Interdomain Multipath Routing

Igor Anatolyevich Ganichev

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-136
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-136.html

December 15, 2011

Interdomain Multipath Routing

by

Igor A Ganichev

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

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Scott Shenker, Chair
Professor Ion Stoica
Professor Ahmet Yildiz

Fall 2011

Interdomain Multipath Routing

Abstract

Interdomain Multipath Routing

by

Igor A Ganichev
Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Scott Shenker, Chair

While astonishingly successful, Internet is still less reliable than the phone system and supports very limited user choice and control. As many researchers observed, multipath routing is a promising paradigm to address these issues. In this thesis, we argue that multipath routing can indeed go a long way towards these goals as well as lead to a more scalable, extensible, and evolvable Internet.

We begin by describing Yet Another Multipath Routing (YAMR) protocol that provably constructs a set of paths resilient to any one interdomain link failure. YAMR uses an efficient scheme to construct the paths and a novel failure hiding technique to further reduce the control plane overhead.

Next, we describe Pathlet Routing, a protocol that departs from the path-vector paradigm. Pathlet routing allows ASes to advertise policy-compliant path segments called pathlets, and allows users to stitch them together, thus forming a complete path suitable for the user's particular needs. Pathlet routing greatly reduces the forwarding table size, can efficiently express a wide class of routing policies, and provide an exponential number of paths to the users. Finally, we investigate how pathlet routing can be a basis for an evolvable Internet architecture.

Professor Scott Shenker
Dissertation Committee Chair

To my wife

Contents

List of Figures

List of Tables

Acknowledgments

During the 5 years of my doctoral studies I have time and again found my advisor Scott Shenker to be the source of solid support, inspiring motivation, acute comments, and immense help. His ability to take raw thoughts, extract fundamental insights, and lay them down in a crystal clear prose has always left me amazed. His eagerness to help in both academic and personal issues, his hard work, and sincere humility, showed me the good of human character.

I would like to thank Prof. Ion Stoica, Prof. Sylvia Ratnasamy, Prof. Ahmet Yildiz for being on my qualifying examination committee. Their valuable and insightful comments helped bring this thesis together.

My fellow lab mates deserve as much credit for this thesis as anybody else. I was privileged to work with truly gifted, knowledgable, and simply amazing Berkeleyans. Bin Dai, Junda Liu, and Brighten Godfrey helped bring the YAMR project to a conclusive end. Working with Brighten Godfrey on the pathlet routing project taught me how to scope and manage the work, as well as bring out the essential ideas with convincing data. Working with Barath Raghavan, Ali Ghodsi, Somaya Arianfar, Dmitriy Kuptsov, and Teemu Koponen on the FII project I could benefit from Barath's imagination and passion, from Teemu's depth and practical insights, and from Ali's breadth of knowledge and sincere opinions. Beyond the work that went into this thesis, I was fortunate to work with many individuals including Kyriakos Zarifis, Andrey Ermolinskiy, Arsalan Tavakoli, Kenes Beketayev, Daekyeong Moon, and many more whose unique personalities never allowed me to live a dull moment all these years.

I am deeply indebted to my family, who has never hesitated a single moment in supporting me in every endeavor, independently of how obnoxious it was. Finally, my beloved wife Nadezhda, is the one person who shared all the joys and difficulties of my graduate studies. Her unwavering love was ever the comfort for my heart and inspiration for my mind.

Chapter 1

Introduction and Related Work

The Internet has changed all of our lives, mostly for the better. As an example of its power to transform in strange and wonderful ways, the Internet enables Tibetan monks to keep track of the latest Silicon Valley news. In this growth sprint, it outgrew the farthest dreams of it original designers and early developers. While all of us are indebted to the original design that did not fall apart under the growth pressures, we must admit that the current Internet architecture faces a number of important challenges. This realization has led many researchers to work on clean-slate redesigns of the Internet. This thesis is a small contribution to this great challenge.

1.1 Architectural Challenges of the Current Internet

Researchers and operators have identified a long list of outstanding architectural challenges with *Reliability*, *Scalability*, and *Poor User Choice* being among the prominent ones ([50], [12], [71], [49]).

Reliability. The Internet is becoming a critical component of our infrastructure, business, and lives, thereby expanding our dependence on it and making it increasingly more painful to go through an outage. Moreover, the possibility of an outage prevents us from developing applications that cannot tolerate a disconnection, an

illustrative example being telesurgery.

Companies realizing this fact develop mechanisms to mitigate the effects of a network disconnection using higher level approaches such as fetching a piece of data from another location that is still connected. Network operators strive to increase the reliability of their networks using mechanisms like MPLS Fast Reroute ([54]). All of these efforts improve the Internet's reliability, but measurement studies continuously find it rather low ([34]). One popularly cited number being 2.5 nines of reliability as compared to 5 nines of the phone system.

