properties.

4.3.1 Preventing Eavesdropping and Impersonation

Property 1. *Let (id, *) be a public forwarding entry inserted by a legitimate user. Then, an attacker cannot insert a forwarding entry with the same ID, id.*

This property prevents eavesdropping and impersonation by preventing an attacker from inserting a forwarding entry with the same ID as that of the victim. The property also covers the case in which the victim has no entry in the FI at the time the attacker inserts its entry. Hence, even if the attacker causes the removal of the victim's entry (*e.g.,* by flooding the victim), it cannot impersonate the victim.

4.3.2 Preventing Attacks on End-Hosts

The following property prevents an attacker from using the FI to: (a) amplify the traffic it sends to a victim host (thus preventing *ID-level confluences*), and (b) redirect traffic meant for other hosts to the victim host (thus preventing *malicious linking* and *cycles involving end-hosts*).

Property 2. *An attacker cannot make a single victim end-host receive more packets than the attacker itself sends.*

4.3.3 Limiting Attacks on FI

While the previous two properties *prevent* attacks on end-hosts, the next property only alleviates attacks on the FI (*e.g.,* confluences, dead-ends). We state the property after introducing a new metric: *damage ratio.*

Definition 1. *Consider a packet m that traverses l links (i.e., the packet is forwarded l times) in the FI and that is received by k receivers. The forwarding cost of m is then*

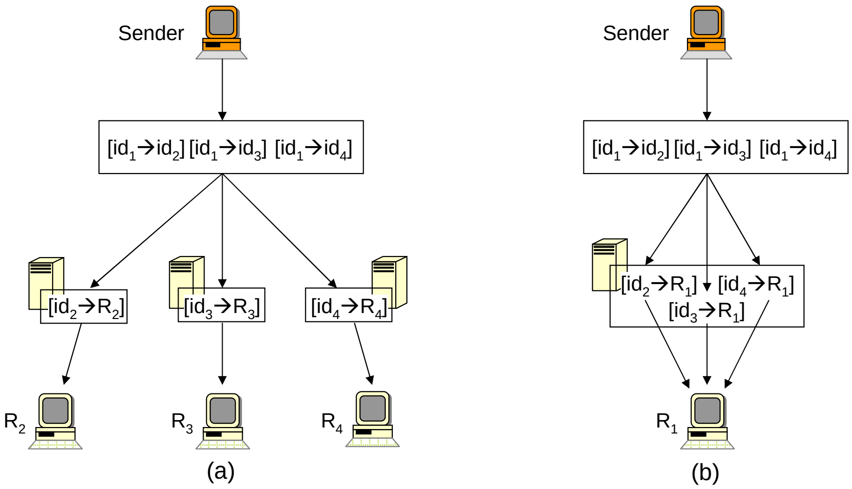


Figure 4.3. Example where a legitimate control plane topology can be exploited for attack at the data plane.

$$FC(m) = \frac{l}{k+1} \tag{4.3}$$

The forwarding cost measures the amount of work the FI does for every unit of work performed by end-hosts involved in the communication, where a unit of work is either sending, receiving or forwarding a packet. (The increment by one in the denominator of Eq. (4.3) accounts for the sender sending a packet.) For example, the forwarding cost of a unicast packet traversing h hops is $h/2$ if it is delivered to the receiver, and h otherwise. A cycle has infinite forwarding cost; by imposing a TTL of l, the cost of a cycle would be bounded by l.

To reduce the ability of an attacker to use the FI to amplify its attack, we should make the forwarding cost as small as possible. The following property captures this requirement.

Property 3. *The forwarding cost is bounded and small.*

This property counteracts *oversubscription* attacks, where an attacker builds a legitimate topology, but makes the FI do extra work by sending packets at a much higher rate than the (colluding) receivers can handle. Consider the legitimate multicast topology in Figure 4.3(a). An attacker can exploit this topology to mount an attack on an FI node, by having all leaves terminate at a colluding receiver (see Figure 4.3(b)) which has limited receiving capability, and make all the IDs in the penultimate level reside on the same FI node. This will cause all the replicated traffic to be directed to that FI node.

Since oversubscription attacks can be mounted using legitimate topologies, a defense mechanism can detect such attacks only after the attacks are started. Thus, we can only alleviate such attacks, not prevent them completely. Property 3 achieves this by linking the damage caused by an attacker to how much communication resources the attacker has, *i.e.,* it bounds the ratio between how many packets an FI forwards on the behalf of the attacker and how much traffic the attacker can send/receive directly to/from the FI.

4.4 Defense Mechanisms

We present defense mechanisms that achieve the properties of a secure FI that we enumerated in the previous section. The first technique, *constrained IDs*, is our main technique. We also use two other well-known techniques—challenge-responses and erasure coding. The constrained IDs technique enforces property 1, and together with the challenge response technique, they enforce property 2. By using all three techniques, we can provide property 3.

4.4.1 Technique 1: Constrained IDs

Constrained IDs is our core technique, which prevents eavesdropping, impersonation, and the construction of topologies that are not trees. Consider an FI node that updates the packet ID from id to id'. We enforce a constraint on the structure of IDs such that the choice of id cryptographically constrains the choice of id' or vice-versa.

To implement the constraints, we divide $id.key$ into two sub-fields: a constrained part ($id.key.c$) and an unconstrained part ($id.key.u$). When a packet is matched at an FI node,

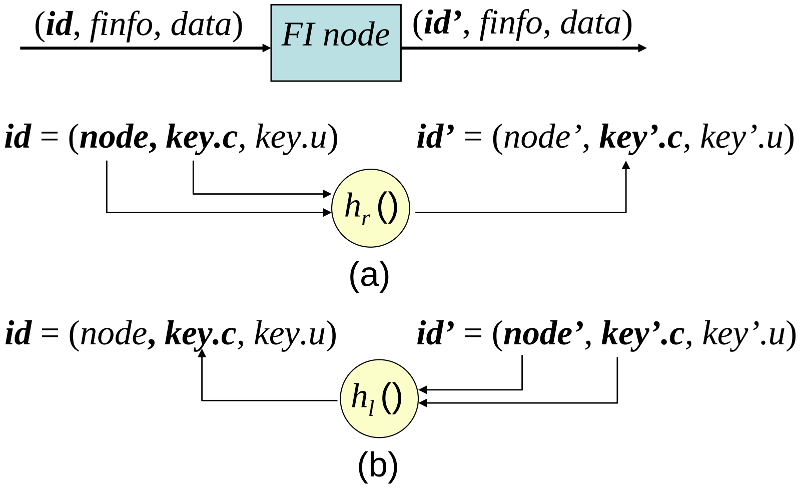

(id, finfo, data)
FI node
(id', finfo, data)
id = (node, key.c, key.u)
id' = (node', key'.c, key'.u)
h_r ()
(a)
id = (node, key.c, key.u)
id' = (node', key'.c, key'.u)
h_l ()
(b)

Figure 4.4. An FI node can update the ID of a packet from id to id' iff id and id' are either (a) right constrained (or r-constrained), or (b) left constrained (or l-constrained).

the constrained part *must* match. The *constrained IDs* rule can be stated as follows (see Figure 4.4):

Constrained IDs Rule: A packet ID, id, can be updated to id', if and only if either *id'.key.c=*h_r*(id.node, id.key.c)* or *id.key.c=*h_l*(id'.node, id'.key.c)* hold.[4]

Functions h_l and h_r are cryptographic hash functions mapping N-bit strings to n-bit strings, where N is the size of an ID excluding the unconstrained part of the key, and n is the size of the constrained part of the key. The properties we require of the cryptographic hash functions (h_l and h_r) are: (a) strong collision resistance, and (b) computationally infeasibility of finding short cycles. Secure one-way hash functions, such as SHA-256 [83], provide these two properties [77].[5] If it is clear from the context, we use $id'{=}h_r(id)$ and $id{=}h_l(id')$ as a shorthand for *id'.key.c=*h_r*(id.node, id.key.c)* and *id.key.c=*h_l*(id.node, id.key.c)*, respectively. (Recall that an identifier has the form *(id.node, id.key.c, id.key.u)*, where *id.key.c* is constrained by the cryptographic function, and *id.key.u* is unconstrained and can be freely chosen.)

[4] We allow both l and r constraints because, as we will show later, l-constraints provide better security properties, whereas r-constraints allow greater functionality.

[5] Given the recent attacks against SHA-1 [122] which reduces the complexity to find a collision to 2^{63}, SHA-256 is required for high security.

