NIRA: A New Internet Routing Architecture

by

Xiaowei Yang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2004

Signature of author ______________________________

Department of Electrical Engineering and Computer Science
September 1, 2004

Certified by ______________________________

David Clark
Senior Research Scientist of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ______________________________

Arthur C. Smith
Chairman, Committee on Graduate Students
Department of Electrical Engineering and Computer Science

This is a modified version of the submitted thesis. A few typos were corrected and Appendix 5.C is added.

NIRA: A New Internet Routing Architecture

by

Xiaowei Yang

Submitted to the Department of Electrical Engineering and Computer Science on September 1, 2004 in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

The present Internet routing system faces two challenging problems. First, unlike in the telephone system, Internet users cannot choose their wide-area Internet service providers (ISPs) separately from their local access providers. With the introduction of new technologies such as broadband residential service and fiber-to-the-home, the local ISP market is often a monopoly or a duopoly. The lack of user choice is likely to reduce competition among wide-area ISPs, limiting the incentives for wide-area ISPs to improve quality of service, reduce price, and offer new services. Second, the present routing system fails to scale effectively in the presence of real-world requirements such as multi-homing for robust and redundant Internet access. A multi-homed site increases the amount of routing state maintained globally by the Internet routing system. As the demand for multi-homing continues to rise, the amount of routing state continues to grow.

This dissertation presents the design of a new Internet routing architecture (NIRA) that simultaneously addresses these two problems. NIRA gives a user the ability to choose the sequence of Internet service providers his packets traverse. It also has better scaling characteristics than today's routing system. The design of NIRA is decomposed into four modular components: route discovery, route availability discovery, route representation and packet forwarding, and provider compensation. This dissertation describes mechanisms to realize each of these components. It also makes clear those places in the design where a globally agreed mechanism is needed, and those places where alternative mechanisms can be designed and deployed locally. In particular, this dissertation describes a scalable route discovery mechanism. With this mechanism, a user only needs to know a small region of the Internet in order to select a route to reach a destination. In addition, a novel route representation and packet forwarding scheme is designed such that a source and a destination address can uniquely represent a sequence of providers a packet traverses.

Network measurement, simulation, and analytic modeling are used in combination to evaluate the design of NIRA. The evaluation suggests that NIRA is scalable.

Thesis Supervisor: David Clark
Title: Senior Research Scientist of Electrical Engineering and Computer Science

Acknowledgments

I am most grateful to my advisor David Clark, who guided me and supported me through this work and my entire graduate study. His vision and wisdom helped me see the importance of this work, and his optimism and enthusiasm encouraged me to tackle numerous difficult issues I would have avoided in this work. Graduate school took me a long time, but Dave was always patient with me. He let me develop my potential at my own pace, and shepherded me in his special way when I wandered off the road. Dave has taught me so much. He taught me to focus on big pictures, to hold high standards to my work, to systematically come up with solutions, to face challenges with confidence, and much more. His teaching made me grow from a student into a young peer of his, and I have no doubt that I will continuously benefit from his teaching throughout my career.

Members of my thesis committee: Hari Balakrishnan and Robert Morris, offered insightful feedbacks on this work. Their perspectives enlarged my view, and helped improve this work.

Arthur Berger helped me develop the analytic model used in this work to evaluate the route setup latency. He also spent much time to meet with me regularly to discuss my research.

Douglas De Couto pointed me to the related work by Spinelli and Gallager, which greatly simplified the network protocol designed for route discovery in this work.

I owe many thanks to the past and present members of the Advanced Network Architecture (ANA) group, and my friends on the fifth floor of the old LCS building. Many of them read early drafts of this work or listened to multiple versions of talks on this work. They provided useful comments and helped mature this work. They include: Mike Afergan, Steve Bauer, Rob Beverly, Peyman Faratin, Nick Feamster, Dina Katabi, Joanna Kulik, George Lee, Ben Leong, Ji Li, Jinyang Li, Karen Sollins, and John Wroclawski. Steve Bauer was my officemate while I was writing this dissertation. He became my sounding board. So many times when I could not come up with the right argument in my writing, a chat with Steve clarified my thinking. Mike Afergan and Rob Beverly, both are very knowledgeable about operational networks. I learned much about networking in practice from them. Becky Shepardson, the administrative assistant of ANA, took care of things ranging from scheduling my thesis defense to FedExing me thesis chapters with readers' comments while I was away. She helped me to get work done.

Many people helped along my journey of graduate school in many ways. John Wroclawski's skepticism greatly helped me catch flaws in my work. I and other graduate students at ANA always know that if we could convince John at group meetings, we could face the most ferocious audience without fear. John's help spread from research, to writing, and to job search.

In my early days at graduate school, Tim Shepard taught me how to use and hack Unix. He also spent much time to help reduce my accent and made me speak English. Even after he left ANA, he always remembered to visit me at my office or to email me. He provided not only technical insights, but inspiration and support.

Tim introduced me to Garrett Wollman. Garrett possesses a tremendous amount of practical knowledge about networking and FreeBSD. He taught me many things I would not learn from books or papers. He showed me how the network of the old LCS building was wired, and how packet filters were deployed at a real router. I also learned many things about FreeBSD from him.

Lixia Zhang, who graduated from ANA even before I joined, has always kept an eye on me remotely. She generously shared her experience at graduate school with me, and provided valuable advice both for research and for life. Despite her busy professional life, Lixia always reserved time to answer various questions I asked.

Finally, I would not have made this without the support from my family. My parents brought up me and my brother with love and discipline. They overcame many hardships to give us a good education, and always put our needs above their own. I am forever indebted to them. My brother, Junfeng Yang, who himself is pursuing a Phd in computer science, had many interesting discussions with me on my work.

I am fortunate to have Daniel Jiang to spend my life with. Dan supported me in every way possible. He let me enjoy the convenience of living within walking distance to work and he himself took nearly two-hour commute every day. He always showed a strong interest in my work, and was eager to contribute his ideas. He took care of all other things in life when I was busy with work. He encouraged me when I felt frustrated, and took pride and pleasure in what I achieved. He deserves my sincerest gratitude.

Contents

List of Figures

Chapter 1

Introduction

The Internet today consists of more than 17,000 [2] worldwide routing domains, each operated under an autonomous organization.[1] Domains make business decisions to interconnect, and technically, the Border Gateway Protocol (BGP) [88], is the uniform inter-domain routing protocol that connects thousands of domains into a coherent internetwork.

This dissertation is concerned with the design of the inter-domain routing system. The present design has largely neglected two important problems: one structural[2] and one architectural. The structural problem is that the present system fails to create and sustain user-driven competition among upper-level Internet service providers (ISPs). The architectural problem is that the present inter-domain routing, accomplished by one global routing protocol, fails to scale effectively in the presence of real-world requirements such as multi-homing.

We present the design of a new Internet routing architecture that scales adequately and gives a user the ability to choose domain-level routes so as to drive ISP competition. Our design focuses on recognizing the proper functional boundaries to modularize the routing system. Modularity is the key principle we apply again and again in our design. We use it not only as a tool to manage complexity, but as a technique to reduce the need to architecturally constrain every part of the routing system. Within a functional module, our design provides basic mechanisms to achieve the desired functionality, yet allows different mechanisms to be introduced to enhance the functionality. We believe that a system designed under such a principle has a great potential to evolve and to survive unpredicted future demands.

In the rest of this Chapter, we elaborate the problems in today's routing system, discuss briefly why existing or proposed solutions do not work well, and outline our solution to these problems.

[1]A routing domain is also referred to as an *autonomous system (AS)*, or simply a *domain*. A domain that provides forwarding service for packets sent or destined to other domains is sometimes called a *provider*, or a *service provider*, or an *Internet service provider (ISP)*.

[2]By "structural," we refer to *industry structure*: the study of how industries and businesses might be organized to achieve specific ends and objectives.

1.1 User Choice

From a structural perspective, user choice is crucial for the creation of a healthy and competitive ISP market [29, 109]. Today, users can pick their own ISPs, but once their packets have entered the network, the users have no control over the overall routes their packets take. With BGP, each domain makes local decisions to determine what the next hop (at the domain level) will be, but the user cannot exercise any control at this level.

This work argues that it would be a better alternative to give the user more control over routing at this level. User choice fosters competition, which imposes an economic discipline on the market, and fosters innovation and the introduction of new services. An analogy can be seen in the telephone system, which allows the user to pick his long distance provider separately from his (usually monopolist) local provider. Allowing a user to select his long-distance provider has created the market for competitive long distance, and driven prices to a small fraction of their pre-competition starting point. The original reasoning about Internet routing was that this level of control was not necessary, since there would be a large number of ISPs, and if a consumer did not like the wide-area choice of a given local access ISP, the consumer could switch. Whether this actually imposed enough pressure on the market in the past might be debated. But for the consumer, especially the residential broadband consumer, there is likely to be a very small number of competitive local ISPs offering service. With cable competing only with DSL (Digital Subscriber Line), the market is a duopoly at best (at the facility level) and often a monopoly in practice. The deployment of the new technology Fiber-To-The-Home (FTTH) [78] is likely to further reduce the number of competing local ISPs. So in the future, the competitive pressures on the wide-area providers will go down.

While it is only speculation, one can ask whether the lack of end to end quality of service in the Internet is a signal of insufficient competitive pressure to drive the deployment of new services. Certainly, there are many consumers who would be interested in having access to enhanced QoS, if it was available end to end at a reasonable price. Such a service might have driven the deployment of VoIP, of various sorts of teleconferencing and remote collaboration tools, and so on. If the consumer could pick the routes his packets took, this might entice some provider to enter the market with a QoS offering, and a set of ISPs might in time team up to make this widely available. But there is no motivation to offer such a service today, since the consumer has no way to get to it. So one can speculate that the lack of competition in the form of user-selected routes is one cause of stagnation in Internet services today.

We cannot prove this hypothesis as a business proposition. Only an experiment in the real world, a real market deployment, can reveal what users actually want, what creative providers will introduce, and what might happen to pricing. But this work takes as a starting point that this experiment is a worthy one, and explores the technical problems that would have to be solved to give users this sort of choice.

1.2 Scaling

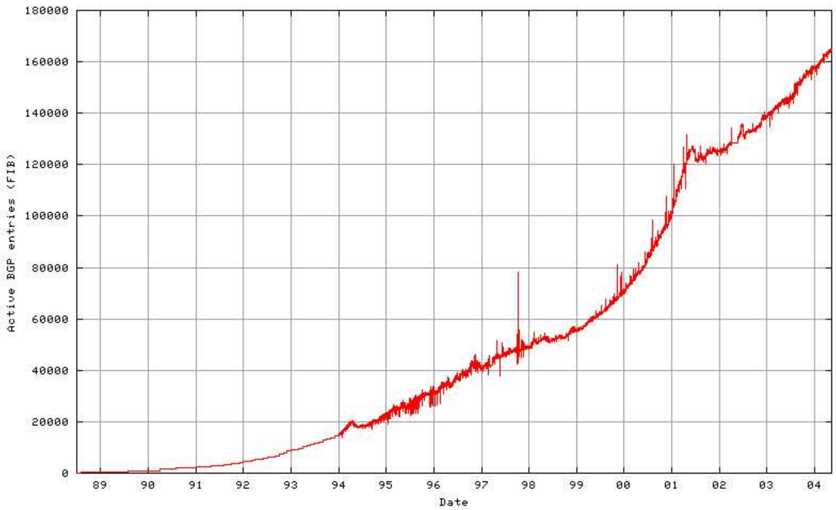


Figure 1-1: The continuing growth of active BGP entries collected from AS1221 (Telstra, an Australian ISP)). Courtesy of http://bgp.potaroo.net/

At an architectural level, the current routing system faces a scaling problem. Figure 1-1 shows the continuing growth of the active BGP table entries. Multi-homing [24] is one of the most significant contributors to the fast growth of BGP tables, as users increasingly demand robust and redundant Internet access paths. A multi-homed site is usually allocated an address prefix from one of its providers. When this address prefix is announced to the site's other providers, it cannot be aggregated into these providers' address space. So these providers would have to announce the address prefix of the multi-homed site in their BGP sessions to their neighbors, thus propagating one extra entry into BGP routing tables.[3]

Today, multi-homing has introduced 20−30% extra prefixes in BGP tables [24]. One might argue that the advance of hardware technology will surpass the growth of the routing table size, thus scaling won't become a problem. However, this optimistic prediction has perhaps overlooked two points. First, the Internet community has already exercised self-discipline to control the growth of BGP tables, making demands for multi-homing difficult to satisfy [76]. It is common practice among ISPs today to have a restriction on the longest address prefix their BGP routers would accept. For example, Sprint [94], Verio [108], and Jippii [12] all have a filter policy that only accepts address prefixes not allocated by themselves with a maximum length /24.[4] If a small or-

[3]The provider that allocates the address prefix to the multi-homed site would also need to announce that address prefix in its BGP sessions, because IP forwarding is based on longest prefix match. If the provider does not announce that prefix separately and only announces an aggregated address prefix, all traffic destined to the multi-homed site would go through the other providers of the site.

[4]An IP address prefix is often written in the format *address/prefix length*. So the notation */number* is used to denote the length of an address prefix, or the size of an address block. A */number* address prefix has $2^{32-number}$ addresses.

ganization, e.g., a five-person stock trading company, gets an address prefix longer than /24 from an upstream provider, it would be difficult for the organization to have another upstream provider to announce its address prefix to the Internet in order to obtain a redundant Internet access path. At the mean time, it is difficult for that small company to obtain a provider-independent address prefix from an Internet Registry, as most regional Internet registries [11] have a policy limiting the minimum size of an address block they would allocate to an organization and require that the organization justify an efficient utilization rate for an address block of that size. For example, the smallest address block ARIN (American Registry for Internet Numbers) [48] would allocate to a multi-homed site is /22, and ARIN requires that a site show a 25% immediate address utilization rate of an address block of that size and a 50% utilization rate within one year. It would be difficult for a small company to justify such a utilization rate.

So if we count the pent up needs for multi-homing, and consider the possibility that millions of business organizations worldwide might all desire redundant and robust Internet access paths, the routing table sizes could have increased much faster than what we have observed.

Second, we are primarily concerned with the control overhead incurred by maintaining the routing tables, although the fast growth of the routing tables also makes it a nuisance for providers to continuously upgrade their routers [58, 37]. The inter-domain routing protocol, BGP, is essentially a flat routing protocol. The status change of one entry in a BGP table could lead to global routing update, causing BGP announcements to ripple through the entire Internet. The larger the tables get, the more the CPU time and the bandwidth a router would have to spend on processing and propagating the announcements in order to maintain an up-to-date routing table. Even for an address prefix to which a router seldom has traffic to forward, the router has to spend its CPU time and bandwidth to keep the entry updated.

If we consider the speculation that if multi-homing were satisfactorily supported, then the routing tables would grow much faster, and the control overhead for updating the routing tables would become much more significant, our conclusion is that a flat inter-domain routing system is likely to have a high maintenance cost and suffers from performance problems.

1.3 Existing Proposals or Solutions

In the past, these two problems, user choice and scaling, have been dealt with in isolation. There exist partial solutions that address each of these problems.

1.3.1 User Choice

IP [84, 35] has a loose source routing option for users to specify routes. However, to fully support domain-level user selected routes, a number of problems must be addressed, including how users discover routes scalably and make intelligent route choices, how user-selected routes are efficiently represented, how packets are forwarded, how route failures are handled, and how providers are compensated if users choose their services. Loose source routing option alone is insufficient to support user-selected routes.

Intelligent route control products, such as products offered by Internap [8], RouteScience [5] and OPNIX [4], are able to assist a multi-homed site to select its first hop provider based on user-configured parameters, e.g., cost, dynamic traffic load. However, those products only offer choices for outbound traffic, and cannot choose beyond the first hop, and are not generally affordable by small sites or individual users.

Overlay networks [16, 97, 102, 90] enable route selection to some extent. An overlay network exploits a small group of end hosts to provide packet forwarding and/or duplication service for each other. However, an overlay network does not have a global scope, and thus does not enable route choice for every user. Only nodes in an overlay network can choose routes formed by other nodes in the same overlay network. Moreover, packet forwarding using an overlay network is usually less efficient than packet forwarding using the underlying IP network. Different overlay links may share the same physical links. A packet forwarded along an overlay path may traverse one physical link multiple times, thus wasting network resource.

1.3.2 Scaling

In the IPv4 routing architecture, limiting the size of BGP tables largely depends on address allocation policies and ISP filter policies. Internet number registries [11] have restrictions on the smallest address block they would allocate. ISPs accept prefix announcements with maximum prefix length requirements [9]. Both these policies are meant to limit the number of address prefixes that would be propagated into BGP routing tables. Evidently, these policies obstruct the deployment of multi-homing.

The IPv6 architecture, still in its deployment phase, proposes that a multi-homed site should obtain an address prefix from each of its providers, and only announce the address prefix allocated from a provider to that provider. This scheme prevents a multi-homed site from “poking holes” into the global routing tables, but introduces the *address selection* problem. An end host with multiple addresses needs to pick a “good” source address to communicate with a destination. This problem is referred to as the “Source Address Selection” problem, and has received much attention [6]. If an end host chooses a “bad” source address, e.g., a source address allocated from a provider to which the end host cannot connect due to a temporary failure, return packets cannot come back to the host and connections cannot be established. Moreover, if ingress filtering [46, 18] were deployed, the source address of a packet sent to a provider must match the address prefix allocated by the provider.

So far no satisfactory solution exists to the source address selection problem. Existing proposals [57, 34, 38, 22] basically suggest two approaches. One approach is to require that ISPs relax source address filtering for a multi-homed site, either turning off source address filtering for a multi-homed site, or having the site communicate with the ISPs its authorized address prefixes. The other approach is to rely on the exit routers of a multi-homed site to provide information to help a user to pick a "good" source address. For the first approach, relaxing source address filtering does not help a user to pick a usable source address when there are temporary failures on its address allocation paths. For the second approach, when address allocation has several levels, a failure on a source address allocation path could occur at a place to which the exit routers of a site are not directly connected. The existing proposals leave it unanswered how exit routers would obtain this information and help a user to pick an address at the first place.

1.4 Our Approach

This dissertation presents a New Internet Routing Architecture (NIRA) that is scalable and supports user route selection. We set out to seek an architectural re-design, rather than the specific mechanisms that fix the problems within today's system. We believe that the difference between a fresh re-design and the current system will offer directions for improvement.

1.4.1 Design Overview

We aim to identify as many modularizable components of the inter-domain routing system as possible. Modularity allows us to decompose the complex design task into a set of smaller sub-problems. More importantly, it gives us the flexibility not to restrict the design of every component in the system. Our design presents the primitive mechanisms to realize each functional module. But within each module, future designers could invent new mechanisms to enhance, complement, or replace our baseline design.

Our work shows that the design of NIRA can be decomposed into three sub-problems: route discovery and selection, route representation and packet forwarding, and provider compensation. The reasoning behind this decomposition is as follows.

Our goal is to build support for user choice into the inter-domain routing system. Before a user[5] could make an intelligent route choice, he should know what options are available to him. The routing system, owned by multiple ISPs, has the knowledge of what routes are available for users to choose. So the first problem we need to address is how to provide mechanisms to expose the route options to a user.

Once a user has learned his route choices and picked a route to send his packets, a user shall specify his choice in his packets so that routers could forward the packets along his chosen route. We recognize that how a user specifies a route is independent of how he discovers and

[5]The word "user" refers to an abstract entity. It is not necessarily a human user. It could be a piece of software running in a human user's computer, making decisions upon a human user's configured preferences.

chooses routes. Multiple mechanisms are possible for a user to discover and select routes without changing the mechanisms for route specification. The reverse is also true. So in our design, we deal with the problem of route discovery and selection separately from that of route representation and packet forwarding.

In a commercial Internet, if ISPs are chosen to forward packets for users, they should be rewarded correspondingly. We refer to this problem as the provider compensation problem. Clearly, users and ISPs could come up with different compensation schemes without altering the mechanisms for users to discover routes or specify routes. So we regard provider compensation as a distinct design component.

Furthermore, we identify that the task of route discovery can be further modularized. Both the physical structure of the Internet and the transit policies of each domain together determine the set of possible domain-level routes between two end users. Routes, in this sense, change slowly. In contrast, the dynamic conditions of a route may change frequently. We use "route availability" to refer to whether the dynamic conditions of a route satisfy a user's requirements, such as whether a route is free from failures, or whether a route has sufficient available bandwidth. Intuitively, if mechanisms for route discovery are also required to supply route availability information, there will be less variety in design choices. Therefore, we separate the design problem of route availability discovery from that of route discovery.

Moreover, the task of route discovery can be divided into a sender half and a receiver half. If an individual user is required to have the knowledge of all possible routes to reach every destination on the Internet, regardless whether he is going to communicate with a destination or not, it is likely that the amount of information a user needs to keep grows as the Internet grows. As a general principle, a user should be limited to choose within the part of the network he agrees to pay for. So in our design, an individual user only needs to know the part of network that provides transit service for him. Usually, this part of the network consists of a user's upstream providers, his providers' providers, and so on. A user could infer the set of route segments he is allowed to use to reach the rest of the Internet from this part of the network. When a user wants to communicate with a destination, he could use any general mechanisms to discover the route segments the destination is allowed to use. A user can then combine these two pieces of information to pick a route to reach the destination.[6]

It is worth noting that the dissertation is primarily written in explaining how individual users choose routes. We acknowledge that an edge domain such as a company might not want to give its employees the ability to choose routes. Our design is completely consistent with this requirement. Choices do not have to be made literally by the end users. For the sake of clarity, we discuss more about where choice could be made in Chapter 6.6, and focus the rest of the dissertation on individual user choice.

[6]Our design focuses on end-to-end communication and does not consider inter-domain multicast or broadcast.

1.4.2 Mechanisms

In our design, we provide basic mechanisms to achieve the functionality of each design module in NIRA. For the part of route discovery, we design a network protocol, Topology Information Propagation Protocol (TIPP), to assist a user to discover his route segments. We also describe a DNS-like (Domain Name System) [74] infrastructure service, Name-to-Route Lookup Service (NRLS) that facilitates a sender to retrieve a destination's route information on-demand. Intermixing his route information and that of the destination, a sender is able to pick an initial route to communicate with the destination.

NIRA uses a combination of reactive and proactive notifications for users to discover route availability. A user is proactively notified of the dynamic status of the route segments on his part of the network, and relies on reactive mechanisms such as router feedback and timeouts to discover the dynamic conditions of a destination's route segments.

We design a route representation mechanism and a corresponding router forwarding algorithm such that a source and a destination address are able to represent a typical type of domain-level route.

Provider compensation in NIRA is still based on contractual agreements. Providers will be properly compensated by billing their customers based on the contractual agreements. Per packet based accounting or micro-payment [73] schemes are unnecessary.

1.4.3 Evaluation

We evaluate the design of NIRA using simulations, analytic models, and network measurements. Simulations help us debug subtle design flaws; analytic models provide high-level descriptions of the system under a variety of operating conditions; network measurements check our intuitions against reality. Our evaluation suggests that NIRA is practical.

1.5 Organization of the Dissertation

The rest of the thesis is organized as follows. We start with related work in Chapter 2. We then present the design rationale of NIRA in Chapter 3. Chapter 4 describes NIRA's route discovery and route availability discovery mechanism. We discuss route representation and packet forwarding algorithm in Chapter 5. Chapter 6 explains how providers might be compensated if users choose to user their services. We evaluate the design of NIRA in Chapter 7. Chapter 8 concludes the dissertation and discusses future work.

Chapter 2

Related Work

At a high level, related work falls into three categories: scalable routing schemes, routing architecture proposals, and current route selection technologies. In this chapter, we first briefly describe the current Internet routing system, and then discuss related work.

2.1 Background: the Present Internet Routing System

Routing in the Internet happens at two levels: intra-domain routing and inter-domain routing. A routing domain (or an AS) is a network that is under a single administration. Each domain uses an intra-domain routing protocol such as Open Shortest Path First (OSPF) [75] or Routing Information Protocol (RIP) [70] to handle the task of finding a route to reach a destination inside the domain. Border routers of each domain run the de facto inter-domain routing protocol, Border Gateway Protocol (BGP) [88], to collectively handle the task of finding a route to reach any network connected to the Internet.

BGP is a path-vector protocol. In BGP, a domain announces to its neighboring domains the range of addresses reachable via itself together with the attributes of the paths to reach those addresses. A range of addresses is encoded as an address prefix. One of the most important path attributes is the ASPath attribute. The ASPath attribute associated with an address prefix announced by a domain includes the sequence of domain identifiers (i.e. Autonomous System Numbers (ASNs)) a packet would traverse to reach the address prefix via that domain. If a domain receives BGP announcements for reaching the same address prefix from multiple neighbors, the domain selects a neighbor as the next hop to reach that address prefix based on its local policies. Again, based on its policies, the domain may announce the address prefix to some of its neighbors, with its own domain identifier (ASN) prepended to the ASPath attribute.

Figure 2.1 illustrates how BGP works. A domain AS_1 announces an address prefix 10.0.0.0/16 with the ASPath attribute set to AS_1 to its neighbors AS_2 and AS_3. AS_2 and AS_3 further announce the address prefix 10.0.0.0/16 with ASPath set to AS_2 AS_1 and AS_3 AS_1 respectively to their neighbor AS_4. AS_4, based on its own policies, selects AS_3 as the next hop neighbor to reach the address prefix 10.0.0.0/16, and announces the address prefix with ASPath set to AS_4 AS_3 AS_1 to AS_5. So packets sent from AS_5 to the address range 10.0.0.0/16 will follow the path $AS_5 \rightarrow AS_4 \rightarrow AS_3 \rightarrow AS_1$.

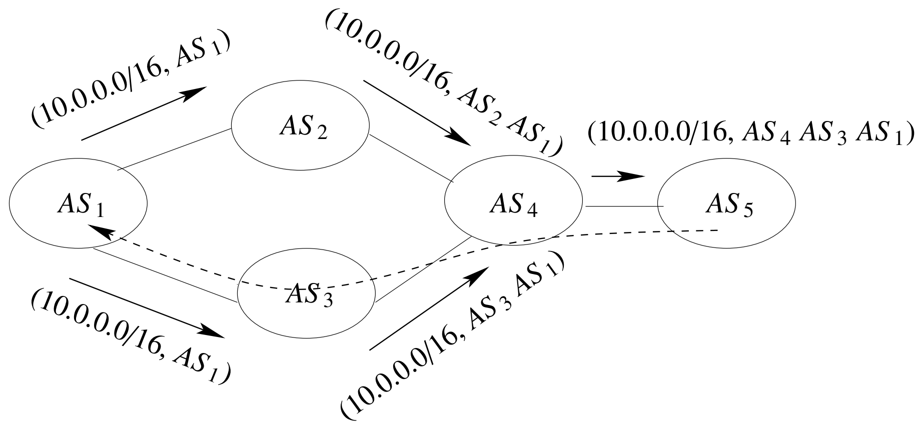


Figure 2-1: This figure illustrates how BGP works. A domain AS_1 announces (10.0.0.0/16, AS_1) to its neighbors AS_2 and AS_3, telling them that it can reach the address prefix 10.0.0.0/16. AS_2 and AS_3 propagate this reachability information by announcing (10.0.0.0/16, AS_2 AS_1) and (10.0.0.0/16, AS_3 AS_1) respectively to their neighbor AS_4. From these announcements, AS_4 learns that it has two paths $AS_4 \rightarrow AS_3 \rightarrow AS_1$ and $AS_4 \rightarrow AS_2 \rightarrow AS_1$ to reach the address prefix 10.0.0.0/16. Based on its policies, AS_4 selects the path $AS_4 \rightarrow AS_3 \rightarrow AS_1$ and announces (10.0.0.0/16, AS_4 AS_3 AS_1) to AS_5. Packets sent from AS_5 to the address range 10.0.0.0/16 will then follow the path $AS_5 \rightarrow AS_4 \rightarrow AS_3 \rightarrow AS_1$.

BGP supports policy routing. With BGP, each domain can select the next-hop neighbor to reach an address prefix and control which address prefixes to announce to a neighbor based on its local policies. By exercising the control over the next-hop selection and address prefix announcements, a domain can select a routing path that complies with its business agreements with its neighbors. For example, if a domain does not provide transit service between two neighbors, then the domain will never announce address prefixes learned from one neighbor to another neighbor. If a domain learns an address prefix announcement from both a provider and a peer, the domain would prefer using the peer as the next hop, because the domain does not need to pay the peer for traffic sent over the peering connection.

Next, we discuss related work.

2.2 Scalable Routing

In a basic routing system, a node needs to find paths and maintain forwarding information to all other nodes in a network. The amount of state stored in a node and the number of control messages for maintaining the state grow as fast as the number of nodes in the network.

Scalable routing schemes aim to reduce the amount of routing state and the control overhead. We first describe several well-known scalable routing schemes and then summarize their tradeoffs and applicability to the Internet inter-domain routing.

2.2.1 The Cluster (or Area) Hierarchical Routing

Back in the seventies, Kleinrock and Kamoun's seminal paper on hierarchical routing [64] showed that through hierarchical clustering of network nodes, the number of routing table entries of a node in a network can be reduced to $HN^{\frac{1}{H}}$, where N is the number of nodes and H is the level of hierarchy.

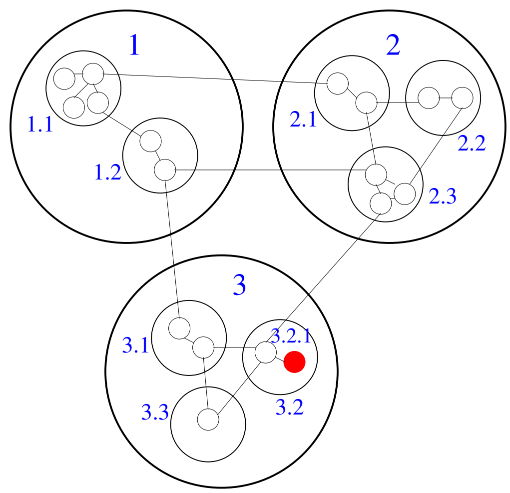


Figure 2-2: An example of a cluster-hierarchy network.

The idea of hierarchical clustering is to divide a big cluster (the entire network) into several small nested clusters, which are in turn divided into smaller nested clusters. A node's hierarchical address used for routing is based on the node's position in the cluster hierarchy, and is equal to the sequence of labels used to identify each of the node's enclosing clusters, with the first label being that of the outmost enclosing cluster, and the last label being the node's identifier. Figure 2-2 shows a 3-level hierarchical network and the addresses of some nodes.

To achieve scalability, detailed topology information inside a cluster is not propagated outside a cluster. A cluster appears as a single node to nodes in its sibling clusters. A node's routing table contains entries only for nodes in the same lowest level clusters, clusters in the same second-level, clusters in the same third-level, and so on untill the top-level clusters. Figure 2-3 shows the visible part of the network for the node 3.2.1. The node's routing table only has entries for those nodes and clusters visible to it.

To forward a packet in a cluster-hierarchical network, a node examines the destination ad-

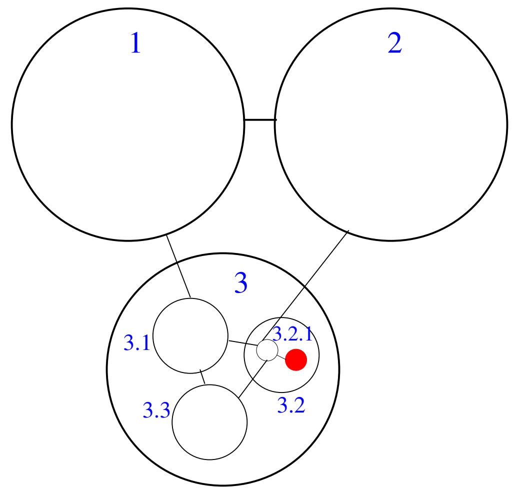


Figure 2-3: A node's routing table contains entries only for nodes in the same lowest level clusters, clusters in the same second level cluster, and top-level clusters.

dress, determines the lowest level of the cluster the destination node is in and visible to the node itself. The node forwards the packet towards that cluster. Nodes inside that cluster will send packets to the sub-cluster the destination is in. This process repeats until the packet reaches its destination.

Intermediate System-Intermediate System (IS-IS) [81] and Private Network to Network Interface (PNNI) [49] are cluster-hierarchical routing systems.

2.2.2 The Landmark Hierarchical Routing

Paul Tsuchiya proposed the landmark hierarchical routing system [103, 104] in late eighties as an alternative to the cluster hierarchical routing. His conclusion is that the dynamic management of a landmark hierarchy is easier than that of a cluster hierarchy. Hence, a landmark hierarchy is more suitable for rapidly changing networks, e.g., mobile ad hoc networks.

Central to the landmark hierarchy is the concept of a landmark. A landmark $LM_i(id)$ is a node with an identifier *id* whose neighbors within a radius of distance r_i[1] have routing entries for, or who are visible to neighbors within r_i. All nodes are landmarks at level 0. A level $i+1$ landmark is elected from level i landmarks, and r_{i+1} is greater than r_i to ensure that a level i landmark will have a level $i+1$ landmark within a distance r_i. The landmarks at the highest level (H) of the hierarchy are visible to all nodes in the network.

A node's address in a landmark hierarchy is a sequence of landmark identifiers $LM_H[id_H]$ ... $LM_1[id_1]LM_0[id_0]$, with the constraints that each landmark in the address must be within the radius of the next lower-level landmark. For the same physical network, Figure 2-4 shows the

[1]Network distance between two nodes is defined as the shortest path length between the two nodes.

three-level landmark hierarchy and the addresses of each node. In this example, $r_0 = 0$, $r_1 = 2$, and $r_2 = 4$.

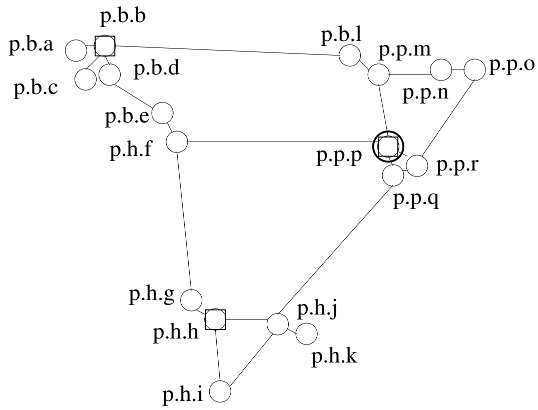


Figure 2-4: An example of a landmark-hierarchy network. A node marked by a square is a level 1 landmark. A node marked by a bigger circle is a level 2 landmark. $r_0 = 0, r_1 = 2, r_2 = 4$.

In a landmark-hierarchy network, a node only has routing entries for landmarks visible to it. Routing table sizes [103] are typically $3\sqrt{N}$, N is the number of the nodes in the network.

To forward a packet, a node examines the landmark address of the destination, and sends the packet towards the lowest level landmark in the destination address it has an entry for. Nodes in the vicinity of that landmark $LM_i[id_i]$ will have entries for the next lower level landmark $LM_{i-1}[id_{i-1}]$ in the destination address. When a packet comes closer to the landmark $LM_i[id_i]$, it will be forwarded towards the lower level landmark $LM_{i-1}[id_{i-1}]$. This process repeats until the packet reaches the destination.

2.2.3 Provider Hierarchical Routing

The interconnections between Internet domains exhibit a natural hierarchical structure. Stub domains connect to a provider domain for transit service. Local or regional providers connect to backbone providers for transit service. Figure 2-5 illustrates the hierarchical structure of the Internet. This hierarchical structure is referred to as “provider hierarchy.”

The provider hierarchy is not a strict tree-like structure. A domain may connect to multiple providers, e.g., `Stub3` shown in Figure 2-5. Two domains may have a lateral connection without forming a provider-customer relationship, e.g., the connection between `Regional2` and `Regional3` shown in Figure 2-5. A domain may have direct connections to providers at different levels of the hierarchy, e.g., `Local1` in Figure 2-5 connected to both `Regional1` and `Backbone1`.

Provider-rooted hierarchical addressing utilizes the natural structure of the Internet and was proposed to scale the Internet routing system [51, 36, 105, 50]. Providers allocate address prefixes to customers, and customers with multiple providers will get multiple addresses. If a cus-

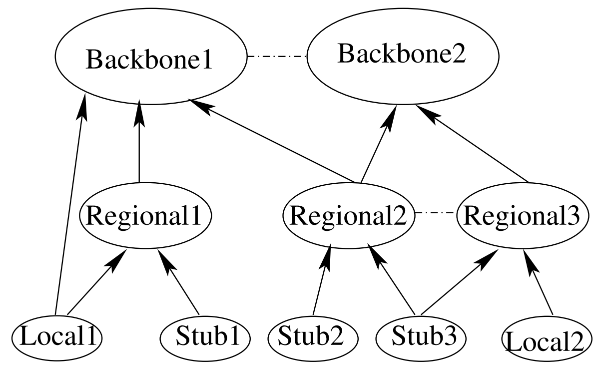


Figure 2-5: The hierarchical structure of the Internet. An arrowed line between two domains represents a provider-customer connection, with the arrow ending at the provider. A dashed line represents a peering connection.

tomer changes its providers, it will obtain addresses from its new provider and return old addresses allocated from its previous provider.

In a provider-hierarchy network, a node only needs to have an entry for a provider's address space in order to route a packet to the provider's customers. The reduction of routing table sizes depends on the structure of the network.

2.2.4 Compact Routing

A tradeoff of various scalable routing schemes is sub-optimal routing paths. A node knows less about the network compared to the basic routing scheme, thus a node may not find the optimal routing path to reach a destination. Compact routing [31, 101] is the area of theoretical study on how to make a good tradeoff between the stretch (the ratio of path length compared to the shortest path length) of routing paths and the routing table size. The best known result [101] in this area is a minimum stretch-3 routing scheme with a per-node routing table size upper bounded by $O(N^{1/2} log^{1/2} N)$ bits, where N is the number of nodes in the network.

2.2.5 Geographical Routing

In a geographical routing system, each node knows its own geographical location, and locations of neighbors within a radius of network distance r. A node only has routing entries for those neighbors. A source labels a packet with a destination node's location.

To forward a packet, an intermediate node will examine the destination's location, and send the packet towards a neighbor that is geographically closer to the destination, until the packet reaches the destination. This forwarding process may run into a "dead end": from a node's routing table, the node itself is the closest node to the destination, yet it does not know how to reach the destination.

Geographical routing was first proposed by Gregory Finn [47], and recent work on ad hoc network routing such as Distance Routing Effect Algorithm for Mobility (DREAM) [20], Location-Aided Routing (LAR) [65], and Grid [69], all uses geographical routing. The Greedy Perimeter Stateless Routing (GPSR) [63] proposal improves geographical routing algorithm and solves the "dead end" problem.

2.2.6 Distributed Hash Table (DHT) Routing

A DHT routing system is often used to provide a (key, value) lookup service. Each node has a unique identifier. The nodes' identifier space overlaps with the key space, i.e., a string *id* can be viewed as a node's identifier as well as a key. A logical routing geometry, e.g., a ring [98], a hypercube [87], or a tree [82] is created on top of the identifier space by defining which two identifiers are adjacent (at distance 1) in the logical geometry. In a ring geometry with an identifier space of size 2^L, an identifier id is adjacent to $(id + 1) \bmod 2^L$. In a hypercube geometry, an identifier with length L are adjacent to any identifier that differs by only one bit. Figure 2-6 shows an example of the ring geometry and the hypercube geometry for an identifier space of size 8. A small circle represents the position of an identifier in a geometry, and a filled small circle represents the position of an identifier taken by a node.

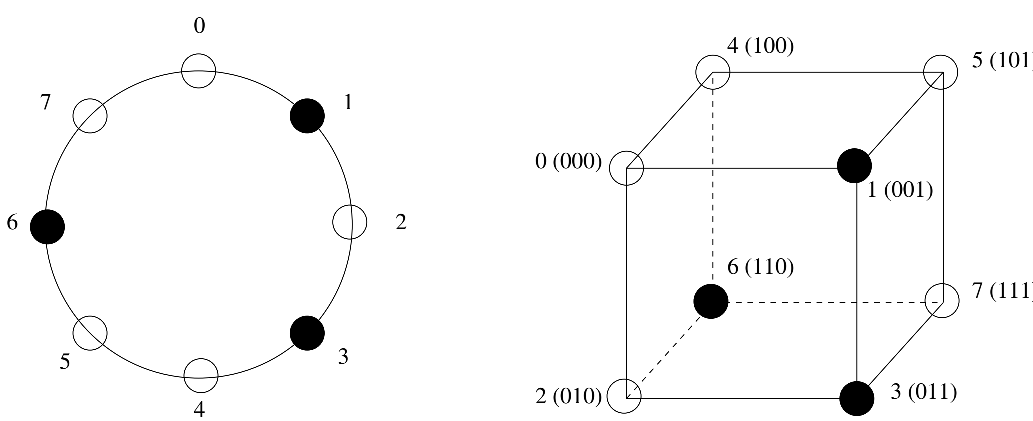


Figure 2-6: An example of the ring geometry and the hypercube geometry for an identifier space of size 8. A circle represents the position of an identifier in a geometry, and a filled circle represents an identifier taken by a node.

In DHT routing, a node *m* only needs to keep routing entries for a small subset of nodes (e.g., $O(logN)$ in a ring geometry, where N is the number of nodes). This subset of nodes is chosen based on the distances of these nodes to node *m* in the logical geometry structure. Different DHT systems have different algorithms for choosing this subset, but these algorithms all guarantee that for any identifier or key *id*, if a node *m*'s identifier is not closest to *id*, it will have a routing entry for a node whose identifier is closer to *id*.

A node stores the value for a key if its identifier is closest to the key.[2] Forwarding a query for a key k in a DHT routing system is similar to the greedy forwarding process in a geographical routing system. Each intermediate node examines k, and forwards the query towards a node whose identifier is closer to k, until the query reaches a node that has the value for k.

A good summary of various DHT routing systems can be found in [19].

2.2.7 Tradeoffs of Scalable Routing Schemes

The primary tradeoff of scalable routing schemes is sub-optimal routing path. When a node has routing entries for all nodes in the network, it is possible for a node to keep an entry for the optimal routing path (usually the shortest path) for every node in the network. For scalable routing, a node only has routing entries for a subnet of nodes, or for groups of nodes. Therefore, a node may not route a packet along the optimal path to reach a destination.

For hierarchical scalable routing schemes, such as cluster-hierarchical routing, landmark-hierarchical routing, and provider-hierarchical routing, another tradeoff is that a node's address is topology dependent. When the network topology changes, for example, a cluster becomes disconnected, a landmark moves out of range, or a site changes a provider, the address(es) of a node will change. A source needs to find out the updated address(es) of a destination in order to send a packet.

2.2.8 Applicability to Inter-domain Routing

Except for provider-hierarchical routing, it is difficult to apply other scalable routing schemes to make the inter-domain routing scalable, primarily because the Internet routing is constrained by both business agreements and physical connectivity.

In the cluster-hierarchical routing, a cluster should be connected so that a node outside a cluster only needs to keep one routing entry for the entire cluster. Once a packet reaches any node in a cluster, it can reach all other nodes inside the cluster as long as there are no intra-cluster partitions.

For Internet routing, business agreements make certain physically connected paths invalid routing paths. For example, a customer domain will not provide transit service[3] between its providers. In Figure 2-7, packets are generally forbidden to traverse the path `Provider1` → `Net1` → `Provider2`.

The restrictions imposed by business agreements make the clustering algorithm difficult. In Figure 2-7, the three domains `Provider1`, `Net1`, and `Provider2` cannot be simply clustered together, and appear as one cluster node to other domains in the Internet, because packets reach

[2]DHT routing systems could be optimized such that the value for a key can be replicated at multiple nodes.

[3]A domain is said to provide transit service between a pair of neighbors if the domain allows packets coming from one neighbor to go to the other neighbor, and vice versa.

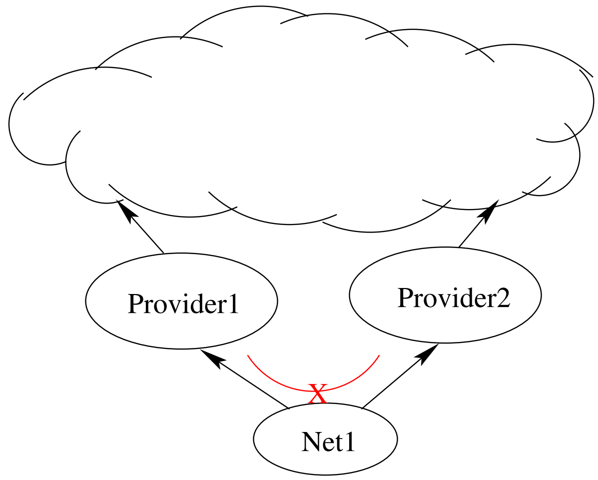


Figure 2-7: Packets are generally forbidden to traverse the path `Provider1` → `Net1` → `Provider2`. An arrowed line in the figure represents a customer-provider connection, with the arrow ending at the provider.

`Provider1` cannot reach `Provide2` without leaving the cluster. It is unclear how a clustering algorithm would take these restrictions into consideration, and how scalable the algorithm would be.

Other scalable routing schemes, landmark routing, compact routing, geographical routing, and DHT routing have similar problems. To apply these routing schemes to the inter-domain routing, when a domain picks its routing entries, the domain needs to consider business agreements between other domains. The scalability of these routing schemes is questionable after such considerations.

In addition, a DHT routing system relies on a lower level routing system to resolve the routing path to the next hop in a DHT routing entry, because two nodes that are adjacent in a logical geometry may not be directly connected in the physical network. IP routing is the lowest globally-available routing, and therefore cannot leverage the scalability of DHT routing.

Our work, NIRA, is a provider-hierarchical routing system, and we design mechanisms to solve problems such as address management and address selection that are inherent to a provider-hierarchical routing system.

2.3 Routing Architecture Proposals

There exist a number of routing architecture proposals that aim to overcome the limitations of the present routing system. We briefly describe them and discuss how our work differs from them.

2.3.1 Nimrod

Nimrod [25, 96, 86] is a cluster-hierarchical routing system. It proposes to use a map-distribution mechanism for nodes to discover network topology and to construct routes. However, Nimrod did not address how to fit its design into a policy-rich inter-domain routing environment. As a result, only an intra-domain routing protocol Private Network to Network Interface (PNNI) [49] was developed based on the design of Nimrod.

NIRA is designed for inter-domain routing. Its design directly incorporates the contractual relationships between different parties in the Internet.

2.3.2 Inter-domain Policy Routing

The Inter-domain Policy Routing (IDPR) [95] protocol was designed according to the original policy routing ideas described by Clark [28]. IDPR uses a link-state routing protocol [21] to distribute routing information. Each AS has a route server that keeps a link state database for the domain-level Internet topology and computes policy routes on the requests of hosts.

NIRA does not require the presence of per domain route servers and domain-level topology information does not flow globally in NIRA.

2.3.3 Scalable Inter-domain Routing Architecture

The Scalable Inter-Domain Routing Architecture [41] contains two routing components: a hop-by-hop node routing (NR) component and a source demand routing (SDR) component. The NR component installs routing entries for default forwarding paths, and is similar to BGP. The SDR component allows a node to specify its own routing path for a packet.

Two approaches [106] were suggested for nodes to discover routes: Routing Information Base (RIB) query and Path Explorer. A node's RIB often contains multiple routes to reach a destination, but the node only selects one route and announces the selected route to its neighbors. So if a node queries a remote node's RIB, the node may get additional routes to reach a destination.

Path Explorer is the technique that a source on-demand requests a destination to compute a path from it to the source. Intermediate nodes do not exert their route selection preferences but rather propagate routes that obey their transit policies to the source.

Recent research has shown quite a few problems of BGP, such as delayed convergence, [67, 66, 55, 107, 71]. Since the NR component proposed in the Scalable Inter-Domain Routing Architecture is similar to BGP, inserting another component, such as the SDR component, into the routing system is likely to cause more problems.

NIRA has a different route discovery mechanism and does not require a separate routing component to establish default routes.

2.3.4 IPNL

IP Next Layer (IPNL) [53] is a Network Address Translation (NAT) extended architecture that aims to solve the IPv4 address depletion problem and the routing scalability problem. It has the advantage of only modifying hosts and NAT boxes.