Besides the fact that these solutions do not achieve desired reliability levels, they complicate network operations and require each company to reinvent the wheel. These patch solutions raise the barrier for new companies to enter the market and boast comparable reliability guarantees.

Scalability. The Internet's poor scalability has been identified as a major problem by The Internet Architecture Board [50]. The current concern is due to the state and processing requirements scaling linearly with the number of advertised prefixes, which in turn has been growing at an increasing rate [36]. The growth of the forwarding state is problematic because the Forwarding Information Base (FIB) has to be kept in fast and expensive SRAM memory. The growth of the control plane state is an issue because maintaining this state requires a proportional number of control messages and leads to convergence times of several minutes [46, 48].

Poor User Choice. Users have very limited control over the path their data takes across the Internet. In most cases, users can choose their edge ISP, but the fate of their packets beyond that ISP is out of their reach. By user choice, we don't mean a grandma actively choosing the path her email should take. Rather, we mean giving the end-hosts, edge routers, or some entities acting on their behalf some control over the path and its properties.

If users are given more control over the path, the competitive landscape of the Internet Service Providers (ISP) should become healthier allowing much of the "tussle" to be resolved within the protocol [13]. With rigid protocols like Border Gateway Protocol (BGP, [59]) different stakeholders have vested interest to push the design to

their benefit, eroding it with time.

Given a choice over the path, users can optimize the metrics they care about. Currently, everyone gets *an average* path. There is no distinction between a user who can tolerate a satellite latency but would prefer larger bandwidth and a user who is making a voice call, needs only 10kbps, but would appreciate lower latency and loss rate.

Andersen et al [4] showed that, had users even a small choice over their paths, they could recover from all the observed outages. Furthermore, 5% of transfers could double their bandwidth and 5% could decrease their loss rate by 0.05.

1.2 BGP is at the Core of the Problems

A large number of researchers analyzing these problems [44, 64, 42, 46, 48, 50, 31, 35, 17, 45] have shown that BGP plays a significant part in all of them.

Reliability. The evidence for BGP's hand in poor reliability is abundant. Kushman's et al. study [42] of 50,000 VoIP calls found that

- Almost 50% of unintelligible VoIP calls occur within 10 minutes of a BGP update, while only 1% of all VoIP calls are within 10 minutes of a BGP update.
- More than 20% of outages correlated to BGP last for more than 4 minutes, while more than 90% of outages not correlated with BGP last for less than 10 seconds.

Labovitz et al. in [46] showed that a route change causes, on average, 30% packet loss for as long as 2 minutes. Wang et al. in [64] found that a singe routing event can produce hundreds of loss bursts lasting up to 20 seconds. Finally, Gummadi et al. [33] showed that routing through a random node (thus deviating from the BGP path) can recover over 50% of network outages.

Scalability. The scalability concerns of the Internet are direct consequences of BGP's basic design as well as historic events like static address allocations and multihoming. BGP operates at a network prefix level, i.e. (almost) all of its advertisements and

updates are per-prefix. Hence, both the forwarding and the control plane memory requirements scale with the number of prefixes.

Being aware of this fact, designers of BGP included an aggregation mechanism intended to collapse neighboring (in the address space) prefixes into a single prefix. However, with the recent trend of multihoming and ISPs' desire for finer control over their address spaces, the aggregation mechanism becomes less and less effective [36]. In fact, a recent initiative ([19]) aims at removing these reasons by a global address translation at the network edge.

Poor User Choice. Recall that in a typical scenario, an AS learns a path for each prefix from one or more neighboring ASes. It then selects only one path to use and advertise further. Thus, most BGP domains do see multiple paths but they cannot offer these paths to the users. In fact, [51] suggests a mechanism to expose these paths and improve user choice.

1.3 Multipath Routing Paradigm

In the previous section, we have seen that the three issues with the current Internet are intimately related to BGP. The reader, however, should not belittle the design of BGP (which stood the test of time, gave each AS autonomy in its routing decisions, and was the pioneer of policy routing). Instead, he should see that interdomain routing is a rather daunting task with a myriad of stringent requirements.

Luckily, a growing number of researchers [51, 74, 68, 73] have identified and argued for an approach that has a promise to address all the aforementioned problems, namely, the multipath routing. The goal of multipath routing research has been to find a protocol that can build and maintain multiple paths to each destination in an efficient manner and can expose them to the user. Intuitively, this approach is attractive because

- By exposing multiple paths to the users, we enable them to respond to network failures by choosing a working path.

- If multiple paths are available, the protocol might not be required to take immediate action to repair the broken paths. Thus, alleviating the scalability problem.