Intuitively, a cryptographic hash function makes it hard for an adversary to construct malicious topologies such as loops. Since h_l and h_r are publicly known hash functions, any FI node or host can check and enforce the constraints. If packet's ID, id, is updated to id' and $id' = h_r(id)$, we say that packet ID is *right*-constrained (r-constrained); otherwise, we say that it is *left*-constrained (l-constrained). Note that we choose different hash functions h_l and h_r to avoid trivial cycles of length two.

Next, we show that constraining packet IDs allows only topologies that are trees. Note however that since we allow flexibility of choosing *id.node*, one can still construct confluences on end-hosts and FI nodes. We deal with these problems in Sections 4.4.2 and 4.4.3 respectively.

Theorem 1. *With constrained IDs, it is infeasible for a computationally-bounded adversary to create topologies other than trees.*

Proof. Define G_d as the directed graph formed by assigning directions to the edges of G (we simplify the notation by dropping the argument of G and G_d). In particular, for each edge (x, y) in G we associate the direction from x to y if $y = h_r(x)$, and from y to x if $x = h_l(y)$. The proof is by contradiction. Assume G has a cycle. We consider two cases:

Case (i) G_d has at least one vertex with in-degree 2. This implies there are vertices x, y, z such that there are distinct edges $(x{\rightarrow}z), (y{\rightarrow}z) \in G_d$. Thus, $h_i(x) = h_j(y)$, for $h_i, h_j \in \{h_l, h_r\}$, such that $x \neq y$ or $h_i \neq h_j$ (otherwise edges $(x{,}z), (y{,}z)$ will not be distinct). In both the cases, finding x, y that satisfy these constraints is infeasible as it reduces to finding hash collisions.

Case (ii) All vertices of G_d are of in-degree at most one. We know that underlying graph G has a cycle, say C_u. Consider the sub-graph of G_d induced on the vertices of C_u, call it C_d. We know that $\forall v \in C_d$, $in\_degree(v) \leq 1$. But C_d is a cycle. Hence $\forall v \in C_d$,

$in\_degree(v) = 1$. Thus, we have $x = \{h_l, h_r\}^*(x)$. This is equivalent to finding a cycle in the hash function and is hence computationally infeasible.

□

The rule that we use to constrain IDs results directly from the dual goal of achieving the desirable security properties and at the same time preserving the FI functionality. To illustrate this point, we enumerate several alternatives to constrain IDs we considered. (In Appendix 4.7, we show that our constrained IDs rule indeed preserves the functionality of the FIs.)

(a) Constraining the entire ID id' using id (or vice-versa) would imply that *id'.node* would depend on *id*. This would limit the flexibility of an end-user or third-party in choosing the nodes along a path.

(b) Constraining the entire *id.key* using some part of *id'* would be restrictive, as some FIs require control on the value of *id.key*. For example, *i3* uses the *id.key*'s suffix to implement anycast [106].

(c) Constraining *id'.key* using only *id.key* would allow an attacker to create confluences on FI nodes by mapping all the leaf IDs to the victim node; e.g., by inserting the entries $[id_1 \rightarrow id_2]$, $[id_1 \rightarrow id_3]$, $[id_2 \rightarrow id_4]$, $[id_3 \rightarrow id_5]$ where all IDs are constrained and $id_4.node = id_5.node$.

Achieving Property 1: Constrained IDs (l-constrained IDs in particular) help achieve property 1 (preventing eavesdropping and impersonation) if we enforce all public IDs to be l-constrained, *i.e.,* if a packet matches a public ID id and is replaced by id', then $id = h_l(id')$. If a host constrains its public ID id using a secret ID id', then, to eavesdrop, an attacker should insert an entry $[id \rightarrow id'']$ pointing to the attacker. Hence, an attacker needs to find an ID id'' such that $h_l(id'') = h_l(id') = id$ which amounts to finding hash collisions.

A simple technique to ensure l-constraints on public IDs would be to separate the key space for public and private IDs. For instance, the first bit of *id.key* could denote whether the entry is l-constrained or r-constrained. Since the key space of l- and r-constrained entries are separate, an attacker cannot insert an r-constrained entry for eavesdropping.

4.4.2 Technique 2: Challenge-Response

To ensure that an attacker cannot insert entries pointing to other benign end-hosts, we use the well-known challenge-response technique. FI nodes challenge the insertion of every entry that points to an end-host using a simple three-way handshake.

Consider end-host A inserting an entry $[id{\rightarrow}B]$ at an FI node I_a. The FI node I_a sends a nonce n to host B, since B is the host contained in the entry $[id{\rightarrow}B]$. Host A, which attempted the insertion, can respond to I_a with the nonce n only if it receives the the traffic sent to B—this condition is trivially true if a node is inserting an entry pointing to itself (*i.e.,* $A = B$ in this case). However, an attacker that is not in the physical path to B cannot respond to the challenge, and hence the insertion does not succeed. To avoid maintaining state in FI nodes for every insertion, the challenge is computed using a message authentication function h_k on the values id and B, where k is a secret key only known to the FI node. To prevent replay of challenges, the FI node can periodically update the key k.

Achieving Property 2: The challenge-response protocol outlined above helps achieve property 2 (preventing amplification attacks on end-hosts) since an attacker cannot insert an entry pointing to an arbitrary end-host it does not control. Hence, to replicate its traffic and direct it towards a particular host E, the attacker must itself create a malicious ID-level topology, and link all leaves with an existing entry already inserted by E. But since we already achieve property 1, such an ID-level topology is not possible.

4.4.3 Technique 3: Defense against Oversubscription

As mentioned in Section 4.3.3, legitimate control plane topologies can be used by an attacker to launch a flooding attack on a victim FI node. For example, an attacker, by controlling both the sender and the receiver, can construct a tree such that all entries at the last level (*i.e.,* entries of the form $[idv{*}{\rightarrow}R]$, where R is the receiver) reside at the victim FI node, V. Each packet sent by the sender will be replicated, and all replicas will be sent to the victim FI node. In general, an attacker can amplify its attack N-fold by inserting $O(N)$ forwarding entries. Unfortunately, it is very hard to prevent such an attack since the resulting topology is legitimate, *i.e.,* it is a tree in which each leaf points to an end-host. What enables this attack is the ability of the attacker to insert forwarding entries at an arbitrary FI node. Since this control is critical to the flexibility of many FIs, we choose to use a reactive technique to alleviate this attack (rather than place restrictions on where the entries are stored).

The main observation we make is that such an attack can be alleviated if the attacker cannot make the FI generate more traffic than the attacker can send or receive. In other words, an attacker should not be able to generate more traffic than the receiver R can handle, in which case the attack on node V would be bounded by R's link capacity. The attacker then would not be able to benefit from replicating its traffic, and hence it cannot do better than attacking the victim directly. A simple way to achieve this property is to ensure that the packet loss along *each edge (link)* in the topology is bounded. Consider a forwarding entry $(id, *)$ located at FI node A that forwards packets to FI node or end-host B. If A detects that B receives less than a fraction f of the packets sent by A, then A raises a pushback signal. Now, there are two questions that we need to answer: how is the loss rate measured, and how does the sender react when the loss rate exceeds f.

To detect high losses, we borrow the mechanism based on erasure codes proposed in [50]. FI node A associates a nonce a with every n consecutive packets forwarded via entry $(id, *)$ to next hop B. In particular, node A uses a (k, n) erasure code to encode nonce a, and then piggybacks the erasures into the n consecutive packets forwarded to B. As long as B receives at least k packets, it can reconstruct the nonce a and send it back to A. If node A doesn't receive the nonce, then it implies that B received less than a fraction $\alpha{=}k/n$ of the packets, *i.e.,* the loss is at least[6] $1{-}\alpha$. The additional traffic generated by this mechanism

[6] A dead-end is a special case in which the pushback can be initiated when no forwarding entry matches the packet.

is very low, since only one small-sized packet is sent every n packets. For example, choosing α=3/4 would help tolerate a loss rate as high as 25% (a loss rate at which TCP would not be able to sustain any reasonable throughput), while worsening a possible attack only by a factor of 1.33. The other important parameter in our design is the block size n. A large value of n makes the test more robust, but increases the vulnerability period during which the attacker can exploit the mechanism. To account for the possible loss of nonce reply messages sent by node B to previous hop A, we require that a loss rate greater than f be observed over c consecutive epochs of encoding before initiating the pushback. In practice we choose $c = 3$.