In the IPNL architecture, a source labels a packet with its Fully Qualified Domain Name (FQDN), the global IPv4 address of a border router in its domain, the destination's FQDN, and the global IPv4 address of a border router in the destination domain. In the global routing region, the packet will be routed to the border router of the destination domain using the existing IPv4 routing architecture. Once the packet reaches the destination domain, IPNL routers are able to route the packet using the destination's FQDN.

As an optimization, a fixed-length numeric domain-specific address is assigned to each host, similar to a private IPv4 address in today's architecture. A source will attach its private address in its first packet to a destination. A destination replies with its private address. These private addresses could replace the FQDNs in subsequent packets for efficient forwarding.

IPNL does not address the problem of user-selected routes, while NIRA does. NIRA preserves the end-to-end routability of IP addresses, and does not require special routing mechanisms in stub domains. The design of NIRA modifies both routers' and hosts' software.

2.3.5 TRIAD

TRIAD [27] is an Internet architecture that provides explicit support for content routing. In TRIAD, routing and naming are tightly integrated. Routers exchange name reachability information and map name to next-hop. Packets could be routed by names, rather than by IP addresses.

NIRA is designed to handle IP layer routing and addressing issues. The design goal is fundamentally different from that of TRIAD.

2.3.6 Feedback Based Routing System

The Feedback Based Routing proposal [110] separates routing information into topology information and dynamic information. A domain's access router keeps the domain-level topology information and transit policies of the entire Internet, monitors the dynamic status of a route, and performs route selection on behalf of users. The design of Feedback Based Routing system does not specify how routers obtain the topology information and domain transit policies.

NIRA is designed to let individual users to select routes. Each user only needs to know a small portion of the inter-domain topology. NIRA also includes protocols to propagate such information to users.

2.3.7 Overlay Policy Control Architecture (OPCA)

OPCA [15] is an architecture that builds on top of today's routing architecture to facilitate BGP policy exchange. Intelligent Policy Agents (PA) are introduced in each AS to form a policy distribution overlay network. PAs in an AS are responsible for sending explicit policy change requests to PAs in other ASes, processing policy announcements, and enforcing policies at border routers of the AS. OPCA can be used to achieve fast fail-over and inbound load-balancing. Unlike OPCA, NIRA is not designed to improve BGP. NIRA has a built-in policy distribution mechanism, and does not need a separate overlay network to distribute routing policies.

2.3.8 Platypus

Platypus [92, 85] is a routing system that aims to provide policy checking at packet forwarding time. If a user has specified a domain-level route in a packet's header, a router would like to check whether its configured policy allows it to forward the packet according to the user-specified route. Platypus describes how to implement efficient checking.

NIRA addresses various issues related to giving a user the ability to specify domain-level routes, such as route discovery, route representation and forwarding. Efficient policy checking is one of NIRA's components.

2.3.9 Routing Control Platform (RCP)

RCP [44] is an architecture that separates the task of route selection and the task of packet forwarding. In this architecture, routes are selected by RCPs of each routing domain, and routers simply forward packets according to the routing decisions made by RCPs. Unlike NIRA, RCP architecture is intended to reduce the complexity of today's Inter-domain routing architecture.

2.4 Current Route Control Technologies

As there is no global support for users to select routes, local and small scale solutions are created to satisfy users' needs for selecting routes. There are both commercial products and academic efforts. We describe these techniques and their limitations.

2.4.1 Commercial Route Control Technologies

Several companies, including Internap [8], OPNIX [4], RouteScience [5] emerged to offer route control services or products. Route control technologies can help a multi-homed site to choose the access link to its providers. The technologies involve real-time path performance monitoring and evaluation. Based on the measurement data, a program is able to automatically change an edge router's forwarding table, so that the traffic will go through the desired access link.

Route control products are limited to selecting the next hop provider for outbound traffic, because to influence inbound traffic, an edge router needs to frequently send out BGP announcements, which harms the stability of the routing system. Besides, these products cannot choose beyond the first hop provider. Moreover, they are generally not affordable by individual users or small sites.

2.4.2 Overlay Networks

Another technique for route selection is overlay network. An overlay network is formed by a small group of end hosts that voluntarily provide packet forwarding and/or duplication service for each other [16, 97, 102, 90].

The limitation of an overlay network is that it is not ubiquitous. Only nodes on an overlay network can control their paths by tunneling traffic through other nodes on the same overlay network. It is unlikely that overlay networks can scale up to include every computer on the Internet.

Besides its limited scope, an overlay architecture is less efficient than an architecture that supports loose source routing. Different overlay links may share the same physical links. For example, a packet relayed by an end host at least traverses the last hop twice. This "detour" wastes link bandwidth and router processing cycles, and increases packet delivery latency compared to a source routing path that does not traverse the last hop twice.

Chapter 3

Design Rationale

NIRA is designed to give a user the ability to select domain-level routes. A domain-level route is a sequence of domains a packet traverses; a router-level route refers to the sequence of routers a packet follows. We emphasize domain-level choices rather than router-level choices because choices at this level are intended to stimulate competition among providers. A domain may offer router-level choices to users if its own policies allow such choices. NIRA neither mandates a domain to do it, nor prevents a domain from doing it. Our design does not include mechanisms for router-level route discovery, but allows a user to specify a router-level route in a packet header.[1] We focus our discussion on domain-level choices. In the rest of this dissertation, unless otherwise specified, a "route" refers to a domain-level route.

3.1 Modularization

As a general principle, the more architectural constraints we put into a system, the more rigid the system is and the less likely it will evolve and survive. So in our design, we aim to reduce the amount of architectural constraints to minimal. Our approach is to identify the set of components in the inter-domain routing system that have little dependency between each other, and make them into separate design modules. For each design module, we analyze whether a global consistent mechanism is needed to implement that module, or multiple mechanisms can be developed and deployed locally. In this dissertation, we design mechanisms that realize each of the modules, and make it clear whether our mechanisms should be used globally or different local mechanisms could be developed.

As we have discussed in Chapter 1, we identify that the design of NIRA can be decomposed into three key design problems:

1. Route discovery and selection: How does a user discover the various routes connecting him

[1] We'll make it clear how to specify a router-level route when we discuss route representation in Chapter 5

to a destination and select one to use?

2. Route representation and packet forwarding: Once a user has chosen a route to reach a destination, what should he put into a packet header so that the packet will be forwarded along the chosen route unambiguously?

3. Provider compensation: In a commercialized Internet, if a provider cannot be properly compensated, it will have little motivation to forward packets according to a user's route specification. Thus, the design of an architecture should take into consideration how a provider is compensated if a user chooses to use its service.

Furthermore, we can separate the problem of route availability discovery from that of route discovery.

4. Route availability discovery: How does a user discover whether the dynamic attributes of a route allow him to send a packet to a destination, e.g, whether a route is failure-free?

With this separation, the primary task of route discovery then becomes discovering routes between a user and a destination that are determined by the physical structure of the Internet and the contractual agreements between ISPs, and does not necessarily include discovering whether the dynamic attributes of those routes would allow a packet to be sent from a user to a destination. Multiple general mechanisms could be developed to accomplish the task of route discovery and that of route availability discovery. Moreover, as the dynamic attributes of a route could change at a high frequency, but the physical structure of the Internet and the contractual agreements between ISPs may remain stable, this separation also allows discovered routes to be cached for reuse.

These four sub-problems are relatively independent. Mechanisms that solve one problem could be changed without changing mechanisms to address the other problems. For example, regardless of what a user takes to discover and select a route to reach a destination, the user could apply various mechanisms to discover the availability of the route. Even if the route representation and the packet forwarding algorithm are changed, how a user discovers a route or the availability of a route could remain the same.

3.2 Design Requirements

There exist many solutions to these problems. We narrow down the solution space by identifying several key design requirements:

- Scalability. With the fast growth of the Internet and the possibility that millions of small devices will be connected to the Internet, we require that no single routing component needs to have the bandwidth, memory, or processing power that scale linearly or super linearly with the size of the Internet.

- Robustness. This requirement has two components. First, if a route is unavailable, a user should be able to find an alternative route within a few round trip times if one exists. We think this fail-over time is reasonable for Internet applications. Second, the failures, mistakes, or malicious behavior of one router should not have a global impact.

- Efficiency. The overhead added for supporting user-specified routes such as extra packet header space should be minimized for common cases.

- Heterogeneous user choices. Individual users in the same domain should be allowed to choose different providers. For example, if a local provider connects to multiple wide-area service providers, users of the local provider should be allowed to have business agreements with the wide-area providers of their own choices. If a user signs agreements with more than one wide-area provider, the user should be able to dynamically choose a provider to carry his packets.

- Practical provider compensation. Provider compensation should not require per-packet detailed accounting or micro-payment [73]. History suggests that these schemes have little market appeal [59, 60, 91].

Next, we describe how we design NIRA towards satisfying these requirements.

3.3 Route Discovery

The goal of NIRA is to support user-specified routes. The first problem we need to address is how a user discovers various routes that connect him to a destination and selects one to send packets. A straightforward approach for route discovery is to use a link-state routing protocol to distribute the entire domain-level topology and the transit policies of each domain. Both IDPR [95] and feedback-based routing [110] suggest this approach. In these proposals, the border router of a domain or a specialized server in a domain stores the domain-level topology and domain transit policies, and does route selections for users.

This approach does not fit our design requirements. To support heterogeneous user choices, we must allow transit policies to be specified in the context of individual users. For example, a local provider may connect to UUnet, but a user in the local provider's network does not sign a business agreement for UUnet's service. Packets destined to that user cannot be sent through UUnet. Hence, it is necessary to specify transit policies at the granularity of individual users. It is not scalable to keep this amount of information at a single router or a server. For this reason, we cannot adopt the previous proposals.

We observe that the natural structure of the Internet allows us to modularize the task of route discovery, so that a user does not need to know the entire inter-domain topology. In the Internet environment, a domain decides whether it will provide transit services between a pair of its neighboring domains or for certain address prefixes, based on its business relationships with its neighbors. A domain's decisions are reflected in the domain's transit policies. There are three most common business relationships [54] between two interconnecting domains: the provider-customer relationship, the peering relationship, and the sibling relationship. In a provider-customer relationship, the customer domain pays the provider domain for service, and the provider domain provides transit service between a customer domain to all its neighbors; in a peering relationship, two peers do not pay each other, but each peer only provides transit service between its customer domains and the other peer; in a sibling relationship, each domain allows the other domain to use its providers for mutual transit service. Compared to the provider-customer relationship and the peering relationship, the sibling relationship is less common, and is usually negotiated between two edge domains to save each other's cost of purchasing additional providers [54] for robust and redundant Internet access service. For reason of clarity, in the main body of the dissertation, we discuss our design without considering sibling relationships. We concentrate discussions on the sibling relationship in Appendix 5.C.

A typical domain-level route is said to be "valley-free." That is, after traversing a provider-to-customer connection, a route cannot traverse a customer-to-provider connection or a peering connection. So when an end user sends a packet, the packet is usually first "pushed" up along a sequence of customer-to-provider links, and then flows down along a sequence of provider-to-customer links. There exists a densely connected region of the network [99] where packets can not be further "pushed" up. We call this region the *Core* of the Internet. Outside the *Core*, the interconnection of the network is relatively sparse, where the dominant interconnection relationship is the provider-customer relationship [54]. We discuss more about the role of the *Core* in our architecture in Chapter 6.5. A packet does not have to be pushed all the way up to the *Core* to reach its destination. Alternatively, a peering link outside the *Core* may connect the sender's provider chain and the receiver's provider chain. A packet can take the short-cut to reach its destination. Research results on topology inference further confirmed the existence of this structure [54] [99]. Figure 3-1 shows a simplified picture of the Internet.

We utilize this structure of the Internet, and divide the task of route discovery into a sender half and a receiver half. On the sender half, a user discovers topology information on domains that provide transit service for him according to contractual agreements, i.e., his providers, his providers' providers, and so on. We call this part of the network a user's "up-graph." Figure 3-2 shows a user Bob's up-graph. From the up-graph, a user would know his most commonly used routes to reach the *Core*, and the peering connections of the providers on his up-graph. We design a network protocol, Topology Information Propagation Protocol (TIPP), for a user to learn such information.

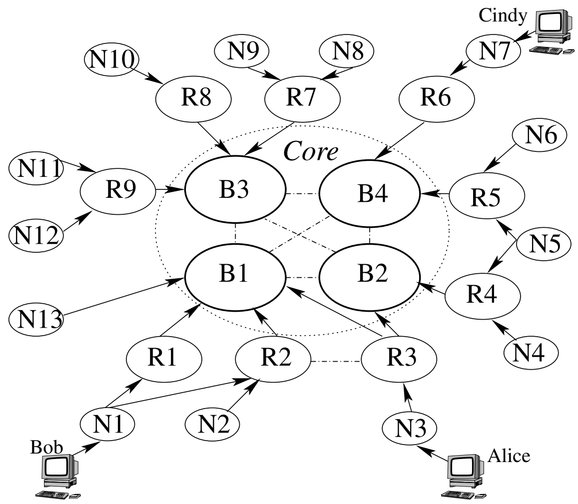


Figure 3-1: A simplified view of the Internet domain-level topology. A domain is represented by an ellipse. An arrowed line between two domains represents a provider-customer connection, with the arrow ending at the provider. A dashed line represents a peering connection.

On the receiver half, a user that wants to become a server stores his commonly used routes to reach the *Core* at a distributed Name-To-Route Lookup service (NRLS). When a user wants to communicate with a destination, the user could query the NRLS to retrieve the route information of the destination. Combining his route information with the destination's route information, a user is able to specify an initial route to reach the destination. Two users may exchange subsequent packets to negotiate a better route based on their preferences.

A routing region refers to a contiguous region of the Internet that runs an inter-domain routing protocol. In our design, we make the *Core* a routing region. If a user-selected route goes across the *Core*, the portion of the route inside the *Core* is therefore chosen by the providers in the *Core*, instead of by users. We make this design choice because in practice, the business relationships between domains in the *Core* are complicated. Domains may belong to different countries and have sophisticated commercial contracts. Therefore, in most cases, we expect that users are not allowed to choose due to the complicated provider compensation problem, or cannot choose because there is only one policy-allowed route. Moreover, our primary goal for enabling user choice is to encourage competition among wide-area providers. We think once a user can choose to which wide-area provider to send his packets, although he cannot choose the route that connects two wide-area providers (the portion of the route in the *Core*), it is sufficient to exert competitive pressure on wide-area providers.

Our route discovery mechanism does not require a single router or a server to keep the route information and transit policies for every user. The tradeoff is that using a distributed lookup service, users may spend several round trip times to retrieve the destination's route information. We do not think this overhead is a significant problem because the common practice of using

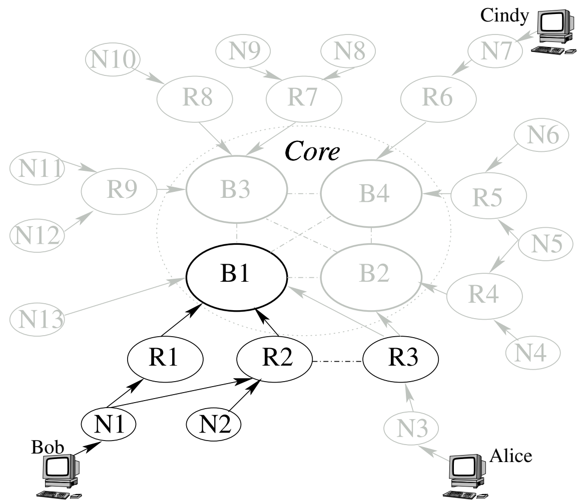


Figure 3-2: User Bob's up-graph is drawn with dark lines.

DNS [74] names to retrieve IP addresses has a similar overhead. Since we deliberately separate the dynamic route availability discovery from static route discovery, routes retrieved on demand can be cached for later use. We expect that caching will amortize the overall cost on round trip times.

TIPP and NRLS are the basic mechanisms we provide for route discovery. The network protocol, TIPP, propagates to a user his up-graph. We design TIPP such that its messages do not propagate globally. So it is unlikely that the fault of a single routing component will have a global impact. The distributed service for looking up a destination's route information, NRLS, is basically an enhanced DNS.

Our modularized design for route discovery allows for generality. We provide basic mechanisms that assist users to discover the typical "valley-free" routes, and do not guarantee to discover all possible routes. Users can use any general mechanisms to discover other types of routes.

3.4 Route Availability Discovery

After a user has learned multiple routes to reach a destination, he also needs to discover whether the dynamic attributes of a route permits him to send packets to the destination. For example, he needs to know whether the route is failure free, or satisfies his delay, bandwidth, or loss requirement. We have considered three general mechanisms for route availability discovery. The first one is reactive notification. If a route is unavailable when a user tries to use it, a router will send a control message to the user, or a user detects it via time-out. This mechanism does not require the dynamic attributes of a route to be flooded globally, but may increase connection setup time. The second one is proactive notification. The network floods the dynamic route conditions to users. The third one is proactive probing. Users send probe packets along routes they are interested in. Given the size of the Internet, the last two mechanisms are unlikely to be efficient and scalable in general, but can form part of a larger solution.

In NIRA, proactive notification and reactive notification are used in combination to compensate for the performance problems of each other. The network protocol, TIPP, proactively notifies a user of dynamic network conditions on a user's up-graph. A user detects failures on other parts of the network via router feedback or timeouts. So again, we exploit the regional modularity of our design.

Our design provides the primitive mechanisms for users to detect route availability. Multiple other mechanisms could be built upon these primitives for a user to handle route unavailability. For example, if a user has learned multiple routes to reach a destination during the route discovery process, and he discovers a failure on his selected route, he could quickly switch to a different route. A user could also probe two routes in parallel, and select the one with better performance for communication. In our design, we do not specify how a user handles route unavailability. In this dissertation, we also refer to route unavailability as route failure.

3.5 Route Representation and Forwarding

After a user has chosen a route to send his packets, he needs to express his route choice in a packet header so that routers know how to forward the packet along his chosen route. In the current Internet, if a user does not specify a route in a packet, he only needs to put a source address and a destination address in a packet header. An architecture that allows a user to specify routes is likely to increase the packet header overhead for route representation.

To minimize the overhead, we introduce a route representation scheme such that a common type of domain-level route can be represented by a source and a destination address, and general routes can be represented by multiple addresses, i.e., a source route. The network protocol we designed, TIPP, automatically propagates to a user his addresses, and the dynamic route conditions represented by the addresses. When sending a packet, in most cases, a user only needs to pick a source address and a destination address, and puts the two addresses in a packet header. This representation is as convenient and efficient as in today's Internet.

3.6 Provider Compensation

In a commercial Internet, providers are not going to give service away. We design the provider compensation schemes in NIRA to be conditioned on pre-negotiated contractual agreements as it is today. Providers decide whether they would provide transit service for a user based on the contractual agreements between the user and the providers. They may install policy filters to prevent illegitimate route usage. Providers will be properly compensated by billing their customers based on contractual agreements.

Our hypothesis is that it is practical to perform policy checking at packet forwarding time, as suggested by recent work [92, 85, 17]. Micro-payment [73] schemes may not require pre-negotiated contractual agreements, but we rule out such schemes as we consider they are inefficient to implement.

3.7 Putting the Pieces Together

We describe how a user picks a route and sends packets when he wants to communicate with a destination. We use the example in Figure 3-1. Suppose Bob wants to communicate with Alice. The following illustrates what happens.

1. With the network protocol TIPP, Bob learns his addresses, his up-graph, and the dynamic network conditions on his up-graph.

2. Bob queries NRLS servers to obtain Alice's addresses that represent Alice's most commonly used routes to the *Core* , in other words, her view of her up-graph.

3. With his addresses, and his up-graph (shown in Figure 3-2), Bob will know that there are multiple addresses representing different routes that go up to the *Core* to reach Alice, and one route that goes across the peering connection $R_2 \rightarrow R_3$ to reach Alice.

4. Bob picks an initial route to send the first packet based on his preference, dynamic network conditions on his up-graph, and Alice's preference specified in her NRLS records. Bob expresses this route by the choice of the source and the destination address he uses.

5. If the initial route suffers from failures, Bob detects the failure from router feedbacks or timeouts, and switches to a different route. If the second route works well, the first packet reaches Alice. Otherwise, Bob repeats the failure detection and route switching process until his first packet reaches Alice.

6. Subsequently, Bob and Alice may exchange packets to negotiate a better route.

3.8 Summary of Design Decisions

We summarize the modularizable components of NIRA we have identified.

- We modularize the design of NIRA into four key design problems, route discovery, route availability discovery, route representation, and provider compensation. Moreover, we disentangle the problem of route availability discovery from that of route discovery. Our design provides primitive mechanisms to address each of the problems, and allows additional general mechanisms to be developed.

- We utilize the natural structure of the Internet to modularize the task of route discovery. Each user is responsible to know about his route information. A sender on demand retrieves a receiver's route information. With his route information and that of the receiver, a sender is able to discover and select routes to reach the receiver. Again, we design mechanisms for a user to learn his route information and to retrieve a receiver's route information, but allows a user to use any general mechanism to achieve either task.

Chapter 4

Route Discovery and Failure Detection

In this chapter, we describe the mechanisms we provide in NIRA for users to discover routes and to detect route failures. NIRA uses three components for scalable route discovery: a strict provider-rooted hierarchical addressing scheme that gives a user multiple addresses, a network protocol TIPP (Topology Information Propagation Protocol) that propagates topology and address allocation information, and a distributed Name-to-Route Lookup Service (NRLS). NIRA uses TIPP, and other mechanisms such as router feedback and timeouts for route failure detection. Figure 4-1 shows the dependency between the function modules and the mechanisms.

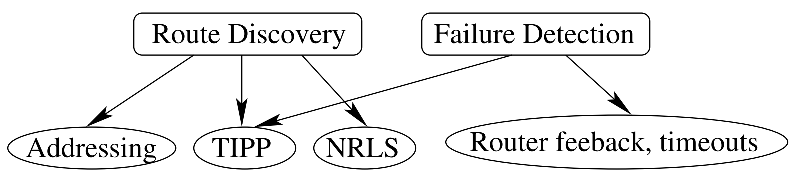


Figure 4-1: Dependency between the function modules (route discovery and failure detection) and the mechanisms.

4.1 Background

We first define a few terms and concepts. In our explanation, we use examples from today's IPv4 routing system as IPv4 addresses are simpler than addresses used in NIRA, but the terms and concepts are equally applicable to NIRA.

- *Address prefix*: an address prefix p is a shorthand for designating a range of addresses that share the same prefix bits as those in p. We also call the range of addresses represented by p an *address space*. An address prefix is often written in the format *address/prefix length*. For example, 18.26.0.0/16 represents an IPv4 address prefix that includes addresses ranging from 18.26.0.0 to 18.26.255.255. An *address prefix* is sometimes shortened as a *prefix* in this document.

An address prefix p is said to enclose another address prefix p' if all addresses in the range of p' are a strict subset of the addresses in the range of p. For example, the IPv4 address prefix 18.26.0.0/16 encloses the IPv4 address prefix 18.26.0.0/24. The address prefix p is also said to be larger than the address prefix p'; likewise, the address prefix p' is said to be smaller than the address prefix p.

- *Longest prefix match*: A routing table usually consists of multiple prefix entries, with each entry being a (prefix, next hop) pair. Longest prefix match is a rule for deciding the entry to which an address is matched. The rule says that if an address falls into the range of multiple address prefixes in a routing table, the entry with the longest prefix length is considered as a match. For example, if a routing table has two entries with address prefixes 18.26.0.0/16 and 18.26.0.0/24, then the address 18.26.0.55 falls into the range of both prefixes, but is considered to match the second entry, 18.26.0.0/24, because the prefix length 24 is longer than 16.

- *Forwarding table* versus *routing table*: both are names for tables that consist of (prefix, next hop) entries and are used to determine the next hop to forward a packet. Usually, in a routing table, a prefix entry is learned from a routing protocol, such as BGP. In this document, we use the name *forwarding tables* to refer to such tables whose entries are not necessarily learned from a routing protocol.

4.2 Provider-Rooted Hierarchical Addressing

A provider-rooted hierarchical addressing scheme takes advantage of the natural structure of the Internet, and has long been proposed to make the Internet routing scalable [51, 36, 105, 50]. As we will see, a provider-rooted hierarchical addressing scheme brings us two advantages. First, it makes the *Core* a scalable routing region; second, it enables an efficient route representation scheme (we discuss this in Chapter 5). In this section, we discuss how provider-rooted hierarchical addressing works.

4.2.1 Network Model

The Internet has a loosely hierarchical structure that is defined by the provider-customer relationship. In a provider-customer relationship, a provider domain will transit packets between a customer domain and all its neighbor domains. In practice, there is no provider-customer loop. That is, a provider domain will not acquire transit service from a domain that is its own customer, or its indirect customer (its customer's customer and so on). Therefore, the region of the network that starts at a provider, and includes the customers of the provider, recursively its indirect customers, and the provider-customer connections between these domains, mimics a tree structure. We call such a region a *provider tree*. Figure 4-2 shows an example of a provider tree rooted at the provider B_1. Note that the Internet is only loosely hierarchical. So the provider tree is not a strict tree structure. One domain may simultaneously attach to multiple providers, such as N_1 and R_3. There are lateral links such as the peering connection between R_2 and R_3. A provider tree does not include those peering connections.

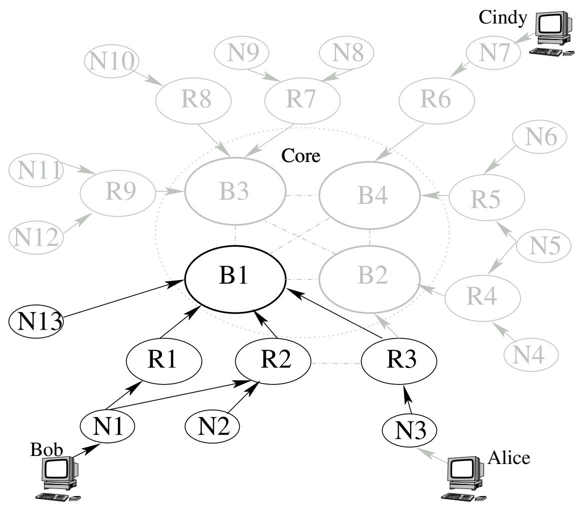


Figure 4-2: The dark region depicts the provider tree rooted at the top-level provider B_1. Note the peering connection between R_2 and R_3 is not part of the tree.

The *Core* of the Internet is where the roots of provider trees meet. A provider in the *Core* is called a top-level provider. In our design, the *Core* is a routing region, and top-level providers in the *Core* are connected by a routing protocol.

4.2.2 The Addressing Scheme

A strict provider-rooted hierarchical addressing scheme refers to the following address allocation scheme. Each top-level provider in the *Core* obtains a globally unique address prefix from a global address space. A top-level provider then allocates non-overlapping subdivisions of the address prefix to each of its customers, and each customer will recursively allocate non-overlapping subdivisions of the address prefix to any customer it might have. If a domain changes its providers, its address prefixes allocated from its previous providers will be returned to those providers, and the domain will obtain address prefixes from its present providers.

It can be seen that in provider-rooted hierarchical address allocation, addresses are recursively allocated "down" a provider tree. Domains that are simultaneously on multiple provider trees will get multiple address prefixes. From a provider to any domain that is on the tree or subtree rooted at the provider, there is always a policy-valid route that consists of only provider-customer connections. Therefore, if a user wants to send packets to a destination address *d*, the user only needs to find a policy-valid route to reach any domain who has an address prefix that encloses *d*, and the user does not need to know the detailed structure of the provider tree that allocates the destination address *d*.

A strict provider-rooted hierarchical addressing scheme makes the *Core* a scalable routing region. Inside the *Core*, a top-level provider only needs to keep routing entries for address prefixes of other top-level providers. As the ISP market is a high barrier market, financial constraints will limit the number of ISP market entries. So the number of domains in the *Core* is small compared to that of the Internet. We think running an inter-domain routing protocol within that region should not have significant performance problem or scaling problem.

4.2.3 Example

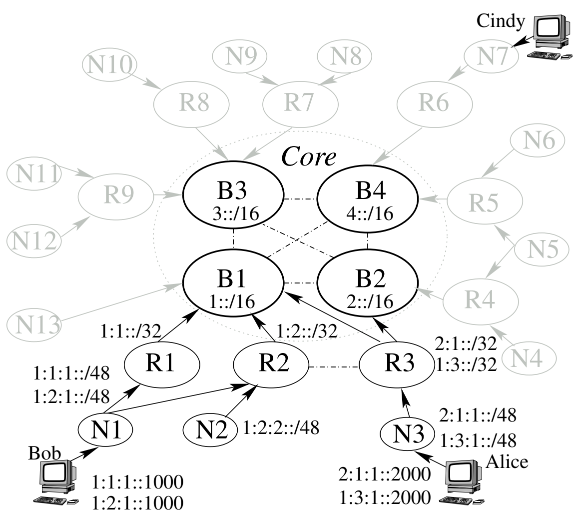


Figure 4-3: An example of strict provider-rooted hierarchical addressing. For clarity, we only shown address allocation for the dark region of the figure.

Figure 4-3 shows an example of the strict provider-rooted hierarchical addressing scheme using the same network topology shown in Figure 3-1. We represent an address and an address prefix using the IPv6 notation as described in [36], where "::" represents a variable number of contiguous zeros, and ":" is a separator that separates every 16-bit piece. Each 16-bit piece is represented by hexadecimals. An address prefix is represented in the format *address/prefix length*. For example, 1:2::/32 designates an address prefix that has a length 32 bits. The first 16 bits of the prefix has the value 1, and the second 16 bits of the prefix has the value 2.

This example shows that the top-level providers B_1 and B_2 each obtain a globally unique address prefix 1::/16 and 2::/16. In this example, we assume that the global address space from which the prefixes in the *Core* are allocated is 0::/1. B_1 allocates non-overlapping subdivisions of 1::/16: 1:1::/32, 1:2::/32, 1:3::/32, to its customers R_1, R_2, and R_3. Recursively, R_1 allocates a subdivision of 1:1::/32: 1:1:1::/48, to its customer N_1. So do R_2 and R_3. Domain N_1 is allocated two address prefixes 1:1:1::/48 and 1:2:1::/48. So a node in N_1, Bob, has two addresses 1:1:1::1000 and 1:2:1::1000. We will explain more about the sub-structure of an address in Section 4.2.5.

4.2.4 Address Allocation Rules

We describe the rules a provider-rooted hierarchical addressing scheme should follow, and the properties these rules induce. Two address prefixes are said to be non-overlapping if no address belongs to both address prefixes. The two important rules for address allocation are non-overlapping and non-looping.

- **Non-Overlapping Rule:**
    1. The global unique prefixes allocated to each top level provider must be non-overlapping.
    2. For a domain M, if M has a prefix p, and allocates subdivisions of p to its customers, then these subdivisions must be non-overlapping.
- **Non-Looping Rule:** If a domain has a prefix p, it is prohibited to take an address prefix allocated from a neighbor if that prefix is within the address space p. This rule prevents address allocation loop. In practice, there should exist no provider-customer loop. Thus, a domain will never be allocated a prefix that is within its address space.

From these two rules, we can infer the following property regarding the allocated address prefixes.

- **Address to Path Mapping Property:** If a domain M has a prefix p, and a domain N has a prefix p', and the prefix p' is within the address space p, then there exists a unique path $M \to P_1 \to P_2 \to ... \to P_i \to ... \to N$ that solely consists of provider-customer connections. Each domain P_i along the path from M to N has an address prefix p_i such that p_i is within the address space p, and p' is within the address space p_i.

This property is useful for efficient route representation. A provider-rooted hierarchical address can then be mapped into a domain-level route. In Chapter 5, we show how we take advantage of this property to design a route representation scheme that uses a source and a destination address to represent a route. We formally prove this property in Appendix 4.A.

4.2.5 Address Format

The addressing scheme describes how addresses are assigned, but does not specify the syntactic form of an address, i.e., address format. In principle, address format is independent of an addressing scheme, and is not a key architectural issue. The difference an address format makes is primarily syntactic. If we change the address format of our addressing scheme, the other mechanisms in NIRA are still applicable. For instance, a user can still use the same route discovery mechanisms to find and select routes.

For a provider-rooted hierarchical addressing scheme, an address could be either fixed length or variable length. Since the depth of provider hierarchy varies at different parts of the network, it seems that a variable-length address format would be a good fit. However, we choose to use a fixed-length address for two reasons, and largely out of convenience. First, the Internet community has spent much effort in deploying IPv6 [35], which uses a 128-bit address format. We think it will be advantageous to reuse some design aspects of IPv6, such as the header format. Second, variable-length addresses are believed to be less efficient compared to fixed-length addresses. Therefore, in our design, we adopt the IPv6 address format. An address is represented by a 128-bit string.

Furthermore, we need to determine the sub-structure of an address. With a provider-rooted hierarchical addressing, an address will have a section that contains a hierarchically allocated address prefix. This section could be of fixed length or variable length. If we make it of variable length, then if a node[1] has multiple addresses, the remaining bits of each address after the hierarchically allocated prefix would be different. If we make it of fixed length, then the remaining bits in the multiple addresses of one node could be the same. This structure allows intra-domain routing to be separated from inter-domain routing. So in our design, the section of an address containing a hierarchically allocated prefix is made to have a fixed length: 96-bit. We call this section the inter-domain address. The remaining 32 bits of an address is called the intra-domain section of an address or the intra-domain address. Figure 4-4 shows the format of a hierarchically allocated address.

Figure 4-4: The format of a hierarchically allocated address.

When a domain D is allocated an address prefix p, it can choose the inter-domain section of an address, the first 96-bit of an address, for all nodes inside itself from the address space p. Since each node inside a domain has an intra-domain address that is unique within the domain, a node inside a domain forms its address by concatenating the inter-domain section of an address derived from p and its intra-domain address. A simple way of picking the inter-domain section of an address is to pad the prefix p with zeroes until the length reaches 96-bit. For the purpose of easy illustration, we assume domains choose their inter-domain addresses using this simple method. For example, in Figure 4-3, when the domain N_1 is allocated the address prefix 1:1:1::/48, it will

[1] A node is either a host or a router.

pick 1:1:1::/96 as the inter-domain address for all nodes in itself. A node inside N_1, Bob, has a 32-bit intra-domain address 0:1000. Bob forms his address 1:1:1::1000 by concatenating the inter-domain address 1:1:1::/96 and his intra-domain address 0:1000.

Separating an address into an inter-domain section and an intra-domain section is another application of modularization. This modularized structure of an address allows intra-domain communication to be isolated from inter-domain address allocation. We can assign a special 96-bit prefix to represent a "site-local" inter-domain section of an address for all domains. A node could use this inter-domain section concatenated with its intra-domain address to form a site-local address for all intra-domain communication, regardless of what address prefixes are allocated to a domain. For example, during bootstrapping, when a node does not know its hierarchically allocated inter-domain address prefixes, it could communicate with a server inside its domain using its site-local address to obtain its hierarchically allocated prefixes. In the process of switching providers, a domain's inter-domain prefixes will change, but having a fixed-length intra-domain address section allows intra-domain communication to stay uninterrupted.

Moreover, this separation facilitates address management. Once a domain is allocated an address prefix p, and has picked an inter-domain section of an address for itself from the address prefix p, any node in the domain could automatically derive its address by concatenating the inter-domain section of an address and its own intra-domain address. A domain does not have to allocate an address from p for every node inside it.

Furthermore, having a separate inter-domain section and an intra-domain section of an address allows a router in a domain to split its routing table into two parts: the inter-domain part and the intra-domain part. This split may be used to reduce the size of the routing tables of a domain with multiple address prefixes. If a domain has K address prefixes (in other words, it has K routes to the *Core*), each node inside the domain will have up to K addresses. So the number of routing table entries for nodes inside the domain might be inflated by K times if a router in the domain keeps a single routing table. To avoid this inflation, a router could split its routing table, and keep entries for the inter-domain addresses of the domain in the inter-domain part, and keep entries for the intra-domain addresses in the intra-domain part of the table. The forwarding module of a router would first consult the inter-domain part of the routing table. If the forwarding module decides a packet is destined to the domain, then the intra-domain part of the table is consulted to determine which node is the destination.

Figure 4-5 shows an example. Suppose a domain has two inter-domain addresses, p_{d_1} and p_{d_2}, and four nodes n_1, n_2, n_3, n_4. If a router keeps one routing table, the number of routing table entries for nodes inside the domain would be inflated by two, with a result of $2 \times 4 = 8$ entries, as shown in Figure 4-5-(a)[2]. Because an address has a fixed-length intra-domain section,

[2]For simplicity, we assume that a routing table has an entry for every node. In practice, the fi nest entry of a routing table is usually an address prefi x for a local network. This simplifi cation does not affect our explanation on how to split a routing table, as one can imagine each entry in the table is for a local network, rather than for a node.

the routing table could be split into two parts, as shown in Figure 4-5-(b) and Figure 4-5-(c). One part contains entries for the inter-domain prefixes, $p_{d_1}/96$ and $p_{d_2}/96$; the other part contains entries for the intra-domain addresses. The number of routing entries is reduced to 2 + 4 = 6.

Address	Next Hop
$p_{d_1}a_{n_1}$	n_1
$p_{d_1}a_{n_2}$	n_2
$p_{d_1}a_{n_3}$	n_3
$p_{d_1}a_{n_4}$	n_4
$p_{d_2}a_{n_1}$	n_1
$p_{d_2}a_{n_2}$	n_2
$p_{d_2}a_{n_3}$	n_3
$p_{d_2}a_{n_4}$	n_4

(a)

Address	Next Hop
$p_{d_1}/96$	self
$p_{d_2}/96$	self

(b)

Address	Next Hop
a_{n_1}	n_1
a_{n_2}	n_2
a_{n_3}	n_3
a_{n_4}	n_4

(c)

Figure 4-5: This example shows that the routing table entries for nodes inside a domain is inflated by 2 when a domain has two address prefixes, p_{d_1} and p_{d_2}. If an address has a fixed-length intra-domain section, the routing table could be split into two parts: the inter-domain part (b) and the intra-domain part(c), to avoid this inflation.

4.2.6 Non-Provider-Rooted Addresses

In addition to provider-rooted hierarchical address, we also introduce a type of non-provider-rooted address. This type of address is useful for bootstrapping communication. When a domain has not acquired address prefixes from its providers, it could use this type of address to communicate with its adjacent domains. It is also useful in specifying a route that cannot be represented by only a source and a destination address.

The format of this address type is shown in Figure 4-6. It consists of a prefix indicating the non-provider-rooted type, a 32-bit domain identifier, and a 32-bit intra-domain address. The 32-bit domain identifier uniquely identifies a domain, and is similar to the AS number in today's Internet.

0	64	95	127
prefix	domain ID	intra-domain	

Figure 4-6: The format of a non-provider-rooted address.

4.2.7 Extended Addressing

We have described the basic version of a strict provider-rooted hierarchical addressing scheme, where addresses are allocated from providers to customers. In an extended version of the addressing scheme, addresses could be allocated in other situations, such as between domains that have a sibling relationship. If two domains M_1 and M_2 have a sibling relationship, then M_1 could allocate a subdivision of an address prefix it obtains from a provider to M_2, and so does M_2.

4.2.8 Discussion

One possible concern with a strict provider-rooted hierarchical addressing scheme is that a domain may get too many address prefixes. However, in practice, due to economic constraints, the depth of provider hierarchy is shallow [59, 60], and a domain is usually located only on a limited number of provider trees. Therefore, we expect that a domain will have a manageable number of address prefixes. In Chapter 7, we use Internet measurement results to show that the number of address prefixes a domain has is likely to be small, ranging from a few to a few dozens.

With a fixed-length address format and variable levels of provider hierarchy, theoretically, hierarchical address allocation may run out of available bits. However, since the inter-domain section of an address has 96 bits, if these bits are allocated prudently, they could address up to 2^{96} domains. In any foreseeable future, the number of domains in the Internet is unlikely to be anywhere close to that number. So we do not think the fixed-length address format will be a problem in the future.

Our fixed-length address format also limits the intra-domain section of an address to be 32-bit long. So the number of intra-domain addresses is at most 2^{32}. Our assumption is that this number is sufficiently large to address all nodes in a domain. If it is not, two or more 96-bit prefixes can be allocated to that domain such that nodes in the domain with the same 32-bit intra-domain addresses will have different 96-bit prefixes. The domain only needs to modify its intra-domain routing table to take into account the 96-bit prefix of an address when it forwards a packet destined to a node inside the domain itself.

4.3 Topology Information Propagation Protocol (TIPP)

A strict provider-rooted hierarchical addressing scheme describes how addresses should be allocated, but does not tell how the results of address allocation are distributed from providers to customers. We design TIPP, a network protocol that distributes address allocation information as well as topology information. TIPP is an inter-domain protocol that runs between border routers of domains. It operates outside the *Core* of the Internet. TIPP also helps routers to establish forwarding entries.

When border routers of a domain receive address allocation information and topology information from TIPP, they can use domain-local mechanisms such as router advertisement [32, 77] to distribute information learned from TIPP to end users. This dissertation does not address the design of these local mechanisms.

For clarity, we abstract each domain as having one router and the possible multiple links connecting two domains as one domain-level link to describe our protocol TIPP. In Appendix 5.B of Chapter 5, we describe the situation when a domain has multiple routers, and two domains are connected by multiple links. When we describe the operations of the router in a domain, we sometimes use the domain to refer to the router as the meaning will be clear from context. For example, when we say "domain A establishes a TIPP connection with domain B", we actually refer to "a router in domain A establishes a TIPP connection with a router in domain B. We sometimes also use the same upper case letter to refer to both a domain and the router in the domain.

We first give a high-level overview of TIPP. Then we discuss the design choices we made. We present the detailed design description of TIPP in Appendix 4.B.

4.3.1 Overview

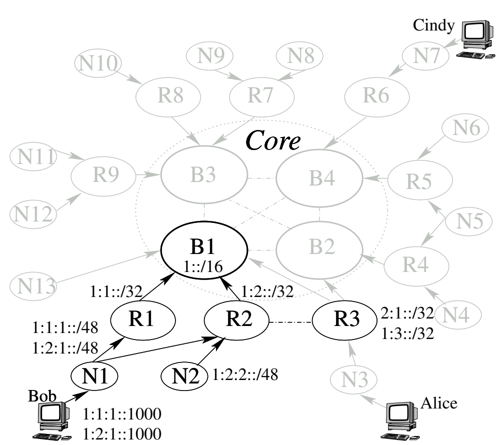


Figure 4-7: What Bob learns from TIPP is shown in black. TIPP propagates to users address allocation information and relevant topology information.

We start the description of TIPP with an example. Figure 4-7 shows what a user, Bob, learns from TIPP. Bob learns the address prefixes allocated to his domain N_1, his up-graph, and the address prefixes allocated to providers on his up-graph. If Bob wants to reach a destination address

d, he could use any algorithm to find a route to a provider that has an address prefix enclosing d, or to the *Core*, because by default, all global address prefixes are connected at the *Core*. For example, if Bob wants to reach one of Alice's addresses (Figure 4-3), 2:1:1::2000, Bob could either use the route, $N_1 \rightarrow R_2 \rightarrow R_3$, to reach R_3, which has a prefix 2:1::/32, or the route, $N_1 \rightarrow R_1 \rightarrow B_1$, to reach the *Core*.

When the domains B_1 and R_1 establish a TIPP connection, R_1 will send a message that requests an address prefix from B_1. Since B_1 is R_1's provider, its policy will allow it to allocate an address prefix 1:1::/32 from its address space 1::/16 to R_1. When R_1 receives this message, it invokes an address allocation process, allocating subdivisions of 1:1::/32 to its customer, and sends the allocation results to its customers. R_1 will also send a message to its customers, telling them its connection to B_1 is up. Such a message will include the address prefixes B_1 owns and R_1 owns. A user could assemble his up-graph from these messages. If R_1 stops using B_1 as a provider, R_1 will send prefix withdrawal messages to its customers. If the connection status between B_1 and R_1 changes, R_1 will also send messages to inform its customers of the status changes. So the customers will know whether they can use R_1 to reach B_1.

4.3.2 Protocol Organization

We apply modularization again in the design of TIPP. We make TIPP deal with the distribution of address allocation information and topology information separately, as the two types of information are relatively independent. Even if the addressing scheme is changed in the future, topology information can still be useful. So TIPP uses separate protocol messages for address allocation and topology distribution so as to minimize the coupling between the two functions.

We adopt BGP's signaling approach to establish TIPP connections. A TIPP connection between two border routers is established on top of a reliable transport protocol, so that TIPP itself does not need to include mechanisms for in-order and reliable message delivery within a TIPP connection.

4.3.3 Address Allocation

The part of TIPP that propagates address allocation information is straightforward: if a domain receives an address prefix from its provider, it allocates a subdivision of the prefix to a customer and sends a message to distribute the allocation result to the customer. What subdivision to allocate to a customer is a domain-local decision. It is also up to a domain to decide whether the decision making process should be automated, or require operator intervention.

One design issue we face is the degree of persistence of a hierarchically allocated address prefix. On one hand, when the router in a domain cannot connect to the router in a provider domain due to temporary failures, or the router crashes and reboots, the domain should not lose its prefixes obtained from its provider, especially when the domain has allocated the subdivisions of the prefixes to its customers. So a prefix should not be “soft” state.

On the other hand, when a customer terminates its service contract with a provider, the provider would like to withdraw address prefixes allocated to the customer. In turn, the customer would have to withdraw all subdivisions of the prefix allocated to its customers, and so on.

Reliably withdrawing an address prefix and all its subdivisions can be difficult. In case of failures or some unexpected operating conditions, an explicit withdrawal message may fail to reach every node that is allocated a subdivision of the address prefix. Therefore, a prefix should not be “hard” state.

To address this dilemma, we make an allocated prefix “leased” state and store it in non-volatile storage, as done in the Dynamic Host Configuration Protocol (DHCP) [40, 39]. “Leased” state is similar to “soft” state in that it can be removed by timeouts [61]. However, “leased” state survives router crashes and connection breakdowns. When a provider allocates an address prefix to a customer, it specifies a leasing period. When the customer allocates subdivisions of the prefix, it specifies the sublet time of each subdivision. When the lease of an address prefix expires, the address prefix is considered being withdrawn by the provider. If a provider decides to withdraw an address prefix, it will send out an explicit withdrawal message and at the same time, it will stop renewing the lease for the prefix. After the lease expires, the provider could reallocate the prefix to other customers.

In our design, a lease specifies the length of a time period. Each router computes the lease expiration time using its local time plus the lease length. We do not require synchronized clocks, i.e., the local time of a provider and that of a customer could be different, but we do assume that the clocks of different routers tick at roughly the same speed, so that a provider’s lease and a customer’s lease expire roughly at the same time point. We believe this is a valid assumption, because lease-based protocols such as DHCP are widely deployed in practice. To be conservative, a provider could wait for a period before it reallocates an expired prefix to other customers in case a customer’s clock ticks slightly slower. If a prefix lease allocated from a provider to a domain is on the order of days, or months, we expect that a waiting time on the order of an hour is sufficient to compensate the clock speed difference.

We expect that when a provider and a customer negotiate a service contract, they will specify the maximum prefix leasing period in the contract. If the connection between the provider and the customer is down for longer than the maximum leasing period, the provider has the right to withdraw a prefix allocated to the customer. A down-time is sometimes caused by failures instead of intentional service termination. Therefore, the leasing period associated with a prefix should be longer than most failure repair time. Measurement results [43, 72] show that most failures last on the order of minutes. So we expect that a lease period of a few days would be longer than what most failures last.

4.3.4 Topology Distribution

We have described how TIPP propagates address allocation information. Another task of TIPP is to distribute topology information. There are at least two general ways to distribute topology for route discovery. One way is to employ a path-vector like protocol. A domain prepends its own identifier to the paths heard from its neighbors to reach a destination, and distributes the new paths to its neighbors. Another way is to distribute link state. A domain sends the information about its connections to its neighbors over the network. Network topology could be assembled from all link-state messages a domain receives.

We design TIPP to use a policy-based link-state method for topology distribution. This method has the disadvantage that a user needs to use a graph search algorithm to find a path to reach a destination after he collects the link-state messages and assembles the network topology. The method to distribute all paths does not require this computation. Moreover, a link-state protocol is usually more difficult to implement because a link-state message needs to be sent reliably over the network, while a path message only needs to be sent to a neighbor reliably.