- If multiple paths are exposed to the user, we have improved user choice.

This thesis describes two interdomain multipath routing protocols called Yet Another Multipath Routing (YAMR) [24] and Pathlet Routing[1] [28]. YAMR stays within the path-vector paradigm of BGP and shows how a policy-based path-vector protocol can be made into an efficient multipath one. Pathlet routing on the other hand is a complete departure from path-vector routing. It is based on new, abstract, and flexible concepts that result in a very efficient, elegant, and extensible protocol. Pathlet routing shows us that the fundamental question of routing still has jewels laden in its design field.

Following the descriptions of YAMR and pathlet routing, we look at how pathlet routing can be a great fit for the interdomain routing protocol for an evolvable Internet architecture[2]. Pathlet routing is a great fit because its building blocks provide a high-enough abstraction level to allow domains to innovate independently. Furthermore, domains can more easily define and roll out new services on top of pathlet routing's building blocks.

1.4 YAMR

Recent multipath routing proposals (e.g. [68, 51]), have made admirable progress and demonstrated that it is possible to provide a set of alternate interdomain paths in a scalable and policy-compliant manner.

The only disquieting aspect of these approaches (and many other multipath proposals in the intradomain case) is that the set of alternate paths is somewhat *ad hoc*;

[1] Pathlet routing was originally developed by Brighten Godfrey and I was privileged to work on the later stages of its design and implementation. In this thesis, I report on the work I contributed as well as the necessary background.

[2] Similarly to pathlet routing, the FII project, [41], was led by Teemu Koponen and Scott Shenker. In this thesis, I report on the work I directly contributed to the project.

they cannot systematically compute a set of alternate paths that has a high degree of path diversity.[3] That is, while they provide a tunable number of alternate paths, these paths may have significant overlap, thereby leaving the possibility that a single failure could take out the entire set.

YAMR, on the other hand, provides high path diversity in a systematic way. There are two components to the YAMR approach.

(1) An efficient BGP-like mechanism for computing a diverse family of policy-compliant paths: This component of YAMR (which we call YAMR Path Construction, or YPC) computes a set of alternate paths that are deviations from BGP's default path.[4] Each alternate path is computed assuming that a link in the default path is down. Considered as static set of paths, there is no single failure that can break all the paths simultaneously, unless that failure disrupts *all* policy-compliant paths between the source and the receiver. When protocol dynamics are taken into account, the story is more complicated (because when BGP recovers from a link failure, it can break paths that did not contain the failed link). We present simulation results on the actual resilience achieved under full dynamics, which show that YAMR improves the reliability of BGP following single link failures by almost three orders of magnitude.

However, computing this family of paths involves higher control plane messaging overhead than BGP. We therefore added another component to YAMR.

(2) A technique for reducing churn by localizing routing updates: Much of the churn created by BGP is due to the fact that every change in a path must be disseminated to all nodes that use that path. YAMR hides some of these updates, and it turns out that this "update hiding" technique not only reduces YAMR's churn, it also increases (by an order of magnitude) YAMR's resilience, by largely avoiding BGP's problem of recovery causing functioning paths to break.

[3]The theory literature has many such algorithms, but they do not lend themselves to scalable, policy-compliant implementation.

[4]This basic idea is borrowed from [21], which computes the cost of the cheapest path that avoids each link on shortest path. The algorithm for path labeling and its convergence properties were developed and analyzed for YAMR for the first time.

1.5 Pathlet Routing

In pathlet routing, each autonomous system (AS) constructs a set of virtual nodes (vnodes) — abstract entities that the AS uses to represent its policies. The AS also constructs and advertises *pathlets*—fragments of paths along which the AS is willing to route. Pathlets are represented as sequences of vnodes. To send a packet, a sender concatenates pathlets into a full end-to-end path.

Pathlet routing has three key features. First, pathlet routing combines simple abstract constructs into a clean protocol that can express a vast number of routing policies and expose a dramatic number of paths to the users. In fact, pathlet routing is a generalization of both BGP and source routing, in terms of the expressible policies (i.e., allowed and prohibited routes). If each pathlet is a full end-to-end route, the scheme is equivalent to BGP. If the pathlets are short, one-hop fragments corresponding to links, then the scheme is equivalent to source routing. Moreover, pathlet routing can emulate the policies of several recent multipath proposals including NIRA [72], MIRO [69], and R-BGP [43], in addition to BGP and source routing. We are not yet aware of any protocol which can emulate pathlet routing, although there are several that pathlet routing cannot emulate [75, 74, 52].