When an FI node detects that the receivers cannot receive the data packets (since it does not receive a correct nonce), it takes action in the form of a *pushback* to ensure that the topology of forwarding entries is pruned all the way to the source. The pushback can be implemented by simply rate-limiting the traffic, or more aggressively, by removing the forwarding entry. In the latter case, even if there are false positives (*i.e.,* an entry is incorrectly removed), soft-state refreshing of entries would ensure that the topology is restored. In the former case, to ensure that pushback signals propagate up the topology, an FI node should reconstruct the nonce (that it uses to prove to the upstream node that it received at least α packets) based on the packets it successfully sent to its downstream nodes. Instead, if a node reconstructs the nonce based on the packets it receives from its upstream node, then pushback from a bottleneck at its downstream node will not propagate upwards.

Providing Property 3

An attacker can exploit the reactive nature of the above mechanism to carry out attacks for short intervals of time. Indeed, the mechanism allows a window of vulnerability from the time the attacker constructs a graph till the time the FI prunes it down. In this section, we bound the damage even when the attacker exploits this window of vulnerability.

Consider an attacker that constructs a tree violating the constraint that the leaves are either dead-ends or end-hosts that receive less than an α fraction of the traffic sent by the FI nodes.[7] Let the maximum height of the tree be h_{max}. This can be enforced using a TTL field. Let $[id{\rightarrow}id']$ be a forwarding entry of this tree stored at node A where id' is a leaf. After receiving the first packet with ID id, A will take t_r time units to remove this entry. If id is a dead-end, t_r is equal to the RTT (τ), since that is how long the pushback mechanism takes to detect a dead-end and propagate a message back one hop. If the leaf is an oversubscribing end-host, t_r is the time it takes the FI node to send $n_c = n{\times}c$ packets of maximum size plus the time it needs to wait for the end-host to send back the nonce: $t_r = n_c l_{max}/r + \tau$, where l_{max} is the maximum packet size.

The only way the attacker can maintain this tree is to replace the leaf edges as soon as they are removed by the pushback mechanism. This attacker strategy would prevent pushback from pruning the rest of the tree. Let λ be the rate at which the attacker can insert new forwarding entries. The maximum number of leaves that an adversary can maintain is then $l = t_r\lambda$. The next result gives a bound on the forwarding cost for this attack scenario.

Theorem 2. *For an attacker that can send packets at an aggregate rate of r, and can insert forwarding entries at a rate λ, the average packet forwarding cost is upper-bounded by:*

$$\frac{h_{max}\ n_c\ l_{max}\ \lambda}{r + \lambda\ o} + h_{max}\tau\lambda, \tag{4.4}$$

where o is the overhead incurred by a host (in bits) when inserting a forwarding entry.

Proof. The rate of sustained attack is proportional to the number of edges in the tree. Since a tree with l leaves has at most lh_{max} edges, an adversary can exploit the system to amplify its attack rate from r to $rh_{max}t_r\lambda$. Now, the total amount of traffic the attacker sends in

[7] Note that the attacker cannot violate the constraints enforced by the "constrained ID" technique.

the FI is $r + \lambda o$, and thus the damage ratio is $(rh_{max}t_r\lambda)/(r + \lambda o)$. Since the maximum value of t_r is achieved when the leaf is an end-host, we take $t_r = n_c l_{max}/r + \tau$. After some simple algebra, Formula 4.4 follows. □

We note that mounting an attack that achieves this bound is not trivial. To utilize the resources optimally, an attacker needs to anticipate when an entry is removed, which is hard due to the fact that the attacker does not know the round-trip time between the FI nodes, and the round-trip times can vary significantly.

Limiting Forwarding Cost

By inspecting Formula 4.4, we see that the forwarding cost can be reduced by increasing the overhead of the insertion operation o and limiting the insertion rate λ. We can increase the insertion overhead by either increasing the size of the response packets so that $o \simeq l_{max}$, or use multiple challenge-response rounds before inserting an entry. In the latter case, at each round the FI node sends a new challenge containing a nonce based on the nonce sent in the previous round (*e.g.,* by hashing the previous nonce). A host will be able to insert an entry only if it answers all challenges sent by the FI node.

Since many systems maintain forwarding entries by soft-state, there is no difference between inserting and refreshing the entries. The refreshing rate can be policed by the first-hop FI node. The rate can be specific to end-hosts, negotiated when hosts sign up for the FI service. However, designing efficient mechanisms for restricting λ for a malicious FI node is a hard problem; initial insights are presented in [61].

Consider an attacker that sends traffic at 5 Mbps. If maximum tree depth of 10, $l_{max} = o = 1400$ bytes, $n_c = 48{\times}3 = 144$, $\tau = 100$ms, and $\lambda = 1$ entries/s[8], we get a forwarding cost of about 2. From the first term in the expression, we also observe that with a higher attack rate, the forwarding cost would only go down.

[8] A web server may negotiate a much higher rate of inserting entries if needed.

4.4.4 Summary of Defense Techniques

The modifications can be classified based on where they are implemented: *data* and *control* plane changes. We first list the data plane modifications.

- Packet IDs should be either l- or r-constrained; *i.e.,* when the packet ID is updated from id to id', then either *id.key.c*=h_l(*id'.node, id'.key.c*) or h_r(*id.node, id.key.c*)=*id'.key.c*. The sub-field *id.key.c* should be long enough (*e.g.,* 128 bit, as discussed in Section 4.4.1) so that it is infeasible for an attacker to guess it. Public IDs should be l-constrained.

- Private IDs should be long enough that it is hard for an external attacker to guess; hence, mounting an attack or eavesdropping on a private entry is hard. The *id.key.c* field can be re-used for this purpose; it is computed by a cryptographic hash function (which guarantees a pseudo-random string) when *id* is the target of a constraint and randomly chosen otherwise.

- To limit the damage ratio, packet headers need to include a TTL field that is decremented at every hop. In practice, a TTL of 8 bits should suffice.

- Packet headers need to include an erasure (about 1-byte) to prevent oversubscription attacks.

- Replicated packets can be delivered to the destination only through a forwarding entry inserted by that destination. This restriction prevents confluences on end-hosts.

Thus, in addition to the fields *id* and *finfo*, a packet header needs to include two other one-byte fields (one for TTL and one for the erasure). Now, we list the control plane changes.

- The *finfo* field of an entry should include a nonce that is hard to guess. This prevents an attacker from updating and removing such entries. As discussed in Section 4.2.1, this nonce should be at least 80-bits long.

- The insertion of a public entry requires a challenge-response mechanism as described in Section 4.4.2. This mechanism prevents malicious linking, but adds one RTT to the entry insertion operation.

- The FI needs to implement the pushback mechanism described in Section 4.4.3 which involves appending an erasure to each packet that is forwarded.

Until now we have implicitly assumed that ID constraints are checked at run-time, *i.e.,* when the ID of a packet is updated. However, in many cases, how the IDs would be updated is known when the forwarding entry is inserted. For example, in *i3* and Network Pointers, a forwarding entry of the form $[id_1 \rightarrow id_2]$ will update the ID of a packet from id_1 to id_2. In such cases, we can check for constraints when the entry is inserted, rather than when the packet is forwarded. As a result, the overhead due to checking constraints on the data path can be eliminated completely.

4.5 Internal Attacks

Thus far, we have considered external attackers alone. In this section, we consider *internal* attackers. We assume that such an attacker can compromise FI nodes and have complete control over their local state, and over packets received or sent by these FI nodes.

While internal attackers have complete control on the traffic forwarded by the FI nodes they compromise, we show that they have very little power on the traffic forwarded by other FI nodes. In particular, the only attack an internal attacker can mount that an external attacker cannot is a *random* attack, *i.e.,* attacking a host through its private forwarding entry *without* knowing the identity of the host. Unlike routing protocols in today's Internet (such as BGP [112]), an internal attacker cannot mount an "off-path" attack, *i.e.,* it cannot affect other FI nodes or end-hosts whose packets are not normally forwarded through the compromised node.

We assume that an attacker cannot eavesdrop the payload of the packets it forwards, including the control packets that manipulate the forwarding entries. In other words, the attacker can read only the information in the control packets regarding forwarding entries that are stored locally. This assumption can be enforced by encrypting the payload of both control and data packets. To authenticate the FI nodes, we can use self-certified node IDs like HIP [85], where a node ID (*i.e.,* more precisely *node.id*) is computed using an one-way hash function on the node's public key. This is equivalent with using public keys to identify FI nodes, instead of IDs.

Even if the attacker is not able to eavesdrop the payload of the packets it forwards, the attacker can still learn the IDs of forwarding entries stored at other FI nodes by inspecting the *finfo* field in packets that are matched locally, or the *finfo* field in the FI entry stored at the compromised node. Hence the advantage of an internal attacker is that it can learn about private IDs of other end-hosts while an external attacker cannot. However, the attacker has no direct way to associate that ID with an end-host, since it cannot learn who inserted the forwarding entries at other nodes. Hence, mounting attacks on a private ID is equivalent to mounting an attack on a random end-host.