We choose the link-state method because it is more scalable than the path-vector method. For general graphs, the total number of paths could be much larger than the number of links. For example, a clique[3] of n nodes has $n(n-1)/2$ links, but there are 2^{n-2} paths connecting any two nodes. A link-state approach only needs to keep state for $n(n-1)/2$ links, but the path-vector approach needs to keep state for 2^{n-2} paths. Although this situation may not be the case for today's Internet (i.e., the number of policy-allowed paths might not be much larger than the number of domain-level links), we do not want the scalability of TIPP's topology distribution mechanism to depend on the structure of the Internet, as the structure may evolve over time.

After choosing a link-state method for topology distribution, we face two additional design issues: policy routing support and topology update algorithm. We discuss them in turn.

Policy Routing Support

The transit policy of a domain N for its neighbor M specifies the following: for packets coming from M to N, to what other neighbors of N these packets can be sent, and under what conditions. A user is supposed to use topology information learned from TIPP to select routes. Without knowing a domain's transit policy, an end user may choose a physically connected but policy invalid route. We think it is necessary to provide policy routing support in TIPP.

To support policy routing, we design TIPP such that a domain can control what topology information to propagate to its neighbors based on its transit policies. TIPP uses a link record as the unit for topology representation, rather than using an adjacency-list as in OSPF [75] and IS-IS [79]. A link record describes the attributes of a domain-level link, and a domain originates a link record per neighbor. Using link records, a domain has the flexibility to control what to dis-

[3] A clique is a graph in which every pair of nodes are connected by a link.

tribute to a neighbor at the granularity of a link record. We recommend the following control: for the purpose of information hiding, a domain may only send a link record about a private peering connection to its customers, not to its providers and other peers; for the purpose of scope enforcement, a domain can choose not to propagate anything heard from a customer to its neighbors so that a link record will only be sent downwards a provider tree, i.e., from a provider to a customer.

In our design, if a domain transits packets between a neighbor M_1 and a neighbor M_2, it explicitly specifies so in its link records using the set of reachable address prefixes from M_1 to M_2 and vice versa. Note that transit policies can be implicitly inferred in BGP. From a route advertisement for a prefix pf with the AS path $(...M_1, N, M_2...)$, one can infer that N provides transit service from M_1 to M_2 for packets destined to the address space pf. So from this aspect, TIPP does not require a domain to reveal more policy related information than BGP. Details about policy specification are described in Appendix 4.B.5.

If a domain does not provide transit service between a neighbor M_1 and a neighbor M_2, then a domain does not have to specify the policy. It can simply hide its connection with M_1 from M_2 and vice versa. Again, this mechanism provides a similar information hiding ability as in BGP: if a domain does not provide transit service between M_1 and M_2, it will not export any route learned from M_1 to M_2, and vice versa.

A domain can specify its policy in TIPP, but it cannot ensure that a user will conform to its policy. The same problem exists in today's Internet. A BGP router can hide route advertisements from a rogue peer, but cannot prevent the peer from dumping traffic to it. So a router should use policy checking at forwarding time to enforce its transit policy. We discuss policy checking in Chapter 5.

Choice of Topology Update Algorithm

When TIPP converges, the link records each domain has should be consistent with the status of the network. It is difficult to keep the consistency of topology information in a dynamic network, because link records may be reordered or lost as they travel along different paths to reach different routers.

Two common approaches to deal with the problem are timestamps and sequence numbers. Timestamps require that clocks are globally synchronized so that routers can perform sanity checking. If a router sends a message with a corrupted timestamps, for instance, with a time far in the future, a router could discard such messages instead of keeping them until the future time. Due to this synchronization requirement, timestamps are considered less preferable to sequence numbers [81]. Existing protocols such as OSPF and IS-IS use sequence numbers and periodic refreshments to keep the consistency of link-state advertisements. A link-state advertisement with the highest sequence number is the most recent advertisement, and can replace advertisements with smaller sequence numbers. However, there are a few difficulties in adopting these techniques to the inter-domain environment.

First, a domain contains multiple routers and hosts. If a domain is partitioned due to failures, the sequence numbers kept by each part may be inconsistent, and do not reflect the order of events.

Second, if a router reboots and forgets its previous highest sequence number, in an intra-domain protocol such as OSPF, the router can choose a random sequence number to start with. Any router who has seen a higher sequence number will flood the record back to the router. The router can then pick up its previous highest sequence number. However, in the inter-domain environment, due to policy controlled topology propagation, topology information is not flooded across the network. A policy may specify that a domain will only receive topology information from its providers, but never leaks any information to its providers. Thus, if a router in a provider loses its previous sequence number, it may be the case that no other routers will send the sequence number back to it. The provider router may choose an initial sequence number that is less than its previous highest sequence number, thus losing the semantic of sequence numbers.

Our implementation experience shows that the sequence number approach is hard to implement, needs to deal with numerous anomalies, and its correctness is hard to prove. Our final decision is to use a modified version of the Shortest Path Topology Algorithm (SPTA) developed by Spinelli and Gallager [93], as TIPP's topology update algorithm. This algorithm solves the link record consistency problem without using sequence numbers or periodic refreshments. It is simple and proven correct. The key idea of SPTA is that a node only uses the messages about a link l received from its neighbor on the shortest path to reach the link l to update link l's record in its topology database. If messages sent over a single link are reliable and in order, then this update rule ensures that the link record about l in a node's topology database are consistent with the status of link l. The primary change we made to SPTA is to add support for transit policies.

The algorithm has two drawbacks. First, its computational cost for per link record update is linear to the number of links in a graph. TIPP is used to propagates up-graphs. The number of links in an up-graph is likely to be small compared to that of the Internet. Therefore, we think that the computational cost will not become a scaling problem. Second, there are unusual situations where the number of messages sent in the convergence process of the algorithm may grow exponentially with the number of link status change. However, such situations involve multiple independent link failures occurring consecutively in a short time. We think that link failure is small probability event. The event for multiple consecutive failures will rarely happen. If it does happen, in practice, rate limiting techniques could be used to reduce the number of messages sent. For detailed analysis and the correctness proof of the algorithm, please see [93].

4.4 Name-to-Route Lookup Service

With TIPP, a user learns his addresses and his up-graph. To reach a destination, a user shall at minimum know about the destination's addresses. Inspired by the success of the Domain Name System (DNS) [74], we design the Name-to-Route Lookup Service (NRLS) to be a distributed hierarchical directory service, which is essentially an enhanced DNS. A server will store its addresses and optionally topology information such as the address allocation paths learned from TIPP at its NRLS servers.

The bootstrapping mechanism and the lookup process of NRLS are similar to those of DNS. A resolver is hard-coded with the addresses of the root servers. A user is hard-coded with the addresses of his resolvers. At each level of the resolution, the addresses of the NRLS servers that are in charge of a lower-level namespace is returned. The lookup process stops when the addresses regarding the queried name is returned.

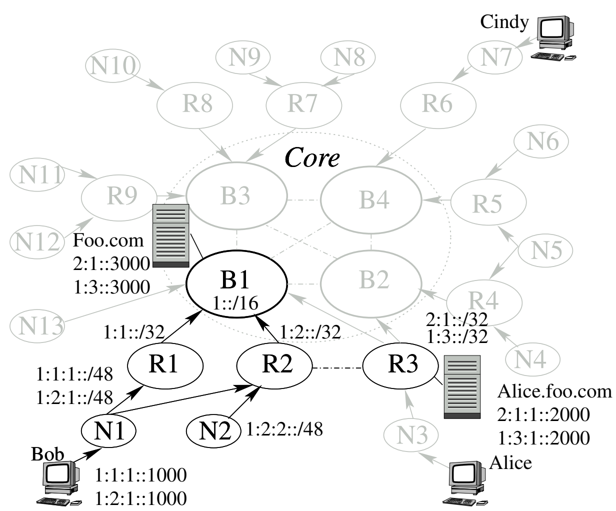


Figure 4-8: When Bob wants to communicate with Alice, Bob will query the Name-to-Route Lookup Service to retrieve Alice's addresses, 2:1:1::2000 and 1:3:1::2000.

Figure 4-8 shows an example. When Bob wants to communicate with Alice, Bob will query the NRLS servers to get Alice's addresses. If Bob does not have the addresses of Alice's NRLS server, e.g., *foo.com*, Bob will send queries to the root NRLS servers to first get addresses of the server responsible for names in the zone *foo.com*. Then Bob will send queries to that server to get Alice's addresses. With his up-graph learned from TIPP and Alice's addresses, Bob is able to send an initial packet to Alice. Optionally, Bob and Alice may exchange subsequent packets to negotiate a different route for communication.

4.4.1 Record Updates

What makes NRLS different from DNS is that it stores the mapping between a name to provider-rooted addresses, which are topology-dependent addresses. When a domain changes its providers, its addresses will change. It is possible that a single topology change may affect the addresses of a large group of users. For instance, when a second-level provider changes its top-level providers, the customers of the second-level provider would all change their addresses. In Figure 4-8, if R_2 stops using B_1 as a provider, and purchases service from B_2, then both N_1 and N_2's addresses will change. The user Bob's address 1:2:1::1000 becomes invalid, and he'll have a new address allocated from B_2's address space.

We recognize that route record update is another modularizable component in NIRA. TIPP propagates address changes to users. We leave it to users to decide how to update their records, and do not architecturally constrain how updates should be done. A user and his NRLS provider could develop any general and ingenious mechanisms to update the user's route records. Stale information of a user may make him unreachable, but does not affect the reachability of other users.

The coupling between topology and address is the logical consequence of any topology-dependent addressing scheme [103, 64], including the strict provider-rooted hierarchical addressing scheme. However, we do not think this coupling effect would cause significant problems for two reasons.

First, the static Internet topology changes at quite a low frequency. According to the study conducted by Chen et al [26], the AS birth and death rate of the Internet is less than 15 per day, the AS-level link birth and death rate is less than 50 per day. Only those changes would affect a user's addresses or route segments. Their study also shows that most of the new or dead ASes are of degree 1 or 2, thus probably being edge ASes. Hence, the changes are likely to affect just users in those domains. Second, static topology changes are caused by the changes in business relationships and will happen in a controlled manner. Network administrators and users could deploy creative scheduling algorithms to reduce the service disruption and route server update overhead. A grace period may be granted before a provider cuts off its service completely so that a user has sufficient time to update its route server records. Randomized algorithms may be used to prevent users from simultaneously update their route server records, and avoid overloading the network and route servers.

4.4.2 Locations of Root NRLS Servers

When the topology of the Internet changes, the addresses of the root NRLS servers may change. However, those addresses are usually hard-coded into every resolver. It can be burdensome to update these hard-coded addresses. For this reason, we require that the root NRLS servers reside in the *Core*, as the addresses of servers in the *Core* are resistant to topology changes. With its up-graph learned from TIPP, a resolver can always find a route to reach the *Core*, thus reaching a root server.

4.5 Route Availability Discovery

We have discussed how a user might discover route information with TIPP and NRLS. This information might not tell a user whether the dynamic attributes of a route satisfies his requirement. In NIRA, we provide additional architectural support for users to discover such information. The basic mechanism we provide is a combination of proactive and reactive notification.

For the part of proactive notification, TIPP messages may include the dynamic status of domain-level interconnections. A user is proactively notified of the dynamic state concerning providers in his up-graph via these TIPP messages. We let an individual domain decide how to distribute TIPP messages to users inside itself. To prevent a user from receiving a large number of TIPP messages, a domain may use standard rate limiting techniques to suppress excessive TIPP messages, or only propagate TIPP messages regarding the changes of a particular dynamic attribute, such as link status. Alternatively, a domain may propagate TIPP messages only to routers inside itself, and a user could retrieve address and topology information from a close-by router at the time of communication.

As TIPP messages do not propagate globally, a user in general does not know the dynamic status of the addresses of other users. When a user wants to communicate with a destination, it is possible that he chooses an address of the destination user that is unreachable. So a user might also discover the route availability via reactive notification. In our design, if a router detects that a route specified in a packet header is unavailable, the router should try its best to send a control message to inform the original sender. Such a control message may include reasons why the route is unusable. When a user receives such a reactive notification, as he knows his addresses, his up-graph, and the addresses of the destination user, he could switch to an alternative route on the order of a round trip time. In this case, NIRA enables fast route fail-over. In cases where a router is unable to send a failure notification, e.g., a router is overloaded, users shall use timeout to detect route failures. The fail-over time then depends on the timeout parameters.

The combination of proactive and reactive notification reduces the amount of dynamic routing information that a user needs to maintain. However, reactive notification may increase the connection setup time when user selected routes suffer from failures. Note that failures that trigger reactive notification are those that result in inter-domain disconnections. For intra-domain failures, routers should always use local repair mechanisms to send packets over alternative paths for rapid fail-over. For inter-domain failures, since a user has expressed his domain-level route choices in a packet header, and an intermediate router does not know the user's route preference, (because it is probably related to a user's financial considerations), it is best for the user to decide on an alternative route. So in NIRA, we prefer to use reactive notification to local repair for inter-domain failures. Users cache recently used routes and avoid using unavailable routes for a new connection. We expect that the amortized set up time will be reduced with such caching.

Since we have modularized route availability discovery as an individual architectural component in NIRA, users or service providers can use additional mechanisms to discover route availability besides using the basic mechanisms provided by NIRA. For instance, a local provider may offer a route monitoring service to its users. The provider can run a server that actively probes the dynamic attributes of popular routes and provides timely information on route conditions to its customers. In addition, popular servers may choose to include dynamic topology information in their NRLS records and update the information at a rate they could afford. When a user retrieves their NRLS records, they may already have the information on which addresses of the servers are unreachable, thus saving the connection setup time.

4.6 Discussion

This chapter describes the basic mechanisms in NIRA for route discovery and route availability discovery. Before we proceed to describe how to represent a route in a packet header in the next chapter, we discuss their limitations.

The basic mechanisms for route discovery, including provider-rooted hierarchical addressing, TIPP, and NRLS, do not guarantee that a user will discover all possible routes. For example, to avoid flooding link records globally, a domain does not propagate link records heard from a customer to other neighbors. If a domain D, for some reason, provides transit service between its providers P_1 and P_2, then there is an additional route connecting any source and destination pair: source $\rightarrow \ldots \rightarrow P_1 \rightarrow D \rightarrow P_2 \ldots \rightarrow$ destination. However, a user may not receive the link records (D, P_i) or (P_i, D) for $i = 1, 2$, and does not know the transit policy. Since we have made route discovery a separate module, domain D could use any other mechanism, such as an advertisement on its web page, to make such routes known to users.

A provider-rooted hierarchical addressing scheme reveals business relationships between domains to some extent. If domain A allocates a subdivision of its address space to domain B, one can conclude that A provides transit service for B, and therefore, A is B's provider. However, the detailed business agreements such as service price are not revealed. We do not expect this revelation will constitute a deployment problem, because even today, business relationships between domains can be inferred to some extent by analyzing BGP announcements [54, 99].

Appendix 4.A Proof of Address to Path Mapping Property

We formally prove the **Address to Path Mapping Property** of a strict provider-rooted hierarchical addressing scheme.

- **Address to Path Mapping Property:** If a domain M has a prefix p, and a domain N has a prefix p', and the prefix p' is within the address space p, then there exists a unique loop-free path $M... \rightarrow ... \rightarrow P_{i-1} \rightarrow P_i \rightarrow P_{i+1} \rightarrow ... \rightarrow N$ that consists solely of provider-customer connections. Each domain P_i along the path from M to N has an address prefix p_i such that p_i is within the address space p, and p' is within the address space p_i.

Proof: Since p' is a subdivision of p, p' cannot be a top-level provider's prefix. Thus, N must have obtained p' from a provider P_k's address space p_k. (We use an upper case letter to represent a domain, and subscripts to differentiate domains.) Similarly, if P_k is not a top-level provider, P_k must have obtained p_k from a provider P_{k-1}'s address space p_{k-1}. Therefore, there must exist a path $P_0 \rightarrow P_1 \rightarrow ... \rightarrow P_i \rightarrow P_{i+1}... \rightarrow N$, where P_0 is a top-level provider, and P_i is the provider of P_{i+1} and has allocated a subdivision of its prefix p_i: p_{i+1}, to P_{i+1}. We call such path the allocation path of N's prefix p'. This path must contain no loops as indicated by the non-looping allocation rule. M must be equal to P_i for some i, because otherwise the non-overlapping address allocation rule is violated. This concludes the proof.

Appendix 4.B TIPP Specification

This section describes TIPP in detail. TIPP runs between border routers of domains outside the *Core*. A border router that runs TIPP is called a "TIPP router."

4.B.1 TIPP State and Logical Data Structures

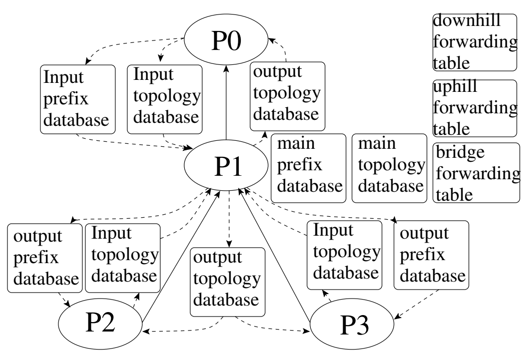


Figure 4-9: The logical data structures the router in P_1 keeps to maintain address prefixes, topology information, and forwarding state.

A TIPP router maintains three types of TIPP state: address prefixes, topology information, and forwarding entries. Figure 4-9 illustrates the logical data structures a TIPP router P_1 keeps to maintain TIPP state. P_0 is its provider; P_2 and P_3 are its customers. For each of its neighbors that allocate address prefixes to it, for example, its provider P_0, P_1 keeps an input prefix database that stores the address prefixes allocated from the neighbor. Similarly, for each of its neighbors to whom it allocates address prefixes, say its customers P_2 and P_3, P_1 keeps an output prefix database that stores prefixes allocated to the neighbor. P_1 also keeps a main prefix database that stores all address prefixes it has.

For each of its neighbors, P_1 keeps an input topology database. P_1 also keeps a main topology database that stores the topology information collected from its input topology databases. A domain categorizes its neighbors into several classes based on its policy configurations. The router in a domain keeps an output topology database for each neighbor class, rather than each neighbor. The classification saves memory and reduces the computation required for topology database update. The two common classes are the provider class, which is for a domain's providers and peers, and the customer class, which is for a domain's customers. As shown in Figure 4-9, P_1 has an output topology database for P_0, and one for P_2 and P_3.

Input databases, main databases, and output databases are logical distinctions. TIPP does not require a router to keep separate copies of the same data items. In its implementation, a router may use pointers instead of copying the same data items to save storage space.

TIPP also helps a router to establish forwarding state. A router keeps three logical forwarding tables: the uphill forwarding table, the downhill forwarding table, and the bridge forwarding table. Since a user will specify a domain-level route in his packets, a router only needs to keep forwarding entries for address prefixes of its neighbors and its own address prefixes. In Chapter 5, we describe how a router forwards packets using these forwarding tables.

4.B.2 Message Types

There are six types of TIPP messages: **open** message, **keepalive** message, **notification** message, **address request** message, **address** message, and **topology** message. As in BGP, each message has a common header, including the length of the message, the type of the message, and the authentication information. The messages for connection control, the **open**, **keepalive**, **notification** messages, are basically the same as those in BGP [88, 56]. The **open** message opens a TIPP connection and contains connection parameters such as the hold time interval. If a router does not receive any message from its neighbor for a hold time interval, it will declare the connection dead. The **keepalive** message is an empty message that is sent out periodically to ensure the liveness of a connection. The **notification** message sends an error notification to a router's neighbor.

The other three messages are unique to TIPP. The **address request** and **address** message handle address allocation, and the **topology** message handles topology updates.

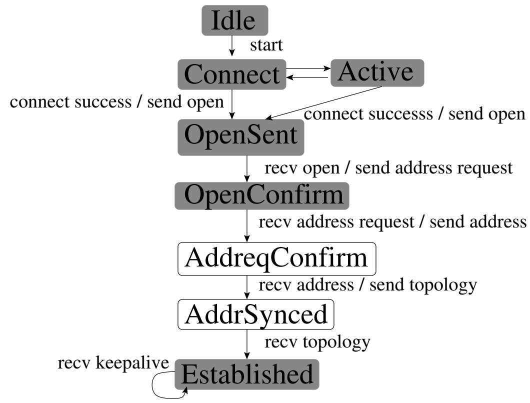


Figure 4-10: TIPP Finite State Machine. Transitions for error events are omitted.

4.B.3 TIPP Finite State Machine

A TIPP router establishes a connection with a neighbor in the same way as a BGP router. It runs a state machine on top of a reliable transport connection, and sends and receives TIPP messages over the transport connection. Figure 4-10 shows the basic TIPP finite state machine. Transitions for error events are omitted for clarity. The shaded states are the same ones as those in BGP. Two additional states ”AddreqConfirm” and ”AddrSynced” are added into TIPP for synchronizing the address prefix databases and topology databases between a router and its neighbor.

Initially, a TIPP finite state machine is in Idle state. A start event changes the state from Idle to Connect. In the Connect state, a TIPP router waits for its transport connection to succeed. If the connection fails, it transfers into the Active state, where it retries its transport connection. When a transport connection succeeds, a TIPP router sends an **open** message to its neighbors.

At the OpenSent state, when receiving an **open** message from its neighbor, instead of sending a **keepalive** message as in BGP, a TIPP router will send an **address request** message to its neighbor. The router then transfers to the OpenConfirm state and awaits an **address request** message from its neighbor.

When the **address request** message from its neighbor arrives at the router, it replies with an **address** message to the neighbor, and transfers to AddreqConfirm state. The address message contains the address prefixes a router allocated to its neighbor.

At the AddreqConfirm state, when a router receives an **address** message from its neighbor, the router synchronizes its input prefix database with the neighbor’s output address database using the contents of the message. It then sends a **topology** message to the neighbor, and transfers to the AddrSynced state. The **topology** message contains the contents of its output topology database for the neighbor.

At the AddrSynced state, when a router receives a **topology** message from its neighbor, it synchronizes its input topology database. At this point, a router’s input databases are synchronized with its neighbor's output databases. The connection is fully established, and the router will transfer to the Established state. A router may send out topology messages to its other neighbors, telling them that the connection between itself and the neighbor is up.

As with BGP, a router periodically sends **keepalive** messages, and closes a connection if it does not receive any message from its neighbor for a hold time interval. During the connection tear-down, a router clears its input topology database with the neighbor, but will not clear its input prefix database.

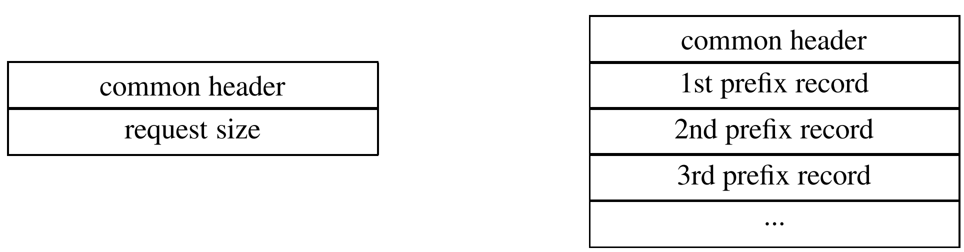
4.B.4 Address Allocation

A hierarchically allocated prefix is a "leased" state, and is stored in a router's prefix databases in non-volatile storage. A record in a router's prefix database is a (prefix, attributes) pair. As the set of attributes may change over time, we use a TLV (type, length, value) [81] triplet to represent an attribute. If a router does not understand the type of an attribute, it can ignore the number of bytes specified by the length field. In our design, the attributes of a prefix include three fields: timestamp, lease period, and allocation path.

The timestamp is a router's local time. Global clock synchronization is not required. For an input database and the main prefix database, the timestamp is the time when the router receives the address message that allocates or renews the prefix. For an output database, the timestamp is the time when the router allocates or renews the prefix to its neighbor. The allocation path specifies the sequence of identifiers of domains that allocates the prefix, followed by the identifier of the domain that leases the prefix.

Address Request and Address Message

The **address request** and the **address** message update a router's prefix databases. An **address request** message contains the size of the address block a router asks for from its neighbor. The **address** message contains a list of prefix records. A prefix record is a tuple with three fields: (opcode, prefix, attributes). Figure 4-11 shows the contents of an **address request** message and an **address** message.



(a) **address request** message (b) **address** message

Figure 4-11: Contents of an **address request** message and an **address** message.

Address Request Message Processing

An **address request** message is sent when a TIPP connection is being established, or an operator decides to request a different size of address prefix from a provider. If a router does not request any address from its neighbor, it sets the request size to zero.

When a router receives an **address request** message from a neighbor, it first checks whether its policy grants such request. If not, it replies with an empty **address** message. Otherwise, it replies with an **address** message that contains prefix records. If the request size is the same as that in a previous request, the address message contains the prefixes allocated to the neighbor. For each prefix, the lease period in the message is set to the remaining lease time. If the size has changed, the router may withdraw previously allocated prefixes and allocates new prefixes to the neighbor.

Pseudocode 1 shows the pseudocode for processing an **address request** message. Note that each domain makes up its own address allocation procedure, i.e., the allocate(M, *req.size*) procedure at line 10 of the pseudocode. TIPP does not standardize it. The process could either be automated or require operator attention.

Pseudocode 1 : **Address request** message processing

```
    N: the TIPP router
    M: N's neighbor
    OPDB_M: the output prefix database for M
    prevSize: the size of a previous request from M
    addrMsg: an address message template

 1: On receiving an address request message req from M
 2: if N's does not allocate req.size to M then
 3:     send empty addrMsg to M;
 4: else
 5:     if req.size ≠ prevSize then
 6:         foreach prefix in OPDB_M do
 7:             add (withdraw, prefix) into addrMsg
 8:         end for
 9:         OPDB_M.clear()
10:         allocate(M, req.size)
11:     end if
12:     foreach prefix in OPDB_M do
13:         lease = prefix.timestamp + prefix.lease − now
14:         path = prefix.allocPath;
15:         push (add, prefix, leaseT, path) into addrMsg
16:     end for
17:     send addrMsg to M
18: end if
```

Address Message Processing

Figure 4-12 shows the steps in processing a non-empty **address** message. When a router receives an **address** message from a neighbor, it first updates its input prefix database if its policy allows. For each prefix record in an **address** message, if the opcode is "add", and the prefix is not in the input database, it will be added into the database. If the opcode is "add", and a prefix is already in the input database, the prefix is renewed. Its timestamp is set to the message receiving time, and the lease time is set to that in the **address** message. Similarly, if the opcode is "withdraw", the prefix will be deleted from the input database.

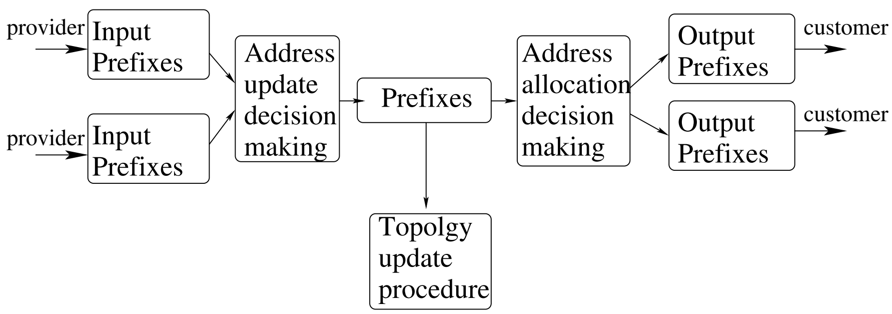


Figure 4-12: When a router receives a non-empty **address** message from a neighbor, it first updates its input prefix database. If its policy allows, it will update its main prefix database, allocate, withdraw, or renew subdivisions of its prefixes, update its output prefix databases to its customers, and send out messages to notify the customers of the update. The router will also invoke the topology update procedure to update its topology databases and its forwarding tables.

If there is any change in the input database, the main prefix database of the router is updated correspondingly. If a prefix is added, renewed, or withdrawn from its main prefix database, a router may update its output prefix databases to allocate, renew, or withdraw the subdivisions of the prefix allocated to its customers and itself. The router will send **address** messages containing the changes to its neighbors.

When a router's main prefix database has a new addition or a withdrawal, the hierarchical addresses of the router will change. As we will explain later, the router will also need to update its topology database and its forwarding tables. Pseudocode 2 shows the pseudocode for processing an **address** message.

Prefix Lease Management

Each router will have a cron job that periodically checks lease expiration for prefixes allocated to them, and prefixes allocated by them. If the cron job finds that a prefix in a router's input prefix database is expired, it will withdraw the prefix as if the router has received an address message that withdraws the prefix. We assume when two domains negotiate a service contract, they also specify a maximum prefix leasing time *maxLease*. The *maxLease* for a subdivision of a prefix

Pseudocode 2 : **Address** message processing

```
IPDB_M: input prefix database for a neighbor M
OPDB_M: output prefix database for a neighbor M
MPDB: main prefix database of the router

on receiving an address message addr from M
foreach record in addr do
    if record.opcode == add then
        if record.prefix ∉ IPDB_M then
            add record.prefix into IPDB_M
        else
            prefix = IPDB_M.find(record.prefix)
            prefix.timestamp = now;
            prefix.leasetime = record.leasetime
        end if
    else if record.opcode == withdraw then
        withdraw record.prefix from IPDB_M
    end if
end for
if any change in IPDB_M then
    update MPDB
end if
if any change in MPDB then
    foreach customer C do
        update OPDB_C
        send address message to C if needed
    end for
end if
if MPDB has new additions or withdrawals then
    update self addresses
    update topology databases
    update forwarding tables
end if
```

should be no more than that of the prefix. If the cron job finds that a prefix subdivision allocated to a customer has a remaining lease time less than $C * maxLease$, where C is a domain specific fraction, and the prefix in its input database has a longer lease, it will renew the lease for the subdivision. For top-level providers, their global unique prefixes have an infinite leasing period. So they can always renew the leases for prefixes allocated to their customers.

When a router receives an address message that renews a prefix, it will also check leases for the subdivisions of the prefix. If the remaining time of any subdivision is less than $C * maxLease$, the router will also renew the prefix subdivision. Pseudocode 3 shows the pseudocode for prefix lease maintenance.

Pseudocode 3 : Periodic prefix lease management

N: the TIPP router
$IPDB_M$: input prefix database for a neighbor M
$OPDB_M$: output prefix database for a neighbor M
$MPDB$: main prefix database of N
$maxLease(N, M)$: max lease time from N to M

```
foreach neighbor M do
    foreach rec in IPDM_M do
        if rec.timestamp + rec.lease < now then
            withdraw rec.prefix as if receiving an address message
        end if
    end for
end for
foreach neighbor M do
    foreach rec in OPDB_M do
        r = rec.timestamp + rec.lease − now
        if r < C ∗ maxLease(N, M) then
            pRec = MPDB.findParent(rec.prefix)
            pr = pRec.timestamp + pRec.lease − now
            if pr > r then
                rec.timestamp = now
                rec.lease = min(maxLease(N, M), pr);
            end if
        end if
    end for
end for
send address messages to notify lease changes.
```

4.B.5 Topology Distribution

Topology information is represented by a set of link records, and are stored in topology databases. A link record is identified by an originator domain identifier and a neighbor domain identifier and specifies the attributes of the domain-level link between the two domains. Due to policy configurations, for a domain-level link, a router may not receive the link records originated by both ends of the link. Therefore, we make a link record contain attributes for both directions of the link so that a router will know the link attributes for its returning packets. A link record itself is unidirectional. A record (N, M) is originated by the router in domain N, but contains attributes for two unidirectional links: link $N \rightarrow M$ and link $M \rightarrow N$.

The link record attributes should include physical connectivity status, transit policy, and optionally other attributes such as delay, bandwidth, jitter, and monetary cost. A link record attribute is encoded as a (type, length, value) triplet [81]. So the set of attributes is easily extensible.

Transit Policy Specification

Since we assume a provider-rooted hierarchical addressing scheme, we can use an address prefix to succinctly specify a domain and all its customers. For example, in Figure 4-3, R_2 has a peering relationship with R_3, i.e., R_2 provides transit service between its customers and R_3. R_2 can use its address prefix, 1:2::/32, to represent itself and its customers reachable from the peering link: N_1 and N_2.

In the transit policy specification of a link record, the reachable neighbors for each direction are represented in two sets. One set is the reachable addresses within a domain's own address space, the internal addresses, and the other set is the reachable addresses outside a domain's own address space, the external addresses. Each set contains a list of prefix records. The internal addresses of a neighbor will also be used to establish a domain's forwarding tables. A set containing a wildcard "*" represents all external addresses of a domain's neighbors.

A prefix record within a link record (N, M) has the format (allocation-relation, prefix, condition). The allocation-relation indicates whether the prefix is allocated along the link from N to M (value 1), or vice versa (value 0), or the prefix is not allocated from any of N's providers (value 2), i.e., either N is a top-level provider and the prefix is the global unique prefix N owns, or the prefix is the non-provider-rooted prefix of N. This field tells whether the link represented by the link record is on an address allocation path, and is useful in route selection, route representation, and forwarding table establishment.

For example, as shown in Figure 4-13, if N and M have a peering relationship, M may have an address prefix pf that is within N's address space, but N does not allocate the prefix to M. Instead, N allocates a prefix that includes pf's address space to a customer C; C is the provider of M, and allocates pf to M. So an address in the address space of pf maps to the route fragment $N \rightarrow C \rightarrow M$, not $N \rightarrow M$.

The prefix field in a prefix record specifies the reachable address space, and the condition field specifies under what conditions packets can be forwarded to the reachable prefix, and could be empty if a domain has no specific restrictions. The conditions should be things a domain can

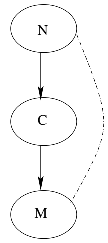


Figure 4-13: M has an address prefix pf that is within N's address space, but is not allocated from N. The allocation-relation bit in a prefix record is used to clarify this.

check, such as a packet header matching rule, or a time slot.

Originator ID:	R_1
Neighbor ID:	B_1
Status:	up
Internal reachable ($R_1 \rightarrow B_1$):	(2, 1::/16)
	(2, $nprPf(B_1)$)
External reachable ($R_1 \rightarrow B_1$):	*
Internal reachable ($B_1 \rightarrow R_1$):	(1, 1:1::/32)
	(2, $nprPf(R_1)$)
External reachable ($B_1 \rightarrow R_1$):	ϵ
Originator ID:	R_2
Neighbor ID:	R_3
Status:	up
Internal reachable ($R_2 \rightarrow R_3$):	(0, 1:3::/32)
	(0, 2:1::/32)
	(2, $nprPf(R_3)$)
External reachable ($R_2 \rightarrow R_3$):	ϵ
Internal reachable ($R_3 \rightarrow R_2$):	(0, 1:2::/32)
	(2, $nprPf(R_2)$)
External reachable ($R_3 \rightarrow R_2$):	ϵ

Figure 4-14: The contents of the link record (R_1, B_1) and (R_2, R_3). The network topology is shown in Figure 4-3.

Figure 4-14 shows the contents of the link record (R_1, B_1) and (R_2, R_3). The network topology is shown in Figure 4-3. The "Internal reachable" field for each unidirectional link specifies the internal address space of the end domain. The "External reachable" field specifies the reachable non-customer neighbors of the end domain. We use the symbol $nprPf(N)$ to represent the non-provider-rooted address prefix of a domain N. In this example, R_1 is a customer of B_1. So a packet from R_1 to B_1 can be sent to any of B_1's neighbors. This policy is specified by the wildcard "*" in the field "external reachable ($R_1 \rightarrow B_1$)". On the reverse direction, a packet from B_1 to R_1 is only allowed to reach the address space allocated from B_1 to R_1: 1:1::/32, and nodes inside R_1: $nprPf(R_1)$, so the external reachable set is empty (denoted by the empty set symbol ϵ). Similarly,

R_2 and R_3 have a peering relationship. So the external reachable addresses for both directions are empty.

Policy-based Topology Information Propagation

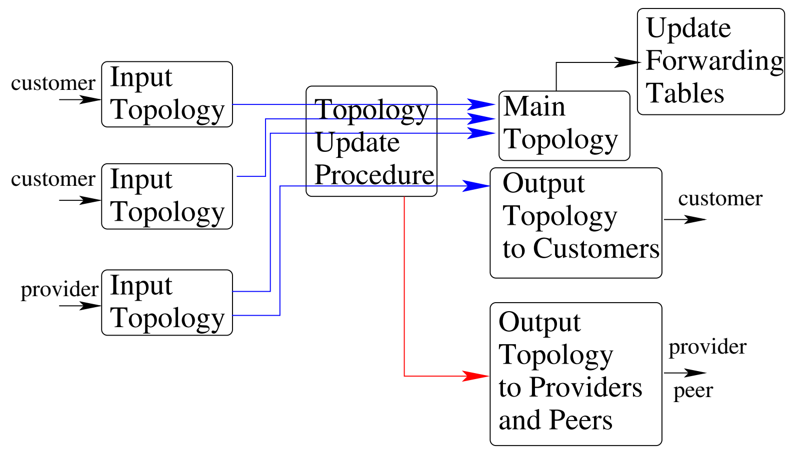


Figure 4-15: When a router receives a **topology** message from a neighbor, it first updates the input topology database for the neighbor. If the input topology database is connected to the output topology database of a neighbor class, the router will invoke the topology update procedure and send changes to the corresponding neighbors. The router may also update its forwarding tables.

Figure 4-15 shows the steps in processing a **topology** message. A router updates its input topology database for a neighbor M when it receives a **topology** message from M, or when the attributes of its connection to M is changed. A **topology** message consists of a list of link records, as shown in Figure 4-16.

common header
1st link record
2nd link record
3rd link record
...

Figure 4-16: Contents of a **topology** message.

If according to a domain's policy, the router in the domain wants to propagate the topology information learned from M to the class of neighbors an output database OT is intended for, it "connects" the input topology database from M, T_M, to the output database OT. Changes in T_M will then invoke the topology update algorithm (described in Section 4.B.6) on OT, and any changes in OT will be sent to the corresponding neighbor class. All input topology databases are connected to a router's main topology database.

Based on a domain's policy, the router in a domain connects different input databases to different output databases. This policy-based propagation serves two purposes. First, it allows information hiding. Second, it controls the propagation scope of topology information, thus improving scalability. If a router connects any input topology database to an output topology database, TIPP becomes a link-state flooding protocol. A router will see the entire inter-domain topology.

We recommend the following propagation policies. For information hiding, if a domain does not provide transit service between two neighbors, it does not connect the input database from one neighbor to the output database for the other neighbor, and vice versa. To control the propagation scope of a link record, a domain does not connect the input database from a customer to any neighbor.

Combining these two policies, the output database for a router's provider class is empty; the output database for a router's customer class only has information about a router's up-graph. In the topology shown in Figure 4-3, the router in domain N_1 only sees link records of its up-tree: (N_1, R_1), (N_1, R_2), (R_1, B_1), (R_2, B_1), (R_2, R_3), and will not see (R_2, N_2) or (B_1, R_3).

Topology Message Origination

An adjacent link record is a link record representing an adjacent link. During a connection establishment process between a domain N and a neighbor M, at the AddreqConfirm state, after N updates its prefix databases based on the **address** message it receives, it will create a partial link record (N, M), consisting of its transit policy for M, and possibly other one-way attributes it knows of. It sends to M this partial link record along with other records in its output database for M and transfers to the AddrSynced state. When N receives a **topology** message from M, the message must contain a partial link record (M, N). It extracts from this record M's transit policy for itself and possibly other one way attributes from M to N. Combining the partial record (N, M) it has, N creates a link record (N, M) with two-way attributes in M's input topology database T_M, and sets the connection status to be up. It creates other link records in T_M using the other records in the **topology** message. Then, N invokes the topology update algorithm (Section 4.B.6) on a topology database T_M is connected to, originates a **topology** message containing the changes, and sends the message to the corresponding neighbors.

When a domain N's transit policy for a neighbor M changes, for example, N obtains a new address prefix pf, and wants to add pf to the internal reachable set, N sends to M a link record (N, M) to inform M the change, updates the (N, M) record in T_M, invokes the topology update algorithm on an output database connected to T_M, and sends a **topology** message to propagate the changes.

Similarly, when the router in domain N detects a connection failure between a neighbor M, it clears the input database T_M, invokes the topology update algorithm, and originates a **topology** message to send out the changes.

4.B.6 Topology Update Algorithm

Our topology update algorithm is a modified version of the Shortest Path Topology Algorithm (SPTA) [93]. We made changes to SPTA to add support for domain transit policies. The main idea of the update algorithm is described as follows.

A link record is correct if it describes the most updated state of the domain-level link it represents. For the router in a domain N, invoking the topology update algorithm on a topology database OT is to ensure that any link record in OT representing a reachable link from N is correct. An adjacent link record for any neighbor M is always correct because N and M have a direct connection. For a remote link record e representing a link l, $e(l)$, if N can find a failure-free and policy-allowed path to reach l using links represented by the correct link records it has so far, then N believes that the link record $e'(l)$ heard from the neighbor on the shortest path to l is correct. N will use the contents of e' to set e. Then e is assumed to be correct, and can be used to reach other remote link records.

Pseudocode 4 shows the pseudocode for the topology update algorithm. At Line 17, the test on whether $fLink = (A, B)$ can reach an adjacent link $e = (B, C)$ takes into consideration the transit policy specification in the link record (A, B).

The algorithm requires that messages sent between adjacent neighbors are reliable and in order. TIPP connection is established on top of a reliable transport connection, so this requirement will be satisfied.

The correctness of the algorithm can be shown using an inductive proof similar to the original proof for SPTA in [93]. The basic idea of the proof is to use induction to show that in a steady network, within a finite time, the router in a domain N will have a correct link record for a link l that is at distance d. The induction hypothesis is clearly true for an adjacent link ($d = 0$). With the condition that messages between two adjacent routers are sent reliably and in order, it can be shown that the hypothesis is true for $d > 0$.

As an optimization, the topology update algorithm only needs to be invoked when a link record is received from a neighbor that is on the shortest path to the link, because link records received from other neighbors will not change the records in a router's main or output topology databases.

4.B.7 Link Records for Removed Links

In a link-state routing protocol, such as OSPF, when a node detects a connection to a neighbor has failed, the node will originate a link-state advertisement that only includes connections that are alive, and does not include the failed connection. Therefore, a failed connection is automatically

Pseudocode 4 : Topology update algorithm

```
T_M: Input topology database from a neighbor M
e_M: the adjacent link record for M
foreach neighbor M do
    if the connection to M is up then
        e_M.parent = T_M.
        set e_M's attributes using the record in T_M
        push e_M into queue.
    else
        clear e_M's attributes
        set the link status in e_M to be down
    end if
end for
while queue is not empty do
    fLink = queue.pop()
    if fLink is processed before then
        continue
    end if
    foreach adjacent e of fLink's end domain do
        if fLink can reach e && fLink is up then
            e.parent = fLink.parent
            set e's attributes using the record in e.parent
            if e is up then
                push e into queue
            end if
        end if
    end for
end while
send changes to neighbors
```

purged out of the link-state database of a node, regardless of whether the failure is due to a temporary link failure, or due to the permanent removal of a link.

In our protocol TIPP, a TIPP node sends messages concerning each adjacent link's status. So if a link goes down, a record for the link is still present in a node's topology database, with its status marked as down. However, if the link is permanently removed from the network, it is desirable that the record for the removed link is eventually purged out of a node's topology databases. Since removed-link records only take up a node's memory, we leave it as a local implementation issue on how to purge out removed-link records. A node may periodically clean up records with down links in its topology databases, or it could simply delete the link record after it has received a message with the link down information and propagated this information to its neighbors. The node will create a new record for the link when it receives a message with the link up information. Alternatively, a node could do nothing about a link record with status down. If an adjacent link is removed, a node would stop originating a record for the link. Therefore, after the node reboots, it should have no record for the removed link. When the node's neighbors reboot, they would synchronize their databases with the node, and their databases will have no record for the removed link either. Eventually, the removed link record will be purged out of the network after all nodes have rebooted.

4.B.8 Topology Database Refreshment

As in BGP, to increase the robustness of the protocol, adjacent TIPP routers may choose to periodically refresh their databases to avoid memory corruption.

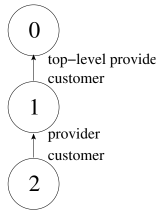
4.B.9 Example

Figure 4-17 shows various TIPP events happened in a simulation run with a three-node topology. Node 0 is the provider of node 1, which is the provider of node 2. The notation $i - j$ means node i's connection to a neighbor j. The first column is the line number, and the second column is the simulated time when a TIPP event happens.

In the simulation, node 1 and node 2 first establish a TIPP connection, and then node 1 acquires node 0 as a provider. At line 4, node 1 receives a prefix allocation message from node 0. It then allocates a subdivision to its customer node 2, and sends an **address** message to node 2 at line 5. It also sends a **topology** message to node 2 to announce its newly acquired prefix at line 6.

At line 13, node 1's connection to node 0 is fully established. It sends a **topology** message at line 15 to inform node 2 that the connection between itself and node 0 is ready to use.

In the simulation, we introduce a failure between node 1 and 0 at simulation time 300 second. At line 18, node 1 detects the failure via timeout and closes the connection. At line 21, it sends a **topology** message to inform node 2 that the connection between node 1 and node 0 is broken. The failure is recovered later, but the events are not shown.




```
 1: ..
 2: ..
 3: 4.833861 1 - 0 FSM change status [OpenConfirm -> AddreqConfirm].
 4: 4.833861 1 - 0 receive ADDRESS size 147
 5: 4.833861 1 - 2 send ADDRESS.
 6: 4.833861 1 - 2 send TOPOLOGY.
 7: 4.833861 1 - 0 FSM event Receive_ADDRESS_message.
 8: 4.833861 1 - 0 send TOPOLOGY.
 9: ...
10:...
11: 24.853879 1 - 0 receive TOPOLOGY size 109
12: 24.853879 1 - 0 FSM event Receive_TOPOLOGY_message.
13: 24.853879 1 - 0 FSM change status [AddrSynced -> Established].
14: 24.853879 1 - 0 send KEEPALIVE.
15: 24.853879 1 - 2 send TOPOLOGY.
16: ...
17: ...
18: 444.813837 1 - 0 FSM event Hold_Timer_expired.
19: 444.813837 1 - 0 send NOTIFY.
20: 444.813837 1 - 0 FSM change status [Established -> Idle].
21: 444.813837 1 - 2 send TOPOLOGY.
22: 444.823843 0 - 1 FSM event Hold_Timer_expired.
23: 444.823843 0 - 1 send NOTIFY.
24: 444.823843 0 - 1 FSM change status [Established -> Idle].
25: 444.823846 2 - 1 receive TOPOLOGY size 69
26: 444.823846 2 - 1 FSM event Receive_TOPOLOGY_message.
27: ...
28: ...
29: 18000.000000 0 - 1 send ADDRESS.
30: 18000.010015 1 - 0 receive ADDRESS size 147
31: 18000.010015 1 - 2 send ADDRESS.
32: 18000.010015 1 - 0 FSM event Receive_ADDRESS_message.
33: 18000.020030 2 - 1 receive ADDRESS size 147
34: 18000.020030 2 - 1 FSM event Receive_ADDRESS_message.
35: ...
36: ...
```

Figure 4-17: Each line shows a TIPP event happened at a simulated time point. The notation $i - j$ means node i's connection to a neighbor j.

At line 29, node 0 renews the subdivision of its address prefix allocated to node 1, and subsequently, node 1 renews the corresponding prefix allocated to node 2 at line 31.

Chapter 5

Route Representation and Packet Forwarding

In the previous chapter, we discussed the basic mechanisms we provide for route discovery and route availability discovery. In this chapter, we discuss how a route is represented in a packet header and how routers forward a packet unambiguously along the route specified in the packet's header.

Packet forwarding requires the coordination of routers of different domains. Routers must agree on where a packet should go in order to avoid forwarding loops and to ensure that a packet arrives at its destination. Therefore, in our design, route representation and packet forwarding is a module on which we impose architectural constraints. Users should use the route representation scheme we design to specify a route, and routers should adhere to our forwarding algorithm to forward a packet.

In this chapter, we first present our route representation scheme, followed by the corresponding packet forwarding algorithm. Second, we briefly describe how a user generates a route representation. Third, we describe how a route representation for a reply packet or an ICMP packet is generated. Finally, we describe how to optimize our route representation scheme.

5.1 Route Representation

We first describe and compare previous work. Second, we identify key design requirements and present our design rationale. Finally, we describe the design details of our route representation scheme. Our design has the feature that a user can specify a common type of domain-level route using a source and a destination address.