The second feature is that pathlet routing can very efficiently represent *local transit* (LT) policies. We call an AS's policies LT if it is only concerned with how traffic travels through its own network. While BGP is perceived to be the ultimate policy routing protocol, it actually forces each AS to choose one complete path for each destination even if the AS is willing to route along any path as long as it traverses AS's network in an allowed way. We believe that the majority of ASes would be well satisfied with local transit policies because the common export policy of valley-free paths is actually an LT policy. Not only can pathlet routing enable local transit policies, it does so with the following great benefits. ASes who choose to use local transit policies can decrease their FIB sizes up to 4 orders of magnitude. This reduction is due to the FIB size scaling with the number of neighboring ASes as opposed to the number of destination prefixes. ASes who adopt local transit policies will dramatically improve the path diversity experienced by their customers, hence improving reliability and

customer satisfaction. Equally important is that the AS that chose to use LT policies gets all these benefits irrespective of what style of policies its neighbors or any other ASes choose to follow.

The third key feature of pathlet routing is that pathlet routing does not require all ASes for follow any particular style of policies. For example, if pathlet routing is adopted as the interdomain routing protocol, some ASes might quickly recognize the benefits of local transit policies and switch to them. Other ASes might be more conservative and continue using BGP-style policies. Yet other ASes might choose to do something in-between by controlling a few hops of the path after themselves.

1.6 Related Work

In this section we describe existing related proposals and the distinctions of YAMR and pathlet routing.

1.6.1 NIRA

In NIRA, [73], each tier-1 ISP, named collectively as the *core*, is assigned an address prefix. Each core ISP then allocates sub-prefixes to its customer ISPs, which in turn allocate to their customers, and so on. If a tier-2 ISP has three tier-1 providers, it will get three prefixes - one from each provider. Smaller ISPs can get exponentially many prefixes if the network is densely interconnected. Each end-host gets an address from each prefix of its direct provider. Thus, addresses essentially become AS-level path descriptors. In other words, given an address of a host, one knows which ISPs the packet will flow through on its way from the core to the host. This process of recursive hierarchical address allocation is accomplished through a BGP-like Topology Information Propagation Protocol (TIPP).

When an end-host wants to send a packet, it looks up all the addresses of the destination (from a DNS-like system that authors call Name-to-Route Resolution Service, NRRS). Then, it chooses one of its own addresses and one of the destination's addresses and sends the packet. The source address in the packet determines the AS-

level path from the source to the core, and the destination address determines the path from the core to the destination. Thus, NIRA provides multipath functionality by encoding the AS-level path in the addresses.

NIRA is similar to source routing, but it does not have the classical source routing problem of not being able to enforce policies because all possible routes are computed by the network and are policy compliant.

While addresses in NIRA are somewhat similar to pathlets, NIRA is distinct from both YAMR and pathlet routing in that it institutionalizes valley-free routing policy. While NIRA suggests some ways to accommodate exceptions to valley-free routes, these mechanisms incur significant overhead and additional complexity. YAMR's policy routing machinery is very similar to that of BGP and it does not assume valley-free routes to be the predominant policy choice. The abstract constructs of pathlet routing never come close to making any similar assumptions. In fact, many policies that NIRA has to treat as exceptions are handled smoothly in pathlet routing. If, for example, an AS wished to provide transit service between two peers or two providers, this would simply involve connecting the two with pathlets.

1.6.2 MIRO

The main routes in MIRO [68] are constructed using BGP. When AS A wants to have an alternative path to a certain destination D, it asks its immediate neighboring ASes for alternative paths they have (recall that with today's BGP, ASes learn many paths to each destination, but choose to use and propagate only one). Neighbors reply. If A does not like any of the paths it gets, it asks its neighbors to ask their neighbors and so on. Say A discovers a suitable path starting at AS B. A and B then establish a tunnel between themselves identified by a tunnel id.

Customers that want their packets to be forwarded on the new path somehow notify A, which then installs filters in its routing tables to select customers' packets, to encapsulate them in an extra IP header destined to B, and to include the tunnel identifier. When that customer's packet destined to D reaches A, it is selected by the filters, encapsulated, and directed into the tunnel. When B receives an encapsulated

packet, it knows to decapsulate it (and possibly perform other actions like bill A) because it sees the tunnel id. Once the packet is decapsulated, it is forwarded using B's regular route to D.

Similarly to YAMR, MIRO uses BGP paths as the default ones. However, MIRO does not describe any systematic way of choosing which paths should be built. It assumes that some party will request a path and the AS will find and install one using MIRO's mechanisms. If paths are installed on-demand, the path establishment latency can be prohibitive for reliability purposes. If paths are to be preinstalled, some efficient automatic mechanism similar to YAMR must be added to MIRO.

While MIRO is arguably easier to deploy in today's BGP speaking routers than pathlet routing, it has significantly higher path establishment and forwarding state overhead if the number of available paths is to be comparable to that of pathlet routing. In fact, pathlet routing can emulate the set of paths that MIRO can build using its general and clean constructs.