Finally, a compromised FI node could also violate the l- and r-trigger constraints. However, the key observation is that all replicated traffic will continue to flow through the compromised FI node, *i.e.,* it cannot create a loop not going through itself or an amplification topology not including itself.

The crucial insight behind the argument that no other forms of attacks are possible is that FIs merely forward packets as the forwarding entries dictate; they do *not* run any routing protocol. All the operations that are performed at an FI node use local state and simple packet update rules. Hence, the operations that a compromised FI node can perform—insert or remove forwarding entries and send packets—is fundamentally no different from the ones performed by end-hosts; the only difference is that FI nodes typically have more resources than the end-hosts.

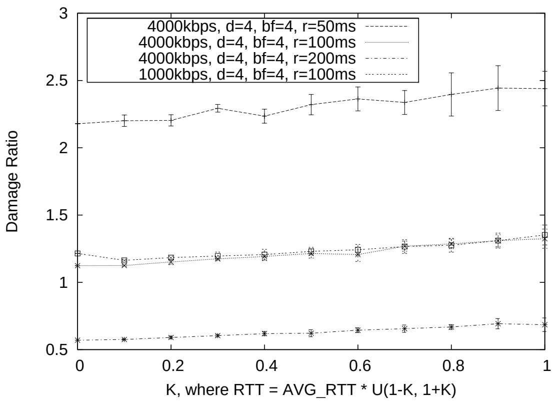


Figure 4.5. Effectiveness of pushback as a function of variability of RTTs of links.

4.6 Implementation and Evaluation

We have implemented the three main mechanisms—constraints of forwarding entries, response to oversubscription and challenges to forwarding entry insertion—over *i3* [106], one of the FIs proposed earlier. We used inverted hash tables to implement pushback (needed for implementing response to oversubscription) and used a one-way hash function for generating the constraints as well as the challenges.

For efficiency, our one-way hash function is based on the Advanced Encryption Standard (AES) [32], using the Matyas, Meyer, and Oseas construction [75]. The key is encrypted by the AES cipher and then the output is XORed with the input. We get two different one-way functions, h_l and h_r, by keying the cipher with two different publicly known keys (different from the keys we hash).

We evaluate the three mechanisms in terms of their overhead and effectiveness. Since cryptographic constraints and challenges completely prevent the attacks that they are designed for, we only evaluate the additional overhead that they introduce. To illustrate the effectiveness of the pushback, we use simulations.

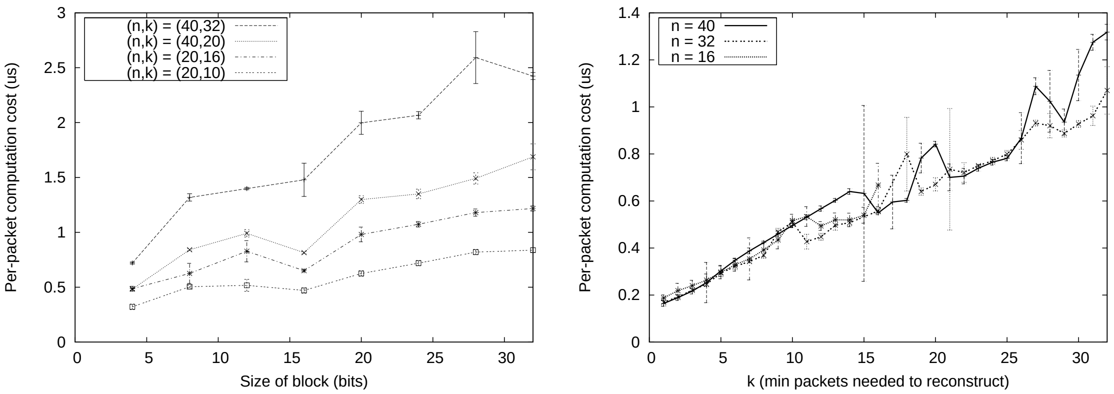


Figure 4.6. Overhead of verification mechanisms for preventing oversubscription

4.6.1 Cryptographic Constraints and Challenges: Computational Overhead

The two security mechanisms, cryptographic constraints as well as challenge mechanism, require operations on the control path. We show by experiments that the cost of both these operations is minimal.

As mentioned in Section 4.4, checking both cryptographic constraints and challenges involve computation of a one-way hash function. If the challenge checking or constraint checking fails, the forwarding entry is not inserted (or in the case of runtime checking, the packet is not forwarded).

To measure the additional overhead, we ran tests on an *i3* node on a 866 MHz Pentium III running Linux 2.4.8. The results are averaged over half a million operations. The running time for a hash-computation is less than 3 μs, which implies that constraints can be checked even on the data path while supporting forwarding rates of a few hundred thousand packets per second. Overall, the computational overhead for checking the challenge and the constraints is only about 28%.

4.6.2 Defense against Oversubscription

To evaluate the sensitivity of the pushback mechanism, we first performed a set of simple experiments using a 5-node chain topology over PlanetLab with an RTT of about 200 ms. In the first experiment, we run a TCP flow across the chain topology; in the subsequent experiments, we run UDP flows of increasing rates. In each experiment, we transferred 3 MB and recorded if the pushback was triggered. We repeat each experiment 25 times.

The TCP transfers experienced an average throughput of 1.6 Mbps and never triggered pushback. Table 4.1 shows the fraction of UDP transfers that did not trigger a pushback. As expected, as the rate of the UDP flow increases, the probability that pushback is triggered also increases. When the rate reaches 4 Mbps, pushback is always triggered immediately. We infer that the probability of false positives of the pushback mechanism when the sending rate is close to the TCP sending rate is negligible.

Rate (Mbps)	2	2.5	3	3.5	4
Fraction of Successful Transfers	1	0.8	0.6	0.5	0

Table 4.1. Fraction of successful UDP transfers for different sending rates.

The analysis of the technique to defend against oversubscription in Section 4.4.3 presents an upper bound assuming that all hops have the same RTT. We now present the effect of variation in RTTs on the mechanism using a simple event-driven simulator. We use simulations rather than experiments as it allows the attacker to precisely control the timing of (re)inserting triggers. This precise timing is very hard to achieve in practice due to the RTT variations, thus we expect the simulation results we present here to be an upper bound for the experimental results.

At the beginning of the simulation, we construct a complete tree given a particular depth, d and branching factor b. We let the adversary refresh the forwarding entries at a particular rate r. The adversary also is assumed to have global knowledge so that it can refresh the forwarding entry that would cause *maximum* damage (*i.e.,* the deepest entry).

In the main experiment that we report, we set the branching factor to be 4. Figure 4.5 shows how the damage ratio varies depending on how the RTT of the links is chosen in the simulation—randomization of zero corresponds to all links having same RTT (of

MAX_RTT/2) and randomization of 1 corresponds to RTTs being chosen uniformly between [*0, MAX_RTT*]. The refresh periods are chosen as $50, 100$ and 200 ms. The main inference from the graph is that the variation in RTTs does not affect pushback by much and almost closely mirrors our analysis. Secondly, even when receivers are allowed to refresh every 50 ms, the damage ratio is only about 2. Finally, varying attacker sending rate had little effect on the damage ratio.

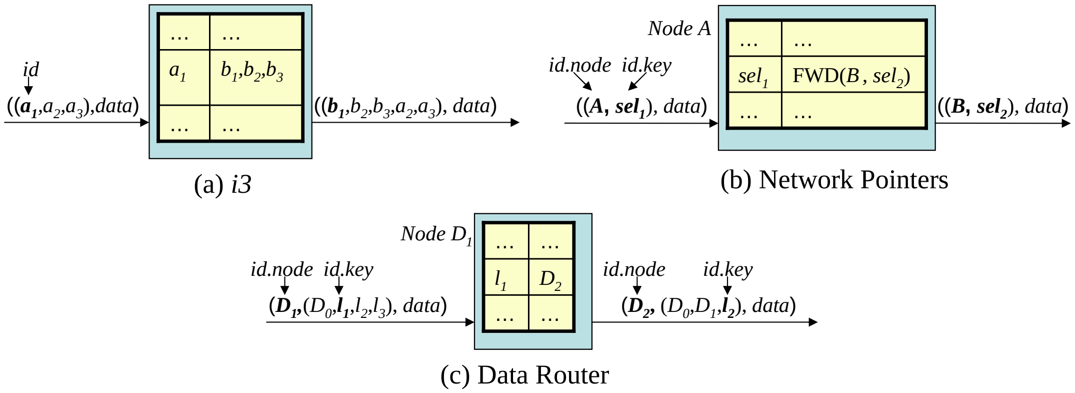
4.6.3 Cost of Erasure Computation

We present the cost of computing the erasures (introduced in Section 4.4.3) for preventing oversubscription attacks. We used the FEC software developed by Rizzo *et al.* [95] for benchmarking the erasure computation. Figure 4.6(a) shows the cost of per-packet erasure computation by varying the size of the block used for different (n, k) combinations. The increase in cost with the increase in block size is marginal—even for 32-bit blocks, and $(n, k) = (40, 32)$, the cost is under $2.5\mu s$. Figure 4.6(b) shows the variation in overhead as n/k is varied for three values of n for 8-bit blocks. The increase is almost linear—it is not precisely linear because the implementation is based on Vandermonde matrices and at certain values, low-level issues such as cache hits/misses would cause deviations from the expected trends.