5.1.1 Previous Work

Route representation schemes can be categorized into two types: *stateful* and *stateless*. In a *stateful* approach, a user sets up a path before he sends packets along the path. Path setup maps a path identifier to a next forwarding hop at each router along the path the user has chosen. So packets carrying a path identifier can be forwarded along a user-chosen path. The mapping from path identifiers to paths is usually temporary. The same path identifier could be assigned to a different path in a different path setup. A *stateful* approach can be optimized such that one path is reused by multiple connections to avoid per-connection path setup overhead. The ATM [1] network, NIMROD [25, 96, 86], IDPR [95], SIDR [41], and Multiple Protocol Label Switching (MPLS) [89] all use a stateful approach for route representation.

In a *stateless* approach, a user does not need to set up a path before he sends packets. A user can represent a path using a sequence of labels in a packet header. A label could either be a globally unique identifier, such as an address, or an identifier local to a router, such as an outgoing interface address [100]. We categorize the first approach as the *stateless global* approach, and the second one as the *stateless local* approach. PIP [50], SIPP [52], SIDR [41], IDPR [95], and IP loose source routing [84, 35] all use the globally unique addresses of intermediate routers, or the globally unique identifiers of intermediate domains to represent a route. The work described in [100] uses a sequence of outgoing interface identifiers of routers to represent a route.

As we are most interested in representing a domain-level route, we describe how to apply these approaches to represent a domain-level route. With a *stateful* approach, a user could set up a path and obtain a path identifier to represent a domain-level route. The user can then insert the path identifier in his packet headers. With a *stateless global* approach, a user can use a sequence of domain addresses[1] or a sequence of domain identifiers to represent a route. With a *stateless local* approach, a domain could assign a domain-local address to each of its neighboring domains. A user can then use a sequence of such addresses to represent a domain-level route.

5.1.2 Comparison

In order to make our design decision, we first compare the three different approaches from various aspects, including header overhead, maintenance overhead, support for reverse path generation, forwarding efficiency, and transit policy checking overhead.

- **Header Overhead**: We compare the number of bytes required in a packet header to represent a domain-level route. For the *stateful* approach, a path identifier could be very short. For example, MPLS [89] uses a 20-bit identifier. Thus, the header overhead is on the order of a few bytes.

  For the *stateless global* approach, the header overhead depends on the length of a domain-

[1]We define a domain address as an anycast address [80, 36] whose destination is any router in a domain.

level route. Measurement studies [33] show that the average number of domains an end-to-end packet traverses in the Internet is about 4. In NIRA, an address is 128-bit long, and a domain identifier is 32-bit long. If we adopt a straightforward *stateless global* approach in our architecture NIRA, the header overhead is about 64 bytes if domain addresses are used, and 16 bytes if domain identifiers are used.

For the *stateless local* approach, since a domain almost never has more than 2^{24} domain-level neighbors, it can use a 24-bit address to identify its neighbors. Then the average header overhead is about $4 * 3 = 12$ bytes.

- **Maintenance Overhead**: We compare the overhead to maintain the mapping from the route representation in a packet header to a route. With a *stateful* approach, at the time of communication, a user will need to set up a path, and routers need to maintain the dynamic mapping from path identifiers to paths.

  With a *stateless* approach, a routing protocol could map an address or a domain identifier to a next forwarding hop at a router, and this mapping could be reused by all users. No extra overhead is required at the time of communication.

- **Support for Reverse Route Generation**: We compare how easy it is to generate a reverse route representation from a forward route representation. If a reverse route representation can be easily generated from a packet with a forward route representation, then a router en route or a receiver is able to quickly send a response back to the sender of the packet by inspecting the packet header only.

  For the *stateful* approach, a reverse path identifier may not be correlated with a forward path identifier. In general, a router or a receiver cannot generate a reverse path identifier from a packet with a forward path identifier.

  For the *stateless global* approach, a reverse route representation can be generated simply by reversing the forward route representation, since a domain address or a domain identifier can uniquely represent a domain in either the forward or the reverse direction. For the *stateless local* approach, a domain-local address only specifies a neighbor of a domain. A reverse route representation cannot be simply generated from a forward route representation because a local address that specifies the neighbor M of a domain N does not necessarily specify the reverse relation, i.e., neighbor N of domain M.

- **Forwarding Lookup Efficiency**: We compare the computation cost for a router to determine the next forwarding hop. With all three approaches, a router could simply look up a path identifier, an address, or a domain identifier in a routing table to determine the next forwarding hop.

- **Transit Policy Checking Overhead**: A domain may want to check whether a route specified in a packet header violates its transit policy before it forwards a packet. We want to compare the overhead for transit policy checking in different route representation schemes.

  For the *stateful* approach, policy checking can be done at path setup time. If a packet carries a valid path identifier, the route specified by the path identifier will conform to the transit policies of providers en route. Additional transit policy checking is not necessary after a router has determined the next hop to forward a packet.

  For the *stateless* approaches, as there is no special forwarding state associated with a policy-valid route, an intermediate router needs to check for transit policy violation in addition to deciding the next hop to forward a packet.

5.1.3 Design Requirements

The comparison of different approaches shows that each approach has its advantages and disadvantages. In our design, we identify the following key design requirements:

- Low maintenance overhead. End users need not set up a path before sending a packet, and routers need not maintain state for every active path. We consider per-path state an unscalable solution, and path setup also increases connection setup latency.

- Easy reverse route generation. A receiver should be able to generate a reverse route representation to send a reply from a packet it receives without invoking a route discovery process. A router should be able to generate a route representation to send an Internet Control Message Protocol (ICMP) [83, 30] packet from a packet it receives without invoking a route discovery process.

  We consider this requirement important because in many cases, a router or a receiver needs to quickly send a response to the sender of a packet. For example, in the case of a route failure, a router needs to send an ICMP message back to the sender of a packet.

- Low header overhead. The number of bytes in a packet header to represent a domain-level route should be minimized.

- Efficient forwarding. The amount of operations needed for a router to find the next forwarding hop of a packet should be minimized.

- Low policy checking overhead. The amount of operations needed for a router in a domain to determine whether a route representation complies with the domain's transit policy should be minimized.

5.1.4 Design Rationale

To satisfy these requirements, we opt for a *stateless global* approach for its low per-connection maintenance overhead compared to a *stateful* approach, and its easiness for reverse path generation compared to a *stateless local* approach. Within the *stateless global* category, we prefer using a sequence of addresses for route representation to using a sequence of domain identifiers. There are two reasons. First, if we use a sequence of addresses, a packet header in NIRA could be exactly the same as that in IPv6. We believe this feature will facilitate the deployment of NIRA. Second, unlike a domain identifier, an address is able to specify both a single node and a group of nodes. So using a sequence of addresses has the flexibility to represent finer granularity routes. For example, a user could use a sequence of router addresses to represent a router-level route. Therefore, although our design does not address how a user discovers a router-level route, NIRA allows a user to specify a router-level route.

The *stateless global* approach has a larger header overhead compared to the other two approaches, and a higher policy checking overhead at packet forwarding time compared to the *stateful* approach. In our design, we would like to reduce the header overhead and the transit policy checking overhead of the *stateless global* approach.

Since we have adopted a provider-rooted hierarchical addressing scheme, an address can be mapped to a sequence of domains that allocate the address (Chapter 4.2). We can utilize this feature to minimize the header overhead and policy checking overhead for the common case. Intuitively, for a packet with a source and a destination address, the source address could represent the sequence of domains that the packet traverses to reach the *Core*, and the destination address could represent the sequence of domains the packet traverses from the *Core* to the destination. Therefore, a source and a destination address can represent a type of commonly used and policy-allowed domain-level route. General routes can be represented by more than two addresses.

Before we proceed to describe our route representation scheme in detail, for clarity, we first define a few terms and conventions.

5.1.5 Notation

We use *nprAddr*(x) to denote the non-provider-rooted address (defined in Chapter 4.2) of a domain x or a node x (a host or a router). We use $nprPf(x)$ to denote the non-provider-rooted inter-domain address prefix of a domain x or a node x.

We have used the word *route* to refer to a sequence of domains between a sender and a receiver. In this section, we will frequently talk about a sequence of domains that might not have a sender in the first domain or a receiver in the last domain, e.g., a sub-sequence of domains within a route. We refer to such a sequence of domains as a *route segment*.

Without loss of generality, we assume that an address with an all-zero intra-domain address represents a domain address. When we use a sequence of addresses to represent a route segment, the addresses we use will be domain addresses. When we represent a route between a specific source and a destination, say between Bob and Alice in Figure 4-3, the first address will be a unicast address of the source, and the last address will be a unicast address of the destination. Other addresses in the route representation are domain addresses.

If two adjacent domains on a route are inter-connected by a routing protocol in the *Core*, we use the symbol $\rightsquigarrow$ to indicate that connection type; otherwise, if they are connected by TIPP, we use $\rightarrow$ to indicate the type.

5.1.6 Details

We take a divide-and-conquer design approach. Note that any end-to-end domain-level route can be divided into a sequence of route segments. We first design the route representation scheme for basic route segments, and then come up with the design for any route.

We identify three important types of route segments: uphill, downhill, and bridge, based on whether an address can be mapped to a route segment. For an uphill route segment, there is at least one address prefix allocated from the last domain to the first domain, and vice versa for a downhill segment. So in Figure 4-3, the route segment $N_1 \rightarrow R_1 \rightarrow B_1$ is an uphill route segment; the route segment $B_1 \rightarrow R_3 \rightarrow N_3$ is a downhill route segment. A bridge route segment contains only two domains, with no address prefix allocated from one to the other. The two domains are either TIPP neighbors, or connected by a routing protocol. In the latter case, the two domains may not be directly connected, and the path between the two domains are chosen by the routing protocol in the *Core*. In Figure 4-3, $R_2 \rightarrow R_3$ is a bridge segment connected by TIPP, and $B_1 \rightsquigarrow B_2$ is a bridge segment connected by the routing protocol in the *Core*.

We also define a compound route segment type: a hill type. A hill segment consists of at most three segments in the following order: an uphill segment, a bridge segment, and a downhill segment. One of the three segments could be missing. Moreover, if a hill segment contains a bridge component, then the first domain of the bridge component must be the root[2] of an address prefix that is allocated along the uphill segment, and the second domain must be the root of an address prefix that is allocated along the downhill segment. For example, in Figure 4-3, $N_1 \rightarrow R_1 \rightarrow B_1 \rightsquigarrow B_2 \rightarrow R_3 \rightarrow N_3$ is a hill route segment.

Not every route segment has a type. For example, in Figure 4-3, the route segment $N_1 \rightarrow R_2 \rightarrow R_3$ does not satisfy our definition for any type, and thus does not have a type. However, a route segment may be divided into a sequence of smaller segments such that each smaller segment has a type. For example, $N_1 \rightarrow R_2 \rightarrow R_3$ can be divided into two smaller segments $N_1 \rightarrow R_2$ and $R_2 \rightarrow R_3$, with the first being an uphill segment, and the second being a bridge segment. A route segment is maximal if it will lose its type when extended to include more domains in a route. For

[2]We use the term "the root of an address prefi x pf" to refer to the top-level domain that allocates the address prefi x pf.

example, the route segment $N_1 \rightarrow R_2$ in the route $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$ is maximal, because we cannot extend the route segment to include the next adjacent domain R_3 since there is no address allocated along the path $R_3 \rightarrow R_2 \rightarrow N_1$. If a route only has one maximal segment, it has the same type as the segment.

By utilizing the Address to Path Mapping Property (Chapter 4.2.4) of a provider-rooted hierarchical address, we can use a source and a destination address to represent a route segment of any type, regardless of how many domains the route segment consists. For an uphill segment, the first domain must have an address whose allocation path overlaps with the route segment. In Figure 4-3, for the uphill route segment $N_1 \rightarrow R_2$, N_1 has an address 1:2:1:: whose allocation path $N_1 \rightarrow R_2 \rightarrow B_1$ overlaps with the route segment $N_1 \rightarrow R_2$. We can use that address as the source address, and the non-provider-rooted address of the last domain as the destination address to indicate where the segment stops. For example, we could use the source address 1:2:1:: and the destination address $nprAddr(R_2)$ to represent the uphill segment $N_1 \rightarrow R_2$.

The reverse of a downhill route segment is an uphill route segment. So is its representation. Therefore, the downhill route segment $R_2 \rightarrow N_1$ can be represented by $nprAddr(R_2)$ and 1:2:1::.

If a bridge segment is connected by TIPP, it can be represented by the non-provider-rooted addresses of the two domains. As an example, in Figure 4-3, the bridge segment $R_2 \rightarrow R_3$ can be represented by $nprAddr(R_2)$ and $nprAddr(R_3)$. If a bridge segment is connected by a routing protocol, such as two top-level providers in the *Core*, then the segment can be represented by two addresses that are routable by the routing protocol. For example, since a top-level provider in the *Core* announces its globally unique address prefix to other top-level providers, the bridge segment $B_1 \rightsquigarrow B_2$ can be represented by 1:: and 2::.

A hill route segment is represented by a source address allocated along its uphill segment, and a destination address allocated along its downhill segment. For example, in Figure 4-3, the hill route segment $N_1 \rightarrow R_1 \rightarrow B_1 \rightarrow R_3 \rightarrow N_3$ can be represented by 1:1:1:: and 1:3:1::. The hill segment $N_1 \rightarrow R_1 \rightarrow B_1 \rightsquigarrow B_2 \rightarrow R_3 \rightarrow N_3$ can be represented by 1:1:1:: and 2:1:1::.

A route consisting of multiple maximal route segments is represented by concatenating the representation of each route segment and deleting duplicate addresses. In Figure 4-3, suppose Bob wants to send a packet to Alice via the route $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$, which consists of three maximal segments: an uphill segment, a bridge segment, and a downhill segment. The uphill segment can be represented by 1:2:1::1000 and $nprAddr(R_2)$; the bridge segment can be represented by $nprAddr(R_2)$ and $nprAddr(R_3)$; the downhill segment can be represented by $nprAddr(R_3)$ and 1:3:1::2000. Concatenating the representation of each segment yields a sequence of addresses: 1:2:1::1000, $nprAddr(R_2)$, $nprAddr(R_2)$ $nprAddr(R_3)$, $nprAddr(R_3)$ and 1:3:1::2000. Deleting the duplicate addresses leads to the final route representation: 1:2:1::1000, $nprAddr(R_2)$, $nprAddr(R_3)$, and 1:3:1::2000.

5.2 Packet Forwarding

To ensure that a packet is forwarded unambiguously along the route specified in its header, we need to design a forwarding algorithm that understands our route representation scheme. A forwarding algorithm describes how a router examines a packet header and determines the next hop to forward the packet. NIRA's packet header format could be the same as that in IPv6. A header contains a source address field, a destination address field, and an optional routing header. The source and the destination address field contain the first two addresses of a route representation. If a route representation has more than two addresses, the additional addresses are stored in the routing header. The routing header also contains other auxiliary fields that can be used to compute the next-to-visit address.

Again, for clarity, we will repeat what we do when we describe TIPP. We abstract a domain as having one router and the possible multiple links connecting two domains as one domain-level link. As we primarily focus on how the domain-level next hop is determined, we describe our forwarding algorithm assuming a router will keep a separate table for intra-domain forwarding, as described in Chapter 4.2.5. We discuss the case where a domain has multiple routers and there are multiple links connecting two domains in Appendix 5.B.

5.2.1 Design Requirements

We identify the key design requirements for our forwarding algorithm before we describe our design rationale. If a user correctly specifies a route in a packet header using the route representation scheme described in Section 5.1, the first thing we want is that the packet is forwarded along the user-specified route to reach its destination, if the route is failure free. Secondly, if at a router, the next hop to forward a packet is unreachable due to failures, we would want the packet to be dropped at that router instead of looping around in the network. Thirdly, we want a route representation to be reversible. That is, a packet with the reverse of a route representation will be forwarded along the reverse of the route, if the route is failure free. This requirement makes it easy for a receiver of a packet to send a reply back to the sender of the packet. It also makes the forwarding service of routers more predictable to users. If a user sends a packet to a destination with a route he chooses, most likely he would expect that the return packets with a reverse route representation could come back from the same route.

A forwarding step is reversible if at domain N, a packet with a route representation is forwarded to a neighbor domain M, and at domain M, a packet with the reverse of the route representation will be forwarded to N. It can be seen that if each forwarding step is reversible, then the third requirement will be satisfied. So in our design of the forwarding algorithm, we focus on satisfying this step-wise reversibility requirement.

A user could specify a route using our route representation scheme, and routers en route are able to find next forwarding hop for the packet, but the specified route might violate a domain's transit policies. For example, in Figure 4-3, the user Bob may specify a route with the following addresses: 1:1:1::1000, 1:2::, $nprAddr(R_2)$, $nprAddr(R_3)$, 1:3:1::2000. This route representation is the concatenation of those of three route segments: a hill segment $N_1 \rightarrow R_1 \rightarrow B_1 \rightarrow R_2$, a bridge segment $R_2 \rightarrow R_3$, and a downhill segment $R_3 \rightarrow N_3$. This route representation strictly follows the scheme we described in Section 5.1. A router en route will be able to find the next hop to forward a packet with such a route representation. However, this route makes R_2 provide transit service between a provider B_1 and a peer R_3, which violates the transit policy of R_2. So, the route representation is incorrect.

A packet with a route representation that violates a domain's transit policy should be dropped instead of being forwarded to the next hop. However, we do not think we should require our forwarding algorithm to be able to detect policy violation. As we have discussed, the forwarding algorithm of a router must be consistent with those of other routers, but transit policies of domains may vary. So we make policy checking a separate module from forwarding. A domain might use any general mechanism to check for its transit policy. We do not restrict how it should be done. But our design of the route representation scheme and the packet forwarding algorithm limits where policy checking needs to be done. In cases where a packet only has a source and a destination address, a router does not need to do extra work for policy checking. We discuss policy checking in Chapter 6.

5.2.2 Design Overview

We have designed a route representation scheme using a divide-and-conquer approach. For any route, we could represent it using the concatenation of a sequence of single-type route segment representations, with duplicate addresses deleted. Similarly, when we design our forwarding algorithm, we first focus on how to design the forwarding algorithm for a route segment of a single type, i.e., uphill forwarding, downhill forwarding, bridge forwarding, and hill forwarding.

Recall that a route segment of any type can be represented by two addresses, with the first address being an address of the first domain of the route segment, and the second address being an address of the last domain. In our discussion about the forwarding algorithm for a single-type route segment, the source address of a packet (the address in the source address field of a packet) is always the first address of the route segment representation, and the destination address (the address in the destination address field of a packet) is the second address of the route segment representation. The source domain (or the destination domain) is the first (or the last) domain of the route segment.

With our route representation scheme, the source address and the destination address of a packet together represent a route. Therefore, both the source address and the destination address could be used to find the next hop. Our design uses different forwarding tables for routers to look up different addresses. When a packet is in its uphill route segment to the *Core*, a router looks up the source address of the packet in its uphill forwarding table to determine the next hop; correspondingly, when the packet is in its downhill route segment to reach its destination, a router looks up the destination address of the packet in its downhill table to determine the next hop.

Next, we describe the detailed design logistics for uphill forwarding, downhill forwarding, bridge forwarding, and hill forwarding.

5.2.3 Details

Uphill and Downhill Forwarding

First, we look at how to design the forwarding algorithm to forward a packet along the uphill or downhill route specified in the packet's header. For a packet with an uphill representation, e.g., 1:1:1:: and $nprAddr(B_1)$ for the uphill route segment $N_1 \rightarrow R_1 \rightarrow B_1$ in Figure 4-3, the forwarding algorithm at a router should forward the packet along the source address allocation path until the packet reaches its destination. For a packet with a downhill representation, the forwarding algorithm at a router should forward the packet according to the destination address until the packet reaches its destination.

Since a packet header does not have a field that describes the type of a route segment, our forwarding algorithm must be able to decide which address, the source or the destination address, is the one to use to determine the next hop. In our design, we keep two separate forwarding tables at a router: the downhill forwarding table and the uphill forwarding table. The downhill forwarding table at a router in a domain keeps entries for the hierarchical address prefixes the domain allocates to its neighbors and the domain's own address prefixes. When a router at a domain receives a packet, the router will first look up the destination address of the packet in its downhill table. If there is a match, then the destination address must be allocated by the domain to a neighbor. So the packet should be forwarded to the next hop indicated by the match.

The uphill forwarding table of a router at a domain keeps entries for the domain's hierarchical address prefixes allocated from the domain's neighbors. When a router cannot find a match for the destination address of a packet in its downhill table, i.e., the router has no knowledge about how to forward towards the destination, the router looks up the source address in its uphill table to decide which provider of the domain the packet should go to.

Second, we design our forwarding algorithm to deal with route failures. For downhill forwarding, if a domain is disconnected from a customer, then the router at the domain might not have an entry for an address prefix allocated to the customer in its downhill table. We definitely do not want the router to look up the source address of the packet in its uphill table. Instead, we want the packet to be dropped at the router. So we add an entry for each address prefix a domain has into the downhill forwarding table of the router at the domain, with the next hop pointing to a *blackhole*. A *blackhole* indicates that a packet should be dropped and an ICMP packet should be sent to notify the sender of the delivery failure. As forwarding lookup is based on longest prefix match, if a customer's address prefix is missing from the downhill table of the router at a domain due to failures and the router receives a packet destined to the customer, the router will find that the destination address of the packet matches an address prefix of the domain with the next hop pointing to a *blackhole*. So the router will drop the packet instead of looking up the packet's source address in its uphill table.

For uphill forwarding, if a domain is disconnected from its provider, then the router at the domain might not have an entry for an address prefix of the domain allocated from the provider. In this case, the router is unable to find a next hop to forward the packet. So the router can drop the packet and send an ICMP packet back to the sender.

Third, we examine the reversibility of our forwarding algorithm. It can be seen that an uphill forwarding step is reversible. If the router in domain N finds a match for a source address a in its uphill table, then the next-hop domain M must have allocated an address prefix pf that encloses the source address a to the domain N. So the router in the next-hop domain M must have an entry in its downhill table for the address prefix pf with the next-hop pointing to the domain N. For a packet with a reverse route representation, its destination address will be a. So the router in domain M is able to find a match for the address a in its downhill table and forward the packet to domain N. Similarly, we can verify that if a packet has a correct downhill route representation, a downhill forwarding step is also reversible.[3]

Bridge Forwarding

We look at how to design the forwarding algorithm to forward a packet with a bridge representation. A bridge route segment contains two domains, connected either by TIPP or a routing protocol. For a packet with a bridge representation, the forwarding algorithm at the router of the first domain (i.e., the source domain) of the bridge segment should forward the packet to the second domain (i.e., the destination domain) of the segment, and the router at the second domain should know that the packet has arrived at its destination.

Since a bridge segment has only two domains, the router at the source domain of the bridge segment will know how to reach the destination domain either via TIPP, or via a routing protocol. At first thought, it seems that for bridge forwarding, a router could simply inspect the destination address, because the router at the source domain knows how to forward to the destination address, and the router at the destination domain knows that the destination address is one of its addresses.

However, a route segment representation with the same second address could indicate either

[3]Note that if a packet has an incorrect downhill route representation, a downhill forwarding step may not be reversible. For example, suppose the router in a provider domain, say B_1 in Figure 4-3, sends a packet to Bob in N_1 with a spoofed source address 1:2:2::, which is an address of N_2, and a destination address 1:2:1::1000. Such a packet will be forwarded to R_2 from B_1. However, a packet with a reverse representation: a source address 1:2:1::1000 and a destination address 1:2:2:: will be forwarded to N_2 at R_2, instead of B_1. Our design focus is to ensure correct forwarding with correct route representations, and with incorrect route representations, our design requirements might not be satisfi ed.

an uphill segment, or a downhill segment, or a bridge segment, depending on the first address. If the first address of a packet is a hierarchically allocated address, then the two addresses represent either an uphill or a downhill route segment. For example, in Figure 5-1, domain D has an address prefix 5::16, and allocates a subdivision of that address prefix, 5:0:1::/48, to domain E, and E allocates a subdivision of 5:0:1::/48, 5:0:1:1::/96, to F. Domain D and domain F also has a shortcut peering link. If a packet at domain F has a source address 5:0:1:1::/96 and a destination address $nprAddr(D)$, it should be forwarded along the segment $F \to E \to D$; if a packet at domain F has the same destination address $nprAddr(D)$, but a different source address $nprAddr(F)$, according to our route representation scheme, the packet should be forwarded along the route segment $F \to D$.

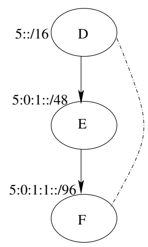


Figure 5-1: If a packet at domain F has a source address 5:0:1:1::/96 and a destination address $nprAddr(D)$, it should be forwarded along the segment $F \to E \to D$; if a packet at domain F has the same destination address $nprAddr(D)$, but a different source address $nprAddr(F)$, according to our route representation scheme, the packet should be forwarded along the route segment $F \to D$. So to forward a packet, a router cannot only examine the destination address.

So to correctly forward a packet along a bridge route segment, a router needs to inspect both the source address and the destination address of a packet. We introduce a bridge forwarding table and a special entry in a router's uphill table to help the router to determine the type of a route representation. A bridge forwarding table at the router of a domain contains the non-provider-rooted address prefix of a neighbor that does not allocate address or take address allocation from the domain, with the next hop pointing to the neighbor. We add an entry for the non-provider-rooted address prefix of the domain at the router's uphill table, with the next hop pointing to the router's bridge table. This entry tells a router at a domain that if the router cannot find a match for the destination address of a domain in its downhill table, and the source address of a packet matches the non-provider-rooted address prefix of the domain, then the router should interpret the route representation in the packet as a bridge representation, and forward the packet along a bridge segment.

So for a packet with a route representation of a bridge segment connected by TIPP, e.g., a packet with a source address $nprAddr(R_2)$ and a destination address $nprAddr(R_3)$ that represent the bridge segment $R_2 \to R_3$ in Figure 4-3, the router at the source domain will not find a match for the destination address in its downhill table. Then it will look up the source address of the packet in its uphill table. There it will find a match for the source address with the next hop pointing to its bridge table. Therefore, the router will look up the destination address in its bridge table and forward the packet to the corresponding next hop.

If the source domain and the destination domain of a bridge segment is connected by the routing protocol in the *Core*, for example, the bridge segment $B_1 \rightsquigarrow B_2$, then the router at the source domain learns how to reach the destination domain through the routing protocol, instead of TIPP. To forward a packet with a route representation of a bridge segment connected by a routing protocol, the router at the source domain should look up the destination address in its routing table built by the routing protocol, and forward the packet to the next hop returned by the lookup. But when a router receives a packet, the router does not know the type of the representation in the packet. For the forwarding algorithm we described so far, the router will first look up the destination address in its downhill table, where it will find no match; then it will look up the source address in its uphill table, where it will find no match either. We have two choices to modify our forwarding algorithm to make a router find the next forwarding hop. First, we can incorporate the routing table built by the routing protocol into the router's downhill table. So the router will find the next hop to forward the packet in its downhill table. Second, we can design our forwarding algorithm to look up the destination address in the routing table after the algorithm cannot find a next hop in the router's forwarding tables. In our design, we take the second approach, as it isolates TIPP from other routing protocols. A change in a router's routing table would not result in a change in the router's downhill table.

We can instruct a TIPP router that also participates in the routing protocol in the *Core* to look up the destination address of a packet in its routing table either right after the router cannot find a match for the destination address in its downhill table, or after the router looks up the source address in its uphill table. The first approach has the advantage of avoiding one unnecessary lookup, as the router will not be able to find the next hop to forward a packet in its uphill table.

The second approach has the advantage of ensuring forwarding reversibility in the *Core* routing region. If a router does not look up the source address in its uphill table before it looks up the destination address in its routing table, a packet with a non-routable source address could be injected into the *Core* . In the example shown in Figure 4-3, the router at B_1 would have a routing entry for the address prefix 2::/16, since this address prefix will be announced by B_2 in the routing protocol. If a packet has a source address $nprAddr(B_1)$ and a destination address 2::/96, the router at B_1 would find a match for the destination address in its routing table, and forward the packet into the *Core* if it does not examine the source address of the packet. But a router in the *Core* might not have a routing entry for the address $nprAddr(B_1)$, since this address is not announced in the routing protocol by B_1. Therefore, the reverse route representation 2::/96 and $nprAddr(B_1)$ might not be routable in the *Core*.

In our design, we take the second approach in order to check the routability of the source address of a packet before injecting the packet into a routing region. In Section 5.6, when we describe the optimization of our route representation scheme, we further discuss the importance of such checking.

We add a special entry in a router's uphill table if the router is at a top-level provider and participates in the routing protocol in the *Core*. This entry contains the globally unique address prefix of the top-level provider, with the next hop pointing to the routing table of the router. When the router fails to find a match for the destination address of a packet in its downhill table, it will follow our algorithm to look up the source address in its uphill table. If it finds a match with the next hop pointing to its routing table, the router can then look up the destination address in its routing table and forward the packet to the next hop computed by the routing protocol in the *Core*.

When a packet with a bridge representation leaves the source domain of the bridge segment and enters the *Core*, we want the packet to follow the path chosen by the routing protocol to reach the destination domain of the bridge segment. It is possible that the path actually consists of multiple providers. So after a packet leaves the source domain of the bridge segment, it might not enter the destination domain of the bridge segment immediately. For example, in Figure 4-3, B_1, B_2 and B_3 might have a business agreement such that if the direct connection between any two of them is broken, the third one is willing to provide temporary transit service between the other two. So if the connection between B_1 and B_2 is broken, the routing protocol in the *Core* might choose a route to reach B_2 via B_3. Thus, when a packet with a source address 1:: and a destination address 2:: which represent the bridge segment $B_1 \rightsquigarrow B_2$ leaves B_1, it will arrive at B_3 first.

When a packet with a bridge representation arrives at a domain in the *Core* that is not the destination domain of the bridge segment, two possible cases might occur. In the first case, the domain is entirely embedded in the *Core* and understands only the routing protocol, then a router in such a domain will behave like a router in today's Internet. It has only one routing table, and will follow the same forwarding algorithm as the one used in today's Internet, looking up the destination address of the packet in its routing table to determine the next hop, and forwards the packet to the next hop. So in this case, the packet will be forwarded towards the destination domain of the bridge segment along the path chosen by the routing protocol.

In the second case, the domain understands both TIPP and the routing protocol in the *Core*, and the router in such a domain has both a routing table and forwarding tables established by TIPP. Therefore, when a packet arrives at the router, the router will follow our forwarding algorithm instead, first looking up the destination address of the packet in its downhill table, then looking up the source address in its uphill table. For the forwarding algorithm we described so far, the router at the domain will have only an entry for the globally unique address prefix of the domain with the next hop pointing to its routing table. Thus, after the router fails to find a match for the destination address of a packet in its downhill table, it might fail to find a match for the source address in its uphill table too. Therefore, we need to extend the entry in the router's uphill table to match all routable addresses in the *Core*. With this entry, the router will forward the packet according to the route chosen by the routing protocol in the *Core*, regardless of where the packet comes from. Since we assume that all global *Core* addresses are allocated from 0::/1, the added entry is thus for the address prefix 0::1 with the next hop pointing to the router's routing table.[4]

Therefore, in both cases, the packet will be forwarded to the next hop that is computed by the routing protocol to reach the top-level provider that allocates the destination address. This is the desired forwarding behavior.

It can be seen that when the source domain of a bridge segment is disconnected from the destination domain of the bridge segment, a router will not find a match for the destination address either in its bridge table, or in its routing table. So the router will drop the packet without forwarding it to a wrong place. Thus, our forwarding algorithm handles route failure properly in bridge forwarding.

It can be verified that bridge forwarding is reversible. If the source domain and the destination domain of a bridge segment are connected by TIPP, then the router at each domain will have the other domain's non-provider-rooted address prefix in its bridge forwarding table. So if a packet with a source address $nprAddr(N)$ and a destination address $nprAddr(M)$ is forwarded from N to M, then a packet with the reverse representation: $nprAddr(M)$ and $nprAddr(N)$ will be forwarded from M to N. Similarly, it can be seen that the bridge forwarded is reversible when the source and the destination domain are connected by the routing protocol in the *Core*.

Hill Forwarding

At last, we consider how to forward a packet with a hill route representation. A hill route is represented by two hierarchically allocated addresses. A packet with a hill route representation should be forwarded first uphill along its source address allocation path, and then downhill along its destination address allocation path. If the two allocation paths share a common domain, then before the packet reaches the common domain, our forwarding algorithm will find no match for the destination address in a router's downhill table, but will find a match for the source address in a router's uphill table. So the packet will be forwarded uphill until it reaches the common domain that allocates both the source and the destination address. After the packet reaches the common domain, our forwarding algorithm will find a match for the destination address in a router's downhill table. So the packet will be forwarded downhill until it reaches its destination. Therefore, it can be seen that since our design already supports uphill forwarding and downhill forwarding, the forwarding algorithm also works for a hill route consisting of an uphill segment and a downhill segment that share a common domain.

For a packet with a hill route representation, if the source address allocation path and the

[4] If not all routable addresses are allocated from the same contiguous address space, we will need multiple address prefi xes to represent all routable addresses.

destination address allocation path do not share a common domain, then our forwarding algorithm described so far will forward the packet all the way up to the top-level provider that allocates the source address, because no router on the source address allocation path will have a match in its downhill table for the destination address. At that top-level provider, the router there will follow the bridge forwarding steps to forward the packet into the *Core*. Inside the *Core*, the packet will be forwarded to the top-level provider that allocates the destination address along the path chosen by the routing protocol in the *Core*, according to what we have discussed in bridge forwarding. When the packet arrives at the top-level provider that allocates the destination address, the router at that provider will find a match for the destination address of the packet in its downhill table. So the packet will be forwarded along the allocation path of its destination address until it reaches its destination.

Therefore, it can be seen that our forwarding algorithm handles hill forwarding correctly. In fact, at any step of a hill forwarding, a TIPP router does either an uphill forwarding, a downhill forwarding, or a bridge forwarding. Since our forwarding algorithm satisfies the design requirements for each forwarding type: forwarding along a specified route, dropping on failure, reversible, it thus satisfies the requirements for hill forwarding.

Degenerate Hill Forwarding

For a packet with any two hierarchically allocated addresses, our forwarding algorithm will interpret the two addresses as a hill route representation, and forward the packet according to the hill forwarding rule. If the allocation paths of the source address and the destination address share a common domain, and one of the two addresses is the address of the common domain, then the route the packet follows is actually either an uphill route or a downhill route. For example, in Figure 4-3, a packet with a source address 1:1:1::1000 and a destination address 1:: will be forwarded along the uphill route $N_1 \rightarrow R_1 \rightarrow B_1$. We consider such a route representation as a degenerate hill route representation, and do not recommend using two hierarchically allocated addresses to represent an uphill or a downhill route segment, because such a route segment representation cannot be concatenated efficiently with a bridge route segment representation. For example, in Figure 4-3, suppose Bob wants to send a packet to Alice along the route: $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$. If the uphill segment in this route $N_1 \rightarrow R_2$ is represented by two hierarchical addresses: 1:2:1::1000 and 1:2::, and the downhill segment $R_3 \rightarrow N_3$ is also represented by two hierarchical addresses: 1:3:: and 1:3:1::2000, then the entire route will be represented by six addresses: 1:2:1::1000, 1:2::, $nprAddr(R_2)$, $nprAddr(R_3)$, 1:3::, and 1:3:1::2000, instead of four addresses: 1:2:1::1000, $nprAddr(R_2)$, $nprAddr(R_3)$, and 1:3:1::2000.

Forwarding for an Arbitrary Route Representation

We have discussed how we design our forwarding algorithm to work with a route representation of a single type. An arbitrary route can be represented by concatenating a sequence of single-type route segment representations. When a user sends a packet, the source address field and the destination address field of a packet specify the first route segment the packet should follow, and the other addresses in the routing header specify the rest of the route segments the packet should follow. So we design our forwarding algorithm to shift the next route segment representation into the source address field and the destination address field of the packet when the packet has arrived at the end of one route segment. Therefore the packet will be forwarded along each route segment specified in its header, thus following the route specified in its header.

So far, we have described how we design the forwarding algorithm. Next, we describe how the forwarding tables at a router: the uphill table, the downhill table, and the bridge table, are established via TIPP.

5.2.4 Forwarding Tables

Forwarding Table Initialization

A router in a domain initializes its uphill table with the domain's non-provider-rooted address prefix, with the next hop pointing to its bridge forwarding table. The router initializes its downhill forwarding tables with the domain's own inter-domain address prefixes, with the next hop for each prefix pointing to itself. These inter-domain address prefixes include both the domain's non-provider-rooted inter-domain address prefix and the domain's provider-rooted inter-domain address prefixes. The router also initializes its downhill forwarding tables with the domain's provider-rooted hierarchical address prefixes, with the next hop for each prefix pointing to a *blackhole*. A router leaves its bridge table empty. Figure 5-2 shows the initial state of the forwarding tables of a domain R_2 that appears in the network shown in Figure 4-3.

R_2's downhill table

$nprPf(R_2)$	$self$
1:2::/96	$self$
1:2::/32	$blackhole$

R_2's uphill table

$nprPf(R_2)$	$bridge$

R_2's bridge table

Figure 5-2: The initial state of R_2's forwarding tables.

If a domain is a top-level provider, then the downhill table of its router will have entries for the domain's globally unique address prefix and its inter-domain address prefix derived from that address prefix. In addition, the router's uphill table also includes an entry that includes all routable

B_1's downhill table

$nprPf(B_1)$	*self*
1::/96	*self*
1::/16	*blackhole*

B_1's uphill table

$nprPf(B_1)$	*bridge*
0::/1	*routing*

B_1's bridge table

Figure 5-3: The initial state of B_1's forwarding tables.

B_1's routing table

2::/16	B_2
3::/16	B_3
4::/16	B_4

Figure 5-4: Contents of B_1's routing table.

addresses in the *Core* routing region with the next hop pointing to its routing table. Figure 5-3 shows the initial state of the forwarding tables of a top-level domain B_1 in the network shown in Figure 4-3. Figure 5-4 shows the possible contents of the top-level provider B_1's routing table, built by the routing protocol running in the *Core*.

Forwarding Table Update

A router updates its forwarding tables using information learned from TIPP, as shown in Figure 4-15. If a domain P allocates an address prefix pf to a neighbor C, then C will send to P an adjacent link record that announces the prefix pf. When the router in P receives the adjacent link record from the router in C, it will add an entry for pf in its downhill table, and sets the next hop to C. For example, in Figure 4-3, when B_1 receives the adjacent link record from R_1, the link record will include a field that tells B_1 1:1::/32 is an address space reachable via R_1. B_1 will insert the prefix 1:1::/32 into its downhill table, with the next hop set to R_1.

There is another way to set the downhill table of P. Since P allocates an address prefix pf to a neighbor C, P can set its downhill table when it sends the **address** message to C, or sets the table using the address records stored in its output address database. We choose to use a link record to update a downhill table for two reasons. First, the adjacent link record from the neighbor C that contains the prefix record for pf confirms that C has accepted the address allocation and is ready to receive packets destined to addresses within the address space pf. Second, using adjacent link records to set the downhill tables offers the flexibility of load balancing. A domain normally has multiple routers. A domain may split up a prefix allocated from a provider into several smaller subdivisions, and announce different subdivisions via different border routers. So traffic destined to different subdivisions of the prefix may traverse different border routers.

If a domain C accepts an address prefix pf allocated from a neighbor P, the router in C will update its forwarding tables after it has sent to P an adjacent link record that announces pf. C inserts pf into its uphill table with the next hop pointing to P. The entry for pf in the uphill table indicates that the address prefix pf is allocated from P. If the prefix pf is a new address prefix allocated to C, C will also insert pf into its downhill table, with the next hop pointing to *blackhole*. Moreover, C will pick an inter-domain address prefix for hosts and routers inside itself from the address space pf and insert the prefix into its downhill table with the next hop pointing to itself. In Figure 4-3, after R_1 sends an adjacent link record to B_1, it will insert 1:1::/32 into its uphill table with the next hop set to B_1, insert 1:1::/32 into its downhill table with the next hop set to *blackhole*, and insert 1:1::/96, its inter-domain address prefix, to its downhill table with the next hop set to itself.

If a domain N does not have an address allocation relationship with a neighbor M, i.e., neither N allocates an address prefix to M, nor M allocates an address prefix to N, then the router at domain N will update its bridge table when it receives the adjacent link record sent by M. The router at domain N will insert the non-provider rooted address prefix of M into its bridge table with the next hop pointing to M.

When a TIPP connection to a neighbor breaks down, a router could either clear the entries in its forwarding tables for which the neighbor is the next hop, or mark the entry as down.

Examples

Figure 5-5 shows the forwarding tables of R_2 after its TIPP connections to all its neighbors are fully established. Figure 5-6 shows those of B_1 after its TIPP connections to all its neighbors are fully established.

R_2's downhill table

$nprPf(R_2)$	$self$
1:2::/96	$self$
1:2::/32	*blackhole*
1:2:1::/48	N_1
1:2:2::/48	N_2

R_2's uphill table

$nprPf(R_2)$	$bridge$
1:2::/32	B_1

R_2's bridge table

$nprPf(R_3)$	R_3

Figure 5-5: R_2's forwarding tables.

B_1's downhill table

$nprPf(B_1)$	*self*
1::/96	*self*
1::/16	*blackhole*
1:1::/32	R_1
1:2::/32	R_2
1:3::/32	R_3

B_1's uphill table

$nprPf(B_1)$	*bridge*
0::/1	*routing*

B_1's bridge table

Figure 5-6: B_1's forwarding tables.

Summary of Forwarding Table Contents

In summary, the downhill forwarding table of a router at a domain N has the following types of entries:

1. $nprPf(N)$ with the next hop pointing to itself;

2. every inter-domain address prefix of N with the next hop pointing to itself.

3. every address prefix allocated to N with the next hop pointing to *blackhole*.

4. every address prefix allocated to a neighbor C with the next hop pointing to C.

The uphill forwarding table of the router at a domain N has the following types of entries:

1. $nprPf(N)$ with the next point pointing to *bridge*;

2. every address prefix of N allocated from a neighbor P with the next hop pointing to P.

If N is a top-level provider, its router's uphill table also an entry that includes all routable addresses in the *Core* routing region with the next hop pointing to its routing table.

The router in a domain N's bridge table includes an entry for the non-provider-rooted address prefix of each neighbor M with which N does not have an address allocation relationship, with the next hop pointing to M.

5.2.5 The Forwarding Algorithm

Next, we describe the forwarding algorithm that works with our route representation scheme. The basic forwarding algorithm involves the following steps:

1. A router at a domain decides whether the destination address of a packet is within the domain's address space by looking up the destination address in its downhill forwarding table using the longest prefix match rule. If a match is found, the packet is forwarded towards the next hop in the entry. If the next hop is the router itself, go to Step 4. If no match is found, go to the next step.

2. Look up the source address in a router's uphill table using the longest prefix match rule. If no match is found, drop the packet and send an ICMP packet to notify the sender of the delivery failure. If a match is found, and the next hop is a neighbor router, forward the packet to the next hop. If the next hop is *bridge* or *routing*, go to the next step.

3. Look up the destination address in a corresponding table based on the look up result from the previous step using the longest prefix match rule. If a match is found, forward the packet to the next hop; else drop the packet and send an ICMP packet to notify the sender of the delivery failure.

4. If in the packet header there is no address left, then the packet has arrived at its destination domain. Consult the intra-domain routing table to determine the final destination. If in the packet header there are addresses left to be visited, then the destination address will be shifted to the source address field; the next-to-visit address will be moved to the destination address field; the original source address will be preserved at the previous next-to-visit address field; and the next-to-visit pointer will move to the next address in the routing header. The changes of the packet header are shown in Figure 5-7. Go back to step 1.

before Step 4 →

source	*src*
destination	*dst*
	...
next-to-visit	V_1
	V_2
	...

after Step 4 →

source	*dst*
destination	V_1
	...
	src
next-to-visit	V_2
	...

Figure 5-7: This figure shows how a packet header is changed after a router executes Step 4 of the forwarding algorithm.

Pseudocode 5 shows the pseudocode for finding the next-hop.

5.2.6 Correctness

We present in detail that the forwarding algorithm satisfies our design requirements in Appendix 5.A. Here we use two examples to show how the route representation scheme and the forwarding algorithm work together. In the first example, we show if Bob in Figure 4-3 sends a packet to Alice with a source address 1:1:1::1000 and a destination address 1:3:1::2000, the

Pseudocode 5 : Next-hop lookup

```
Down: downhill forwarding table;
Up: uphill forwarding table;
Bridge: bridge forwarding table;
self: the current router;
src: source address;
dst: destination address;
AddrLeft: number of addresses left to visit;
Addrs: addresses in the optional routing header;
Lookup(a, T): longest prefix match for address a in table T.
Begin:
nextHop = Lookup(dst, Down).
if nextHop == self then
    if AddrLeft == 0 then
        return self
    else
        compute next-to-visit address index i
        shift dst to src, Addrs[i] to dst;
        preserve the original src in Addrs[i].
        goto Begin
    end if
end if
if nextHop == blackhole then
    goto Drop
else if nextHop ≠ noMatch then
    return nextHop
else
    nextHop = Lookup(src, Up)
    if nextHop == noMatch then
        got Drop
    else if nextHop == bridge then
        nextHop = Lookup(dst, Bridge)
    else if nextHop == routing then
        nextHop = Lookup(dst, Routing)
    else
        return nextHop
    end if
    if nextHop ≠ noMatch then
        return nextHop
    else
        goto Drop
    end if
end if
Drop:
drop p; send an ICMP error notification
return noMatch
```

packet will be forwarded along the domain-level route $N_1 \rightarrow R_1 \rightarrow B_1 \rightarrow R_3 \rightarrow N_3$. Since N_1 is allocated an address prefix from each of its providers R_1 and R_2, and does not allocate address prefixes to other domains, N_1's downhill table will only have the following entries: $nprPf(N_1)$, 1:1:1::/96, 1:2:1::/96, each with the next hop pointing to N_1, and 1:1:1::/48 and 1:2:1::/48 each with the next hop pointing to *blackhole*. When N_1 receives a packet with a source address 1:1:1::1000 and a destination address 1:3:1::2000, it will first follow Step 1 of the forwarding algorithm, looking up the destination address in its downhill forwarding table. The destination address 1:3:1::2000 will not match any entry in N_1's downhill table, so N_1 will follow Step 2 of the forwarding algorithm. N_1's uphill table will have an entry for $nprAddr(N_1)$ with the next hop pointing to N_1's bridge table, an entry for 1:1:1::/48 with the next hop pointing to R_1, and an entry for 1:2:1::/48 with the next hop pointing to R_2. So the source address 1:1:1::1000 will match the entry for 1:1:1::/48. Henceforth, the packet will be forwarded to R_1. Similarly, at R_1, the packet will be forwarded to B_1. At B_1, as shown in Figure 5-6, B_1's downhill table will have an entry for 1:3::/32 with the next hop pointing to R_3. So the destination address 1:3:1::2000 will match this entry, and the packet will be forwarded to R_3. Similarly, at R_3, the packet will be forwarded to N_3. As N_3 is allocated the address prefix 2:1:1::/48 and the address prefix 1:3:1::/48, its downhill table will have an entry 1:3:1::/96 with the next hop pointing to itself. The destination address 1:3:1::2000 will match this entry. N_3 will know that the packet has arrived at its destination domain. By consulting its intra-domain forwarding table, N_3 is able to forward the packet to the destination Alice.

source	1:2:1::1000	
destination	$nprAddr(R_2)$	
	...	
next-to-visit	$nprAddr(R_3)$	start of the routing header
	1:3:1::2000	

Figure 5-8: This figure shows the packet header when Bob sends the packet. The network topology is shown in Figure 4-3.

source	$nprAddr(R_2)$	
destination	$nprAddr(R_3)$	
	...	
	1:2:1::1000	start of the routing header
next-to-visit	1:3:1::2000	

Figure 5-9: This figure shows the packet header after R_2 finishes executing Step 4 of the forwarding algorithm. The network topology is shown in Figure 4-3.