1.6.3 Routing Deflections

Routing Deflections (RD) [74], as well as Path Splicing (PS) [51], concentrate on intra-domain multipath routing and handle inter-domain case as an extension. Currently, forwarding state at each router contains a single outgoing port for each destination. RD replaces this outgoing port with a carefully selected *set* of outgoing ports – a *deflection set*. Each port in the deflection set is equally good to forward packets – no matter what port is selected at each router, the packet will be delivered to the destination without any loops in the path. To ensure this property, authors design a set of rules that determine which ports are to be included in the deflection sets. Exact rules are not very important, but as an illustration, first rule states that if the cost of the path from the next hop router is less than the cost from the router to the destination, the port to the next hop router is included in the deflection set. Each router's deflection sets are computed locally using the information of the router itself and of its neighbors.

Once the deflection sets are in place, RD allows users to select different paths by

allowing them to include an opaque tag in the packets they send. When a router receives a packet with a tag, it maps the tag to a port in the deflection set and sends the packet out on that port. The mapping can be arbitrary but deterministic so that all the packets with the same tag follow the same path.

Besides the fact that RD's inter-domain extension is rather limited, RD does not provide any guarantee about the path diversity it achieves on a general network with general policies. As [74] shows, RD rules for deflection set construction work well on certain intradomain topologies with no policies, but there is little analysis of what happens on interdomain topologies with general routing policies.

YAMR is specifically designed to provide a guarantee of single failure resilience. Pathlet routing, if not atrociously misused by the ASes, exposes most (or even all) of policy-compliant paths.

1.6.4 Path Splicing

Similarly to RD, Path Splicing (PS) [51], builds a set of outgoing ports for each destination and uses tags to allow user path selection. However, the ports in the sets are not equally good and the mapping from tag values to ports is not arbitrary.

Sets of ports are built by running k instances of the regular routing protocol with different link costs. For a given destination, each instance builds a minimal cost tree rooted at the destination. Because link costs are different for each instance, the trees are expected to be different and, hence, each router will know k routes to each destination. The outgoing ports of these routes make up the set and routers remember which port belongs to which tree.

The tag is interpreted as a sequence of trees to be used. In essence, by choosing a certain tag, the user says "Route this packet on this tree for this long, then start routing on that tree for that long, and so on". Unlike routes in RD, routes in Path Splicing can have loops, but because each tree provides a path to the destination, the packet is guaranteed to reach the destination on the last tree specified in the tag.

Similarly to [74], Path Splicing, provides no guarantees on the diversity of the available paths. Its tunable parameter k gives more control over the path diversity

than in RD, but k is not available in the interdomain case. In the interdomain case, the tag in the packet is used to select one of BGP-learned paths. Thus, the paths available through BGP essentially restrict the maximum path diversity.

Another important distinction between PS and YAMR/Pathlet Routing is that policies must be checked and enforced on the data plane. In interdomain Path Splicing, users can put a tag that would make the packet travel on a non-policy-compliant path. The solution PS proposes is for routers to check the policies on the data plane. It is relatively easy to do for valley-free policies (a single extra bit in the packet is enough), but it remains an open question for more general policies. In YAMR and pathlet routing, it is simply impossible for a user to construct a packet that would travel along a non-policy-compliant path.

Finally, both PS and RD use opaque tags to designate paths. While it is sufficient for reliability purposes, many other uses (e.g. avoiding an AS) require some knowledge of the path by the sender. Pathlet routing exposes AS-level information about each path. With an easy extension, pathlet routing can expose arbitrary metadata about each pathlet allowing a user to construct a path that perfectly satisfied her needs.

1.6.5 IP's strict source routing and loose source routing

The existing strict and loose source routing extensions to IP [18] enable senders to control their paths. Pathlet routing borrows this fundamental idea of source routing but applies it to policy-compliant pathlets rather than to the physical infrastructure. The main issue with these extensions is that they have limited respect for policies. For this reason and due to security and privacy concerns, these extensions have been mostly disabled in today's Internet. This fact was our primary motivation for considering interdomain routing policies as the first order requirement in the YAMR and pathlet routing designs.

Another issue with strict and loose source routing is that each hop is specified as an IP address (along with other header fields). This representation can have a high header size overhead in strict source routing and in loose source routing if many hops are specified. The problem gets exacerbated if IPv6 is used. Realizing this issue, we

designed forwarding identifiers in pathlet routing to be much more compact. The header size overhead in YAMR is just 4 bytes.