4.7 Realization over Specific Proposals

The generic FI model helped us abstract away the details of FIs, and concentrate on fundamental problems. We presented a range of techniques that can be used in specific FI designs. However, we do not advocate a "one-size-fits-all" approach; particular FIs present tradeoffs that need to be considered before making decisions on which techniques are relevant. Here, we present a few examples to illustrate this point.

The FI model, for generality, assumed that FI nodes perform packet replication. Consistent with this assumption, we constrained forwarding entries such that malicious topologies



(a) *i3*

(b) Network Pointers

(c) Data Router

Figure 4.7. The forwarding operation for three forwarding infrastructure proposals: (a) *i3*, (b) Network Pointers, and (c) DataRouter.

that allow misuse of packet replication—such as confluences—are impossible to construct. However, certain legitimate applications construct *control plane topologies* that are identical to confluences but the *data plane topologies* formed by the IDs that the packets take are benign due to additional operations performed during packet forwarding. Examples of such applications include multipath routing and load balancing; packets can be forwarded either along path $(id_s, id_1, id_2, \ldots, id_e)$, or path $(id_s, id'_1, id'_2, \ldots, id_e)$—the union of the two paths is indeed a confluence. If the FI does not allow packet replication, then we can allow load balancing by constraining the IDs based on the keys alone (*i.e., id.key.c*=h_l(*id'.key.c*) or h_r(*id.key.c*)=*id'.key.c*). In practice, an FI that performs both packet replication as well as load balancing can have separate ID spaces and allow packet replication on one ID space and load balancing on the other.

4.7.1 Internet Indirection Infrastructure

We divide the 256-bit identifier in *i3* into three fields: a 64-bit prefix (roughly corresponds to *id.node*), a 128-bit constrained key (corresponds to *id.key.c*), and a 64-bit suffix (corresponds to *id.key.u*). IDs id and id' are matched based on the longest prefix matching rule, given the constraint that both their keys and prefixes match exactly. If an l-constrained trigger (x, y) points to an end-host, we use only $y.key.c$ to constrain $x.key.c$. Ignoring $y.node$ and $y.key.u$ when computing $h_l(y)$ allows us to preserve support for anycast and mobility.

Since the packet's ID is always replaced with the first ID in the matching trigger's stack, constraints can be checked when the trigger is inserted instead of at run-time, thus avoiding any overhead on the data path. Next, we argue that constrained IDs have limited impact on the functionality provided by *i3*.

Mobility. Since constraints are not computed over the IP addresses of hosts (which is stored in *id.node*), there is no impact on mobility.

Multicast. Applications can still build legitimate multicast trees as in *i3* by using r-constrained triggers. The triggers that are used to build multicast trees are private triggers and hence having r-constrained triggers would not expose the multicast group to eavesdropping.

Anycast. Anycast functionality is not affected by trigger constraints. However, in an anycast group with l-constrained triggers, each end-host must have the same *id.key.c*; this key needs to be distributed out-of-band.

Service composition. Disallowing insertion of arbitrary triggers still allows sender-driven service composition, but weakens the flexibility of receiver-driven service composition. In particular, it will not be possible for a receiver to redirect packets with a *given* ID x to an intermediate node with a *given* ID y since this would require the receiver to insert a trigger of the form (x, y), where x and y are fixed. However, this situation can be dealt with at the application level by negotiating a private trigger out-of-band. We expect this restriction to be acceptable to a majority of applications.

4.7.2 Network Pointers

Network Pointers is a link layer mechanism that gives end-hosts fine-grained control over forwarding in the network by inserting *pointers* [117] (see example in Figure 4.7(b)). The incoming packet contains the address of the next hop A as well as a selector sel_1 which is used to index the forwarding table at node A. The packet is forwarded to the next hop after its selector is updated to (B, sel_2). The *p.id.node* field is the next-hop address, and *p.id.key* is the selector. The **match** operation is exact matching. The **p_update** operation is specified in the *e.finfo* field. The packet can be either forwarded to a next hop by updating its *p.id* or delivered to a local application.

Since a node uses exact matching to match the selectors, one can use the entire selector as the key to incorporate ID constraints. However, the length of the selector should be increased as it is only 64 bits long in their design. For supporting the forwarding operation described in [117], the constraints can be checked at the time of inserting the entries. For more complex forwarding operations, one might need to check the constraints at run-time. All proposed uses of Network Pointers involve only chain topologies [48,117], which will not be affected by constrained IDs.

4.7.3 DataRouter

DataRouter is a forwarding engine that provides generic string matching and rewriting capabilities at the IP layer based on application-specific needs [116]. DataRouter is a high performance generic alternative to application-layer overlays. In addition to the IP header, a packet carries a generalized source route. The source route can contain arbitrary strings, which is used to index the forwarding table. Figure 4.7(c) shows an example in which the packet with destination address D_1 arrives at the next hop. The source route carried by the packet consists of the IP path traversed by the packet so far (D_0), and a list of string labels, $[l_1, l_2, l_3]$, used to index the forwarding tables of the hops along the path. In this example, when the packet arrives at node D_1, the node swaps the first string label l_1 with its address, and forwards the packet to the next hop, as indicated by the forwarding entry, D_2.

In general, *p.id.node* is the destination address of the packet, and *p.id.key* consists of a *class* that identifies a forwarding table at the next hop (not shown in the example), and a *string* used to search in that forwarding table. The **match** operation can be either exact matching, longest prefix matching, or proximity matching; this is specified in the *p.finfo* field. The **p_update** operation, in general, updates the destination IP address and the forward information *p.finfo* in the packet. The only constraint is that the node cannot update the prefix of the source route (*i.e., p.info*) that shows the path followed by the packet so far.

To enforce ID constraints, the strings used to index into the forwarding tables should have a sub-string of bits which are matched exactly and which represent the *id.key.c* field. Even when alternate matching algorithms are selected by the application (such as range matching), exact matching must be performed on the constrained part before the specific matching algorithm can be invoked on the remainder of the tag. While this does not undermine the specific matching algorithms, it might require additional bits for the field *id.key.c*. Since a packet's ID can be updated based on the packet's *finfo* field, checking the ID constraints needs to be done at run-time. We are not aware of any application in the context of the DataRouter [116] that requires cyclic topologies or confluences, thus constrained IDs will not limit their functionality.

4.8 Related Work

Traditionally, new network architectures have suffered from many security issues. With active networks, achieving security is difficult and has often come at the expense of restricting the flexibility (such as ESP [23]) or use of per-use policy and authentication (such as SANE [8]). In fact, loose source routing is disabled by many ISPs because of security issues [15].

Mechanisms for addressing seemingly simple problems such as loop prevention [124] have involved operations on the data path. Furthermore, loop prevention techniques in literature have been reactive, and do not guarantee loop-free topologies.

In the process of designing security mechanisms for FIs, we have leveraged techniques that have been proposed earlier in the literature. Challenge-response protocols have been used for a long time in diverse areas. The idea of using erasure codes to ensure that uncooperative hosts do not oversubscribe to high-bandwidth streams was proposed recently in the context of multicast [50]. Pushback has been proposed for rate-limiting the traffic of IP aggregates [71].

Proposals that deal with DoS attacks based on packet floods [68,10,56] are orthogonal to ours; we devise mechanisms to prevent end-hosts from *using* the infrastructure to aggravate attacks.

We do not consider the issue of securing the underlying routing layer, since the work in that space is largely orthogonal. We note that there are several ongoing research efforts to address these issues both in the context of IP routing [59,65,101,112] and DHT-routing [25, 103].

4.9 Summary

Giving hosts control over forwarding in the infrastructure has become one of the promising approaches in designing flexible network architectures. In this chapter, we addressed the security concerns of these forwarding infrastructures.