In the second example, we show if Bob in Figure 4-3 sends a packet to Alice with four addresses 1:2:1::1000, $nprAddr(R_2)$, $nprAddr(R_3)$, and 1:3:1::2000, the packet will be forwarded along the route $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$. The first two addresses will be put into the source and

source	$nprAddr(R_3)$	
destination	1:3:1::2000	
	...	
	1:2:1::1000	start of the routing header
	$nprAddr(R_2)$	
next-to-visit	*end*	

Figure 5-10: This figure shows the packet header after R_3 finishes executing Step 4 of the forwarding algorithm. The network topology is shown in Figure 4-3.

the destination address field of the packet header, and the other two addresses will be put into the routing header. The header of the packet sent by Bob is shown in Figure 5-8. Similar to the first example, at N_1, the packet will be forwarded to R_2 since N_1 has an entry in its uphill table for the address prefix 1:2:1::/48 with the next hop pointing to R_2. When the packet arrives at R_2, R_2 will first follow Step 1 of the forwarding algorithm, looking up $nprAddr(R_2)$ in its downhill table. A match for the entry $nprPf(R_2)$ in R_2's downhill table will be found. Step 4 will be followed, with $nprAddr(R_2)$ shifted to the source address field, $nprAddr(R_3)$ shifted to the destination address field, 1:2:1::1000 preserved in the routing header, and 1:3:1::2000 being the next-to-visit address. The shifted packet header is shown in Figure 5-9. Then R_2 will follow Step 1 of the forwarding algorithm and look up $nprAddr(R_3)$ in its downhill table, where it will find no match. Next R_2 will follow Step 2 of the forwarding algorithm, looking up $nprAddr(R_2)$ in its uphill table, where it will find a match with the next hop pointing to its bridge forwarding table. Step 3 of the forwarding algorithm will be followed, where R_2 looks up $nprAddr(R_3)$ in its bridge table. A match will be found with the next hop pointing to R_3. So the packet will be forwarded to R_3. When the packet arrives at R_3, R_3 will follow Step 1 of the forwarding algorithm, looking up $nprAddr(R_3)$ in its downhill table, where it will find a match for the entry $nprPf(R_3)$ with the next hop pointing to itself. Similarly, Step 4 of the forwarding algorithm will be followed, with $nprAddr(R_3)$ shifted to the source address field, 1:3:1::2000 shifted to the destination address field, $nprAddr(R_2)$ preserved in the routing header, and the next-to-visit address pointing to the end of the routing header. Figure 5-10 shows the packet header after R_3 executes Step 4 of the forwarding algorithm. Then R_3 will go back to Step 1 of the forwarding algorithm, where it will look up the address 1:3:1::2000 in its downhill table. A match will be found for the entry 1:3:1::/48 with the next hop pointing to N_3. The packet will be forwarded to N_3 and then be forwarded to Alice.

5.3 How a User Creates a Route Representation

We have described how a sequence of domains can be represented by two or more addresses. In NIRA, we do not restrict how a user selects a route and represents the route with multiple addresses according to our route representation scheme. Any mechanism could be developed to select a route on a user's behalf and convert the route into two or more addresses. In this section, we discuss how a user might create a route representation from the information he has obtained using our basic route discovery mechanisms, i.e., TIPP and NRLS.

From the information learned from TIPP and NRLS, a user might not know the exact sequence of domains that connect him to a destination user, because the user does not have the up-graph of the destination user. However, a user is able to create a route representation without knowing the exact route. With TIPP, a user learns his up-graph in the form of a set of link records. A link record contains the address allocation information between two adjacent domains. It also imparts the set of address prefixes a domain has. With NRLS, a user discovers the addresses of a destination user. Combining these two pieces of information, a user can tell if one of the domains in his up-graph has allocated one of the destination addresses, and which destination addresses are allocated from a top-level provider in the *Core* routing region.

Suppose there is a domain in a user's up-graph that allocates a destination address. We refer to that domain as the *rendezvous* domain. Then the user would know that there is a downhill route segment connecting the rendezvous domain and the destination, although he might not know the exact sequence of domains in the downhill route segment. If the user selects a route to reach the destination via the rendezvous domain in his up-graph, the user could tell whether the route segment from himself to the rendezvous domain has a type, or the route segment can be decomposed into a sequence of maximal segments. The end-to-end route the user has chosen is the route segment concatenated with a downhill segment from the rendezvous domain to the destination, so the user is able to tell the type of the entire route. Thus, he is able to represent the route using the route representation scheme described in Section 5.1.

For example, in Figure 4-8, after Bob learns his up-graph from TIPP and Alice's addresses from NRLS, Bob could tell that the domain R_3 with address prefixes 2:1::/32 and 1:3::/32 allocates Alice's addresses: 2:1:1::2000 and 1:3:1::2000. So Bob would know that there is a downhill route segment from R_3 to Alice. Suppose Bob decides to use the provider R_2, the peering link between R_2 and R_3, and the downhill segment from R_3 to Alice to reach the destination Alice. Bob is able to tell that the partial route from him to R_3: $N_1 \rightarrow R_2 \rightarrow R_3$ consists of an uphill segment: $N_1 \rightarrow R_2$, and a bridge segment: $R_2 \rightarrow R_3$. So the entire route from him to Alice consists of an uphill segment, a bridge segment that is outside the *Core*, and a downhill segment. The uphill segment ($N_1 \rightarrow R_2$) can be represented by 1:2:1::2000 and $nprAddr(R_2)$, and the bridge segment ($R_2 \rightarrow R_3$) can be represented by $nprAddr(R_2)$ and $nprAddr(R_3)$. Although Bob does not know the exact sequence of domains in the downhill segment from R_3 to Alice, he knows that a downhill segment can be represented by the start domain's non-provider-rooted address $nprAddr(R_3)$ and a destination address 1:3:1::2000. So Bob is able to compose a route representation with the addresses: 1:2:1::2000, $nprAddr(R_2)$, $nprAddr(R_3)$, and 1:3:1::2000 and send packets to Alice.

Similarly, if a destination address is allocated from the address space of a top-level provider in the *Core*, although the user might not see the top-level provider in his up-graph, the user could tell that there is a downhill route segment from the top-level provider in the *Core* to the destination. If the user chooses a route that goes up to a top-level provider in the *Core* to reach the destination, he could combine his address allocated from that top-level provider and the destination address to compose a hill route. For example, in Figure 4-8, from NRLS, Bob knows that Alice has two addresses 1:3:1::2000 and 2:1:1::2000 that are both allocated from the *Core* routing region. So if Bob selects to use the provider R_1 and B_1 to reach Alice, he could specify a route either using a source address 1:1:1::1000 and a destination 1:3:1::2000, or using a source address 1:1:1::1000 and a different destination address 2:1:1::2000.

5.4 Route Representation for a Reply Packet

When a receiver receives a packet, it may want to send a reply back to the sender. Multiple mechanisms are possible for a receiver to generate a return route representation. For example, if the receiver knows the sender's name, it can follow the procedure described in Chapter 4 to perform its own route discovery and pick a return route to send the reply; or a sender could attach its addresses or topology information in its initial packet to the receiver so that the receiver can select a return route without invoking a route discovery process.

We provide a basic mechanism for a receiver to generate a return route from the packet it receives. A return route can be generated by simply swapping the source and the destination address in the received packet, and reversing the sequence of addresses in the routing header if the received packet has one. For example, in Figure 4-3, if Bob sends Alice a packet with a source address 1:1:1::1000 and a destination address 1:3:1::2000, the packet will follow the domain-level route $N_1 \rightarrow R_1 \rightarrow B_1 \rightarrow R_3 \rightarrow N_3$ to reach Alice. When Alice receives the packet, she can send a reply by swapping the source and the destination address, i.e., using 1:3:1::2000 as the source address, 1:1:1::1000 as the destination address. The reply packet will follow the route $N_3 \rightarrow R_3 \rightarrow B_1 \rightarrow R_1 \rightarrow N_1$ to reach Bob. If Bob sends Alice a packet with 1:2:1::1000 in the source address field, $nprAddr(R_2)$ in the destination address field, and $nprAddr(R_3)$ and 1:3:1::2000 in the routing header, the packet will follow the route $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$ to reach Alice. The packet header is shown in Figure 5-8. When Alice receives the packet, according to our forwarding algorithm, $nprAddr(R_3)$ will be in the source address field, 1:3:1::2000 in the destination address field, and 1:1:1::1000 and $nprAddr(R_2)$ in the routing header. The received packet header is shown in Figure 5-11. Alice could generate a return route representation by swapping $nprAddr(R_3)$ and 1:3:1::2000, and reversing the sequence of 1:2:1::1000 and $nprAddr(R_2)$. The reply packet header sent by Alice is shown in Figure 5-12. It can be seen that the reply packet will follow the route $N_3 \rightarrow R_3 \rightarrow R_2 \rightarrow N_1$.

source	$nprAddr(R_3)$	
destination	1:3:1::2000	
	...	
	1:2:1::1000	start of the routing header
	$nprAddr(R_2)$	
next-to-visit	*end*	

Figure 5-11: The packet header when Alice receives the packet. The network topology is shown in Figure 4-3.

source	1:3:1::2000	
destination	$nprAddr(R_3)$	
	...	
next-to-visit	$nprAddr(R_2)$	start of the routing header
	1:2:1::1000	

Figure 5-12: The reply packet header sent by Alice to Bob. The network topology is shown in in Figure 4-3.

5.5 Route Representation for an ICMP Error Notification Packet

NIRA provides a reactive mechanism for a user to detect route failures. When a router forwards a packet along the route specified in a packet header, if the router notices a failure on the route, the router should make its best effort to send an ICMP error notification packet to the sender of the packet.

We describe how a route representation for an ICMP packet might be composed from a received packet. The key problem to address is how a router picks the source address of an ICMP packet. The source address will be combined with the other addresses in a received packet to generate a reverse route representation from the router to the sender of the received packet. In general, the router at a domain needs to figure out the domain's position in the route the packet is supposed to follow, e.g., whether the domain is on the uphill portion or on the downhill portion. A router could figure out its position based on at which step of the forwarding algorithm the ICMP packet is triggered. Then the router could pick a source address that is compatible with its domain's position in the route. We briefly describe how a router might pick a compatible source address.

A router might want to send an ICMP packet due to route failures in the following cases:

1. At Step 1, the router finds that the destination address in a packet header matches an entry with the next hop pointing to *blackhole*.
2. At Step 2, the router finds that the source address in a received packet matches no entry in its uphill forwarding table.
3. At Step 3, the router finds that the destination address in the received packet has no match

in its bridge forwarding table or in its routing table.

In the first case, a *blackhole* indicates that the destination address matches an address prefix of the domain of the router, but the router is unable to find a subdivision of the address prefix to match the destination address. So the domain of the router is on the allocation path of the destination address, i.e., the downhill portion of the route a received packet is supposed to follow. The reverse route from the router to the sender is either an uphill route, or a hill route. In general, the router could send an ICMP packet with its address that shares the longest prefix with the destination address in the received packet as the source address, the address in the source address field of the received packet as the destination address. If the received packet has a routing header, the router should reverse the sequence of the visited addresses in the routing header and insert them into the routing header of the ICMP packet.

For example, in the network shown in Figure 4-3, if the connection between R_3 and N_3 is broken, the downhill table of R_3 might not have an entry for the prefix 2:1:1::/48 and the prefix 1:3:1::/48 allocated to N_3. When a packet sent by Bob with a source address 1:2:1::1000 and a destination address 2:1:1::2000 arrives at R_3, R_3 will find that the destination address 2:1:1::2000 matches the prefix 2:1::/32 with the next hop pointing to *blackhole*. R_3 could then send an ICMP packet back to the sender Bob with the source address 2:1:: and 1:1:1::1000. The ICMP packet will follow the route $R_3 \rightarrow B_2 \rightarrow B_1 \rightarrow R_1 \rightarrow N_1$ to reach Bob.

In the second case, a router at a domain finds no match for the source address of a received packet in its uphill forwarding table. This situation usually indicates that the domain of the router is on the uphill portion of the route the received packet is supposed to follow, but the connection between the domain and the provider that allocates the address prefix containing the source address is broken. The router could send an ICMP packet with its non-provider-rooted address as the source address, and the address in the source address field of the received packet as the destination address. If the received packet has a routing header, the router should insert the reverse sequence of the visited addresses in the routing header into the ICMP packet.

In the sample network shown in Figure 4-3, if the connection between R_1 and B_1 is broken, the uphill forwarding table of R_1 might not have an entry 1:1::/32 with a next hop pointing to B_1. When a packet sent by Bob with a source address 1:1:1::1000 and a destination address 2:1:1::2000 arrives at R_1, R_1 will find no match for the packet in its uphill table. R_1 could then send an ICMP packet back to the sender Bob with the source address $nprAddr(R_1)$, and the destination address 1:1:1::1000. The ICMP packet will follow the route $R_1 \rightarrow N_1$ to reach Bob.

In the third case, a router at a domain finds no match for the destination address of a received packet in its bridge forwarding table, or in its routing table. A no-match in a router's bridge forwarding table indicates that the domain of the router is the first domain of a bridge segment connected by TIPP. So the router must have executed Step 2 of the forwarding algorithm and found that the address in the source address field of the packet[5] matches its non-provider-rooted address prefix. The router could send an ICMP packet with its non-provider-rooted address as the source address, the address before the next-to-visit address in the routing header of the received packet as the destination address. The router should reverse the rest of the visited addresses in the routing header of the received packet and insert them into the routing header of the ICMP packet.

For example, in the network shown in Figure 4-3, if the connection between R_2 and R_3 is broken, R_2's bridge table might not have an entry for the non-provider-rooted address of R_3. When a packet sent by Bob with a header shown in Figure 5-8 (i.e., with the source address 1:2:1::1000, the destination address $nprAddr(R_2)$, and $nprAddr(R_3)$ and 1:3:1::2000 in the routing header) arrives at R_2, R_2 will first execute Step 1 and 4 of the forwarding algorithm, and then go back to Step 1 with $nprAddr(R_2)$ in the source address field of the received packet, $nprAddr(R_3)$ in the destination address field of the received packet, and 1:2:1::1000 and 1:3:1::2000 in the routing header with the next-to-visit pointer pointing to 1:3:1::2000. With this shifted packet header as shown in Figure 5-9, R_2 will execute Step 3 of the forwarding algorithm and find no match for $nprAddr(R_3)$ in the destination address field. R_2 could then send an ICMP with $nprAddr(R_2)$ as the source address, and the address before the next-to-visit address, 1:2:1::1000, as its destination address. The ICMP packet will follow the route $R_2 \rightarrow N_1$ to reach Bob.

In the case that a router finds a no-match in its routing table, the router must be at a top-level provider in the *Core* routing region, and the source address in the received packet is a routable address in the *Core* routing region. To send an ICMP packet, the router could use its routable address in the *Core* routing region as the source address, the address in the source address field of the received packet as the destination address. If the received packet has a routing header, the router should reverse the visited addresses in the routing header and insert them into the routing header of the ICMP packet. For example, in the network shown in Figure 4-3, if due to failures, the top-level provider B_1 cannot reach the top-level provider B_2, then the routing table of B_1 may not have an entry 2::/16 with a valid next hop to reach B_2. When B_1 receives a packet sent by Bob with an address 1:1:1::1000 and 2:1:1::2000, B_1 will reach Step 3 of the forwarding algorithm, and finds no match for the destination address in its routing table. B_1 could generate an ICMP packet with the source address 1:: and the destination address 1:1:1::1000. The ICMP packet will follow the route $B_1 \rightarrow R_1 \rightarrow N_1$ to reach Bob.

5.6 Optimization

In our basic route representation scheme, a valley-free route whose uphill segment and downhill segment intersects at a common domain or are connected by a routing protocol in the *Core* can be represented by a source and a destination address. A valley-free route that includes a

[5] Due to the shift in Step 4 of our forwarding algorithm, this address might not be the original source address in the received packet.

peering connection outside the *Core* is represented by at least four addresses. This type of route is not represented by two addresses because there are insufficient addresses to represent routes. A hierarchically allocated address can be uniquely mapped into an allocation path that starts at a top-level provider. But the allocation paths of two addresses may interconnect at various places, e.g., in the *Core*, at a common domain, or via a peering connection. In our basic route representation scheme, we choose to use two hierarchically allocated addresses to represent the route in which the allocation paths of the two addresses are either connected in the *Core* or intersect at a common domain.

It is possible to optimize our route representation scheme so that the valley-free route that consists of a peering connection outside the *Core* can also be represented by a source and a destination address. If two domains outside the *Core* have a peering connection, each domain could obtain a private but globally unique address space[6] and allocate address prefixes from this address space to its customers. Similarly, the customers will allocate subdivisions of these address prefixes to their customers, and so on. The allocation results will be propagated by TIPP from providers to customers.

We can extend our route representation scheme to allow a source and a destination address to represent a valley-free route that consists of a peering connection outside the *Core*, if the source address is allocated from the private address space of one domain of the peering connection, and the destination address is allocated from the private address space of the other domain of the peering connection. As an example, Figure 5-13 shows the same network as shown in Figure 4-3, with R_2 and R_3 each obtaining a private address space: FFFF:1::/32 and FFFF:2::/32. R_2 allocates an address prefix from this address space to each of its customers N_1 and N_2. So does R_3. The user Bob obtains an address FFFF:1:1::1000 allocated from R_2's private address space, and the user Alice obtains an address FFFF:2:1::2000 from R_3's private address space. If Bob wants to send a packet to Alice via the route $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$, Bob could use a source address FFFF:1:1::1000 and a destination address FFFF:2:1::2000 to represent the route.

To make our forwarding algorithm work with this optimization, we need to add entries for address prefixes allocated from a domain's private address space into routers' forwarding tables. A hierarchical address allocated from a private address space could be treated the same as a hierarchical address allocated from a globally unique address space that belongs to a top-level provider in the *Core* routing region. A router will announce such prefixes using the adjacent link records in TIPP to its neighbors in the same way as it announces other hierarchically allocated prefixes, except that for a domain that is the root of a private address space, the router in the domain will not announce its private address space to its providers. Accordingly, a router could set up entries in its downhill table or its uphill table for an address prefix allocated from a private

[6]We use the world "private" to imply that the address space will not be leaked into the *Core* routing region. So it is not globally routable. But it should be globally unique so that an address allocated from this address space could uniquely identify an allocation path.

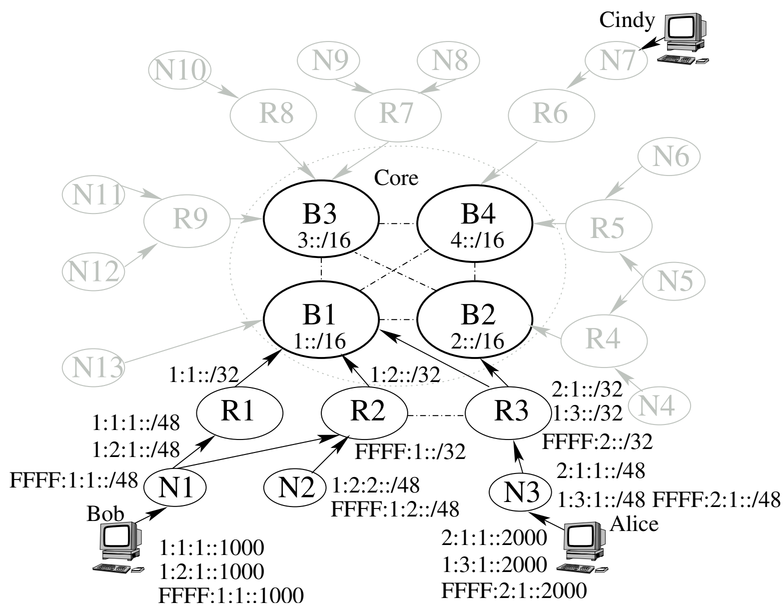


Figure 5-13: R_2 and R_3 each obtain a private address space: FFFF:1::/32 for R_2 and FFFF:2::/32 for R_3, and allocates an address prefix from the private address space to their customers. So Bob gets an address FFFF:1:1::1000 and Alice gets an address FFFF:2:1::2000. This optimization allows a valley-free route that contains a peering connection outside the *Core* to be represented by two addresses. The route $N_1 \rightarrow R_2 \rightarrow R_3 \rightarrow N_3$ can be represented as FFFF:1:1::1000 and FFFF:2:1::2000.

address space in the same way as it sets up entries for other hierarchically allocated addresses. For instance, if a domain is allocated an address prefix pf from the private address space of its provider, the router in the domain will insert pf into its downhill table with the next hop pointing to *blackhole* and insert pf into its uphill table with the next hop pointing to its provider. The router will also pick an inter-domain address from pf and insert it into its downhill table with the next hop pointing to itself.

How the router in a domain sets up its bridge table is slightly different when there are private addresses. If a domain does not have an address allocation relationship with a TIPP neighbor, and the neighbor announces to the domain a private address prefix to which the neighbor is the root[7], then the router in the domain will insert the private address prefix of the neighbor into its bridge table, with the next hop pointing to the neighbor.

For a domain that is the root of a private address space, the router in the domain will insert an entry for the private address space in its uphill table with the next hop pointing to the router's bridge table. This entry indicates that a packet with a source address allocated from the private address space should not be pushed up further. Instead, the packet should be forwarded across a bridge segment.

R_2's downhill table

$nprPf(R_2)$	$self$
1:2::/96	$self$
1:2::/32	*blackhole*
1:2:1::/48	N_1
1:2:2::/48	N_2
FFFF:1::/96	$self$
FFFF:1::/32	*blackhole*
FFFF:1:1::/48	N_1
FFFF:1:2::/48	N_2

R_2's uphill table

$nprPf(R_2)$	*bridge*
1:2::/32	B_1
FFFF:1::/32	*bridge*

R_2's bridge table

$nprAddr(R_3)$	R_3
FFFF:2::/32	R_3

Figure 5-14: This figure shows the contents of R_2's forwarding tables after we optimize our route representation scheme.

Figure 5-14 shows the contents of R_2's forwarding tables after we optimize our route representation scheme. When Bob sends a packet with a source address FFFF:1:1::1000 and a destination address FFFF:2:1::2000, the packet will be forwarded to R_2, since N_1's uphill table will have an entry for FFFF:1:1::/48 with the next hop pointing to R_2. R_2 will execute Step 1 of the

[7]A domain could learn whether a neighbor is the root of an address prefix from the allocation-relation field of the prefix announcement in a link record (Appendix 4.B.5). A specific value of the allocation-relation field indicates whether the neighbor is the root of the address prefix.

forwarding algorithm, looking up the destination address in its downhill table, where it will find no match. Then R_2 will execute Step 2 of the forwarding algorithm, looking up the source address FFFF:1:1::1000 in its uphill table, where it will find a match with the next hop pointing to its *bridge* table. Afterwards, R_2 will look up the destination address FFFF:2:1::2000 in its bridge table, where it will find a match with the next hop pointing to R_3. So R_2 will forward the packet to R_3. Similarly, at R_3, the packet will be forwarded to N_3, and then be forwarded to its destination Alice.

This optimization simplifies the representation for a valley-free route that consists of a peering connection outside the *Core*. The tradeoff is that it increases the number of addresses a user has.

5.6.1 Multiple Routing Regions

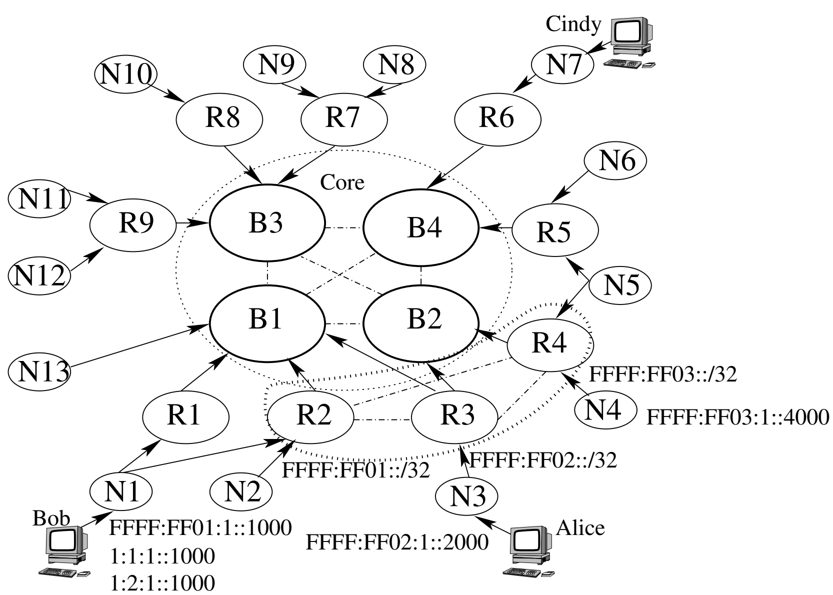


Figure 5-15: R_2, R_3, and R_4 may decide to form a routing region. The routing region is assigned an address space. Each domain obtains a unique address prefix from that address space, and recursively allocate the address prefix to its customers. R_2 obtains an address prefix FFFF:FF01::/32, and R_3 obtains an address prefix FFFF:FF02::/32. The user Bob gets an address FFFF:FF01:1::1000, and the user Alice gets an address FFFF:FF02:1::2000. With this address allocation scheme, a valley-free route that includes a bridge segment connected by a routing protocol can be represented by two addresses. The route $N_1 \rightarrow R_2 \rightsquigarrow R_3 \rightarrow N_3$ between Bob and Alice can be represented by FFFF:FF01:1::1000 and FFFF:FF02:1::2000.

Our route representation scheme and forwarding algorithm can be extended to support multiple routing regions. It is possible that a group of domains outside the *Core* also desire to form a routing region. For example, in Figure 5-15, R_2 and R_3 may notice that their customers send a lot of traffic to R_4. So it would be profitable for them to have a direct peering connection with

R_4 instead of sending traffic to their providers. The three domains, R_2, R_3, and R_4 each have a peering connection with the others. They may decide to negotiate a business agreement such that if the peering connection between any two domains is broken, the third domain would provide transit service between the other two domains. The three domains may decide to run a routing protocol and not to let a user pick the route segment that connects them.

If a private address space is allocated to a routing region, and each domain in the routing region obtains a unique address prefix from the address space, and allocates subdivisions of the address prefix to its customers, and the customers recursively allocate the subdivisions to their customers, then a valley-free route that includes a bridge segment connected by the routing protocol in the routing region can also be represented by two addresses. In Figure 5-15, if the address prefix FFFF:FF00::/24 is assigned to the routing region that includes R_2, R_3, and R_4, and each of the domains obtains a unique address prefix respectively: R_2 gets FFFF:FF01::/32, R_3 gets FFFF:FF02::/32, and R_4 gets FFFF:FF03::/32. The customer of R_2: N_1, obtains an address prefix FFFF:FF01:1::/48. The user Bob obtains an address FFFF:FF01:1::1000. Similarly, the user Alice obtains an address FFFF:FF02:1::2000. Then the route $N_1 \rightarrow R_2 \rightsquigarrow R_3 \rightarrow N_3$ between Bob and Alice can be represented by FFFF:FF01:1::1000 and FFFF:FF02:1::2000.

Similar to a router at a top-level provider inside the *Core*, a router at a domain inside a routing region would add a special entry for the address prefix of the routing region in its uphill table, with the next hop pointing to the routing table of the routing region. For example, the router at R_2 will have an entry for FFFF:FF00::/24 in its uphill table, with the next hop pointing to its routing table. When the router receives a packet with a source address FFFF:FF01:1::1000 and a destination address FFFF:FF02:1::20000, it will first look up the destination address in its downhill table, where it finds no match; then it will look up the source address in its uphill table, and follow the next hop to look up the destination address in its routing table. There it will find a match and forward the packet to the next hop computed by the routing protocol in the routing region.

5.6.2 Source Address Compatibility

In Section 5.2.2, when we discuss our design rationale for the forwarding algorithm, we mention that there are two choices to make a TIPP router that also participates in a routing protocol to look up the destination address of a packet in its routing table. The first option is to make the router look up the destination address right after the router cannot find a match for the destination address in its downhill table. The second option is to make the router do so after it looks up the source address in its uphill table.

We choose the second option because it checks for the forwarding reversibility in a routing region. When there is only a *Core* routing region, the importance of this checking is not that obvious, because if a packet is forwarded all the way up to a top-level provider, then its source address must be an address allocated from the top-level provider's address space. So it is not necessary to check for the source address again. The checking is only useful when a node inside a top-level provider sends a packet. It is possible for the node to use its non-provider-rooted address as the source address of the packet and inject the packet into the *Core*.

With multiple routing regions, as shown in Figure 5-15, this checking becomes more important. A user Bob, due to either mistake or malice, may use a source address 1:2:1::1000 and a destination address FFFF:FF02:1::2000 to send a packet to Alice. When this packet arrives at R_2, if R_2 looks up its routing table right after it cannot find a match for the destination address FFFF:FF02:1::2000, the packet will be forwarded to R_3, and then reach Alice. However, Alice cannot send a reply back to Bob with the source address FFFF:FF02:1::2000 and the destination address 1:2:1::1000. As R_3 is the root domain that allocates the source address FFFF:FF02:1::2000, the reply will not be further pushed up from R_3, but R_3 will not have forwarding information for the destination address 1:2:1::1000.

We consider this mismatch between a source and a destination address harmful. It consumes network resource to forward a packet. If a reply packet cannot be sent back to the sender of the packet, a connection might not be established, thus wasting network resource in vain. Therefore, in our design, we make a router check for the routability of the source address of a packet before the router looks up the destination address in its routing table.

5.7 Forwarding Cost Analysis

Our route representation scheme is able to represent a common type of domain-level route with only two addresses, regardless how many domains the route consists of. As a tradeoff, our forwarding algorithm needs to inspect not only the destination address of a packet, but sometimes the source address of a packet. In this section, we analyze the computational cost for our forwarding algorithm to find the next hop to forward a packet.

Suppose the computational cost for a longest prefix match is L, the cost for one header field rewritten is M. For a packet that has only a source and a destination address, in the best case, our forwarding algorithm will find the next hop to forward the packet with one lookup (L) in a router's downhill table; in the worst case, our forward algorithm will find the next hop with three lookups ($3L$): one lookup in a router's downhill table, one lookup in a router's uphill table, and one lookup in a router's bridge or routing table. So the cost for finding the next hop to forward a packet with a source and a destination address ranges from L to $3L$.

For a packet that contains an optional routing header, the cost for looking up the next hop may include header rewritten and additional longest prefix lookups. As the domain of a router might be both the end domain of a route segment and the start domain of the next route segment, in the worse case, a route representation may contain two addresses of the same domain. When a packet reaches that router, Step 4 of the forwarding algorithm will be executed twice. Each execution includes a longest prefix lookup (L) for the destination address in the router's downhill table, and four header fields rewritten ($4M$): the source address field, the destination address field, the next-to-visit address field, and the next-to-visit address pointer. After executing Step 4 twice, the forwarding algorithm may spend additional three lookups ($3L$) to find the next hop in the router's bridge or routing table. So the worst case cost for looking up the next hop for a packet with an optional routing header is $3L + 2(L + 4M) = 5L + 8M$.

Therefore, the cost for finding the next hop to forward a packet ranges from L to $5L + 8M$. We expect that in most common cases, packets will have no routing header, because a typical Internet route is valley-free, and a valley-free route can be represented by two addresses. So the forwarding cost in most cases will range from L to $3L$. We do not expect this computational cost will become a problem. Advanced hardware technology such as parallel lookup on the destination address field and the source address field may speed up the lookup latency. In addition, forwarding tables of a TIPP router only contain entries for its own address prefixes and those of its neighbors, and therefore are expected to be much smaller than BGP tables. A small table size may also help to reduce the longest prefix lookup latency (L).

Appendix 5.A Correctness of the Forwarding Algorithm

By correctness, we mean that our forwarding algorithm satisfies the following design requirements: if a user specifies a route following the route representation scheme described in Section 5.1, then

1. Correct forwarding: the packet should be forwarded along the route to reach its destination if the route is failure free.

2. Drop on failure: if at a router, the next hop to forward a packet is unreachable due to failures, the packet will be dropped at that router instead of looping around in the network.

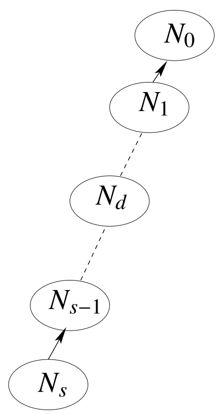
3. Step-wise reversibility: if at domain N, the packet is forwarded to a neighbor domain M, then at domain M, a packet with the reverse route representation will be forwarded to N.

In this section, we verify that our forwarding algorithm is correct. We focus on showing that the first two conditions are satisfied, as reversibility is obvious due to the symmetry of the forwarding algorithm and the route representation scheme, and we have discussed reversibility in the main text Section 5.2.2.

We first verify that the correctness of the forwarding algorithm for a route segment that has a type, i.e., an uphill route segment, a downhill route segment, a bridge route segment, or a hill route segment.

5.A.1 An Uphill Route Segment

According to our route representation scheme, an uphill route segment is represented by two addresses: a source address S_a whose allocation path overlaps with the uphill route segment, and a destination address D_a that is the non-provider-rooted address of the destination domain. Suppose the source address allocation path is $N_0 \rightarrow N_1 \rightarrow N_2 \rightarrow ... \rightarrow N_i ... \rightarrow N_s$, where N_s is the source domain, and N_0 is the top-level provider that allocates the source address. Without loss of generality, suppose N_d is the destination domain, where $0 \leq d \leq s$, and $D_a = nprAddr(N_d)$. Let $Addr(N_0, N_1, N_2, ..., N_s)$ denote the address of N_s that is allocated along the path $N_0 \rightarrow N_1 \rightarrow N_2 ... \rightarrow N_s$. An uphill segment and its route representation are shown in Figure 5-16.



source	$Addr(N_0, N_1, N_2, ..., N_d, ..., N_{s-1}, N_s)$
destination	$nprAddr(N_d)$

Figure 5-16: This figure shows an uphill route segment and its representation.

We will show that at each N_i, $d < i \leq s$, if the connection between N_i and N_{i-1} is up, the packet will be forwarded to N_{i-1}. If the connection between N_i and N_{i-1} is down, the packet will be dropped and an ICMP message will be sent.

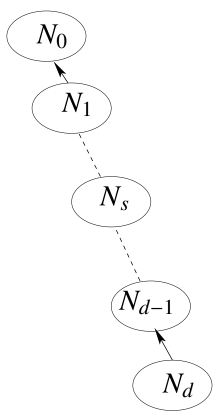
When a packet with a source address $Addr(N_0, N_1, N_2, ..., N_s)$ and a destination address $nprAddr(N_d)$ arrives at N_i, since $nprPf(N_d)$ will only appear in N_d's downhill table, N_i will not find a match for the destination address in its downhill table. N_{i-1} has allocated to N_i an address prefix pf that encloses the source address $Addr(N_0, N_1, N_2, ..., N_d, ..., N_s)$. So if the connection between N_i and N_{i-1} is up, N_i's uphill table will have an entry for the address prefix pf with the next hop pointing to N_{i-1}. The source address $Addr(N_0, N_1, N_2, ..., N_d, ..., N_s)$ of the packet will match the entry. If there is another longer prefix *bad* that matches the source address with the next hop pointing to a different next hop M, $M \neq N_{i+1}$, then we have pf encloses *bad*, but they are both address prefixes allocated to N_i, which violates the non-looping address allocation rule as stated in Section 4.2.4. Therefore, pf must be the longest prefix that matches the source address. So the packet will be forwarded to N_{i-1}.

If the connection between N_{i-1} and N_i is down, then the address pf that encloses the source address $Addr(N_0, N_1, N_2, ..., N_d, ..., N_s)$ may not be present in N_i's uphill table. Then the lookup on the source address in N_{i-1}'s uphill table will find no match. If there is a match for the prefix *bad*, then either *bad* is longer than pf or pf is longer than *bad*. In either case, the non-looping address allocation rule is violated. Therefore, the packet will be dropped and an ICMP message will be sent.

We also need to show that when the packet has arrived at N_d, it will stay in N_d. Since $nprPf(N_d)$ will appear in N_d's downhill table with the next hop pointing to N_d itself, when N_d receives the packet and executes Step 1 of the forwarding algorithm, N_d will find that the destination address $nprAddr(N_d)$ matches the prefix $nprPf(N_d)$, hence recognizing the packet has arrived at its destination domain.

5.A.2 A Downhill Route Segment

A downhill segment is the reverse of an uphill segment, and so is its representation. A downhill segment is represented by a source address which is the non-provider-rooted address of the source domain, and a destination address whose allocation path overlaps with the downhill segment. Let $N_0 \rightarrow N_1 \rightarrow ... \rightarrow ...N_i \rightarrow N_d$ be the destination address allocation path, N_s be the source domain, where $0 \leq s \leq d$. A downhill segment and its representation are shown in Figure 5-17.



source	$nprAddr(N_s)$
destination	$Addr(N_0, N_1, N_2, ..., N_s, ..., N_{d-1}, N_d)$

Figure 5-17: This figure shows a downhill route segment and its representation.

Similarly, first, we want to show that at N_i, $s \leq i < d$, if the connection between N_i and N_{i+1} is up, a packet with a route representation shown in Figure 5-17 will be forwarded to N_{i+1}; if the connection between N_i and N_{i+1} is down, the packet will be dropped and an ICMP packet will be sent. N_i has allocated to N_{i+1} an address prefix pf that encloses $Addr(N_0, N_1, N_2, ..., N_s, ..., N_d)$. So if the connection between N_i and N_{i+1} is up, then in N_i's downhill table, there must be an entry for pf with the next hop pointing to N_{i+1}, and the destination address $Addr(N_0, N_1, N_2, ..., N_s, ..., N_d)$ will match the entry.

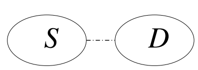
When N_i executes Step 1 of the forwarding algorithm, if there is another longer prefix *bad* that matches the destination address with the next hop pointing to a different next hop M, $M \neq N_{i-1}$, then we have both pf, a prefix allocated to N_{i+1} by N_i, and *bad*, a prefix allocated to M by N_i, both enclose the destination address, which violates the non-overlapping address allocation rule as stated in Section 4.2.4. Therefore, pf must be the longest prefix that matches the destination address. So the packet will be forwarded to N_{i+1}.

If the connection between N_i and N_{i+1} is down, then the entry for pf may not be present in N_i's downhill table. The destination address cannot match any other address prefixes allocated to N_i's other neighbors, because it will violate the non-overlapping allocation rule. However, it will match N_i's own address prefix with the next hop pointing to *blackhole*. Therefore, the packet will be dropped and an ICMP packet will be sent.

At the destination domain N_d, N_d's downhill table will have an entry for the inter-domain address prefix that encloses the destination address $Addr(N_0, N_1, N_2, ..., N_s, ..., N_d)$ with the next hop pointing to N_d itself. The destination address will match that entry. Hence, the packet will stay at N_d.

5.A.3 A Bridge Segment

A bridge segment consists of two domains that have no address allocation relationship, and can be represented by a source address that is the source domain's non-provider-rooted address *nprAddr*(S) and a destination address that is the destination domain's non-provider-rooted address *nprAddr*(D). If S and D are top-level providers that are connected by the routing protocol running inside the *Core*, then the source address would be *Addr*(S), the address of S derived from the globally unique address prefix owned by S, and the destination address would be *Addr*(D), the address of D derived from the globally unique prefix owned by D. Figure 5-18 shows a bridge segment and its representation.



source	*nprAddr*(S)
destination	*nprAddr*(D)

Figure 5-18: This figure shows a bridge route segment and its representation.

Let's first consider the case where S and D are connected by a TIPP connection. At the source domain S, when S executes the first step of the forwarding algorithm, looking up the destination address *nprAddr*(D) in its downhill table, it will find no match for *nprAddr*(D), because the downhill table will only contain S's address prefixes, S's own non-provider-rooted address prefix, and prefixes allocated to S's neighbors. S will move to Step 2 of the forwarding algorithm, looking up the source address *nprAddr*(S) in its uphill table. There, it will find a match for *nprAddr*(S) with the next hop pointing to *bridge*. So S will follow the third step of the forwarding algorithm, looking up the destination address *nprAddr*(D) in its bridge table.

At this stage, if the connection between S and D is up, then S will have an entry for $nprPf(D)$ in its bridge table with the next hop pointing to D. So the destination address will match this entry and the packet will be forwarded to D. If the connection between S and D is down, then S will not find a match for the destination address in its bridge table. The packet will be dropped and an

ICMP packet will be sent.

If S and D are connected by the routing protocol running in the *Core*, and there is no route failure, S's routing table will have an entry for the globally unique prefix pf of domain D with the next hop pointing to D. When S receives a packet with a source address $Addr(S)$ and a destination $Addr(D)$, S will first execute Step 1 of the forwarding algorithm, looking up the destination address in its downhill table. There it will find no match. Then S will execute Step 2 of the forwarding algorithm, looking up the source address in its uphill table. There it will find a match with the next hop pointing to *routing*. Therefore, S will look up the destination address in its routing table, and find a match for the destination address $Addr(D)$. Again, if there is another longer prefix match with a different next hop, then the non-overlapping address allocation rule is violated. So the packet will be forwarded to D.

If S and D are normally connected by the routing protocol, but are temporarily disconnected due to failures, then at Step 3 of the forwarding algorithm, S will find no match for the destination address in its routing table because of the non-overlapping address allocation rule. The packet will be dropped and an ICMP message will be sent.

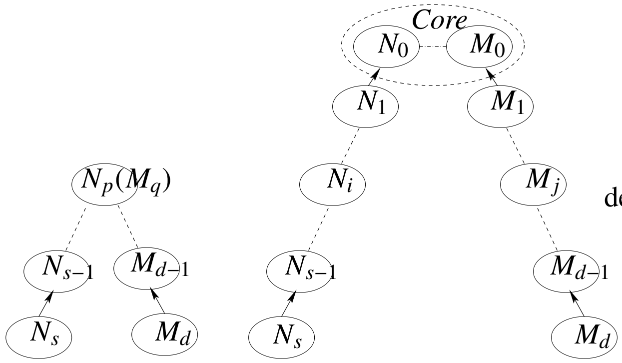
The destination domain D will have an entry for its own address in its downhill table. So when the packet has arrived at D, D will recognize that the packet has arrived at its destination domain.

5.A.4 A Hill Segment

A hill route segment consists of at most an uphill segment, a bridge segment connected by the routing protocol in the *Core*, and a downhill segment. One of these three segments could be missing. A hill segment could be represented by a source address whose allocation path overlaps with the uphill section of the segment, and a destination address whose allocation path overlaps with the downhill section of the segment. Suppose the source address allocation path is $N_0 \rightarrow N_1 \rightarrow ... \rightarrow N_s$, and the destination address allocation path is $M_0 \rightarrow M_1 \rightarrow ... \rightarrow M_d$. If the hill segment does not contain a bridge segment, then the two allocation paths intersect at a common domain N_p, with $N_p = M_q$, $0 \leq p \leq s$ and $0 \leq q \leq d$. Examples of typical hill segments are shown in Figure 5-19. If a hill segment does not have an uphill section, then $s = 0$, as shown in Figure 5-20; if it does not have a downhill section, then $d = 0$, as shown in Figure 5-21. A hill segment can be represented by two addresses: $Addr(N_0, N_1, ..., N_s)$ and $Addr(M_0, M_1, M_2, ..., M_d)$, as shown in Figure 5-19, Figure 5-20, and Figure 5-21.

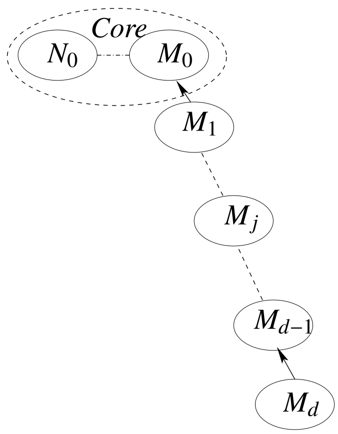
We first show that if the hill route segment is failure free, a packet with a source address and a destination address that represent the hill segment will be forwarded along the route segment and reach the destination domain. If there is a failure on the route, the packet will be dropped and an ICMP message will be sent.

It can be seen that at N_i, where N_i is neither the top-level domain that allocates the source



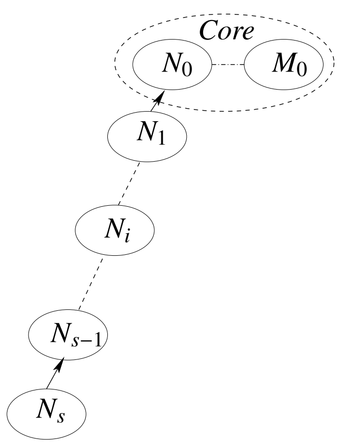
source	$Addr(N_0, N_1, N_2, ..., N_s)$
destination	$Addr(M_0, M_1, M_2, ..., M_d)$

Figure 5-19: This figure shows two typical hill route segments and their representations.



source	$Addr(N_0)$
destination	$Addr(M_0, M_1, M_2, ..., M_d)$

Figure 5-20: A hill route segment without an uphill portion and its representation.



source	$Addr(N_0, N_1, N_2, ..., N_s)$
destination	$Addr(M_0)$

Figure 5-21: A hill route segment without a downhill portion and its representation.

address, nor the common domain N_p (or M_q) that is on the allocation paths of both the source and the destination address, then N_i's downhill table will not have a matched entry for the destination address $Addr(M_0, M_1, M_2, ...M_d)$. Otherwise, either N_p allocates overlapping address prefix to N_{p+1} and M_{q+1}, or N_0 and M_0 have overlapping address space, which violates the non-overlapping address allocation rule. From our proof on the uphill route segment representation, it can be seen that the packet will be forwarded along the uphill segment $N_s \rightarrow N_{s-1} \rightarrow ..$ until it reaches N_0, or N_p if there is a common domain on the allocation paths of the source and the destination address, or the packet will be dropped if there is failure.

If the packet has reached the common domain N_p, then since N_p has allocated an address prefix to M_{q+1} that encloses the destination address $Addr(M_0, M_1, M_2, ..., M_d)$, according to our proof on the downhill route segment representation, the packet will be forwarded along the downhill segment $M_q \rightarrow M_{q+1} \rightarrow ...M_d$ and reaches the destination domain.

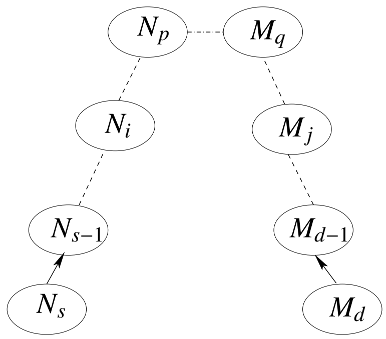
If the packet has reached N_0, and $N_0 \neq M_0$, then N_0's routing table will contain a prefix entry for the globally unique prefix of M_0, if N_0 and M_0 are connected by the routing protocol. That entry will be the longest matched prefix entry for the destination address according to the non-overlapping address allocation rule. Therefore, the packet will be forwarded to M_0. If N_0 and M_0 are temporarily disconnected due to failures, then the N_0 will not find a match for the destination address in its routing table. Therefore, the packet will be dropped and an ICMP message will be sent.

When the packet has reached M_0, according to our proof on the downhill route segment representation, the packet will be forwarded along the downhill segment and reach the destination domain, or be dropped if there is a failure.

5.A.5 Any Route Segment

Any route segment can be decomposed into a sequence of route segments of basic types, with the end domain of a previous route segment being the start domain of a next segment. The route segment can be represented by concatenating the representation of each route segment of a basic type, and deleting duplicate adjacent addresses. Figure 5-22 shows an example of a compound route segment and its representation. That route segment consists of an uphill segment $N_s \rightarrow N_{s-1} \rightarrow ...N_p$ which can be represented by $Addr(N_0, N_1, N_2, ..., N_s)$ and $nprAddr(N_p)$, a bridge segment $N_p \rightarrow M_q$, which can be represented by $nprAddr(N_p)$ and $nprAddr(M_q)$, and a downhill segment which can be represented by $nprAddr(M_q)$ and $Addr(M_0, M_1, M_2, ...M_d)$.

In a packet header with a route representation of a compound route segment, the starting source and the destination address together represent the first route segment the packet should follow. Since we have shown that the route representation of a route segment of any type works correctly with the forwarding algorithm, to show that for any route segment, we only need to demonstrate the following:



source	$Addr(N_0, N_1, N_2, ..., N_s)$
destination	$nprAddr(N_p)$
	...
next-to-visit	$nprAddr(M_q)$
	$Addr(M_0, M_1, M_2, ..., M_d)$

Figure 5-22: This figure shows an arbitrary route segment and its representation.

- When a packet arrives at the end domain of one route segment, E, which is also the start domain of the next route segment, the two addresses representing the next route segment will be shifted into the source and the destination address field of the packet header, and the router at E will execute Step 1 of the forwarding algorithm with such a packet header.