1.6.6 MPLS

Multiprotocol Label Switching (MPLS) [60] is a mechanism to carry packets along a preconfigured path. Each path is assigned a label. When a packet enters the path, the ingress router encapsulates the packet in a small MPLS header that contains the label. Intermediate routers only look at the label when forwarding MPLS packets. The labels are short and require only exact matching (which is less expensive than longest prefix match of IP addresses). The egress router strips the MPLS label and forwards the packet based on its original header (usually IP).

The tunnels of MPLS are similar to pathlets and the labels are similar to pathlet routing forwarding identifiers. Moreover, MPLS allows label stacking to concatenate path segments into longer MPLS paths similarly to concatenating pathlets into a complete end-to-end path. However, MPLS is not intended to act as a policy-aware interdomain routing protocol.

While it is true that MPLS has been growing in popularity for intradomain ISP routing as well as some limited tunneling between neighboring ISPs, the labels are not globally exposed and there is no abstract notion of vnodes. Without some notion of vnodes (that allows the abstraction from physical routers), ASes cannot define policy-compliant path segments that can be concatenated by end-users.

1.6.7 Platypus

Platypus ([57]) is similar to loose source routing except that it uses network capabilities to ensure policy-compliance. Platypus can achieve high path diversity but its use of network capabilities and cryptography entails considerable configuration challenges and run time overhead.

It is interesting to juxtapose how pathlet routing and Platypus solve the policy enforcement problem of source routing. Platypus uses an external mechanism of network capabilities, while pathlet routing decides to follow a totally different ap-

proach. Pathlet routing essentially recognizes that it is hard to do source routing on the *physical* topology and decides to make up a *virtual* topology (with vnodes as nodes and pathlets as edges) that is *inherently* policy-compliant. In other words, the logic of pathlet routing is "if we cannot solve the problem on the physical topology, let's change the topology." Using a virtual topology instead of a physical one let's pathlet routing get rid of external enforcement mechanisms - if something is not policy-compliant it is simply not present in the virtual topology.

YAMR does not need to solve this problem because it is not a variant of source routing. Like BGP, YAMR constructs a set of paths and makes then available to the users – it does not allow users to make their own paths.

1.6.8 LISP

At a high level LISP ([19]) can be viewed as a giant tunneling system. LISP begins with the current Internet (i.e. running BGP). Each stub network obtains a topologically significant IP address called a routing locator (RLOCs) from each provider. To route a packet, a location-independent end-host identifier (EID) is mapped to a location-dependent RLOC, tunneled across the Internet to the destination's stub network, and then de-encapsulated and delivered. LISP can select tunnel ingress and egress points to produce multiple possible routes, but not an exponentially large number as in pathlet routing or a provable guarantee as in YAMR.

LISP alleviates the scalability problem with today's Internet, but does not change its fundamental characteristics - the control and forwarding table sizes are still proportional to the number of networks making up the translating core of the Internet (i.e. all but the stub networks). Pathlet routing on the other hand, can make the scaling behavior proportional to the number of neighboring ASes.

1.6.9 R-BGP

R-BGP [43] represents a small change to BGP that, under valley-free policies assumption, can guarantee that the connectivity is not disrupted because of a single interdomain link failure. It is a surprisingly slim mechanism that achieves guarantees

similar to YAMR. However, it is not a multipath solution, as it does not expose the failover paths to the users. Only the network exercises these paths in response to a failure. Users cannot use them proactively to improve their path quality.

1.6.10 Discussion

Before we move onto YAMR, we would like to note that both YAMR and pathlet routing teach us unique design lessons that distinguish them from previous proposals. YAMR shows that maintaining more can actually cost less. The logic behind this counterintuitive fact is that when you have more (paths) and one fails, you don't need to immediately repair the failure, you can use the still working ones, thus saving on maintenance costs.

Pathlet routing gives us a novel and quite a beautiful example of how abstraction is able to solve problems. Pathlet routing represents ASes' policies as a graph and the whole Internet becomes one giant policy-representing topology with the property that every path in this graph is policy-compliant by definition.

Chapter 2

Yet Another Multipath Routing

In this chapter we describe the YAMR design in details. As outlined in the introduction, YAMR consists of two basic mechanisms: YAMR Path Construction (YPC) and the Hiding Technique. In Section 2.1 we present the YAMR Path Construction algorithm and, in Section 2.2, we present the hiding technique. Following that we describe our simulation results in Section 2.3. Appendix A contains the formal proofs of the theorems in this chapter.

2.1 YAMR Path Construction

2.1.1 Overview

The core of YAMR is a policy-based multipath construction mechanism that is very similar to the way BGP constructs its paths. We borrow all of BGP's basic mechanisms including path advertisement, import filters, path selection process, and export filters. Each mechanism is slightly modified to produce YPC. First, we introduce some notation.