We presented a general FI model, analyzed potential security vulnerabilities and presented several mechanisms to alleviate attacks. Our key defense mechanism, based on light-weight cryptographic constraints, provably prevents a large set of attacks. In contrast to previous efforts that detect and mitigate malicious activity, the cryptographic mechanism prevents attacks altogether. Our mechanisms are applicable to many earlier proposals such as *i3* [106] and DataRouter [116] while requiring only modest changes. In providing secure forwarding, we make the deployment of these promising architectures much more viable.

Chapter 5

Conclusions and Future Work

We conclude the dissertation by summarizing the contributions and proposing future directions.

5.1 Summary of Contributions

In this thesis, we have presented an architecture for decoupling the resilience goals and customizability goals for a network infrastructure. Addressing all the issues that arise in such an architecture is a gargantuan task. We have proposed solutions for three important issues that arise from this architectural split.

First, we designed a protocol called Failure-Carrying Packets (FCP) that provides resilient underlying routing. FCP does so by completely eliminating the convergence period that routing protocols typically experience. Once a failure is detected locally, a packet is guaranteed to be routed to its destination as long as a path to the destination exists in the network. While the basic FCP protocol assumes that all nodes have a consistent link-state of the network, we also designed a variant called Source-Routing FCP (SR-FCP) that provides similar properties even if the network maps are inconsistent, at the expense of additional overhead in packet headers.

Though we primarily present FCP to introduce a new routing paradigm that is qualitatively different from previous approaches, we show through simulation that it provides quantitative benefits as well. Using real-world ISP topologies and failure data, we show that the overhead of using FCP is very small. We also compare FCP with OSPF and show that, unlike OSPF, FCP can simultaneously achieve both low loss and low overhead. We also show that compared to prior work in backup path precomputations, FCP provides much lower loss-rates while maintaining less state at the routers.

Second, we presented an architecture for providing customizable routing as a service (RAS). In this architecture, we argue for allowing third-party providers to form business relationships with ISPs and end-users (that require customized routing); RSPs then install customized routes on the behalf of end-users. Since the third-party providers contract with the ISPs, the ISPs still have control over the traffic that is forwarded.

Finally, we provided a secure forwarding mechanism for allowing third-party providers and hosts to select routes across the underlying network infrastructure. Our forwarding mechanism, which is based on simple cryptographic constraints on the entries inserted by the third-party providers, ensures that such entries cannot be used to launch eavesdropping or amplify packet-flooding attacks on the hosts of the network. To the best of our knowledge, this is the first system that exploits the difficulty in finding short loops in cryptographic hash functions for designing a secure routing system.

5.2 Future Directions

We now identify some of the future research directions, identifying the stumbling blocks in each case.

5.2.1 Extending FCP to Interdomain Routing

While FCP was primarily discussed as a link-state routing protocol applicable to intradomain routing networks, we presented a strawman solution for using FCP to improve

BGP. One of the main issues in extending FCP for interdomain policy routing is resolving policy conflicts during BGP updates. It can be impossible to tell the difference between a selfish AS routing a packet along its neighbor to reduce its own traffic load and a protocol-conformant AS doing the same since there was a failure. This tussle is captured in the meta-policy decision we outline in the description of how a node routes a packet.

We believe that there is scope for improvement here, since the AS getting a packet that contains a source-route not conforming to its policies can perform certain checks. For instance, it can record failures it hears from other nodes, and check if the failure header contained in the packet matches its local cache of failures it has heard from other nodes. Alternatively, it can optimistically believe the neighboring AS and forward the packet for a small duration of time, but if the information the neighbor uses does not propagate to this node in a certain period of time, it can penalize the neighbor.

5.2.2 Making FCP Robust to Internal Attackers

If a node within the network infrastructure is malicious (i.e. an internal attacker), it can cause a denial-of-service attack on FCP by generating a large number of packets with a failure header causing all nodes in the network to spend time performing recomputations. First of all, we note that by treating packets with failure headers to have lower priority than normal packets, we can ensure this attack doesn't affect the packets that do not encounter failures.

To deal with this attack, a node N can perform certain checks to see if a packet could have encountered the failure that is contained in the header before reaching N. Also, N can check if the source route in the packet is legitimate given the failures that the packet header contains. What makes the problem particularly hard is that it is possible for an attacker to construct a fake failure header and source route that is not distinguishable from a legitimate one by just using local checks at a downstream node. Naively performing such checks could also cause resource exhaustion since these checks could be almost as expensive as the route computation operation itself. One possible solution is performing randomized checks coupled with high penalties for nodes that are not conformant to the protocol.

5.2.3 Deployment Issues with RAS

There are several issues of incentives and trust that need to be solved before the RAS model can be deployed. For instance, we need to answer why an ISP would wish to contract with a third-party vendor to provide customized routes rather than provide some sort of specialized routing within its network thereby trying to grab all the end-users to use its network alone. From the past, we know that ISPs have allowed third party providers (such as Akamai) to have boxes installed within their networks (or at least at the edges), but for ISPs to allow external providers to control routes (albeit for a small fraction of traffic) is still a huge step forward. Finally, users need to trust the routes that the RSPs provide them. While they can calibrate the routes against the performance they get out the customized routes, a more comprehensive solution that automates the process of determining how good a customized route is would reduce the barrier for end-users to buy into such a service.

5.2.4 Stability of Independent End-to-End Route Selection

In the RAS architecture, since the routing decisions of end-hosts are controlled by RSPs, the RSPs can ensure the stability of routes that are used. However, when end-hosts receive service from multiple non-cooperating RSPs, or if the size of traffic handled by RSPs becomes very large, stability might be a concern. In such cases, RSPs can use randomization techniques to return a path which is almost as good, but not the same as the best path. Furthermore, techniques such as hysteresis can be employed while performing load-sensitive path switching at the end-hosts.

5.3 Concluding Remarks

In this dissertation, we have presented a new architecture for a resilient and customizable network infrastructure. Deployment of such an architecture seems to be a far cry at this stage. We however believe that the architectural split we presented along with the specific protocols and techniques we have investigated would lead to more exciting research work, both from a theoretical and a practical standpoint, in the future.

Bibliography

[1] Cisco IOS Optimized Edge Routing. http://www.cisco.com/warp/public/732/Tech/routing/oer/.

[2] NANOG Mailing List Archives. http://www.merit.edu/mail.archives/nanog/.

[3] OSPF charter. http://www.ietf.org/html.charters/ospf-charter.html.

[4] Abilene observatory data collections. http://abilene.internet2.edu/observatory/data-collections.html.

[5] Anonymized OC48 traces, CAIDA. http://data.caida.org/.

[6] Adler, M., Halperin, E., Karp, R. M., and Vazirani, V. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proc. of STOC* (2003).

[7] Alaettinoglu, C., Jacobson, V., and Yu, H. Towards Millisecond IGP Convergence. IETF Draft, 2000.

[8] Alexander, D. S., Arbaugh, W. A., Keromytis, A. D., and Smith, J. M. A Secure Active Network Environment Architecture. *IEEE Network* (1998).

[9] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. Resilient Overlay Networks. In *Proc. SOSP* (2001).

[10] Anderson, T., Roscoe, T., and Wetherall, D. Preventing Internet Denial-of-Service with Capabilities. In *Proc. of Hotnets* (2003).

[11] Argyraki, K., and Cheriton, D. R. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Proc. USENIX Annual Tech. Conference* (2005).

[12] Atlas, A. U-turn Alternates for IP/LDP Fast-Reroute. Internet Draft draft-atlas-ip-local-protect-uturn-03.txt, February 2006.

[13] Bartlett, J. Optimizing Multi-homed Connections. *Business Communications Review 32*, 1 (January 2002), 22–27.

[14] Behrens, J., and Garcia-Luna-Aceves, J. J. Distributed, scalable routing based on link-state vectors. In *Proc. ACM SIGCOMM* (1994).

[15] Bellovin, S. Security Concerns for IPng. RFC 1675, 1994.

[16] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. An Architecture for Differentiated Service. RFC 2475, 1998.

[17] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. ACM SIGCOMM* (1994), pp. 24–35.

[18] Bryant, S., Filsls, C., Previdi, S., and Shand, M. IP Fast Reroute using Tunnels. Internet draft draft-bryant-ipfrr-tunnels-01.txt, Oct 2004.

[19] Bryant, S., and Shand, M. A Framework for Loop-free Convergence. Internet Draft draft-bryant-shand-lf-conv-frmwk-03, October 2006.

[20] Bryant, S., Shand, M., and Previdi, S. IP Fast Reroute Using Not-via Addresses. Internet Draft draft-bryant-shand-ipfrr-notvia-addresses-03, 2006.