We show this using induction. It can be seen that at the source domain of the packet, the above condition is true. Suppose the condition is true at the start domain of the ith route segment. The address in the source address field is $Saddr$, and the address in the destination address field is $Daddr$. Figure 5-23 shows the packet header.

source	$Saddr$	
destination	$Daddr$	
	...	
next-to-visit	$Vaddr$	start of the routing header
	$Vaddr'$	
	...	

Figure 5-23: This figure shows the route representation for a compound route.

By previous proofs on route segments of any basic type, the packet will be forwarded to the end domain, E_i, of the ith route segment. When E_i executes the first step of the forwarding algorithm, it will find a match for $Daddr$ with the next hop pointing to itself, and transfer to Step 4. At Step 4, if there are still addresses left in the routing header, $Daddr$ will be shifted to the source address field, and the next-to-visit address $Vaddr$ will be shifted to the destination address, the pointer next-to-visit will be moved to the next address in the routing header. E_i will go back to Step 1 of the forwarding algorithm. Figure 5-24 shows the packet header after the above operations.

If $Vaddr$ is the second address in the representation of the next route segment, then the packet header has $Daddr$, the first address of the next route segment representation, in the source address field, $Vaddr$ in the destination address field, and E_i will execute Step 1 of the forwarding algorithm. This shows that our forwarding algorithm will correctly forward the packet along the

source	*Daddr*	
destination	*Vaddr*	
	...	
	Saddr	start of the routing header
next-to-visit	*Vaddr′*	
	...	

Figure 5-24: This figure shows the route representation of a packet for a compound route after the forwarding algorithm modifies the packet header when the packet arrives at the end domain of an intermediate route segment.

next route segment. By induction, our forwarding algorithm will forward the packet along the route specified in the packet's header.

source	*Vaddr*	
destination	*Vaddr′*	
	...	
	Saddr	start of the routing header
	Daddr	
next-to-visit	end	

Figure 5-25: This figure shows the route representation of a packet for a compound route after the forwarding algorithm modifies the packet header when the packet arrives at the end domain of an intermediate route segment.

Suppose *Vaddr* is the first address in the route representation of the next route segment, and the next-to-visit address *Vaddr′* is the second address in the representation. Since a route segment of any type is represented by two addresses, and the start domain of a next segment is the end domain of a previous segment, both *Vaddr* and *Daddr* must be the addresses of E_i. When E_i executes Step 1 of the forwarding algorithm, it will find a match for *Vaddr* with the next hop pointing to itself, and transfer to Step 4 again. At Step 4, *Vaddr* will be shifted to the source address field, and the second address in the representation of the next segment will be shifted to the destination address field. Figure 5-25 shows the packet header after these operations. The domain E_i, which is the start domain of the next route segment, will go back to Step 1 of the forwarding algorithm with *Vaddr* in the source address field and *Vaddr′* in the destination address. These two addresses together represent the next route segment. This again concludes our proof.

5.A.6 Any Route

If a sender wants to communicate with a receiver over a domain-level route $P_0 \to P_1 \to P_2 \to \ldots \to P_n$, the route is represented in the same way as the route segment $P_0 \to P_1 \to P_2 \to \ldots \to P_n$ is represented, except that the first address is a unicast address of the sender, instead of the domain address of the sender's domain, and the last address is a unicast address of the receiver, instead of the domain address of the receiver's domain. It can be proved that this route representation works correctly with the forwarding algorithm in the same way as the correctness of the route segment representation is proved.

Appendix 5.B Multiple Nodes

We have described TIPP and a router's forwarding algorithm assuming each domain has one border router. In practice, a domain usually has multiple routers and hosts. We can use a similar approach as in BGP to synchronize TIPP information learned from different border routers and select router-level next forwarding hop.

As in BGP, TIPP runs between border routers. Border routers of a domain will need to relay information learned from external TIPP connections to each other. This relaying can either be done using a fully meshed connection among all border routers, or using a central server. When one domain has multiple border routers connecting to another domain, address allocation and topology information should be maintained consistent among the routers. If a provider domain withdraws a prefix allocated to a customer domain, then all border routers in the provider domain connected to the customer domain should send an **address** withdraw message to the routers in the customer domain. A domain-level link is considered down and a topology message is sent only when all router-level connections between two domains are down.

For the forwarding part, if a domain has multiple exit routers connecting to a neighbor domain, then one reachable address prefix of that neighbor will be heard from all those exit routers. Through relay, a border router in the domain will know that it has multiple next hops to reach that address prefix. Similar to BGP, the border router can pick the best next hop based on domain policies.

A domain could perform its intra-domain routing in the same way as in today's Internet. Inter-domain address prefixes learned from border routers can be distributed to internal routers in the same way as how prefixes learned from BGP are distributed to internal routers today.

Appendix 5.C Sibling Relationships

In this section, we discuss how sibling relationships should be treated in NIRA. If two domains have a sibling relationship, they provide mutual transit service for each other, i.e., one domain allows the other domain to send packets to or to receive packets from all its neighbors. A sibling relationship usually exists between two edge domains, and saves each domain the cost of purchasing an additional provider for redundant Internet access [54]. A sibling relationship may also exist due to ISP mergers [54].

If two domains have a sibling relationship, as we describe in Chapter 4.2.7, one domain could allocate a subdivision of an address prefix it has to its sibling domain. With this allocation scheme, a valley-free route that consists of sibling links can also be represented by a source and a destination address.

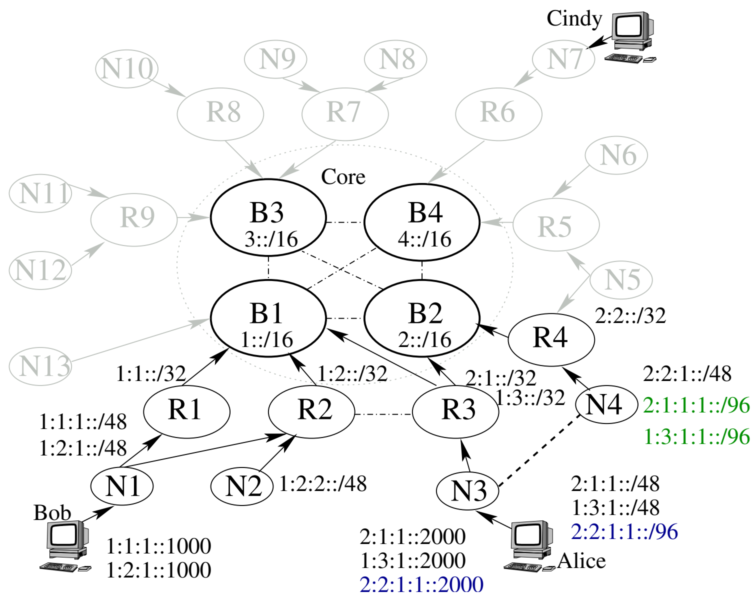


Figure 5-26: In this example, the two domains N_3 and N_4 have a sibling relationship. So N_3 will allow N_4 to use its provider R_3, and N_4 will allow N_3 to use its provider R_4. N_4 allocates a subdivision (2:2:1:1::/96) of an address prefix (2:2:1::/48) obtained from R_4 to its sibling domain N_3. So does N_3.

Figure 5-26 shows an example. The two domains N_3 and N_4 have a sibling relationship. So N_3 will allow N_4 to use its provider R_3, and N_4 will allow N_3 to use its provider R_4. If the connection between N_3 and R_3 is broken, a user Alice in N_3 can use the uphill route segment $N_3 \rightarrow N_4 \rightarrow R_4 \rightarrow B_2$ to reach the rest of the Internet. For instance, if Alice wants to communicate with Bob, she can pick the route $N_3 \rightarrow N_4 \rightarrow R_4 \rightarrow B_2 \rightarrow B_1 \rightarrow R_1 \rightarrow N_1$. This route can be represented by a source address of Alice, 2:2:1:1::2000, which is inside the address space 2:2:1:1::/96 allocated from N_4 to N_3, and a destination address of Bob 1:1:1::1000.

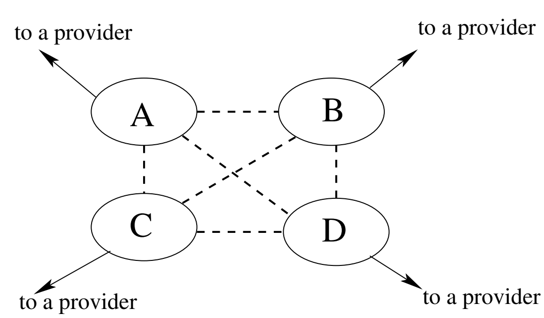


Figure 5-27: Domains A, B, C, and D form a clique, and each has a sibling relationship with the other domains.

If a group of domains all have sibling relationships with each other, which may happen due to ISP mergers, and if each domain allocates a subdivision of an address prefix it has to every sibling domain, and each sibling domain does the same thing in turn, then a domain in the group may suffer from address implosion. For instance, in Figure 5-27, domains A, B, C, and D form a clique, each having a sibling relationship with the other domains. Suppose each domain has one address prefix allocated from a provider. If each domain recursively allocates subdivisions of address prefixes it has to its sibling domains, then each domain will end up with having 15 address prefixes. For example, domain A will have address prefixes allocated from the following paths: $B \to A$, $C \to A$, $D \to A$, $B \to C \to A$, $B \to D \to A$, $C \to B \to A$, $C \to D \to A$, $D \to B \to A$, $D \to C \to A$, $B \to C \to D \to A$, $B \to D \to C \to A$, $C \to B \to D \to A$, $C \to D \to B \to A$, $D \to B \to C \to A$, $D \to C \to B \to A$.

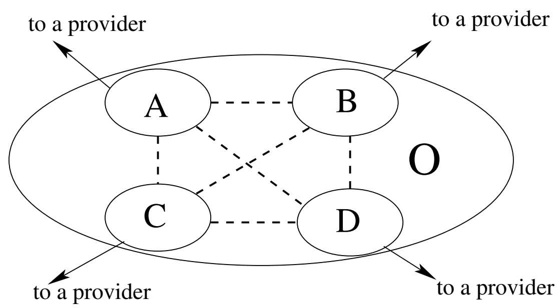


Figure 5-28: Domains A, B, C, and D can be consolidated into one domain.

There are two possible approaches to avoid the address implosion problem. First, the group of sibling domains can be consolidated into one compound domain. The internal domains inside the compound domain run a routing protocol to connect themselves. Address prefixes assigned to each internal domain are therefore assigned to the compound domain. Each internal domain does not need to allocate address prefixes to each other. For example, in Figure 5-28, domains A, B, C, and D can run a routing protocol among themselves, and appear to the outside as a compound domain O. O will have the address prefixes allocated by each of the internal domains' providers. Each node in O will have four addresses, instead of 16.

When multiple sibling domains are merged into a compound domain, a user cannot use NIRA's route discovery mechanism to pick route inside the compound domain. We do not think this is a problem, because our purpose for user route choice is to stimulate wide area ISP competition. Even if sibling domains are merged into one compound domain, a user can still pick different providers of the compound domain. Besides, in practice, competitive service providers usually do not negotiate sibling relationships with each other. If a situation like the one shown in Figure 5-27 occurs, it is most likely due to ISP mergers. The domains with sibling connections actually belong to the same ISP. Therefore, domain merging will not prohibit a user from choosing different ISPs.

The second approach to alleviate the address implosion problem is to allocate address prefixes only along the most commonly used routes. In Figure 5-27, it is unlikely that all possible routes in the sibling cluster will be equally used. So address prefixes do not have to be allocated along every policy-allowed route. For instance, each domain in Figure 5-27 can be configured not to recursively allocate address prefixes that are allocated from a sibling domain. So domain A will only have an address prefix allocated from its provider, and address prefixes allocated from its sibling domains: $B \to A$, $C \to A$, and $D \to A$. Because addresses are not allocated along other policy-allowed routes in the sibling cluster, a sender that wants to communicate with a user of domain A may not discover those routes via NRLS servers. But a user of domain A will learn those routes via TIPP, and can tell the sender about those routes once he receives a packet from the sender. Therefore, the two users can still communicate using those routes.

Chapter 6

Provider Compensation

NIRA is a scalable routing architecture that gives a user the ability to select domain-level routes. In previous chapters, we have focused on mechanisms that support user route choice, which include route and route availability discovery, and route representation and packet forwarding. In this chapter, we explore an orthogonal but equally important problem: how a provider gets paid if a user chooses to use its service. We refer to this problem as the provider compensation problem.

We consider the provider compensation problem important because providers have control over various network resources. If they can not benefit from giving a user the ability to choose from multiple routes, it is unlikely that a provider will honor a user's choice. In this chapter, we first discuss how the providers are compensated in today's Internet. Then we describe how providers might be compensated in NIRA. We also discuss miscellaneous economic issues that could arise in NIRA.

6.1 Provider Compensation in Today's Internet

In the current Internet, providers are paid by their directly connected customers based on pre-negotiated contractual agreements. Local providers are paid by individual users that sign up for their services and use their local access facility. In the sample network shown in Figure 6-1, N_1 is paid by users like Bob and Mary. Regional or wide-area providers are paid by other providers or organizations that purchase their wholesale Internet service and are directly connected to them. For example, R_2 is paid by local providers such as N_1 and N_2; B_1 is paid by regional providers such as R_1, R_2, and R_3.

In today's Internet, local providers, such as Verizon online DSL [13] or Comcast [7], usually charge a flat-fee to end users. Providers that sell wholesale Internet access service to other providers or organizations may either charge a flat-fee or a usage-based fee to their customers. When a wholesale provider charges a usage-based fee to a customer domain, the charge is based on the aggregated traffic on the customer's access link(s). The wholesale provider does not need

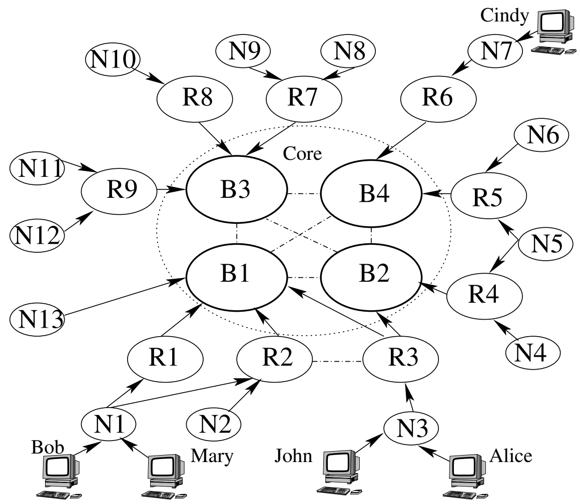


Figure 6-1: In today's Internet, a provider is paid by its directly connected customers based on pre-negotiated contractual agreements. In this example, N_1 is paid by users like Bob and Mary; R_2 is paid by local providers such as N_1 and N_2; B_1 is paid by regional providers such as R_1, R_2, and R_3.

to track the usage of end users. For example, in Figure 6-1, if R_1 charges a usage-based fee to N_1, then the charge will be based on the aggregated traffic on N_1's access link to R_1, and R_1 does not need to track the link usage of users such as Bob or Mary.

BGP provides a good technical support for this type of provider compensation. One hop contractual agreements map to one hop routing decisions. With BGP, each domain makes local decisions to select the next hop a packet follows. A domain could pick the cheapest next hop to reach a destination. The common BGP policies state that if there are multiple routes to reach a destination, a domain prefers a customer route to a peer route or a provider route, as a customer route potentially brings revenue to the domain; a domain prefers a peer route to a provider route, as a peer route usually involves no payment; the least preferable route is a provider's route, as it costs the domain money.

6.2 Design Rationale

With NIRA, an end user, instead of BGP routers of each domain, selects the route his packets follow. A provider no longer has the control to pick the cheapest next hop to reach a destination. It is possible that this technical difference between NIRA and today's Internet brings about different solutions for provider compensation. To come up with the design for NIRA's provider compensation schemes, we first examine the general approaches to compensate a provider.

At a high level, there are two possible approaches. The first approach requires no pre-negotiated contractual agreements between users and providers. At the time when a user sends a packet, the user could freely choose any providers to carry the packet and pays while he sends the packet. In our design, we consider this approach has little market appeal, as it may require that a provider takes micro-payment from packets it receives. History suggests that such approaches are unlikely to be widely deployed in the Internet.

For the second approach, providers are compensated based on pre-negotiated contractual agreements. At the time when a user sends a packet to a destination, the user is restricted to choose a route using those providers he or the destination has agreed to pay. This approach can be further divided into two compensation methods based on how contractual agreements are negotiated. In the first method, contractual agreements are negotiated between directly connected parties as in today's Internet. We refer to this method as *direct compensation model*. In the second method, contractual agreements may be negotiated between non-directly connected users and providers. For example, a user may negotiate a contract with a second-hop provider. We refer to this method as *indirect compensation model*. We discuss these two models in turn.

6.3 Direct Compensation Model

The direct compensation model is the widely adopted compensation approach in today's Internet. This compensation model does not require a wide-area provider to deal with the accounting details of individual users. Thus, it is scalable. We think that this model can also be adopted for provider compensation in NIRA.

It is true that with NIRA, a domain cannot control to which next hop provider a packet will be sent. It is thus more difficult for a domain to provision its bandwidth purchase from a provider to avoid congestion. However, this problem can be solved if a domain provisions its access link to a provider according to its aggregated peak traffic, but negotiates a usage-based contract with its providers. In Figure 6-1, if N_1 provisions its access link to R_1 and R_2 according to the aggregated peak traffic from its users, then even in the worse case, when all its users pick R_1 (or R_2) to send their traffic, the link to R_1 (or R_2) will not be congested. If the contracts are usage-based, then N_1 pays R_1 or R_2 according to how much its users use R_1 or R_2, and does not pay R_1 and R_2 according to its access link capacity. So the overall operational cost of N_1 will not be increased much by its over-provision of the access link to R_1 or to R_2.

When routes are picked by users, a domain is unable to pick the cheapest next hop to reach a destination. The operational cost of a domain might be increased. For example, in Figure 6-1, with BGP routing, if B_1 is a cheaper provider than B_2, then R_3 could always route traffic to B_1. But with NIRA, a user may prefer to send traffic using the more expensive provider B_2, thus increasing the operational cost of R_3. This problem can be solved if a provider charges different routes differently. R_3 can recover its cost by charging N_3 according to N_3's usage of the route $N_3 \to R_3 \to B_1$ and the route $N_3 \to R_3 \to B_2$ and the cost of each route; N_3 can then transfer this charge to users who pick the route.

In NIRA, a route can be represented by an address. So it is not difficult for a provider to track route usage. A domain could monitor traffic at the access point of a customer, and track the usage of a source address for inbound traffic from the customer, and the usage of a destination address for outbound traffic to the customer.[1] The cost of an address prefix could be advertised in TIPP messages when a provider allocates the address prefix to a customer. In Figure 6-1, R_3 could charge N_3 according to how much of N_3's traffic uses an address allocated from B_1, and how much of N_3's traffic uses an address allocated from B_2. The amount of traffic using an address from B_2 will be charged more as B_2 is more costly.

Strictly speaking, the usage cost of a route is determined by both the source address and the destination address of a packet. For example, for an inbound packet from N_3 to R_3 with a source address allocated from the more expensive route $B_2 \rightarrow R_3 \rightarrow N_3$, if the destination address is the address of R_3, the packet would take the route $N_3 \rightarrow R_3$, which does not cost R_3 anything. But if the destination address is the address of user Bob and is allocated from $B_1 \rightarrow R_1 \rightarrow N_1$, then the packet would follow the route $N_3 \rightarrow R_3 \rightarrow B_2 \rightsquigarrow B_1 \rightarrow R_1 \rightarrow N_1$, which would cost N_3 to pay B_2. Therefore, routes with the same source address but different destination addresses may incur different cost to a provider.

So on the one hand, tracking the usage of only a source address or a destination address cannot precisely determine the usage cost of a customer's traffic. On the other hand, it will increase the complexity of accounting if usage-based charge needs to consider the combination of different source and destination addresses. However, there is a similar problem in today's Internet. Although a provider can pick the cheapest next hop to reach a destination, different destinations cost a provider differently. For example, in Figure 6-1, with BGP routing, R_3 could always prefer to use the free peering link between itself and R_2 to send traffic coming from N_3 and destined to customers of N_1, But for traffic coming from N_3 and destined to customers of N_4, R_3 has to send the traffic to its provider B_2 or B_1. Therefore, traffic destined to N_4 will cost R_3 more than traffic destined to N_1. However, in today's Internet, providers still manage to charge customers usage-based fees without doing fine-grained destination-based accounting. We expect that a provider could employ similar heuristics to determine the usage charge of customer's traffic based on source address for inbound traffic and destination address for outbound traffic without doing fine-grained accounting on the combinations of source and destination addresses.

There are multiple ways for N_3 to transfer route usage based charge from R_3 to its customers. For example, N_3 could charge a user according to his address usage. N_3 monitors a user's traffic sent or received with the address allocated from $B_1 \rightarrow R_3 \rightarrow N_3$ or the address allocated from $B_2 \rightarrow R_3 \rightarrow N_3$, and charge the user accordingly. Differently, N_3 could sell addresses at a flat-fee to users. If a user wants to pick between the route $N_3 \rightarrow R_3 \rightarrow B_1$ and the route $N_3 \rightarrow R_3 \rightarrow B_2$, N_3 could sell two addresses to the user; if instead, a user wants a cheap service, N_3 could sell a user only one address allocated from $N_3 \rightarrow R_3 \rightarrow B_1$, and the user is only allowed to use that

[1]The usage of a destination address might not be the intention of a receiver. We discuss this problem in Section 6.4.

route. What's more, N_3 could also offer a service that restricts how a user might choose routes. That service might give a user two addresses, but require that a user use the address allocated from the cheaper route $B_1 \to R_3 \to N_3$ most of the time, unless that route fails. As N_3 will learn about route conditions via TIPP, it can monitor whether a user follows this service agreement or not.

6.3.1 Policy Checking

When provider compensation is based on pre-negotiated contractual agreements, it would be desirable for a provider to verify that the route specified in a packet header conforms to its transit policies before the provider actually transits the packet. We refer to this verification as *policy checking*.

When contractual agreements are negotiated between directly connected parties, a domain usually has transit policies that state whether it will provide transit service between two neighbors, based on whether the neighbors have agreed to pay the domain or not.

Similar to what it is done today, a domain could employ physical connectivity to distinguish incoming traffic from different neighbors for policy checking. If a packet comes from an interface connected to a neighbor domain, a domain could assume that the packet comes from that neighbor domain. So if the domain provides service between the incoming domain and the outgoing domain of the packet, the domain should forward the packet; otherwise, the domain should drop the packet.

If a router checks transit policies based on the incoming and outgoing interface of a packet, policies would be specified in terms of interface identifiers, which involves hardware dependency. It is desirable for a domain to specify transit policies in terms of constraints on packet header fields, such as the source address field, or the destination address field, so as to get rid of hardware dependency. As we have discussed in Chapter 5.2, the outgoing interface of a packet is actually determined by the source address or the destination address in a packet header. If a border router performs source address verification, i.e., verifying whether the source address of a packet matches the incoming interface of the packet, then the incoming interface of a packet can be associated with the source address of a packet. Thus, both incoming and outgoing interface of a packet can be bound to packet header fields. As a result, transit policies could be specified as constraints on those fields.

A border router could then enforce its transit policy by matching the header of a packet against its policy filters. A simple policy filter could be of the format *(source address, destination address, action)*. If a packet header matches a policy filter, a router will take the corresponding action. If the action is *forward*, the packet will be forwarded according to the next-hop returned by the forwarding algorithm. If a packet has an optional routing header, the source address field and the destination address field of the packet might be changed by the forwarding algorithm during the next-hop lookup process (described in Chapter 5.2). So a border router should match the original source address (the source address when the packet first arrives at the router) and the final destination address (the destination address when the packet leaves the router) against its policy filters. Policy filter matching is a simplified type of packet classification. Recent research [17] suggests that it can be done at a high speed.

NIRA adopts a provider-rooted hierarchical addressing scheme, which makes it easy for a border router to verify the source address of a packet. A domain knows what addresses it allocates to a customer domain. So if a packet comes from a customer domain, then the source address of the packet should be within the address space allocated to the customer; otherwise, if a packet comes from a provider or a peer domain, the packet should not have a source address that is within the address space allocated to a customer of the domain. This verification prevents a provider or a peer from stealing transit service from a domain by spoofing a source address allocated to a customer of the domain.

NIRA's route representation scheme simplifies policy checking in the case where the source address and the destination address of a packet are not modified after a router finds the next-hop to forward a packet. In this case, a router does not need to do extra work for policy checking, because a packet with only a source and a destination address will be forwarded along a policy-allowed valley-free route. If the next-hop is found in a domain's downhill forwarding table, then the packet is destined to a customer of the domain. Policy checking is not needed as a domain will forward packets destined to its customers, regardless of where the packets come from. If the next-hop is found in a domain's uphill forwarding table, and the source address of the packet matches the interface from which it comes, then the packet is from a customer of the domain. Again, policy checking is not needed, as a domain will transit packets from a customer domain to any neighbor. If the next-hop is found in a domain's bridge table, then the source address of the packet either matches the non-provider-rooted address prefix of the domain, or matches the private address prefix rooted at the domain (see Chapter 5.6). So the packet either comes from nodes inside the domain or comes from a customer of the domain. In either case, the packet can be forwarded to any neighbor of the domain without further policy checking. If the next-hop is found in a domain's routing table, then the routing protocol that builds the table should have already handled transit policies in selecting the next hop to reach the destination address in the packet header, as it is done in today's Internet. Thus, no more policy checking is required.

6.3.2 Indirect Compensation Model

Contractual agreements can also be negotiated between non-directly connected parties. For example, a user may negotiate business relationships with a non-directly connected provider. If he wants to send packets through that provider, NIRA allows a user to specify so in his packet headers.

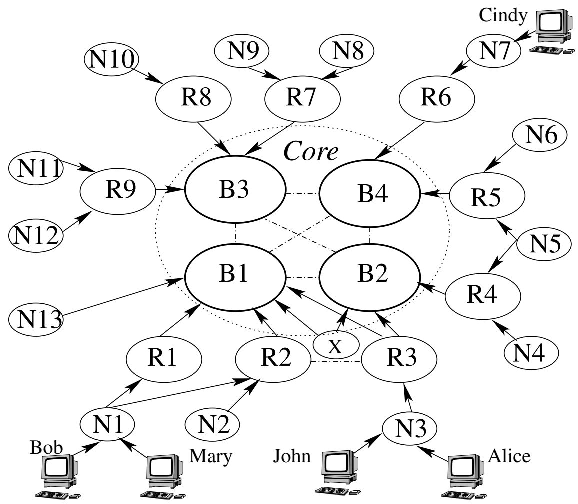


Figure 6-2: X may purchase service from both B_1 and B_2, and sell transit service to end users. If Bob purchases service from X, Bob could send packets to Alice using a route that goes through the route segment $B_1 \to X \to B_2$, rather than the default congested peering link between B_1 and B_2. However, if Mary does not purchase the service, she cannot send packets to Alice using the route that goes through $B_1 \to X \to B_2$.

For this type of indirect business relationship, policy checking becomes a challenging problem. Packets coming from the same adjacent domain may be subject to different transit policies. Thus, a provider can not tell what policies to apply to a packet based on from which interface the packet comes, and to which interface the packet goes. Instead, a provider needs to tell whether the packet is sent to or from one of its customers. Figure 6-2 shows an example. Suppose the peering connection between B_1 and B_2 is often congested. A provider X may consider it as a good business opportunity. So X could purchase service from both B_1 and B_2, and sell transit service to end users. If Bob purchases service from X, Bob may send packets to Alice using a route that goes through the route segment $B_1 \to X \to B_2$, rather than the default congested peering link between B_1 and B_2. However, if Mary does not purchases the service, she cannot send packets to Alice using the same route. When X receives a packet from B_1, X needs to verify that the sender of the packet is Bob in order to transit the packet to B_2.

The need to authenticate the sender or the receiver of a packet at the packet forwarding time is not unique to NIRA. Even in today's Internet, an overlay provider, or a second-hop provider that wants to sell QoS to a customer, also has this requirement. As this problem is general enough to become a separate problem, we do not discuss it in detail in this dissertation. Any solution to this problem could be plugged into NIRA to prevent unauthorized route usage. For instance, a provider could authenticate the sender of a packet before it forwards the packet using the approach proposed by Platypus [92, 85].

6.4 Financial Risks of Exposing Routes

In NIRA, we assume a sender-and-receiver joint payment model between a sender and a receiver at the network layer. This payment model is the widely deployed payment model in today's Internet. For an end-to-end route, the sender of a packet compensates some providers en route, and the receiver compensates other providers en route. In our design, it is usually the case that a sender compensates the providers on his up-graph (the uphill portion of a route), and a receiver compensates the providers on the receiver's up-graph (the downhill portion of a route).

In a sender-and-receiver joint payment model, it is difficult for a receiver to avoid being sent unwanted traffic. This problem prevails in today's Internet. The manifestations include spam and Denial of Service (DoS) attacks. However, in today's Internet, an edge domain can control to which provider to announce its address prefixes using BGP. Therefore, the domain could control from which provider its incoming traffic comes. With NIRA, a receiver exposes its route segments and its route preference via its NRLS servers. A non-cooperative sender may override a receiver's route preference and intentionally send large volume of traffic over an expensive route to a receiver. This non-cooperative behavior of the sender could cause a financial loss to the receiver.

This financial risk is the unfortunate side effect of user empowerment. However, there are two possible counter measures. First, with NIRA's route representation scheme, the source address field of a packet reveals the network location of the sender, and the forwarding algorithm depends on the correctness of the source address. This prevents a user from spoofing an arbitrary source address. For example, in Figure 6-2, if a customer of N_3 does not send a packet with a source address allocated from $B_1 \rightarrow R_3 \rightarrow N_3$ or a source address allocated from $B_2 \rightarrow R_3 \rightarrow N_3$, according to our forwarding algorithm, N_3 will not forward the packet to R_3. This situation is true at each level of the provider hierarchy. If a packet coming from N_3 to R_3 does not have a source address that is within the address space of R_3 allocated from $B_1 \rightarrow R_3$ or $B_2 \rightarrow R_3$, the packet will not be forwarded to B_1 or B_3. If a non-cooperative sender has to reveal his real network location in order to send packets to the victim receiver, this requirement may discourage him from being non-cooperative in the first place. Moreover, if the non-cooperative sender cannot fake his network location in a packet header, the victim receiver could ask his upstream providers to filter out the sender's traffic. Second, a user can store at his NRLS servers a subset of his addresses, and keep more expensive addresses private. He can reveal the private information to those users he trusts after they have exchanged packets via the public addresses he stores at his NRLS servers.

Briscoe [23] also suggested a solution to this problem. The solution says that it is customary for both the sender and the receiver to pay, but the ultimate liability should remain with the sender. Any receiver could dispute his payment unless the sender had proof of a receiver request.

We do not claim that the above measures could eliminate the non-cooperative behaviors of users. We think these measures will reduce the financial risk of a receiver to minimal.

6.5 The *Core*

In our design of NIRA, we utilize the provider-customer relationship between domains to divide the network into smaller regions, such as the up-graph of a user, or a provider tree. We also assume there is a *Core* structure of the Internet, where roots of provider trees are connected by a routing protocol. Routes within the *Core* are chosen by the routing protocol, instead of users.

We make this assumption because in practice, there are providers that do not pay any providers for transit service. Such providers are often called tier-1 providers. A tier-1 provider has a peering connection with all other providers, and there is no money exchange between tier-1 providers. Since a user's payment stops at a tier-1 provider, we think it is reasonable not to let users pick routes in the *Core*.

However, we have never strictly defined what the *Core* is. We deliberately omit the definition because we expect that the *Core* will be self-defined with the deployment of NIRA. A domain in NIRA has two choices. First, it could obtain a globally unique address prefix, connect to those providers with globally unique address prefixes, and convince those providers to announce its own prefix in the *Core* routing region. Second, the domain could accept address allocations from the providers it connects to. If the domain takes the first choice, then the domain becomes a top-level provider in the *Core*; otherwise, the domain is not part of the *Core*. At a minimum, the *Core* will include those tier-1 providers, which does not purchase transit service from any provider and has a peering connection with all other tier-1 providers. Other domains could make their own decisions on whether to become a *Core* provider, based on the cost to obtain a globally unique address prefix, the cost to get that prefix announced to other providers in the *Core*, and the benefit of doing so.

Therefore, the *Core* will be shaped by market force. If users welcome route choice, then a provider has the motivation to stay outside the *Core* to let users pick routes; otherwise, a provider might stay inside the *Core*. It is worth noting that even in the worse case, where every provider has decided to join the *Core*, with NIRA, a multi-homed edge domain will not poke a hole in the global routing tables. Thus, the growth of the global routing state is still limited by the growth of the *Core*, instead of the growth of the Internet. We think this rate of growth will scale well.

6.6 Where Choice can be Made

In this dissertation, we focus our discussion on how to let users choose routes. The default usage model of NIRA is that a software agent running on a user's computer selects routes based on the user's preference. The software agent could select routes using simple static policies, e.g., always choosing the cheapest route, or using a learning based approach to infer a user's preference from his behavior, as described in [68, 42].

However, in NIRA, choice is not restricted to be made only by end users. In situations where an end user is not financially responsible for his Internet access, the one that is responsible might not want the user to select routes. For example, a company might not want its employees to select routes according to their own preferences. NIRA fully supports such situations.

If an edge domain does not want to give route choice to users inside itself, the domain does not need to propagate TIPP messages to those users. Instead, the domain could have a route server to select routes for its users. TIPP messages propagated to the domain will be passed to the route server. So the route server knows of the set of address prefixes allocated to the domain and the domain's up-graph. When a user inside the domain wants to contact a destination, the user could contact the route server with the destination's name. The route server can then retrieve the destination's addresses and select a route on behalf of the user, and send the user the selected route representation. In this case, when the selected route fails, the user might need to contact the route server again for another route. Or the route server could send the user a set of selected routes with an order of preference if the route server trusts that the user will try out each route according to the specified preference.

Alternatively, a domain could have its border routers act as name-to-route translation boxes, as described in the IPNL [53] architecture. The user could send all his packets with the destination's name. Since a border router receives TIPP messages, it has the knowledge to select routes on behalf of users. When a border router receives the first packet to a destination from a user, the router does an NRLS query, caches the results, selects a route for the user, attaches the route representation to the packet, and sends the packet. For subsequent packets, the router can translate the destination name into a previously selected route representation using cached results.

It is also possible that an edge domain by default does not want its users to select routes, but allows specific users to choose routes. For example, in a campus network, route choice in general might not be allowed. But if there are students that have special needs and are willing to pay for their network usage, for instance, a student from China wants to use a provider that has a hight speed voice over IP service to China, the network administrator might give special permissions to those users to have access to TIPP messages and select routes based on their own preferences. The domain could directly forward TIPP messages to those users or let those users have access to route servers to obtain TIPP messages.

Even for individual consumers that directly purchase Internet service from local service providers, if they are unsatisfied with the default route selection agents running on their desktop computers, they could purchase route selection service and let the service select routes for them.

Therefore, it can be seen that NIRA supports multiple usage models. Choice can either be made by an individual user, or by a user's domain, or by a service purchased by the user.

Chapter 7

Evaluation

In this chapter, we present our evaluation on the design of NIRA. We have decomposed the design of NIRA into four sub-problems: route discovery, route availability discovery, route representation and packet forwarding, and provider compensation. In Chapter 5, we have shown the correctness of our route representation scheme and the forwarding algorithm, and analyzed the forwarding cost. So in this chapter, we do not further discuss route representation and packet forwarding. We have also described how providers might be compensated in Chapter 6. Users and providers pre-negotiated contractual agreements, and providers use policy checking to enforce service agreements. Policy checking can be done in various ways, and is relatively independent of the routing architecture. So we leave it as future work to evaluate various policy checking schemes.

Our evaluation in this chapter focuses on the basic route discovery and route availability discovery mechanisms. We use network measurement, simulation, and analysis for our evaluation.

7.1 Route Discovery Mechanisms

The basic mechanisms we provide for scalable route discovery include three components: provider-rooted hierarchical addressing, TIPP, and NRLS. Correspondingly, we provide an evaluation of each part.

7.1.1 Provider-rooted Hierarchical Addressing

The principal concern we have about the provider-rooted hierarchical addressing scheme is the number of addresses a user has. Theoretically, the number of addresses a user has might grow exponentially with the level of provider hierarchy. If there are h level of hierarchy, and at each level, a domain will on average have p providers, then the number of addresses a user has scales as p^h.

However, financial factors in practice will limit the provider hierarchy to be shallow, and the number of providers to which a domain is connected to be small. Therefore, the number of addresses a domain has should be small.

Verifying this intuition requires knowledge of domain topology and domain relationships. Unfortunately, such information is not publicly available. The best current practice is to infer domain topology and relationships by analyzing BGP tables. There are two well-known inference algorithms: degree based [54] and ranking based [99]. We compared the inference results of the two algorithms when they are applied to the same BGP data, and found that the results differ by about 10%. This difference suggests that these algorithms are inaccurate, but the majority of the results are probably correct.

Our evaluation is primarily based on the inference results from the ranking based approach, because the results also include the classification of core domains, and we can obtain the updated results online [14]. We downloaded one data set for each year since 2001, a total of four data sets. Each data set summarizes domain-level topology and domain relationships using BGP table dumps from multiple vantage points. The results categorize three types of core domains: the dense core, the transit core, and the outer core. The classification is based on the degree of domain interconnectivity. The dense core is almost a clique, where every domain is connected to all other domains.

In our evaluation, we approximate the *Core* structure in NIRA by including all domains in the dense core and the transit core into the *Core*. Domains with degree greater than 200 in the outer core are also included into the *Core*. Figure 7-1 summarizes the data sets. Each domain-level link is unidirectional. The largest topology (the topology on January 13, 2004) has 16809 domains, 74368 links, and 167 core domains.

Date	# domain	# link	# core
2001/04/18	10915	47514	150
2002/04/06	13155	56634	141
2003/01/09	14695	61630	140
2004/01/13	16809	74368	167

Figure 7-1: Domain-level topologies.

We first evaluate the number of prefixes a domain will have using the inferred domain-level topologies. In our evaluation, each domain in the *Core* is initialized with a global unique prefix. We allocate address prefixes to other domains using the provider-rooted hierarchical addressing scheme. Figure 7-2 shows the number of hierarchically allocated prefixes of each domain as a cumulative distribution, and the mean, median and the 90th percentile of the distribution. It can seen that the 90% of the domains will have less than 20 prefixes. However, the largest number is more than a thousand. Hand-debugging a few examples suggest that the tail part of the distribution may be caused by inference errors, e.g. a peering relationship is mistaken into a provider-customer relationship.

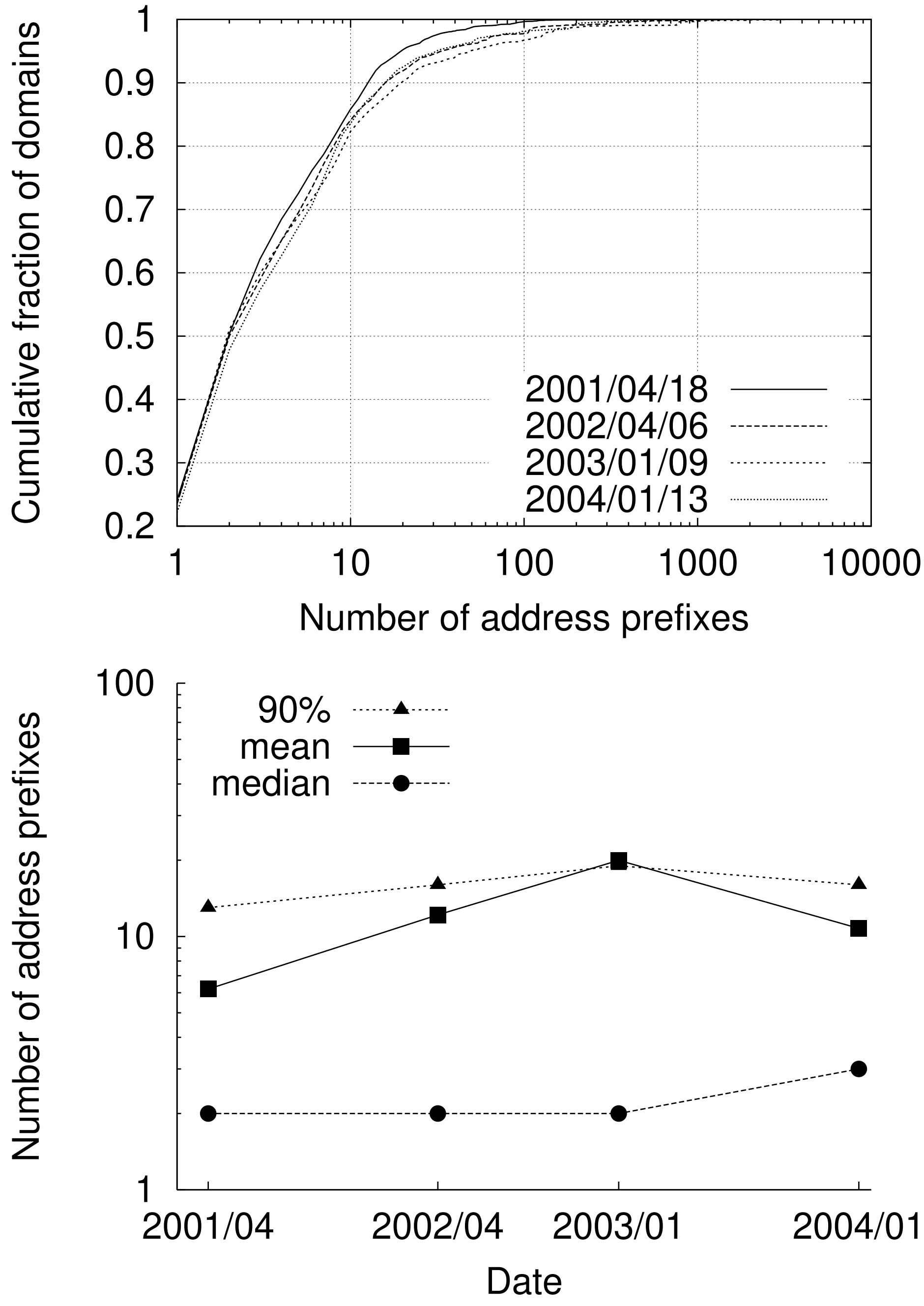


Figure 7-2: The number of hierarchically allocated prefixes of each domain as a cumulative distribution, and the mean, median, and the 90th percentile.

This result confirms our intuition that provider-rooted hierarchical addressing is practical in today's Internet. With NIRA, the Internet topology may change. But we expect that the same financial constraints would apply to the Internet with NIRA. When two domains interconnect, there is cost involved for laying fibers, renting circuits, buying switches etc. A provider will always need to balance the cost to interconnect with another domain and the profit that interconnection would bring. So, we think that even in the future Internet with NIRA, a user will have a moderate number of addresses.

7.1.2 TIPP

Next, we evaluate the scalability and the dynamic behaviors of TIPP.

Scalability of TIPP

To evaluate the scalability of TIPP, we use the inferred domain-level topologies and assume TIPP is turned on between domains outside the *Core*. We then compute the size of the topology database, and the size of the forwarding tables of each domain. The size of the topology database is measured both in the number of link records and in the number of bytes. The size of the forwarding tables is measured in the total number of entries in a domain's uphill, downhill, and bridge forwarding tables. Since the address prefixes allocated to a domain from its providers and the address prefixes allocated by a domain to its customers will show up in a domain's forwarding tables, the number of entries in a domain's forwarding table is a good estimator for the number of entries in a domain's address databases. So we omit the evaluation results for the address databases of each domain.

Figure 7-3 shows the cumulative distribution of the number of link records in the main topology database[1] of a domain, assuming each domain follows the propagation policy described in Appendix 4.B.5. The mean, median, and 90th percentile of the distribution are also shown in the figure. Again, 90% of the domains will have less than 30 link records in their main topology databases.

Figure 7-4 shows the cumulative distribution of the size of a domain's topology database measured in byte, and the mean, median, and 90th percentile. More than 90% of the domains will have a topology database with size less than 10K bytes.

An edge domain is a domain that has no customers. Figure 7-5 and Figure 7-6 show the results of the size of the topology database for an edge domain, which is essentially the size of the up-graph an end user sees. As can be seen, the size of the up-graph an end user sees is small.

Finally, Figure 7-7 shows the cumulative distribution of the number of forwarding entries in a domain's uphill, downhill, and bridge forwarding tables. The mean, median, and 90th percentile

[1] Recall that the main topology database is the topology database that summarizes all input topology databases of a domain.

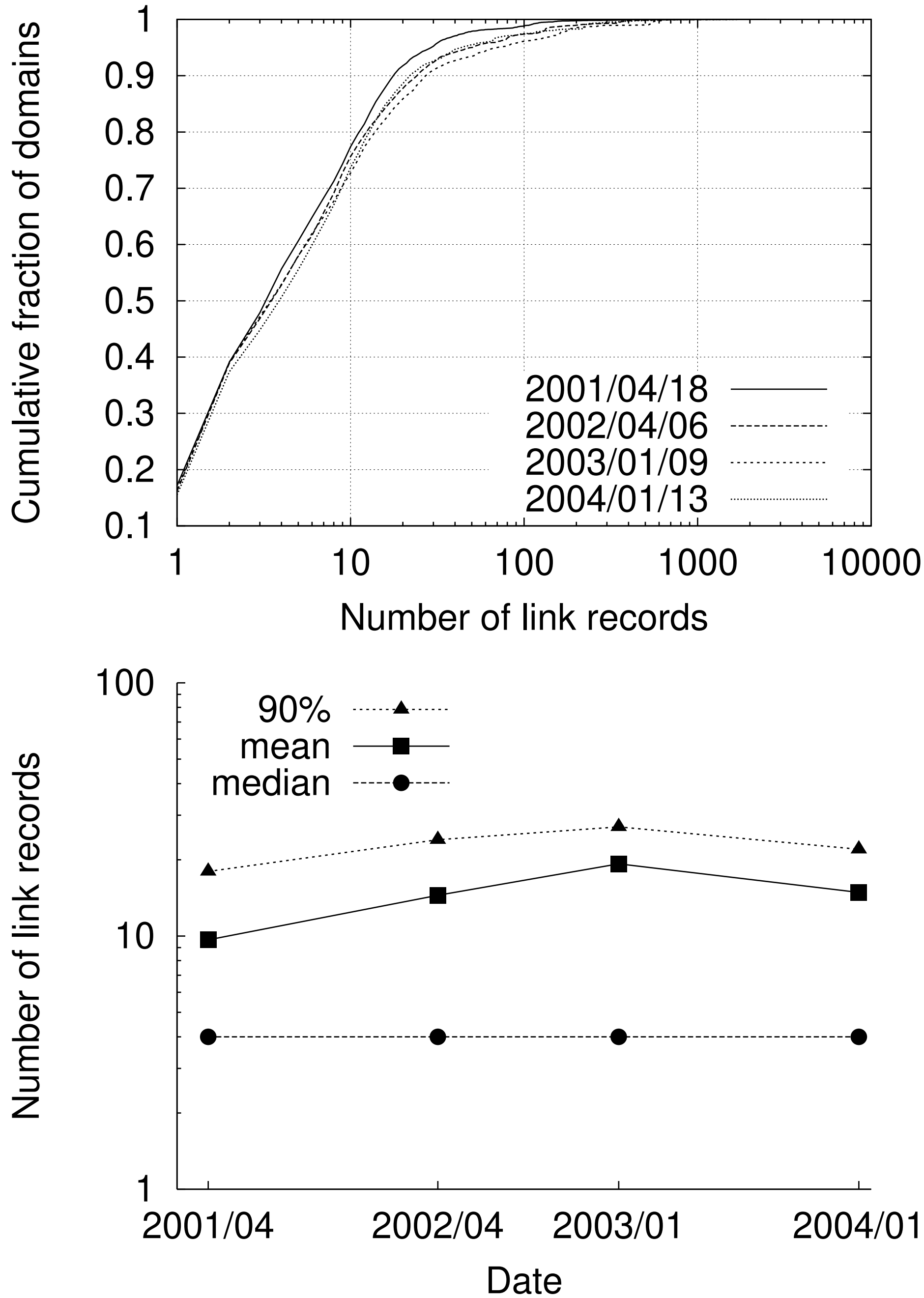


Figure 7-3: The number of link records in a domain's main topology database as a cumulative distribution, and the mean, median, and the 90th percentile.