We describe paths by a series of ASes, such as $[A, B, C, D]$. This path contains three interdomain links – (A, B), (B, C), and (C, D). We use *RIB_IN* to describe the set of routes learned from neighbors and *RIB_LOCAL* to describe the set of selected routes that is actually used for forwarding. For convenience, we use a failure model

where the interdomain links are the units of failure. YAMR can be generalized to cover domain failures (where all links $(*, A)$ and $(A, *)$ fail for some A) but we choose to spare the reader of these details as all the basic mechanisms remain the same but the notation becomes rather cumbersome.

The goal of YPC is to compute a *default* path p_d (that is identical to what BGP would compute) and for each link L in p_d also to compute (if one exists) a policy-compliant *alternate* path p_L that does *not* contain the link L. It turns out that we can only guarantee this under a set of restrictive conditions.

We say that the network is in a *canonical condition* when the network has converged and all ASes follow *next-hop* [22] and *widest-advertisement* [44] policies (these policies include the customer-peer-provider policy [27]), and there are no dispute wheels [32]. Under these conditions, we can show:

Theorem 1 *Assuming the canonical condition, for any destination D, AS A, and interdomain link e, if there is a policy-compliant path from A to D that avoids e, then YPC computes a policy-compliant e-avoiding path to that destination.*

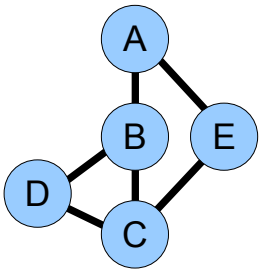
We now describe YPC in more detail, starting with the control plane and then moving to the data plane.

2.1.2 Control Plane

Similar to BGP, ASes in YAMR construct their default and alternate paths from the paths advertised by their neighbors, applying local policies, import and export filters and actions. These multiple paths to the same destination are differentiated using *labels*. Default paths have a special label which we denote by d, while alternate paths are labeled by the link they avoid.

Within this framework, YAMR achieves Theorem 1 by selecting paths as described in Algorithm 1, where

- p_L denotes an L-labeled path
- $best_A(U)$ denotes the A's most preferred path from the set U



This figure presents a simple run of YPC on the topology shown on the left. AS C announces a single prefix and other ASes build their paths to this prefix. In the table below, the first column shows the messages send by the protocol. The other five columns show the state of the routing tables after all the messages in the first column have been processed. Messages are denoted by "src->dst : msg", where msg contains a number of paths. Each path is denoted by "label:AS-path". In this figure, we denote the default path label by (0,0). Messages that don't result in changes to the routing tables are omitted. Also, note that we picked a particular order of the messages. If another order were picked, intermediate routing tables would have been different.

Messages	A	B	C	D	E
C->D: (0,0):[C] C->B: (0,0):[C] C->E: (0,0):[C]	none	(0,0):[B,C]	local path	(0,0):[D,C]	(0,0):[E,C]
D->B: (0,0):[D,C] B->D: (0,0):[B,C]	none	(0,0):[B,C] (B,C):[B,D,C]	local path	(0,0):[D,C] (C,D):[D,B,C]	(0,0):[E,C]
B->A: (0,0):[B,C], (B,C):[B,D,C] E->A: (0,0):[E,C]	(0,0):[A,B,C] (A,B):[A,E,C] (B,C):[A,B,D,C]	(0,0):[B,C] (B,C):[B,D,C]	local path	(0,0):[D,C] (C,D):[D,B,C]	(0,0):[E,C]
A->E: (0,0):[A,B,C], (A,B):[A,E,C], (B,C):[A,B,D,C]	(0,0):[A,B,C] (A,B):[A,E,C] (B,C):[A,B,D,C]	(0,0):[B,C] (B,C):[B,D,C]	local path	(0,0):[D,C] (C,D):[D,B,C]	(C,E):[E,A,B,C] (0,0):[E,C]

Figure 2.1: A complete run of YPC on a simple topology.

- U_L denotes the set of L-labeled paths A knows from its neighbors
- U_d^L denotes the set of default paths A knows that avoid link L.

AS A first selects its default path from the default paths it knows from its neighbors. Because default paths are selected only from default paths, the default paths in YAMR are exactly the same as in BGP. Then, for each interdomain link L on the default path, A selects an L-labeled alternate path from the set of default paths avoiding L (the AS can tell whether a path avoids a given interdomain link simply by looking at the ASes it traverses) and alternate L-labeled paths.