[21] Byers, J., Considine, J., and Mitzenmacher, M. Simple Load Balancing for Distributed Hash Tables. In *Proc. IPTPS* (2003).

[22] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. Design and Implementation of a Routing Control Platform. In *Proc. of NSDI* (2005).

[23] Calvert, K. L., Griffioen, J., and Wen, S. Lightweight Network Support for Scalable End-to-End Services. In *Proc. ACM SIGCOMM* (Pittsburgh, PA, 2002).

[24] Castineyra, I., Chiappa, N., and Steenstrup, M. The Nimrod Routing Architecture. RFC 1992, 1996.

[25] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. Secure Routing for Structured Peer-to-peer Overlay Networks. In *OSDI* (Boston, MA, Dec. 2002).

[26] Chen, W., and Liu, X. Enforcing Routing Consistency in Structured Peer-to-Peer Overlays: Should We and Could We? In *Proc. IPTPS (To appear)* (2006).

[27] Cheriton, D. R., and Gritter, M. TRIAD: A New Next Generation Internet Architecture, Mar. 2000. `http://www-dsg.stanford.edu/triad/triad.ps.gz`.

[28] Choudhury, G. Prioritized Treatment of Specific OSPF Version 2 Packets and Congestion Avoidance. RFC 4222, October 2005.

[29] Choudhury, G., Atlas, A., Torvi, R., Martin, C., Imhoff, B., and Fedyk, D. Basic Specification for IP Fast-Reroute: Loop-free Alternates. IETF draft, 2005.

[30] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. Tussle in Cyberspace: Defining Tomorrow's Internet. *IEEE/ACM Trans. on Networking* (June 2005).

[31] Crowcroft, J., Hand, S., Mortier, R., Roscoe, T., and Warfield, A. Plutarch: An Argument for Network Pluralism. In *Proc. of FDNA* (2003).

[32] Daemen, J., and Rijmen, V. AES proposal: Rijndael, Mar. 1999.

[33] Davie, B., and Rekhter, Y. MPLS: Technology and Applications. In *Morgan Kaufmann* (2000).

[34] Dean, D., and Stubblefield, A. Using Client Puzzles to Protect TLS. In *Proc. of the 10th USENIX Security Symposium* (Washington, D.C., Aug. 2001), USENIX.

[35] Dolev, D., Dwork, C., Waarts, O., and Yung, M. Perfectly Secure Message Transmission. *Journal of the ACM* (1993).

[36] Duan, Z., Zhang, Z.-L., and Hou, Y. T. Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning. *IEEE/ACM ToN* (2003).

[37] Dwork, C., and Naor, M. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology — CRYPTO '92* (1993), E. Brickell, Ed., vol. 740 of *Lecture Notes in Computer Science*, International Association for Cryptologic Research, Springer-Verlag, pp. 139–147.

[38] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and van der Merwe, J. The Case for Separating Routing from Routers. In *Proc. Future Directions in Network Architecture* (Aug. 2004).

[39] Franciosa, P., Frigioni, D., and Giaccio, R. Semi-dynamic shortest paths and breath-first search in digraphs. In *STACS* (1997).

[40] Francois, P., and Bonaventure, O. An evaluation of IP-based Fast Reroute Techniques. In *Proc. CoNEXT* (2005).

[41] Francois, P., and Bonaventure, O. Avoiding transient loops during IGP convergence in IP networks. In *Proc. INFOCOM* (2005).

[42] Freedman, M. J., Freudenthal, E., and Mazieres, D. Democratizing Content Publication with Coral. In *Proc. NSDI* (2004).

[43] Freedman, M. J., Lakshminarayanan, K., Rhea, S., and Stoica, I. Non-Transitive Connectivity and DHTs. In *Proc. WORLDS* (2005).

[44] Garcia-Luna-Aceves, J. J. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Transactions on Networking* (1993).

[45] Gerding, S., and Stribling, J. Examining the tradeoffs of structured overlays in a dynamic non-transitive network, 2003. Class project: `http://pdos.csail.mit.edu/~strib/docs/projects/networking_fall2003.pdf`.

[46] Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I. Load Balancing in Dynamic Structured P2P Systems. In *Proc. INFOCOM* (2004).

[47] Goel, A., Zhang, H., and Govindan, R. Incrementally Improving Lookup Latency in Distributed Hash Table Systems. In *Proc. of ACM Sigmetrics* (2003).

[48] Gold, R., Gunningberg, P., and Tschudin, C. A Virtualized Link Layer with Support for Indirection. In *Proc. of FDNA* (2004).

[49] Goodell, G., Aiello, W., Griffin, T., Ioannidis, J., McDaniel, P., and Rubin, A. Working around BGP: An Incremental Approach to Improving Security and Accuracy in Interdomain Routing. In *Proc. of NDSS* (Feb. 2003).

[50] Gorinsky, S., Jain, S., Vin, H., and Zhang, Y. Robustness to Inflated Subscription in Multicast Congestion Control. In *Proc SIGCOMM* (2003).

[51] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM CCR 35*, 5 (Oct. 2005).

[52] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I. The Impact of DHT Routing Geometry on Resilience and Proximity. In *SIGCOMM* (2003).

[53] Gummadi, K. P., Madhyastha, H., Gribble, S. D., Levy, H. M., and Wetherall, D. J. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proc. of OSDI* (2004).

[54] Gupta, A., Liskov, B., and Rodrigues, R. Efficient Routing for Peer-to-Peer Overlays. In *Proc. of NSDI* (2004).

[55] Haeberlen, A., Hoye, J., Mislove, A., and Druschel, P. Consistent Key Mapping in Structured Overlays. Tech. Rep. 05-456, Rice University, 2005.

[56] Handley, M., and Greenhalgh, A. Steps Towards a DoS-resistant Internet Architecture. In *Proc. of FDNA* (2004).

[57] Hildrum, K., Kubiatowicz, J. D., Rao, S., and Zhao, B. Y. Distributed Object Location in a Dynamic Network. In *Proceedings of SPAA* (Winnipeg, Canada, August 2002), ACM.

[58] Hu, Y., Perrig, A., and Sirbu, M. SPV: Secure Path Vector Routing for Securing BGP. In *Proc. ACM SIGCOMM* (2004).

[59] Hu, Y.-C., Perrig, A., and Sirbu, M. SPV: Secure Path Vector Routing for Securing BGP. In *Proc. of ACM SIGCOMM* (2004).

[60] Iannaccone, G., Chuah, C., Bhattacharyya, S., and Diot, C. Feasibility of IP Restoration in a Tier-1 Backbone. *IEEE Networks* (March 2004).

[61] Jain, A., Hellerstein, J., Ratnasamy, S., and Wetherall, D. A Wakeup Call for Internet Monitoring Systems: The Case for Distributed Triggers. In *Proc. of Hotnets* (2004).

[62] Juniper-Networks. Configure alternate backup paths using fate-sharing. `http://www.juniper.net/techpubs/software/junos/junos53/swconfig53-mpls-apps/html/mpls-signaled-config37.html`.

[63] Karger, D., and Ruhl, M. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *Proc. SPAA* (2004).

[64] Kent, S., and Atkinson, R. Security Architecture for the Internet Protocol. Internet RFC 2401, IETF, Nov. 1998.

[65] Kent, S., Lynn, C., and Seo, K. Secure Border Gateway Protocol (S-BGP). *IEEE JSAC 18*, 4 (Apr. 2000), 582–592.

[66] Kushman, N., Kandula, S., Katabi, D., and Maggs, B. R-BGP: Staying Connected in a Connected World. In *Proc. NSDI* (2007).

[67] Labovitz, C., Ahuja, A., Bose, A., and Jahanian, F. Delayed Internet Routing Convergence. In *Proc. ACM SIGCOMM* (2000).

[68] Lakshminarayanan, K., Adkins, D., Perrig, A., and Stoica, I. Taming IP Packet Flooding Attacks. In *Proc. ACM HotNets-II* (Cambridge, MA, Nov. 2003).

[69] Lenstra, A. K., and Verheul, E. R. Selecting Cryptographic Key Sizes. *Journal of Cryptology 14*, 4 (2001), 255–293.

[70] Li, J., Stribling, J., Morris, R., Kaashoek, M. F., and Gil, T. M. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proc. INFOCOM* (2005).

[71] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. Controlling High Bandwidth Aggregates in the Network. *CCR 32*, 3 (July 2002), 62–73.

[72] Mahajan, R., Wetherall, D., and Anderson, T. Understanding BGP Misconfigurations. In *Proc. of ACM SIGCOMM* (2002).