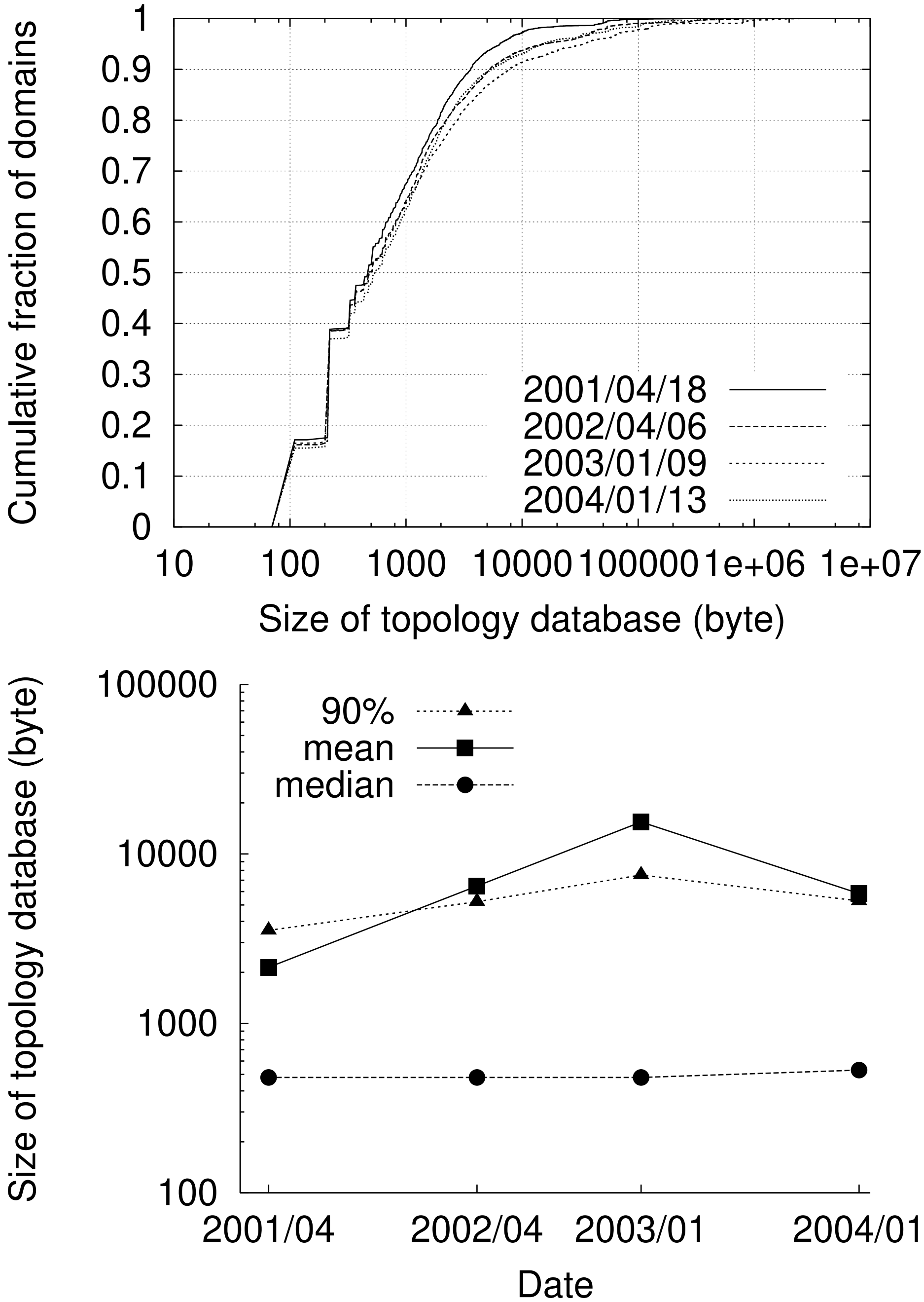


Figure 7-4: The size (in byte) of a domain's main topology database as a cumulative distribution, and the mean, median, and the 90th percentile.

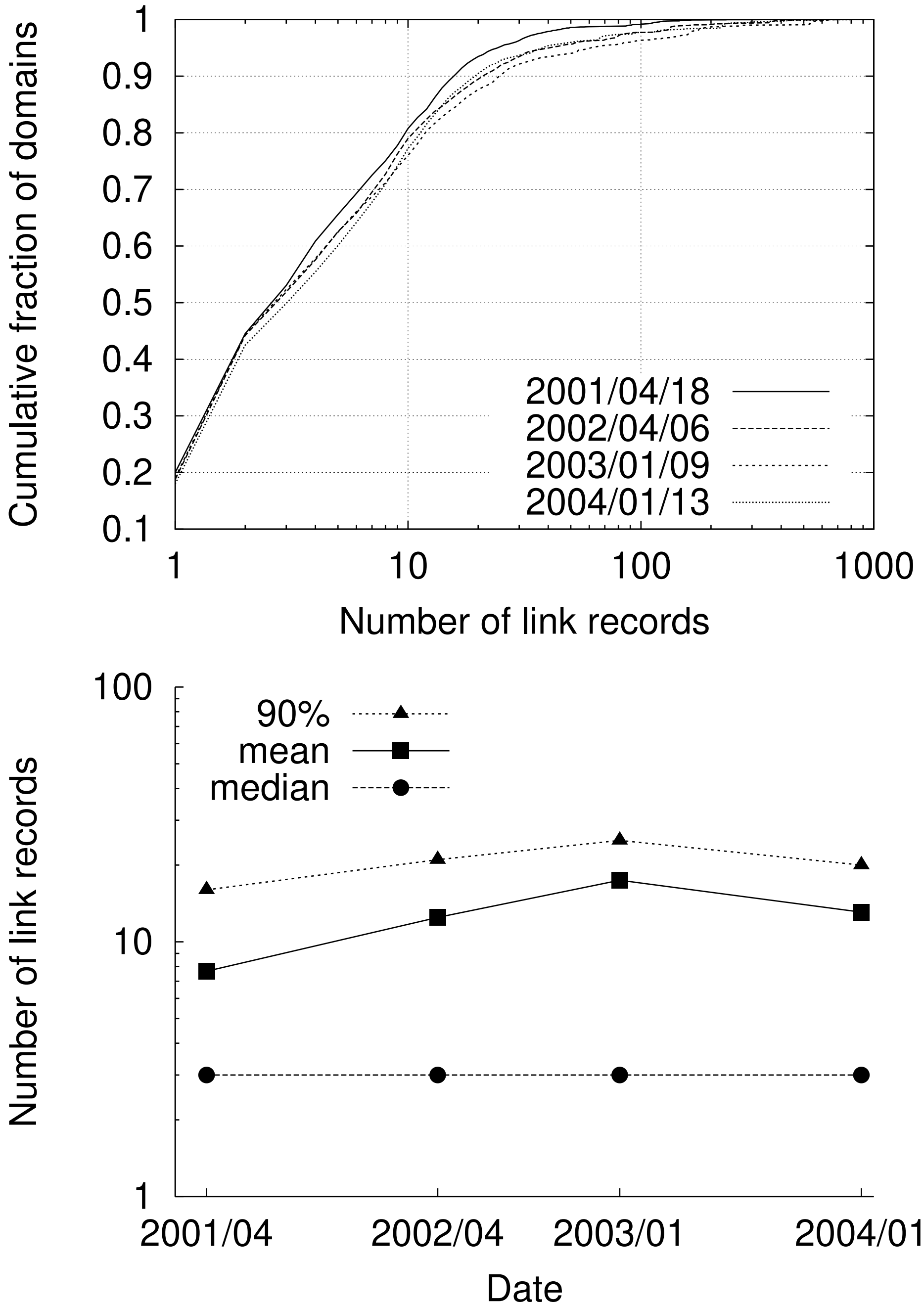


Figure 7-5: The number of link records in an edge domain's main topology database as a cumulative distribution, and the mean, median, and the 90th percentile.

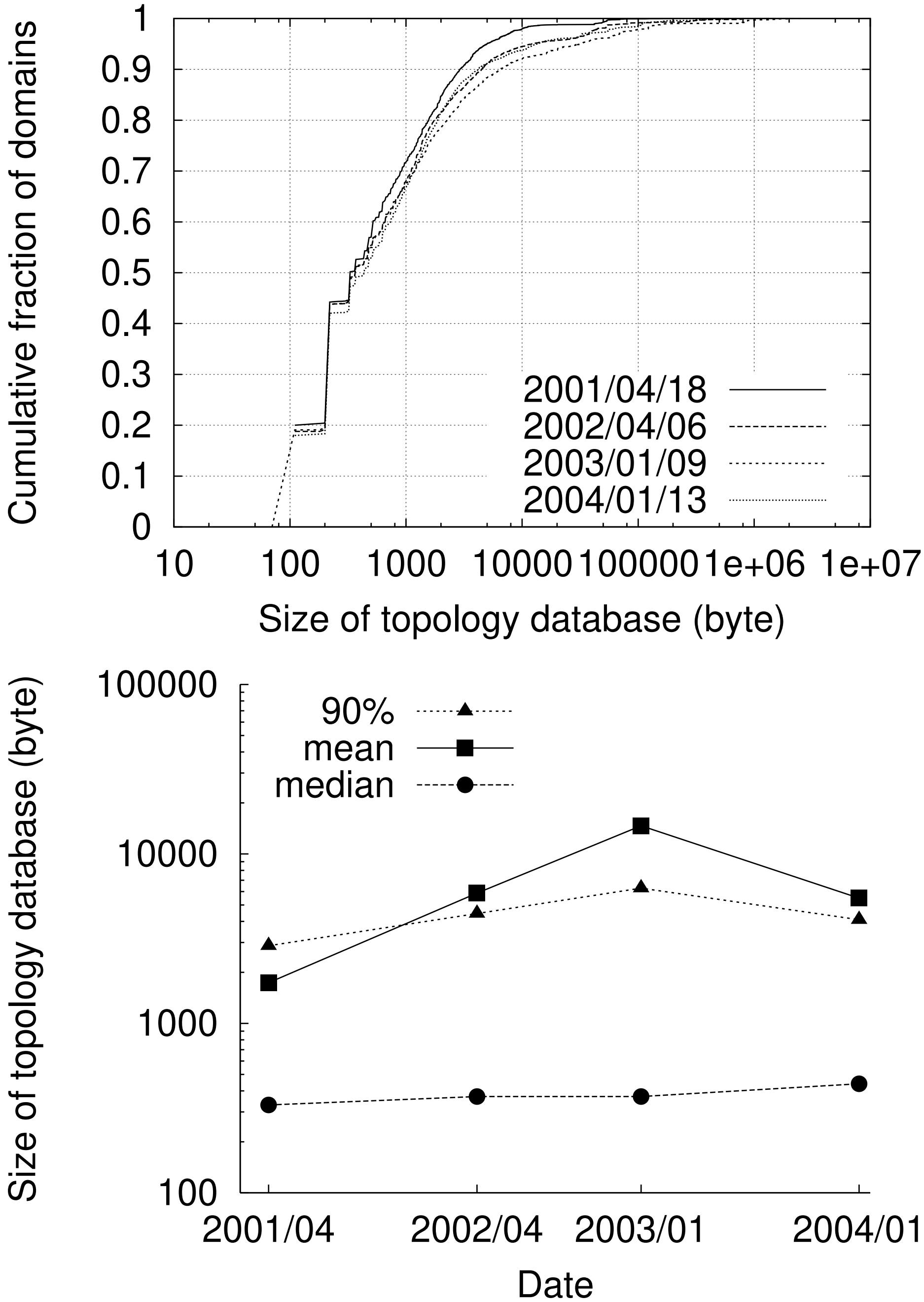


Figure 7-6: The size (in byte) of an edge domain's main topology database as a cumulative distribution, and the mean, median, and the 90th percentile.

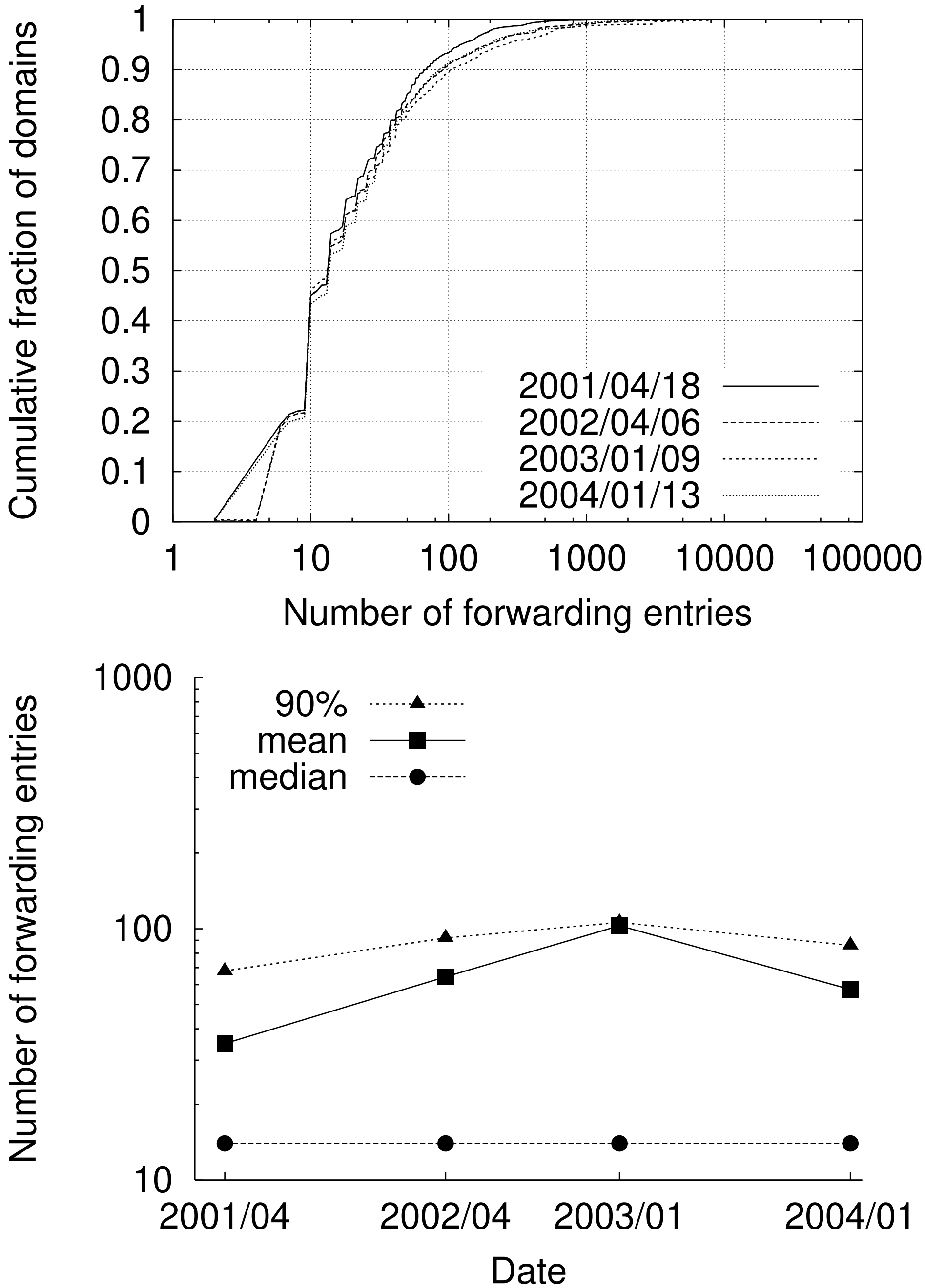


Figure 7-7: The number of forwarding entries in a TIPP router's three logical forwarding tables as a cumulative distribution, and the mean, median, and the 90th percentile.

of the distribution are also shown. 90% of the domains have less than 100 forwarding entries, but the largest number is more than 10,000.

As in the case for NIRA's addressing scheme, there is no theoretic guarantee on the scalability of TIPP, as the number of link records a domain sees may grow exponentially with the depth of provider hierarchy. Suppose there are h level of hierarchy, and at each level, a domain will on average have p providers and q peers, then the number of link records an edge domain has in its topology database scales as $\sum_{i=1}^{h} p^{i-1}(p+q)$.

However, our evaluation results derived from real Internet measurement suggest that TIPP is scalable in practice, because the number of provider hierarchy is shallow, and the number of providers and peers each domain has is quite small.

Dynamic Behaviors of TIPP

We evaluate the dynamic behaviors of TIPP using simulations. We implemented TIPP in the ns-2 [3] simulator. For each inferred domain-level topology, we sampled ten smaller topologies for simulations. To sample a simulation topology, we first pick twenty random edge domains from an inferred topology, and then include domains and domain-level links on the up-graphs of the edge domains. Figure 7-8 summarizes the sampled topologies. On average, each topology has 120-160 domains, and 430-700 unidirectional links.

	Avg		Max	
Date	# Domain	# link	# Domain	# link
2001/04/18	119.2	432.0	199	982
2002/04/06	156.9	676.8	257	1360
2003/01/09	153.3	697.4	220	1262
2004/01/13	147.6	656.8	278	1840

Figure 7-8: Simulation topologies.

We first study the message overhead and the convergence time for single failure events. In a simulation for a sampled topology, we randomly select up to 200 bidirectional links and let them sequentially fail and recover. So at anytime of the simulation, only one bidirectional link fails and recovers. We count the total number of messages and bytes triggered by the failures (excluding the initialization messages sent during a connection re-establishment, since these messages are not further propagated), and average them over the number of failures and the number of links. We also record the maximum sent over a link. Figure 7-9 shows the results for ten simulation runs for each data set. As can be seen, the average message overhead per link is very low, demonstrating that TIPP messages are propagated in controlled scopes. If a message is propagated globally, a failure and a recovery event will generate four messages (two for each direction of a connection). Each unidirectional link will receive two messages on average, and four maximum. The maximum number of messages sent over a link is less than 2, indicating that there is no message

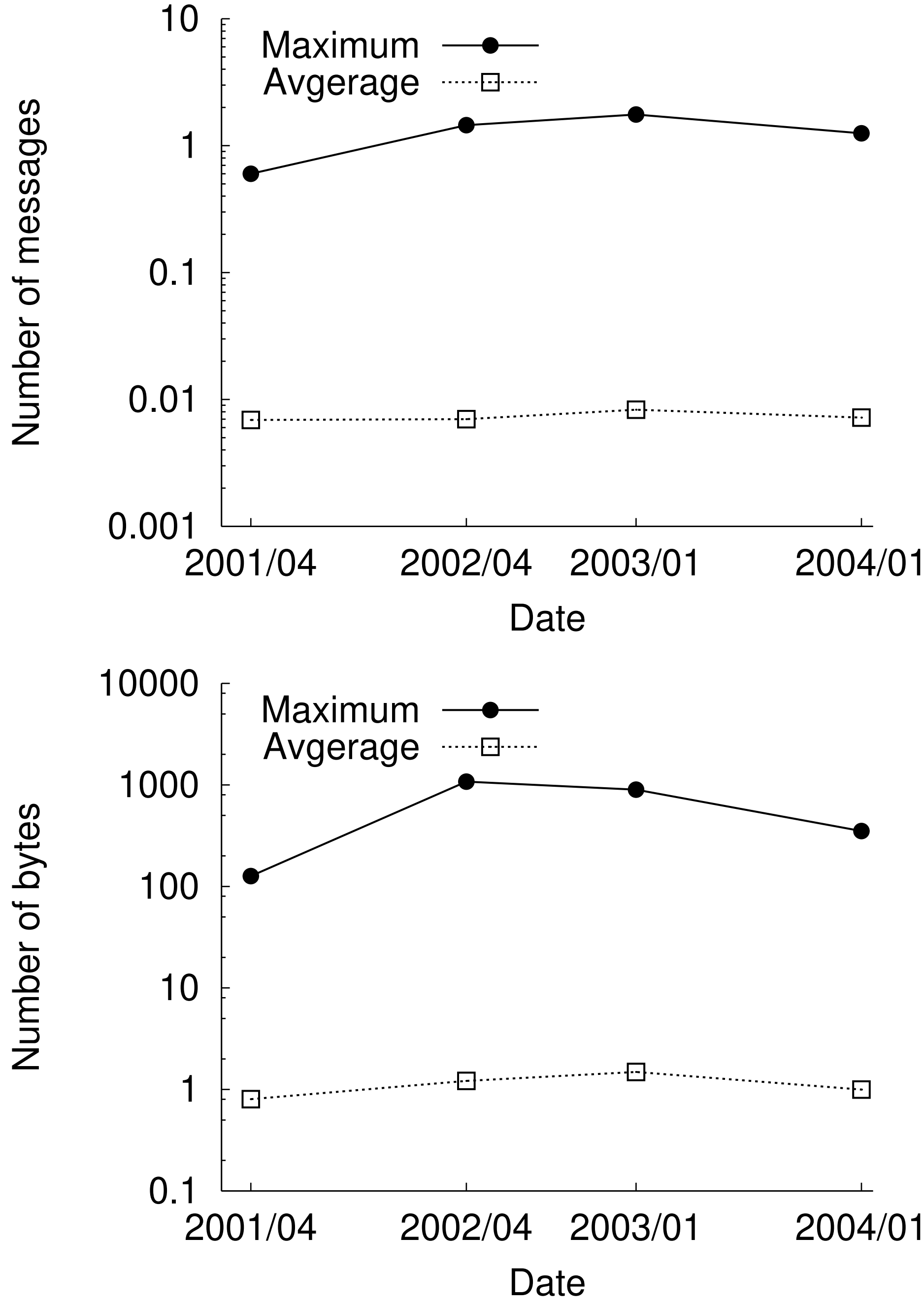


Figure 7-9: The average and maximum number of messages and bytes sent per failure per link.

churning.

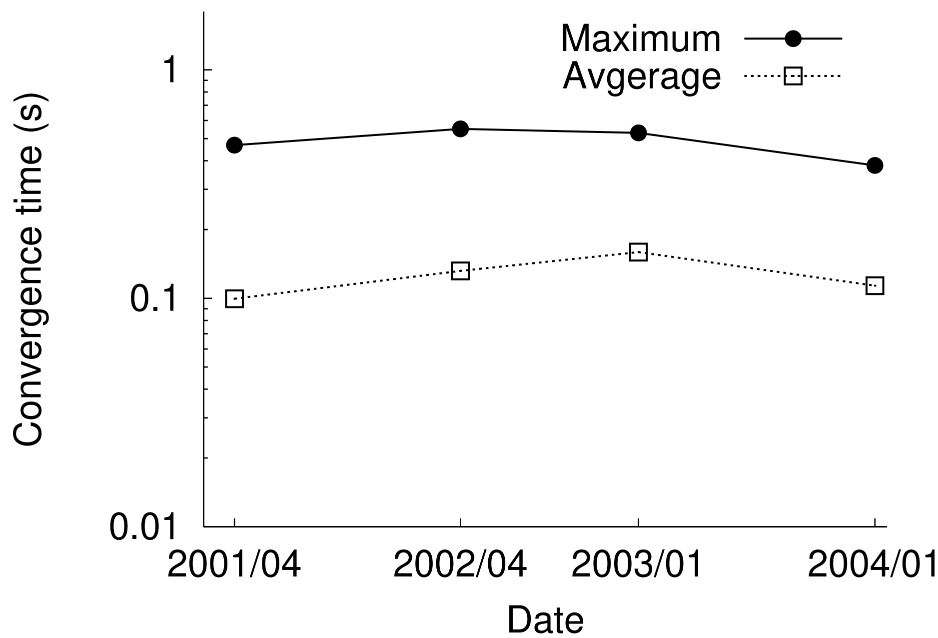


Figure 7-10: The average and maximum time elapsed between the time a failure is detected and the time the last topology message triggered by the failure is received.

To evaluate the convergence property of TIPP, we record the time period between the time when a failure is detected and the time when the last topology message triggered by the failure is received for each sequential failure event. In the simulation, the propagation delay of a link is randomly distributed between 10ms and 110ms. Figure 7-10 shows the average and the maximum convergence time over ten simulation runs. It can be seen that without considering processing time, TIPP could converge within a second for single failure events.

We then look at the message overhead when there are multiple failures. In our simulation, we let a bidirectional link randomly fail and recover. Each failure lasts for 200 seconds, because shorter failures may not be detected by a TIPP router's hold timer. The up time of a link is uniformly distributed between 200 seconds and an upper bound. The value of the upper bound determines the percentage of down time. Each run simulates 3600 seconds. Figure 7-11 shows the message overhead when there are multiple link failures. These results suggest that both the average message overhead and the maximum message overhead are low.

Our results from simulation studies suggest that TIPP has good dynamic behaviors. It has low message overhead and converges fast.

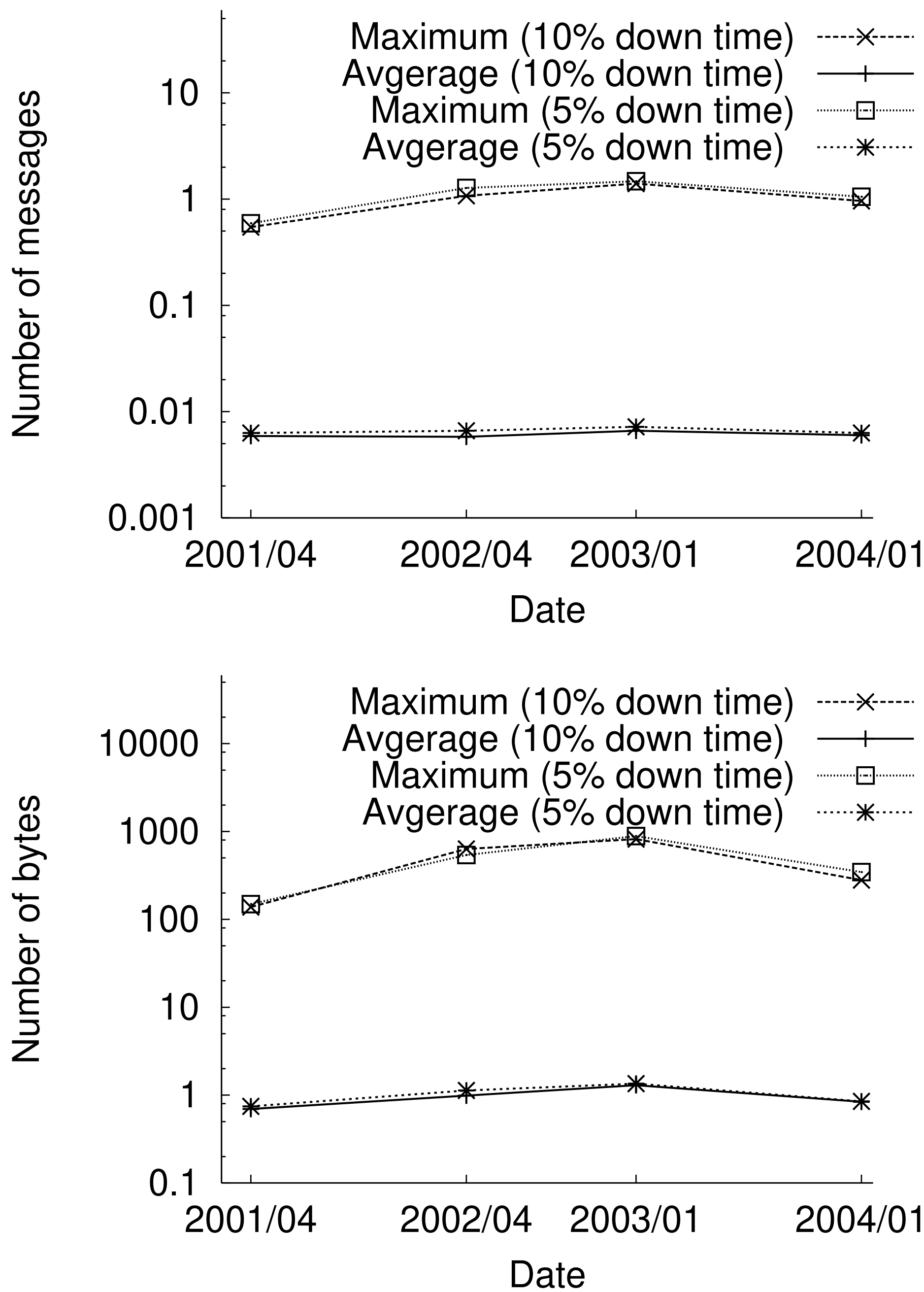


Figure 7-11: The average and maximum number of messages and bytes sent per failure per link when each link randomly fails and recovers.

7.1.3 NRLS

NRLS is an enhanced version of DNS. A user could use NRLS to obtain the addresses of a destination. Since a provider-rooted hierarchical address is a topology dependent address, domain-level topology change may change the addresses a user has. In the design of NIRA, we do not restrict how a user updates his records at NRLS servers. As we have discussed in Chapter 4.4.1, we think the workload for NRLS updates is likely to be manageable, because domain-level Internet topology alters at a frequence of a few dozen changes per day, and the changes can be scheduled so that a grace period could be granted for users to update their records before the actual changes happen. Here, we use back of envelop calculation to confirm our intuition that the workload for NRLS updates is manageable.

Suppose an NRLS server could handle 1000 updates per second. If a topology change affects the records of 100,000 users at an NRLS server, then it takes on the order of 100 seconds of NRLS server processing time to update the records. If an update is on average of the size 1000 bytes, and we restrict the bandwidth consumed on updating NRLS records to be 5% of 100Mb/s, then it takes about 160 seconds to transfer 100,000 record updates. These numbers suggest that the workload for NRLS updates is likely to be manageable. Moreover, note that if a single NRLS server is overloaded with route records, the server can always use standard load-balancing techniques to shed load to multiple machines. So we do not think NRLS updates would become the performance bottleneck of NIRA.

7.2 Route Availability Discovery

The basic mechanism we provide for route availability discovery is a combination of proactive and reactive notification. If a user does not know ahead of time the availability of a route, the user will rely on router feedback or timeouts to detect route failures.

The reactive notification mechanism reduces the need to propagate route availability information throughout the network, but could potentially increase the latency for sending a packet when a route is unavailable. If a user sends a packet along a route that is unavailable, the packet would not reach its destination. If the user waits until he receives a router notification or a timeout event to try a different route to resend the packet, then the overall latency to send the packet will increase.

Although in our design of NIRA, we do not limit how users handle route availability discovery and failure handling, we would like to analyze the latency tradeoff of the basic route availability discovery mechanism provided in NIRA. A user might send probes along multiple routes to monitor the availability of those routes, and always send packets along a route that is available to reduce the latency for sending a packet. Our analysis assumes a conservative approach for failure detection and handling: a user depends on the basic proactive and reactive notification mechanisms to detect route unavailability, and will resend a packet along a different route when he discovers the original route he chooses to send the packet is unavailable. This conservative assumption gives us an upper bound on the increase of packet-sending latency.

Intuitively, if routes are highly available, then most likely a packet will reach its destination the first time when a user sends it. So the increase on packet-sending latency incurred by reactive notifications will not be significant. We use a simple analytic model to test this hypothesis.

In our model, we assume that when a user sends a packet to a destination, the user knows the name of the destination. Each user keeps a NRLS cache that stores the query results from NRLS. When a user sends a packet, if there is a cache hit for the destination name, the user will pick a route and send the packet without querying NRLS; if there is a cache miss, the user will query NRLS to obtain the addresses of the destination. When a user sends a packet, if the user detects a route is unavailable, the user will resend the packet via a different route. A user will use this trial-and-error mechanism to send all his packets, including the packets to query NRLS.

For simplicity, we assume that the round trip time from a user to any destination is the same, and the name hierarchy of any destination is the same, and the timeout value for detecting any route unavailability (or simply route failure) is the same, and the cache hit probability for any destination name is the same. We also assume that the probability a route is unavailable is independent of that of any other route and is the same for all routes, and a router sends a notification back to a user with certain probability. In our model, the round trip time, the name hierarchy, the timeout value, the cache hit rate, the route unavailable probability, and the router notification probability are all tunable parameters. These assumptions do not accurately capture every detail of network operations, but allow us to get a first order estimation of the latency to successfully send a packet.

We model the process of successfully sending a packet as a negative multinomial distribution [45]. In Appendix 7.A, we describe the details of the analytic model. This model could be used to estimate both the connection setup latency and the latency for successfully sending a packet in the course of a connection. If we set the cache hit probability to be 100%, then the latency computed by the model estimates the latency for successfully sending a packet in the course of a connection; if we cache hit rate is less than 100%, then the latency computed by the model approximates the latency for sending the first packet to a destination, i.e., the connection set up latency.

We use the analytic model to compute the latency for successfully sending a packet. Figure 7-12 shows the cumulative distribution of the latency with 80%[2] NRLS cache hit probability, a 3-level name hierarchy (i.e. two queries to NRLS servers are needed to resolve a destination name), a 100ms round trip time, a 3-second timeout value for route unavailability detection, 1% route failure probability, and variable fractions of router notification out of all failure events.

[2]We pick 80% cache hit probability as Jung et al [62] have shown that DNS cache hit rate exceeds 80% with a time-to-live of 15 minutes.

Figure 7-14 shows the result when we change the route failure probability to be 5%. Figure 7-13 and Figure 7-15 show the complementary distribution[3] for 1% and 5% route failure probability respectively. From the complementary distribution, we can see the tail part of the distribution clearly.

From those figures, we can see that when routers could always send failure notification to senders, the connection setup latency is within a couple of round trip times with a high probability (> 99.99%). When senders have to depend on timeouts to detect route failures, the connection setup latency increases. Still, with a probability of > 99.9% for 1% route failure probability and 99.5% for 5% route failure probability, the connection setup latency is less than one timeout period plus a few round trip times.

Figure 7-16 shows how the expected connection setup latency changes with different route failure probabilities. The other parameter values are the same: 80% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 3-second timeout value for route failure detection. With 1% failure probability, the expected latency changes from 141ms to 182ms as the router notification fraction decreases from 100% to 0; With 5% failure probability, the expected latency changes from 147ms to 361ms as the router notification fraction decreases from 100% to 0.

Figure 7-17 and Figure 7-18 show the requirements on the cache hit probability and the fraction of router notification in order to meet the performance requirement that the connection setup latency is less than 0.5 second with certain probability for 1% and 5% route failure probability respectively. The other parameter values are the same as in previous figures. The area above each curve is the area that satisfies the specific performance requirement noted in the legend. From these figures, we can see that when the cache hit probability is 80%, the fraction of router notification needs to exceed 28% out of 1% route failure probability and 87% out of 5% route failure probability for the connection set up latency to be less than 0.5 second 99% of time.

Figure 7-19 and Figure 7-20 show the requirements on the cache hit probability and the fraction of router notification in order to meet the performance requirement that the expected connection setup latency is less than a certain value for 1% and 5% route failure probability respectively. The other parameter values are the same as in previous figures. From these figures, we can see that when the cache hit probability is 80%, the fraction of router notification needs to exceed 80% out of 1% route failure probability and 99% out of 5% route failure probability for the expected connection set up latency to be less than 0.15 second. If we relax the requirement on the expected connection set up latency to be less than 0.20 second, then we do not need router notification in the case of 1% failure probability, and the fraction of router notification out of 5% failure probability needs to exceed 75%.

We estimate the latency for successfully sending a packet in the middle of a connection by setting the cache hit probability to be 100%. We also assume that the timeout value for detecting a

[3] The complementary distribution function for a random variable X is defined as $Prob\{X > x\} = 1 - Prob\{X \leq x\}$.

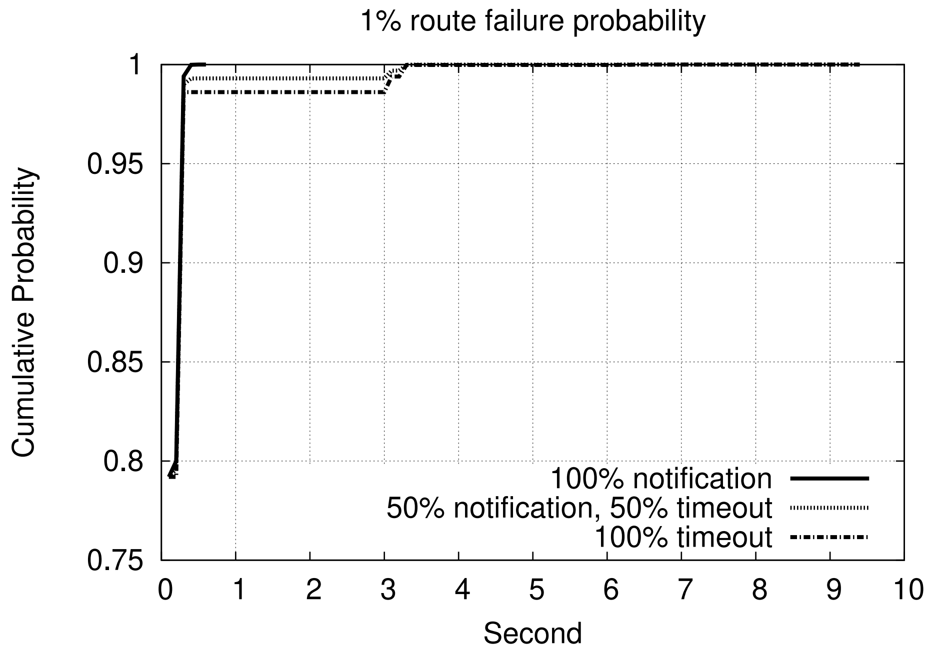


Figure 7-12: The cumulative distribution of the connection setup latency with 1% route failure probability. Other parameter values: 80% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 3-second timeout value for route failure detection.

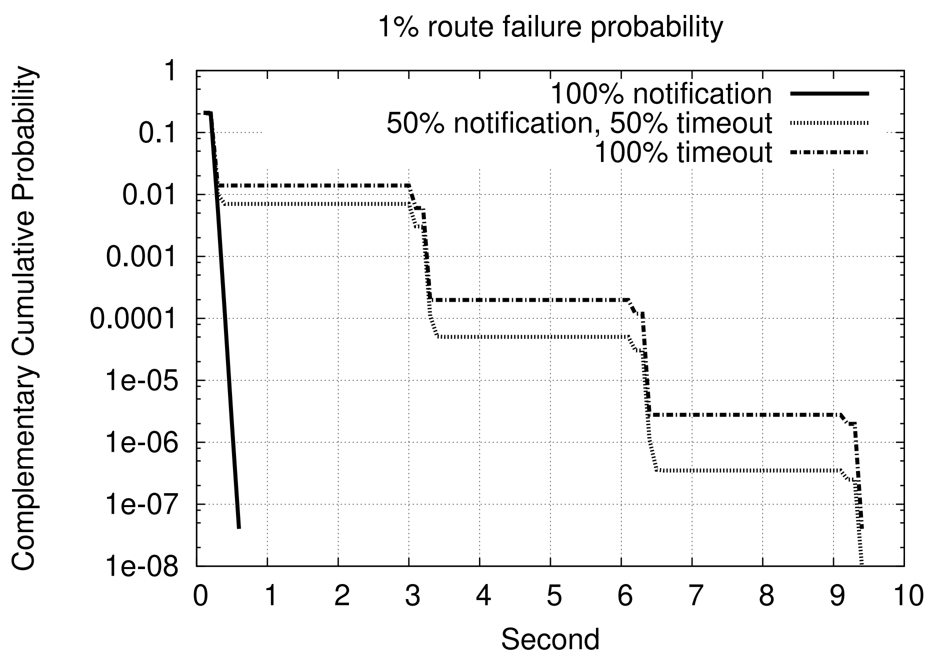


Figure 7-13: The complementary distribution of the connection setup latency with 1% route failure probability. Other parameter values are the same as above.

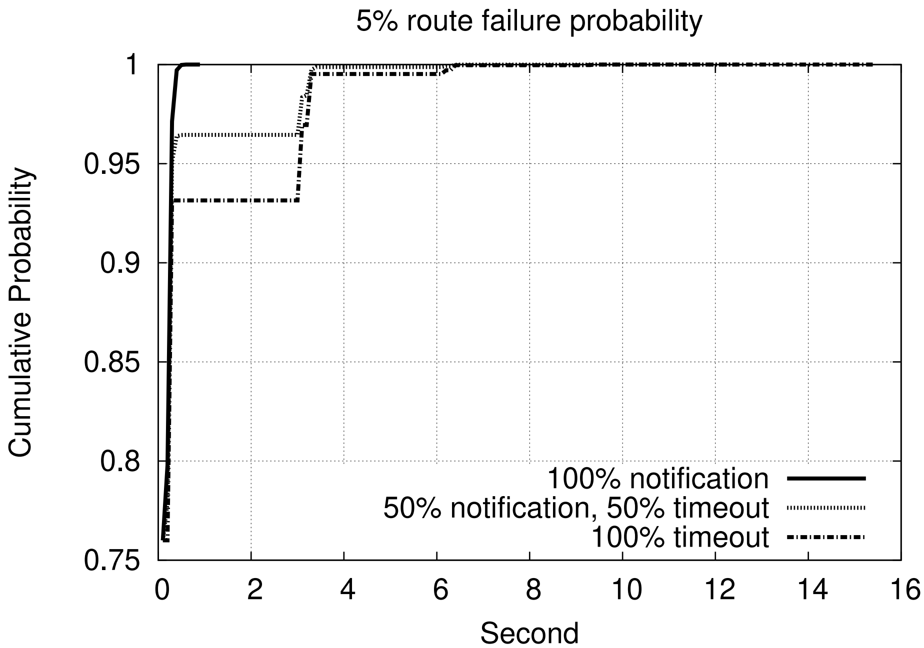


Figure 7-14: The cumulative distribution of the connection setup latency with 5% route failure probability. Other parameter values: 80% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 3-second timeout value for route failure detection.

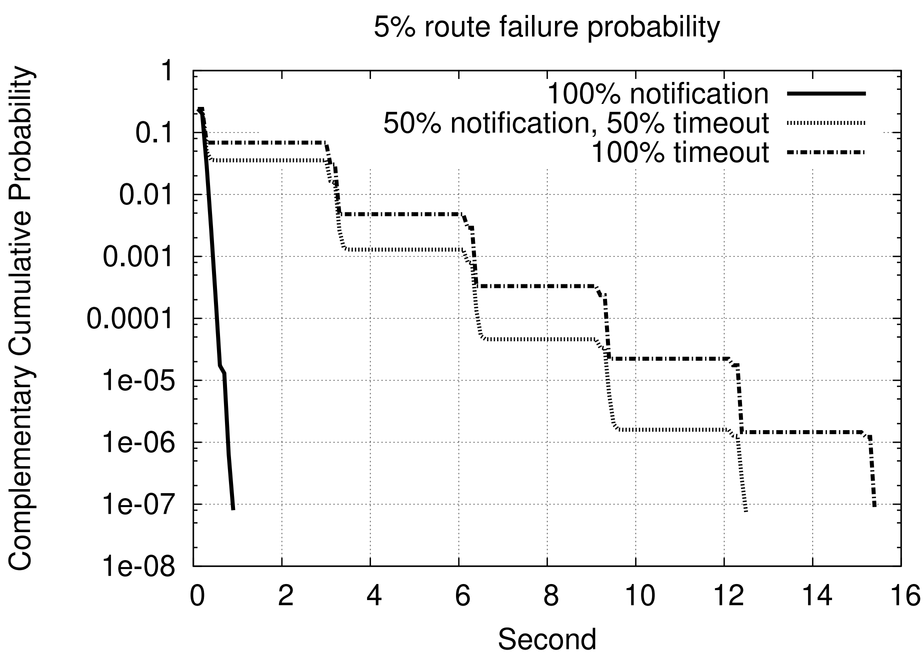


Figure 7-15: The complementary distribution of the connection setup latency with 5% route failure probability. Other parameter values are the same as above.

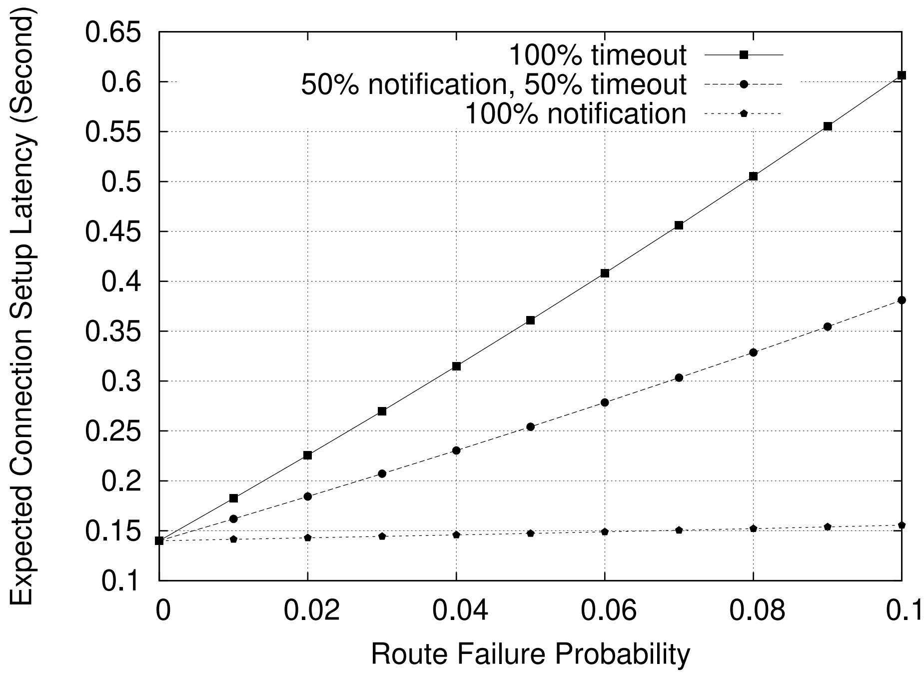


Figure 7-16: How the expected connection setup latency varies with different route failure probabilities. Other parameter values: 80% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 3-second timeout value for route failure detection.

route failure could be shortened in the middle of a connection because the application that opens the connection would know better about the round trip time of the connection. Since we assume a 100ms round trip time, we assume a 800ms timeout value in the middle of a connection. It is about the value of three exponential backoffs of the round trip time: 200ms, 400ms, 800ms. We still assume 3-level of name hierarchy. Figure 7-21 and 7-23 show the cumulative distribution of the latency for successfully sending a packet in the middle of a connection with 1% and 5% route failure probability. Figure 7-22 and Figure 7-24 show the complementary distribution for 1% and 5% route failure probability respectively.

The results shown in these figures are intuitive. Without considering NRLS cache miss, it will take one round trip time to send a packet when there is no route failure. When routers can always send notifications out of all route failures, the probability that it takes n round trip times to successfully send a packet decreases exponentially with n. So for 1% route failure probability, with probability 99%, a packet will go through in one round trip time, with probability 99.99%, a packet will go through in two round trip time. If routers cannot always send notifications when a route failure occurs, then the probability it takes n timeouts plus a round trip time to successfully send a packet decreases exponentially with n. So for 1% route failure probability, without any router notification, with probability 99.99%, the packet will go through within a timeout interval plus a round trip time. Similar results are shown for the case with 5% route failure probability.

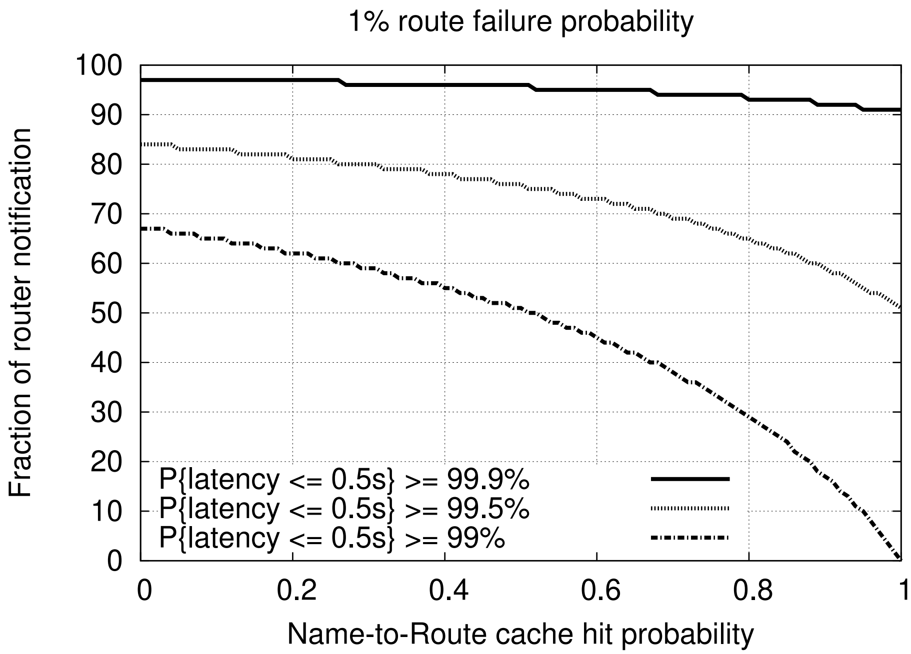


Figure 7-17: The required cache hit probability and the fraction of router notification out of all route failure events when the connection setup latency is less than 0.5 second with certain probability. Other parameter values: 1% route unavailable probability; 3-level of name hierarchy; 3 second timeout.