[73] Mahajan, R., Wetherall, D., and Anderson, T. Negotiation-based routing between neighboring ISPs. In *Proc. Networked Systems Design and Implementation* (2005).

[74] Mao, Z., Govindan, R., Varghese, G., and Katz, R. Route Flap Damping Exacerbates Internet Routing Convergence. In *Proc. SIGCOMM* (2002).

[75] Matyas, S., Meyer, C., and Oseas, J. Generating Strong One-way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin 27* (1985), 5658–5659.

[76] Maymounkov, P., and Mazieres, D. Kademlia: A Peer-to-peer Information System based on the XOR Metric. In *Proceedings of (IPTPS)* (2002).

[77] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography.* CRC Press series on discrete mathematics and its applications. CRC Press, 1997. ISBN 0-8493-8523-7.

[78] Merkle, R. Secure Communication Over Insecure Channels. *Commun. ACM 21*, 4 (Apr. 1978), 294–299.

[79] Moy, J. T. OSPF Database Overflow. RFC 1765, March 1995.

[80] Moy, J. T. OSPF Version 2. RFC 2328, April 1998.

[81] Moy, J. T. OSPF complete implementation. In *Addison-Wesley, New York* (2001).

[82] Narvaez, P., Siu, K.-Y., and Tzeng, H.-Y. New Dynamic Algorithms for Shortest Path Tree Computation. *IEEE/ACM Transactions on Networking* (2000).

[83] National Institute of Standards and Technology (NIST), Computer Systems Laboratory. Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-2, Aug. 2002.

[84] Nelakuditi, S., Lee, S., Yu, Y., Wang, J., Zhong, Z., Lu, G.-H., and Zhang, Z.-L. Blacklist-Aided Forwarding in Static Multihop Wireless Networks. In *Proc. of SECON* (2005).

[85] O'Shea, G., and Roe, M. Child-proof authentication for MIPv6 (CAM). *Computer Communication Review Apr.*, 31 (2001), 2.

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

[87] Park, K., Pai, V. S., Peterson, L., and Wang, Z. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proc. OSDI* (2004).

[88] Perlman, R. *Network Layer Protocols with Byzantine Robustness.* PhD thesis, MIT, 1988.

[89] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A Scalable Content-Addressable Network. In *Proc. ACM SIGCOMM* (San Diego, 2001).

[90] Rekhter, Y., and Li, T. A Border Gateway Protocol 4 (BGP-4). RFC 1771, 1995.

[91] Rhea, S. *OpenDHT: A Public DHT Service.* PhD thesis, University of California, Berkeley, Aug. 2005.

[92] Rhea, S., Chun, B.-G., Kubiatowicz, J., and Shenker, S. Fixing the embarrassing slowness of OpenDHT on PlanetLab. In *Proc. WORLDS* (Dec. 2005).

[93] Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J. Handling Churn in a DHT. In *Proc. of the USENIX Annual Technical Conference* (2004).

[94] Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and Yu, H. OpenDHT: A Public DHT Service and Its Uses. In *Proc. of ACM SIGCOMM* (2005).

[95] Rizzo, L. http://info.iet.unipi.it/ luigi/fec.html.

[96] Roscoe, T., Hand, S., Isaacs, R., Mortier, R., and Jardetzky, P. Predicate Routing: Enabling Controlled Networking. In *Proc. Workshop on Hot Topics in Networking* (2002).

[97] Rosen, E., Viswanathan, A., and Callon, R. Multiprotocol Label Switching Architecture. RFC 3031, Jan. 2001.

[98] Rowstron, A., and Druschel, P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. In *Proceedings of IFIP/ACM Middleware* (Nov. 2001).

[99] Russ Cox and Athicha Muthitacharoen and Robert Morris. Serving DNS using Chord. In *IPTPS* (2002).

[100] Savage, S., Anderson, T., Aggarwal, A., Becker, D., Cardwell, N., Collins, A., Hoffman, E., Snell, J., Vahdat, A., Voelker, G., and Zahorjan, J. Detour: a Case for Informed Internet Routing and Transport. *IEEE Micro 19*, 1 (Jan. 1999).

[101] Secure Origin BGP (soBGP). ftp://ftp-eng.cisco.com/sobgp.

[102] Shaikh, A., Isett, C., Greenberg, A., Roughan, M., and Gottlieb, J. A Case Study of OSPF Behavior in a Large Enterprise Network. In *Proc. IMW* (2002).

[103] Sit, E., and Morris, R. Security Considerations for Peer-to-peer Distributed Hash Tables. In *Proc. of IPTPS, 2002* (Cambridge, MA, Mar. 2002).

[104] Spring, N., Mahajan, R., and Wetherall, D. Measuring ISP Topologies with Rocketfuel. In *Proc. of SIGCOMM* (2002).

[105] Stoica, I. *Stateless Core: A Scalable Approach for Quality of Service in the Internet.* PhD thesis, Carnegie Mellon University, Dec. 2000.

[106] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. Internet Indirection Infrastructure. In *Proc. ACM SIGCOMM* (2002).

[107] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM* (2001).

[108] Stoica, I., Morris, R., Karger, D. R., Kaashoek, M. F., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. In *Proc. ACM SIGCOMM* (San Diego, CA, Aug. 2001), pp. 149–160.

[109] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Tech. Rep. TR-819, MIT, 2001.

[110] Subramanian, L., Agarwal, S., Rexford, J., and Katz, R. H. Characterizing the Internet hierarchy from multiple vantage points. In *Proc. IEEE INFOCOM* (June 2002).

[111] Subramanian, L., Katz, R. H., Roth, V., Shenker, S., and Stoica, I. Reliable Broadcast in Unknown Fixed Identity Networks. In *Proc. PODC* (2005).

[112] Subramanian, L., Roth, V., Stoica, I., Shenker, S., and Katz, R. H. Listen and Whisper: Security Mechanisms for BGP. In *Proc. of NSDI* (2003).

[113] TEIXEIRA, R., DUFFIELD, N., REXFORD, J., AND ROUGHAN, M. Traffic Matrix Reloaded: Impact of Routing Changes. In *Proc. of PAM* (2005).

[114] TEIXEIRA, R., GRIFFIN, T., SHAIKH, A., AND VOELKER, G. Network Sensitivity to Hot-Potato Disruptions. In *Proc. of SIGCOMM* (2004).

[115] TEIXEIRA, R., SHAIKH, A., GRIFFIN, T., AND REXFORD, J. Dynamics of Hot-Potato Routing in IP Networks. In *Proc. of ACM SIGMETRICS* (2004).

[116] TOUCH, J., AND PINGALI, V. DataRouter: A Network-Layer Service for Application-Layer Forwarding. In *Proc. IWAN* (2003).

[117] TSCHUDIN, C., AND GOLD, R. Network Pointers. In *Proc. ACM HotNets-I* (2002).

[118] TURNER, J., ANDERSON, T., PETERSON, L., AND SHENKER, S. Virtualizing the Net: A strategy for network de-ossification. `http://www.arl.wustl.edu/~jst/talks/hotI-9-04.pdf`.

[119] WALFISH, M., BALAKRISHNAN, H., AND SHENKER, S. Untangling the Web from DNS. In *Proc. of NSDI* (Mar. 2004).

[120] WALFISH, M., STRIBLING, J., KROHN, M., BALAKRISHNAN, H., MORRIS, R., AND SHENKER, S. Middleboxes No Longer Considered Harmful. In *Proc. of OSDI* (2004).

[121] WANG, F., AND GAO, L. Inferring and Characterizing Internet Routing Policies. In *Proc. IMC* (2003).

[122] WANG, X., YIN, Y., AND YU, H. Finding collisions in the full sha-1. In *Proceedings of Crypto* (Aug. 2005).

[123] WETHERALL, D. Active Network Vision and Reality: Lessons from a Capsule-based System. In *Proc. of SOSP* (1999).

[124] WHITAKER, A., AND WETHERALL, D. Forwarding Without Loops in Icarus. In *Proc. of IEEE OPENARCH 2002* (June 2002).

[125] White, R. Deployment Considerations for Secure Origin BGP (soBGP), draft-white-sobgp-bgp-deployment-01.txt. IETF Draft, IETF, June 2003.

[126] Xie, G., Zhang, J., Maltz, D., Zhang, H., Greenberg, A., Hjalmtysson, G., and Rexford, J. On static reachability analysis of IP networks. In *Proc. IEEE INFOCOM* (2005).

[127] Yang, X. NIRA: A New Internet Routing Architecture. In *Proc FDNA-03* (2003).

[128] Zhu, D., Gritter, M., and Cheriton, D. Feedback-based Routing. In *Proc Hotnets-I* (2002).