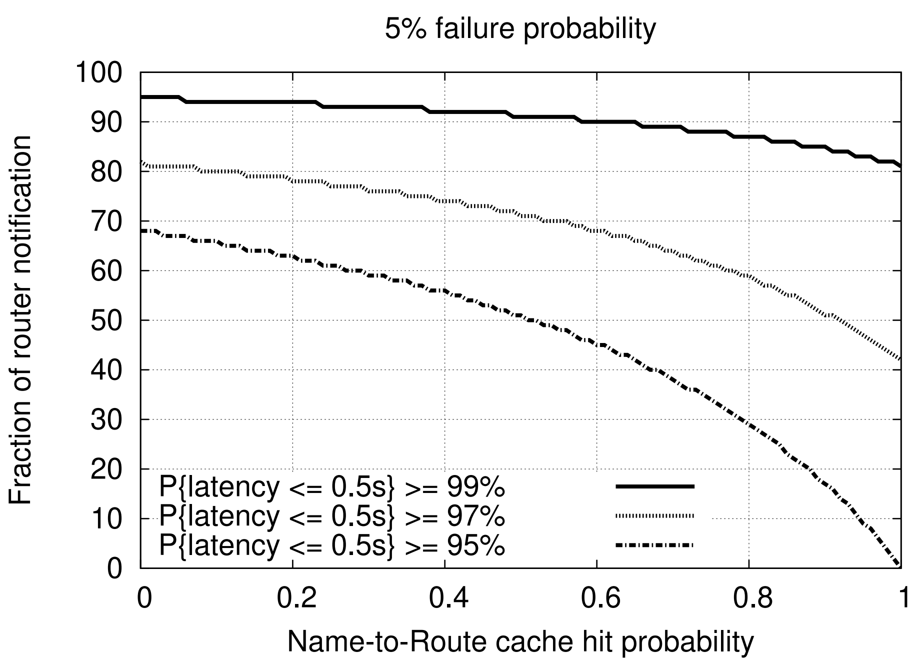


Figure 7-18: The required cache hit rate and the fraction of router notification out of all route failure events when the connection setup latency is less than 0.5 second with certain probability. Other parameter values: 5% route unavailable probability; 3-level of name hierarchy; 3 second timeout.

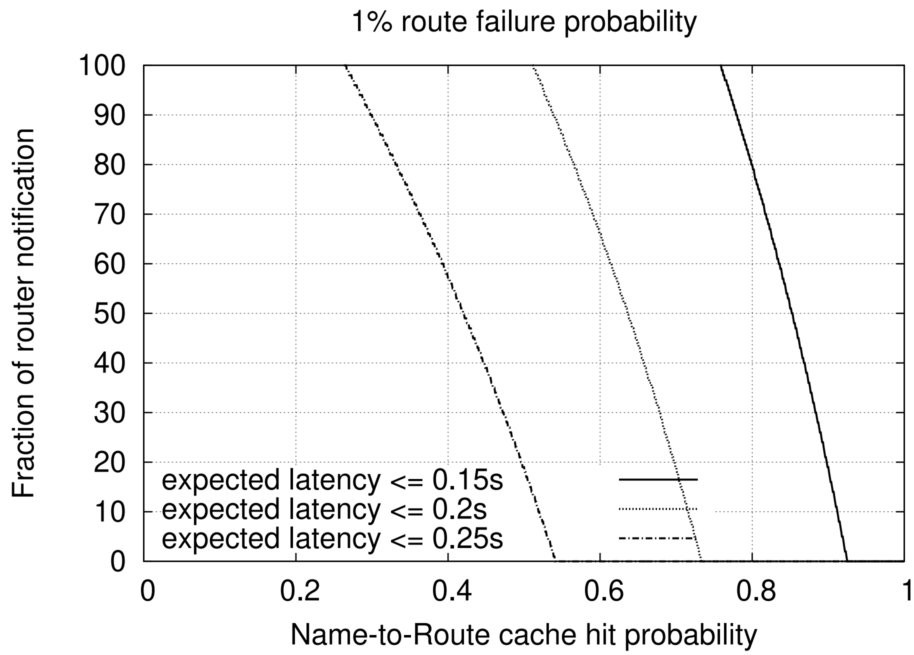


Figure 7-19: The required cache hit rate and the router notification probability when the expected connection setup latency is less than certain value. Other parameter values: 1% route unavailable probability; 3-level of name hierarchy; 3 second timeout.

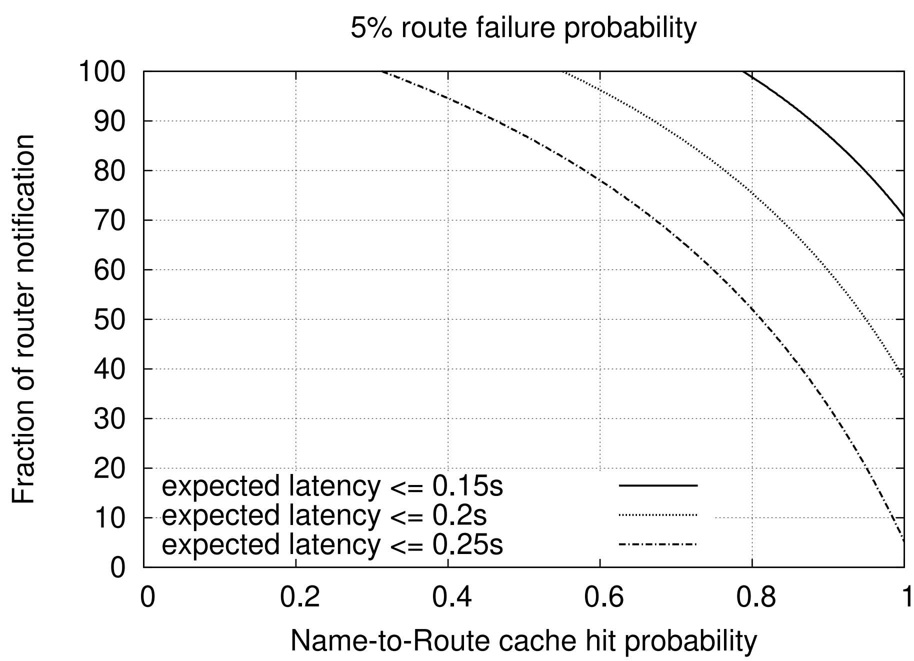


Figure 7-20: The required cache hit rate and the router notification probability when the expected connection setup latency is less than 1 second. Other parameter values: 5% route unavailable probability; 3-level of name hierarchy; 3 second timeout.

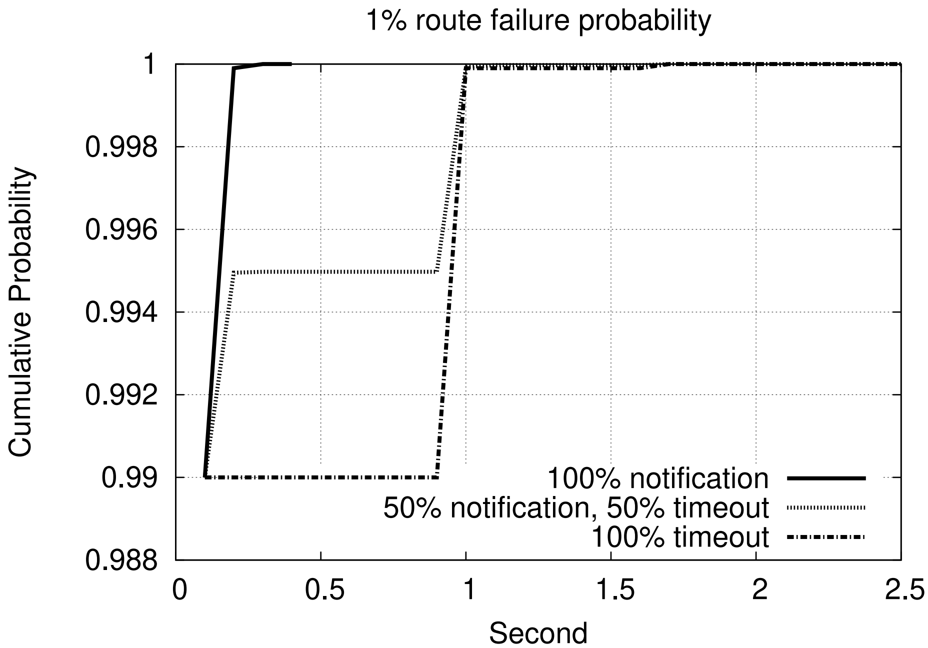


Figure 7-21: The cumulative distribution of the latency for successfully sending a packet in the middle of a connection with 1% route failure probability. Parameter values: 100% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 800ms timeout value for route failure detection.

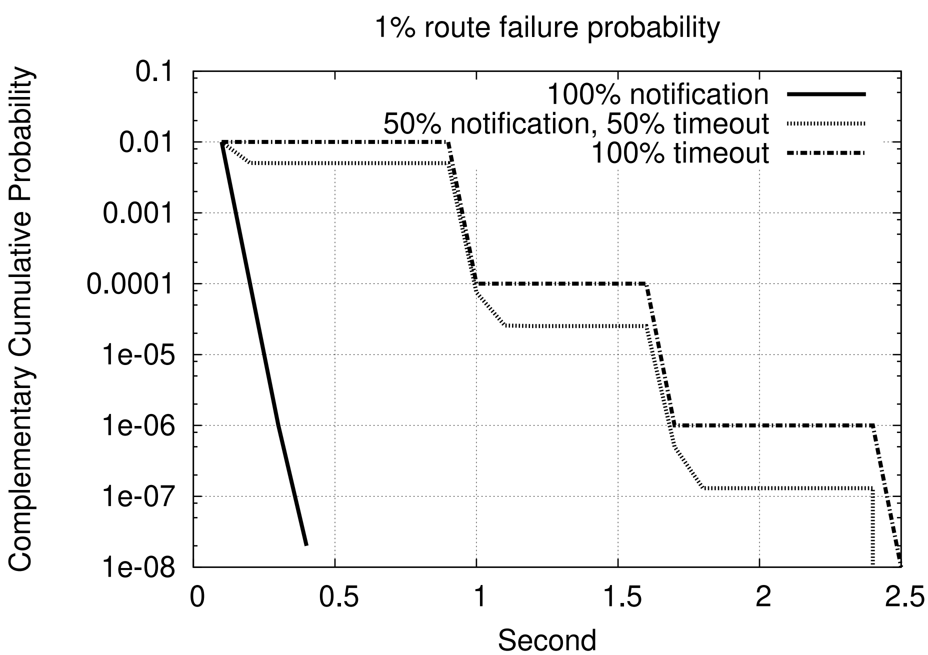


Figure 7-22: The complementary distribution of the latency for successfully sending a packet in the middle of a connection. Parameter values are the same as those in the figure above.

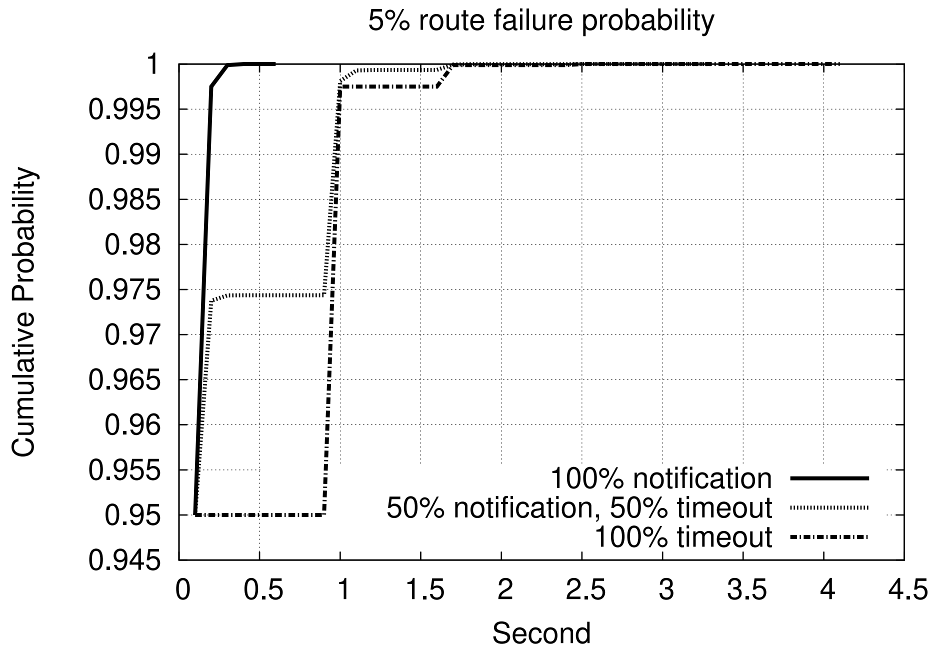


Figure 7-23: The cumulative distribution of the latency for successfully sending a packet in the middle of a connection with 5% route failure probability. Parameter values: 100% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 800ms timeout value for route failure detection.

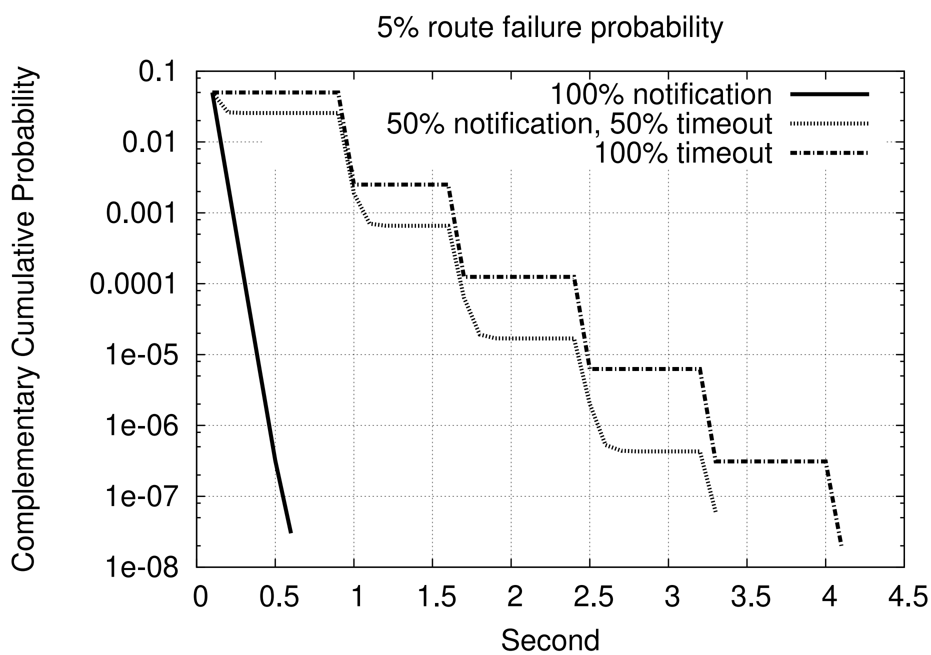


Figure 7-24: The complementary distribution of the latency for successfully sending a packet in the middle of a connection. Parameter values are the same as those in the figure above.

Figure 7-25 shows how the expected latency for sending a packet in the middle of a connection varies with different route failure probabilities. With 1% route failure probability, the expected latency increases from 101ms to 108ms when the fraction of route notification changes from 100% to 0; with 5% route failure probability, the expected latency increases from 105ms to 142ms when the fraction of route notification changes from 100% to 0.

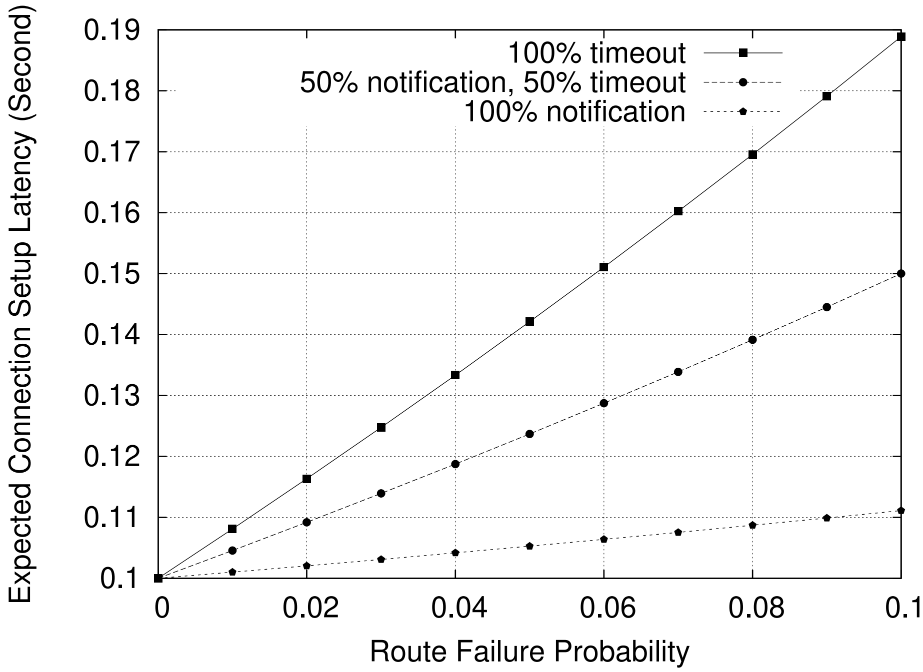


Figure 7-25: How the expected latency for successfully sending a packet in the middle of a connection varies with different route failure probabilities. Parameter values: 100% NRLS cache hit probability, 3-level of name hierarchy, 100ms round trip time, 800ms timeout value for route failure detection.

Appendix 7.A Modeling the Latency for Successfully Sending a Packet

In this section, we describe the analytic model we use to estimate the latency for successfully sending a packet under various operating conditions. As we discussed in the main text Section 7.2, we assume when a user sends a packet to a destination, he uses either router notifications or timeouts to detection route failures. When a failure is detected, the user will try to resend the packet over a different route. We assume failures on different routes are independent and identically distributed.

We present the notations for describing various events and variables related to sending a packet as follows.

- A: NRLS cache hit.
- $\bar{A}$: NRLS cache miss.
- B: the number of successful NRLS queries needed in order to learn the addresses of a destination. For a 3-level of name hierarchy, $B = 2$.
- C: the time interval to send a NRLS query and get a response that is a success.
- D: the time interval to send a NRLS query and get a timeout at the sender.
- E: the time interval to send a NRLS query and get a route failure notification from a router.
- F: the time interval to send a packet to its destination without encountering a route failure.
- G: the time interval to send a packet and get a route failure notification from a router.
- H: the time interval to send a packet and gets a timeout at the sender.
- I: the total latency to successfully send a packet.
- L: the latency to successful learn the addresses of a destination from recursive NRLS queries.
- M: the latency to successfully send a packet when there is NRLS a cache hit.

Various sample paths could occur in the process of sending a packet. For example,

- When the event A occurs, a few possible sample paths are shown as follows:

$$
\begin{aligned}
I &= F \\
I &= G + F \\
I &= G + G + ... + F \\
I &= G + H + G + H + ... + F
\end{aligned}
$$

Let's call a sample path when A occurs α.

- When the event $\bar{A}$ occurs and $B = n$, the following sample paths are possible:

$$
\begin{aligned}
I &= \underbrace{C + ... + C}_{n \text{ success}} + \underbrace{D + ... + D}_{k \text{ timeouts}} + \underbrace{E + ... + E}_{l \text{ failure notifications}} + \alpha \\
&= \sum_{i=1}^{n} C_i + \sum_{k=1}^{K} D_k + \sum_{l=1}^{L} E_l + \alpha
\end{aligned}
$$

7.A.1 Distribution of I

We derive the latency distribution using conditional probabilities. We first divide the sample paths into two groups: 1. the group when an NRLS cache hit occurs, and 2. the group when an NRLS cache miss occurs.

$$
\begin{aligned}
Prob\{I \leq x\} &= Prob\{I \leq x|A\} \cdot Prob\{A\} + Prob\{I \leq x|\bar{A}\} \cdot Prob\{\bar{A}\} \\
&= Prob\{M \leq x\} \cdot Prob\{A\} + Prob\{L + M \leq x\} \cdot Prob\{\bar{A}\}
\end{aligned}
$$

A typical sample path in the first group (NRLS cache hit) includes a random number (N_G) of event Gs (router notifications), a random number (N_H) of event Hs (timeouts), and an event F (a packet gets through without encountering a route failure). The probability that the latency is less a value x ($Prob\{M \leq x\}$) can be computed by adding up the probabilities conditioned on the values of N_G and N_H.

$$
\begin{aligned}
Prob\{M \leq x\} &= Prob\{\sum_{i=1}^{N_G} G_i + \sum_{i=1}^{N_H} H_i + F \leq x\} \\
&= \sum_{n_G, n_H} Prob\{\sum_{i=1}^{n_G} G_i + \sum_{i=1}^{n_H} H_i + F \leq x|N_G = n_G, N_H = n_H\} \cdot Prob\{N_G = n_G, N_H = n_H\} \\
&= \sum_{n_G, n_H} Prob\{\sum_{i=1}^{n_G} G_i + \sum_{i=1}^{n_H} H_i + F \leq x|N_G = n_G, N_H = n_H\} \\
&\quad \cdot \frac{(n_G + n_H)!}{n_G! n_H!} (1 - P_G - P_H) \cdot P_G^{n_G} \cdot P_H^{n_H}
\end{aligned}
$$

A typical sample path in the second group (NRLS cache miss) includes events that occur in resolving a destination name, and events that occur for sending a packet after the addresses of the destination is learned, which are essentially events that occur with an NRLS cache hit. So a typical sample path in this group includes a random number (N_C) of event Cs (a successful response of an NRLS query), a random number (N_D) of event Ds (a timeout during an NRLS query), a random number of (N_E) of event Es (a router failure notification during an NRLS query), and events that could occur when there is an NRLS cache hit. The probability that the latency is less than a value x ($Prob\{L+M \leq x\}$) can be computed by summing up the probabilities conditioned on the number of occurrences of different events.

$$\begin{aligned}
Prob\{L + M \leq x\} &= Prob\{\sum_{i=1}^{N_C} C_i + \sum_{i=1}^{N_D} D_i + \sum_{i=1}^{N_E} E_i + \sum_{i=1}^{N_G} G_i + \sum_{i=1}^{N_H} H_i + F \leq x\} \\
&= \sum_{n_C,n_D,n_E,n_G,n_H} Prob\{\sum_{i=1}^{n_C} C_i + \sum_{i=1}^{n_D} D_i + \sum_{i=1}^{n_E} E_i + \sum_{i=1}^{n_G} G_i + \sum_{i=1}^{n_H} H_i + F \leq x\} \\
&\qquad \cdot Prob\{N_C = n_C, N_D = n_D, N_E = n_E, N_G = n_G, N_H = n_H\} \\
&= \sum_{n_C,n_D,n_E,n_G,n_H} Prob\{\sum_{i=1}^{n_C} C_i + \sum_{i=1}^{n_D} D_i + \sum_{i=1}^{n_E} E_i + \sum_{i=1}^{n_G} G_i + \sum_{i=1}^{n_H} H_i + F \leq x\} \\
&\qquad \cdot Prob\{N_C = n_C, N_D = n_D, N_E = n_E\} \cdot Prob\{N_G = n_G, N_H = n_H\} \\
&= \sum_{n_C,n_D,n_E,n_G,n_H} Prob\{\sum_{i=1}^{n_C} C_i + \sum_{i=1}^{n_D} D_i + \sum_{i=1}^{n_E} E_i + \sum_{i=1}^{n_G} G_i + \sum_{i=1}^{n_H} H_i + F \leq x\} \\
&\qquad \cdot \frac{(n_C + n_D + n_E - 1)!}{n_D! n_E! (n_C - 1)!} (1 - P_D - P_E)^{n_C} P_D^{n_D} P_E^{n_E} \cdot Prob\{N_C = n_C\} \\
&\qquad \cdot \frac{(n_G + n_H)!}{n_G! n_H!} (1 - P_G - P_H) P_G^{n_G} P_H^{n_H}
\end{aligned}$$

For simplicity, we assume that the round trip time to reach any destination in the network is the same, and the timeout values are the same, and the probabilities for receiving a router notification or a timeout event are the same over all routes. That is,

- $G_i = E_i = T_{rtt}$
- $C_i = F = T_{rtt}$
- $D_i = H_i = T_{out}$
- $P_G = P_E = P_n,$ probability of router notification
- $P_D = P_H = P_o,$ probability of timeout

With these assumptions, we can obtain the latency distribution as follows:

$$\begin{aligned}
Prob\{I \leq x\} &= Prob\{A\} \cdot \sum_{n_G,n_H} Prob\{n_G T_{rtt} + n_H T_{out} + T_{rtt} \leq x\} \frac{(n_G + n_H)!}{n_G! n_H!} (1 - P_n - P_o) P_n^{n_G} P_o^{n_H} + \\
&\quad Prob\{\bar{A}\} \cdot \sum_{n_C,n_D,n_E,n_G,n_H} Prob\{n_C T_{rtt} + n_D T_{out} + n_E T_{rtt} + n_G T_{rtt} + n_H T_{out} + T_{rtt} \leq x\}
\end{aligned}$$

$$\cdot\frac{(n_C+n_D+n_E-1)!}{n_D!n_E!(n_C-1)!}(1-P_n-P_o)^{n_C}P_o^{n_D}P_n^{n_E}Prob\{N_C=n_C\}$$
$$\cdot\frac{(n_G+n_H)!}{n_G!n_H!}(1-P_n-P_o)P_n^{n_G}P_o^{n_H}$$

In our numerical computation, we assume that the name hierarchy has a fixed level 3. So two successful NRLS queries are needed to resolve a destination name. We have $Prob\{N_C = 2\} = 1$, and $Prob\{N_c = n\} = 0, \forall n \neq 2$.

7.A.2 Expected Value of I

Similarly, we can compute the expected latency for successful sending a packet.

$$\begin{aligned}
E[I] &= E[I|A]\cdot Prob\{A\} + E[I|\bar{A}]\cdot Prob\{\bar{A}\} \\
&= E[M]\cdot Prob\{A\} + E[L+M]\cdot Prob\{\bar{A}\} \\
&= E[\sum_{i=1}^{N_G} G_i + \sum_{i=1}^{N_H} H_i + F]\cdot Prob\{A\} + \\
&\quad E[\sum_{i=1}^{N_G} G_i + \sum_{i=1}^{N_H} H_i + F + \sum_{i=1}^{N_C} C_i + \sum_{i=1}^{N_D} D_i + \sum_{i=1}^{N_E} E_i]\cdot Prob\{\bar{A}\} \\
&= (\sum_{n_G,n_H}(n_G E[G_i] + n_H E[H_i] + E[F])\cdot\frac{(n_G+n_H)!}{n_G!n_H!}(1-P_G-P_H)\cdot P_G^{n_G}P_H^{n_H})\cdot Prob\{A\} + \\
&\quad (\sum_{n_G,n_H,n_C,n_D,n_E}(n_G E[G_i] + n_H E[H_i] + E[F] + n_C E[C_i] + n_D E[D_i] + n_E E[E_i]) \\
&\quad \cdot\frac{(n_C+n_D+n_E-1)!}{n_D!n_E!(n_C-1)!}(1-P_D-P_E)^{n_C}P_o^{n_D}P_n^{n_E}Prob\{N_C=n_C\} \\
&\quad \cdot\frac{(n_G+n_H)!}{n_G!n_H!}(1-P_G-P_H)P_n^{n_G}P_o^{n_H})\cdot Prob\{\bar{A}\}
\end{aligned}$$

Chapter 8

Conclusion and Future Work

In this chapter, we summarize the contributions and limitations of this work, and discuss directions for future research.

8.1 Contributions

This dissertation presents the design of NIRA, a new Internet routing architecture. NIRA aims to solve two problems in the present Inter-domain routing architecture. First, the current architecture has little support for user choice in the form of user-selected routes. User choice is believed to play an important role in creating a healthy and competitive ISP market. Second, it does not scale effectively due to the failure to aggregate address prefixes from multi-homed sites.

To the best of our knowledge, this is the first work that addresses a full range of the design challenges for a scalable architecture that supports domain-level route selection, which include route discovery and selection, route availability discovery, route representation and packet forwarding, and provider compensation. We evaluated most parts of our design using simulation, analysis, and network measurement.

At an architectural level, our work demonstrates a new modularized approach to design the inter-domain routing architecture. We leverage modularity not only for reducing complexity, but for reducing architectural constraints and allowing generality. We decompose the design into modules, design basic mechanisms to achieve the functionality of each module, and allow new mechanisms to be invented to achieve the same functionality of one module without significantly affecting mechanisms designed for other modules. Below are examples that illustrate this design approach:

- We decomposed the design of the inter-domain routing architecture into three modules: route discovery and selection, route representation and packet forwarding, and provider compensation.

- We unbundled the route availability discovery from route discovery.
- We divided the task of route discovery into two halves, a sender half and a receiver half. Each user could independently learn his half of the network. A sender retrieves the receiver part of the network on demand, and combines his knowledge on his part of the network to select routes.

At a technical level, our work has two primary contributions.

1. We designed a new network protocol, TIPP, that propagates inter-domain address allocation information and topology information to users. TIPP facilitates inter-domain address management and the task of route discovery. Simulation studies based on topologies derived from real Internet measurement show that TIPP scales well and has good dynamic behaviors.
2. We designed a route representation scheme and a packet forwarding algorithm, such that a source and a destination address could represent a common type of domain-level route. We used induction proof to show that our route representation scheme and forwarding algorithm work correctly.

8.2 Limitations

One of our motivations for this work is to come up with an engineering design that has the potential to shape ISP market structure. We hope to encourage and reward competition in the wide-area ISP market by supporting user choice in the form of domain-level routes.

It is true that whether the ISP market will adopt our design is out of our control. Yet we argue this work is a worthy experiment. It is a proof-of-concept that technically it is doable to give a user the ability to select domain-level routes. In the future, if the ISP market welcomes user choice, our work can serve as a viable solution.

8.3 Future Work

This dissertation attempts to address the full range of design problems that arise from supporting domain-level route selection. Unavoidably, some areas of this work are preliminary, and require future study.

First, when users are able to select domain-level routes, providers need to perform policy checking to prevent illegitimate route usage. We discussed how policy checking might be done, but have not designed detailed algorithms, or compared existing algorithms for policy checking. So how to implement policy checking in an efficient way is an area that needs much future work.

Second, we discussed how a user might pick a route and send a packet, but we have not studied in detail how users select routes that best satisfy their performance requirements and minimize their costs. This problem itself has two future directions. One direction is to look at what information is needed for optimizing user choice, and how to provide this information via TIPP and NRLS. The other direction is to investigate how to design the user agent software that automatically manages route selection on a human user's behalf.

Third, in this work, the primary motivation to support user route selection is to stimulate ISP competition. We have not looked at the technical impacts of such choice. First, will user choice lead to traffic oscillation between congested and uncongested route? If yes, then what is the performance penalty of such oscillation, and how can we design control mechanisms to prevent such oscillation? Second, how domain-level user choice will interact with intra-domain traffic engineering, and do we need new mechanisms for intra-domain traffic engineering? Third, can we quantify the performance gain if a user could pick domain-level routes?

Fourth, user's ability to choose routes might impact the ISP market. Presently, business agreements between ISPs or business agreements between users and ISPs are often long lasting (i.e., on the scale of a month or a year). With NIRA, users can adjust their route choices to optimize cost and performance on a much finer time scale, such as a round trip time. Dynamic choices may encourage ISPs to offer more dynamic contracts or to deploy more dynamic pricing strategies. It is interesting to investigate how the ISP market might change and what new services the routing system needs to provide to support the change.

Fifth, in NIRA, a sender picks an initial route to send the first packet to a receiver. If the receiver has a different route preference from that of the sender, the receiver and the sender might negotiate to pick a different route. It is possible that the sender and the receiver have conflicting interests and each behaves selfishly. For example, the sender prefers a route that is cheap to him but expensive to the receiver, but the receiver prefers a different route that is cheap to him but expensive to the sender. It is worth future effort to look at how to design protocols to help a sender and a receiver resolve conflicts and select a route for communication.

Finally, we conducted our evaluation in a simulation environment. In the future, we would like to test our design in a more realistic environment, such as the PlanetLab [10].

Bibliography

[1] *The ATM Forum*. http://www.atmforum.com/.

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

[3] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[4] Opnix. http://www.opnix.com, June 2003.

[5] RouteScience. http://www.routescience.com/, June 2003.

[6] Source Address Selection in IPv6 Multihomed Multi-addressed Sites. IETF multi6 mailing list, July 2003. http://dict.regex.info/ipv6/multi6/2003-07.mail/0038.html.

[7] Comcast. http://www.comcast.com/, July 2004.

[8] Internap. http://www.internap.com/products/route-optimization.htm, May 2004.

[9] ISP Filter Policies. http://www.nanog.org/filter.html, June 2004.

[10] The PlanetLab. http://www.planet-lab.org/, July 2004.

[11] RIR Comparative Policy Overview. http://www.ripe.net/rs/rir-comp-matrix-rev.html, July 2004.

[12] Route Filtering Used by AS6667. http://www.jippii.net/routefiltering.shtml, June 2004.

[13] Verizon online DSL. http://www22.verizon.com/, July 2004.

[14] Sharad Agarwal. *Domain Relationship Inferrence Data*. http://www.cs.berkeley.edu/~sagarwal/research/BGP-hierarchy/data/.

[15] Sharad Agarwal, Chen-Nee Chuah, and Randy H. Katz. OPCA: Robust Interdomain Policy Routing and Traffic Control. In *Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH '03)*, San Francisco, California, USA, April 2003.

[16] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating System Principles*, Banff, Canada, October 2001.

[17] Florin Baboescu and George Varghese. Scalable Packet Classification. In *Proceedings of ACM SIGCOMM*, San Diego, California, USA, August 2001.

[18] F. Baker and P. Savola. *Ingress Filtering for Multihomed Networks*. Internet Engineering Task Force, March 2004. Best Current Practice, RFC 3704.

[19] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, 46(2), 2003.

[20] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 76–84, Dallas, Texas, United States, 1998.

[21] Dimitri Bertsekas and Robert Gallager. *Data Networks*, chapter 5. Prentice Hall, Upper Saddle River, NJ 07458, second edition, 1992.

[22] Nigel Bragg. *Routing support for IPv6 Multi-homing*. Internet Engineering Task Force, November 2000. Internet Draft, Expired, `http://www.watersprings.org/pub/id/draft-bragg-ipv6-multihoming-00.txt`.

[23] Bob Briscoe. The Direction of Value Flow in Multi-service Connectionless Networks. In *Proceedings of International Conference on Telecommunicatons and E-Commerce (ICTEC'99)*, Nashville, USA, October 1999.

[24] Tian Bu, Lixin Gao, and Don Towsley. On Routing Table Growth. In *Proceedings of Global Internet*, Taipei, Taiwan, November 2002.

[25] I. Castineyra, N. Chiappa, and M. Steenstrup. *The Nimrod Routing Architecture*. Internet Engineering Task Force, August 1996. Informational, RFC 1992.

[26] Qian Chen, Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Willinger. The Origin of Power Laws in Internet Topologies Revisited. In *Proceedings of IEEE INFOCOM*, New York, New York, USA, June 2002.

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

[28] David Clark. *Policy Routing in Internet Protocols*. Internet Engineering Task Force, May 1989. RFC 1102.

[29] David Clark, John Wroclawski, Karen Sollins, and Robert Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proceedings of ACM SIGCOMM*, Pittsburgh, Pennsylvania, USA, August 2002.

[30] A. Conta and S. Deering. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. Internet Engineering Task Force, December 1998. Draft Standard, RFC 2463.

[31] Lenore J. Cowen. Compact Routing with Minimum Stretch. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–260, Baltimore, Maryland, United States, 1999.

[32] M. Crawford. *Router Renumbering for IPv6*. Internet Engineering Task Force, August 2000. Proposed Standard, RFC 2894.

[33] The Team Cymru. As path length graph. http://www.cymru.com/BGP/asnpalen01.html, May 2004.

[34] C. de Launois, O. Bonaventure, and M. Lobelle. *The NAROS Approach for IPv6 Multihoming with Traffic Engineering*, April 2003. Submitted, http://www.info.ucl.ac.be/people/delaunoi/naros/naros.html.

[35] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force, December 1998. Draft Standard, RFC 2460.

[36] S. Deering and R. Hinden. *Internet Protocol Version 6 (IPv6) Addressing Architecture*. Internet Engineering Task Force, April 2003. Proposed Standard, RFC 3513.

[37] A. Doria, E. Davies, and F. Kastenholz. *Requirements for Inter Domain Routing*. Internet Research Task Force, December 2003. Internet Draft, Expired May 31, 2004, http://www.ietf.org/internet-drafts/draft-irtf-routing-reqs-02.txt.

[38] Richard Draves. *Default Address Selection for Internet Protocol version 6 (IPv6)*. Internet Engineering Task Force, February 2003. Proposed Standard, RFC 3484.

[39] R. Droms. *Dynamic Host Configuration Protocol*. Internet Engineering Task Force, March 1997. ftp://ftp.isi.edu/in-notes/rfc2131.txt.

[40] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. Internet Engineering Task Force, July 2003. Proposed Standard, RFC 3315.

[41] Deborah Estrin, Yakov Rekhter, and Steven Hotz. Scalable Inter-Domain Routing Architecture. In *Proceedings of ACM SIGCOMM*, pages 40–52, Baltimore, Maryland, USA, August 1992.

[42] Peyman Faratin, John Wroclawski, George Lee, and Simon Parsons. Social Agents for Dynamic Access to Wireless Networks. In *Proceedings of the AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, Stanford, Pennsylvania, USA, March 2003.

[43] Nick Feamster, David G. Andersen, Hari Balakrishnan, and M. Frans Kaashoek. Measuring the Effects of Internet Path Faults on Reactive Routing. In *Proceedings of ACM SIGMETRICS 2003, San Diego, California, USA*, June 2003.

[44] Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. The Case for Separating Routing from Routers. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Portland, Oregon, USA, August 2004.

[45] William Feller. *An Introduction to Probability Theory and Its Applications*, chapter VI. John Wiley & Sons, 1968.

[46] P. Ferguson and D. Senie. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. Internet Engineering Task Force, May 2000. Best Current Practice, RFC 2827.

[47] Gregory Finn. Routing and Addressing Problems in Large Metropolitan-scale Internetworks. Technical Report ISI/RR-87-180, ISI, University of Southern California, March 1987.

[48] American Registry for Internet Numbers. Policy 2002-3: Address Policy for Multi-homed Networks. `http://www.arin.net/policy/2002_3.html`, June 2004.

[49] The ATM Forum. Private Network-Network Interface Specification Version 1.0, March 1996.

[50] Paul Francis. A Near-Term Architecture for Deploying Pip. *IEEE Network*, 7(3):30–37, May 1993.

[51] Paul Francis. Comparison of Geographical and Provider-rooted Internet Addressing. *Computer Networks and ISDN Systems*, 27(3):437–448, 1994.

[52] Paul Francis and Ramesh Govindan. Flexible Routing and Addressing for a Next Generation IP. In *Proceedings of ACM SIGCOMM*, pages 116–125, London, UK, 1994.

[53] Paul Francis and Ramakrishna Gummadi. IPNL: A NAT-Extended Internet Architecture. In *Proceedings of SIGCOMM*, pages 69–80, San Diego, California, USA, August 2001.

[54] Lixin Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions on Networking (TON)*, 9(6):733–745, December 2001.

[55] Timothy G. Griffin and Gordon Wilfong. An Analysis of BGP Convergence Properties. In *Proceedings of ACM SIGCOMM*, pages 277–288, Cambridge, Massachusetts, USA, August 1999.

[56] Sam Halabi. *Internet Routing Architectures*. Cisco Press, 2001.

[57] C. Huitema, R. Draves, and M. Bagnulo. *Host-Centric IPv6 Multihoming*. Internet Engineering Task Force, January 2003. Internet Draft, `http://www.it.uc3m.es/marcelo/draft-huitema-multi6-hosts-02.txt`.

[58] G. Huston. *Commentary on Inter-Domain Routing in the Internet*. Internet Engineering Task Force, December 2001. Informational, RFC 3221.

[59] Geoff Huston. Interconnection, Peering and Settlements – part i. *The Internet Protocol Journal*, 2(1):2–17, March 1999. `http://www.cisco.com/ipj`.

[60] Geoff Huston. Interconnection, Peering and Settlements – part ii. *The Internet Protocol Journal*, 2(2):2–24, June 1999. `http://www.cisco.com/ipj`.

[61] Ping Ji, Zihui Ge, Jim Kurose, and Don Towsley. A Comparison of Hard-state and Soft-state Signaling Protocols. In *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[62] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS Performance and the Effectiveness of Caching. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, San Francisco, USA, November 2001.

[63] Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of ACM/IEEE MobiCom*, August 2000.

[64] Leonard Kleinrock and Farouk Kamoun. Hierarchical Routing for Large Networks: Performance Evaluation and Optimization. *Computer Networks*, 1(3):155–174, January 1977.

[65] Young-Bae Ko and Vaidya Nitin H. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of ACM/IEEE MobiCom*, pages 66–75, October 1998.

[66] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed Internet Routing Convergence. In *Proceedings of ACM SIGCOMM*, pages 175–187, Stockholm, Sweden, August 2000.

[67] Craig Labovitz, G. Robert Malan, and Farnam Jahanian. Internet Routing Instability. In *Proceedings of ACM SIGCOMM*, pages 115–126, Cannes, France, September 1997.

[68] George Lee, Peyman Faratin, Steven Bauer, and John Wroclawski. A User-Guided Cognitive Agent for Network Service Selection in Pervasive Computing Environments. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)*, Orlando, FL, USA, March 2004.

[69] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *Proceedings of ACM/IEEE MobiCom*, August 2000.

[70] G. Malkin. *RIP Version 2*. Internet Engineering Task Force, November 1998. RFC 2453.

[71] Zhuoqing Morley Mao, Ramesh Govindan, George Varghese, and Randy H. Katz. Route Flap Damping Exacerbates Internet Routing Convergence. In *Proceedings of ACM SIGCOMM*, pages 221–233, Pittsburgh, Pennsylvania, USA, August 2002.

[72] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, and Christophe Diot. Characterization of Failures in an IP Backbone. In *Proceedings of INFOCOM 2004*, HongKong, China, March 2004.

[73] Silvio Micali and Ronald L. Rivest. Micropayments Revisited. In Bart Preneel, editor, *Proceedings of the Cryptographer's Track at the RSA Conference*, LNCS 2271, pages 149–163. Springer Verlag CT-RSA, 2002.

[74] P. Mockapetris and K. J. Dunlap. Development of the Domain Name System. In *Proceedings of ACM SIGCOMM*, pages 123–133, Stanford, California, USA, August 1988.

[75] John Moy. *OSPF Version 2*. Internet Engineering Task Force, april edition, 1998. STD, RFC 2328, `http://www.ietf.org/rfc/rfc1583.txt`.

[76] BGP Filtering Policies. `http://www.merit.edu/mail.archives/nanog/2002-04/msg00229.html`, April 2002.

[77] T. Narten and E. Nordmark. *Neighbor Discovery for IP Version 6 (IPv6)*. Internet Engineering Task Force, December 1998. Draft Standard, RFC 2461.

[78] America's Network. FTTH Moves into Mainstream in Japan. http://www.americasnetwork.com/americasnetwork/article/articleDetail.js%p?id=93689, May 2004.

[79] D. Oran. *OSI IS-IS Intra-domain Routing Protocol*. Internet Engineering Task Force, February 1990. Informational, RFC 1142.

[80] C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*. Internet Engineering Task Force, November 1993. RFC 1546.

[81] Radia Perlman. *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, chapter 14. Addison-Wesley, 2000.

[82] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, Newport, Rhode Island, United States, 1997.

[83] J. Postel. *Internet Control Message Protocol*. Internet Engineering Task Force, September 1981. RFC 792.

[84] J. B. Postel. *Internet Protocol*. Internet Engineering Task Force, September 1981. Standard, RFC 791.

[85] Barath Raghavan and Alex C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *Proceedings of ACM SIGCOMM 2004*. Portland, Oregon, USA, September 2004.

[86] Ram Ramanathan and Martha Steenstrup. *Nimrod Functionality and Protocol Specifications, Version 1*. Nimrod Working Group, March 1996. Internet Draft, Expires 30 August 1996, http://ana-3.lcs.mit.edu/~jnc/nimrod/prospec.txt.

[87] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-addressable Network. In *Proceedings of ACM SIGCOMM Conference (SIGCOMM '01)*, pages 161–172, San Diego, California, United States, September 2001.

[88] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, 1995. Draft Standard, RFC 1771.

[89] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*. Internet Engineering Task Force. Proposed Standard, RFC 3031.

[90] Stefan Savage, Tomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, January 1999.

[91] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in computer networks: Reshaping the research agenda. *ACM Computer Communication Review*, 26:19–43, April 1996.

[92] Alex Snoeren and Barath Raghavan. Decoupling Policy from Mechanism in Internet Routing. In *the Proceedings of HotNets-II*, Cambridge, Massachusetts, USA, November 2003.

[93] John Spinelli and Robert Gallager. Event Driven Topology Broadcast without Sequence Numbers. *IEEE Transactions on Communications*, 37(5):468–474, 1989.

[94] Sprint. BGP Route Aggregation and Filtering Policy. `http://www.sprintlink.net/policy/bgp_filters.html`, June 2004.

[95] M. Steenstrup. *An Architecture for Inter-Domain Policy Routing*. Internet Engineering Task Force, June 1993. Proposed Standard, RFC 1478.

[96] Martha Steenstrup. *A Perspective on Nimrod Functionality*. Internet Engineering Task Force, May 1995. Internet Draft, Expires November 1995, `ftp://ftp.bbn.com/pub/nimrod-wg/perspective.ps`.

[97] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM*, Pittsburgh, Pennsylvania, USA, August 2002.

[98] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, California, USA, August 2001.

[99] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. In *Proceedings of IEEE INFOCOM*, New York, New York, USA, June 2002.

[100] Carl A. Sunshine. Source Routing in Computer Networks. *ACM Computer Communication Review*, 7(1):29–33, January 1977.

[101] Mikkel Thorup and Uri Zwick. Compact Routing Schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10, Crete Island, Greece, 2001.

[102] Joe Touch and Steve Hotz. The X-Bone. In *Proceedings of IEEE Global Internet Conference*, Sydney, Australia, November 1998.

[103] Paul F. Tsuchiya. The Landmark Hierarchy: Description and Analysis. Technical Report MTR-87W00152, The MITRE Corporation, June 1987.

[104] Paul F. Tsuchiya. Landmark Routing: Architecture, Algorithms, and Issues. Technical Report MTR-87W00174, The MITRE Corporation, May 1988.

[105] Paul F. Tsuchiya. Efficient and Robust Policy Routing Using Multiple Hierarchical Addresses. In *Proceedings of ACM SIGCOMM 1991*, pages 53–65, Zurich, Switzerland, September 1991.

[106] Kannan Varadhan, Deborah Estrin, Steve Hotz, and Yakov Rekhter. *SDRP Route Construction*. Internet Engineering Task Force, February 1995. Internet Draft, Expires August 27, 1995.

[107] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent Route Oscillations in Inter-Domain Routing. *Computer Networks*, 32(1):1–16, January 2000.

[108] Verio. Bgp Peer Filter Policy. `http://info.us.bb.verio.net/routing.html`, June 2004.

[109] Xiaowei Yang. Nira: A new Internet routing architecture. In *the Proceedings of ACM SIGCOMM FDNA 2003 Workshop*, Karlsruhe, Germany, August 2003.

[110] Dapeng Zhu, Mark Gritter, and David R. Cheriton. Feedback Based Routing. In *First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ USA, September 2002.