We now walk through a complete run of YPC shown in Figure 2.1. First, C announces its default path $[C]$ to its neighbors, which then construct their default paths. None of the neighbors is able to construct an alternate path yet. Next, B and

Algorithm 1: YPC path selection run by AS A.

```
/* select the default path */
p_d := best_A(U_d)
foreach link L in p_d do
    /* select the L-labeled alternate path */
    p_L := best_A(U_d^L ∪ U_L)
end
```

D send their default paths to each other. Upon processing these messages, each of them is able to construct the alternate path it needs. Next, B and E send to A the updates to their RIB_LOCALs. A can construct its default path either from $[B, C]$ or $[E, C]$. A prefers to have $[A, B, C]$ as its default path and now needs to construct alternate paths avoiding links (A, B) and (B, C). For the (A, B)-avoiding path A has the path $[A, E, C]$ as the only choice because the path $[A, B, C]$ goes through (A, B) and the path $[A, B, D, C]$ cannot be considered because of its label (and would be unsuitable anyway, since it does not avoid (A, B)). Finally, A sends updates to its RIB_LOCAL to E, which is now able to pick its alternate path.

Putting all this together, we can show:

Theorem 2 *If there are no dispute wheels, YPC always converges to a unique final configuration that has no loops.*

While the formal proof of this theorem is available in Appendix A, we would like to provide the intuition behind this fact thereby giving the reader a feel for YAMR's path construction dynamics. First, note that default paths are constructed completely independently from the alternate paths. In other words, while default paths serve as building blocks for alternate paths, the latter never influence default paths in any way. Thus, the convergence properties of YPC's default paths are exactly the same as those of BGP's paths. As the theorem 2 is true for BGP, it is true for YPC's default paths.

To extend this result to alternate paths, we first note that default paths will converge at some point. At this point, they become like a boundary condition (in a

differential equation) for alternate paths. As we noted, these boundary conditions don't change after some point. Next, note that alternate paths for each link are completely independent of each other. In other words, alternate paths for link (A, B) have no way to influence any alternate path for link (C, D) because (A, B)-labeled paths are built only from default paths or other (A, B)-labeled paths. Thus, the dynamics of each set (avoiding a specific link) of alternate paths is completely independent of other alternate paths and of already converged default paths. Looking at one such set in isolation, we see that its dynamics are exactly the same as the dynamics of BGP with the only exception of a more complicated boundary condition. Thus, they will also converge. The reader should now see the reason behind the peculiar Algorithm 1.

We would also like to draw the readers' attention to a curious distinction between YAMR's alternate paths and BGP paths computed when a link fails. Consider an AS A, a destination prefix p, and some link (B, C) on the default path from A to p. Let P_1 be the path that BGP would give A were link (B, C) to fail. Let P_2 be the path that YPC gives A as the (B, C)-avoiding path. It is interesting to note that paths P_1 and P_2 are not necessarily the same. While the dynamic properties of YAMR and BGP are very similar they don't arrive to the same solution. In general, either of P_1 and P_2 can be "better" than the other, so its hard to compare the path quality, but in our experiments they were equivalent in the vast majority of cases.

2.1.3 Data plane

YAMR requires a single addition to the IP header: a 32-bit field for the path label. A packet arriving at AS A destined to D with label L is forwarded along the L-labeled path towards D if A has such a path in its RIB_LOCAL. Otherwise, the packet is forwarded along the default path towards D (without overwriting the label). If A does not have a default path towards D, the packet is dropped. Once the control plane has converged, this algorithm is guaranteed to produce no loops.

For each destination, a YAMR router needs to have a forwarding entry for the default path and for each alternate path whose next-hop is different from the next-hop

of the default path. Thus, the state requirements of YAMR are roughly $1 + k$ times that of BGP, where k is the average interdomain path length. Recent measurements suggest that this is around 3.6 [5] in the Internet.

2.1.4 Discussion

We don't take a firm stance on who should perform the actual label insertion, the end-host or one of its domain's routers. If an end-host does not want to worry about the label insertions, it can place some tag in the packet that will get converted to the appropriate label by the domain's router.

As Theorem 1 shows, YPC is guaranteed to give each AS a policy-compliant path that avoids any given interdomain link (if such a path exists), thus greatly improving reliability. Moreover, users can use all of the paths simply by inserting the appropriate path label into their packets. (The default path lists all the AS links, and so the edge will know which labels will produce different paths; YAMR does not include mechanisms to tell the edge which of these AS links might be providing subpar service.) The paths are constructed and made available to users with moderate increases in the control messaging (or churn) as we will see in Section 2.3, and in RIB and FIB sizes.

BGP scalability is considered a critical challenge ([50]) and YPC makes it worse. Among the many dimensions of scalability, churn appears to be the most intractable. Indeed, the comparison of technology trends and projected growth of RIB and FIB sizes in [3] suggests that technology advances are expected to satisfy RIB and FIB memory requirements at a constant cost. We now present a method that reduces YAMR's churn below that of BGP. This churn reduction method is based on hiding path withdrawals and leaving most of the Internet completely oblivious to the failure.

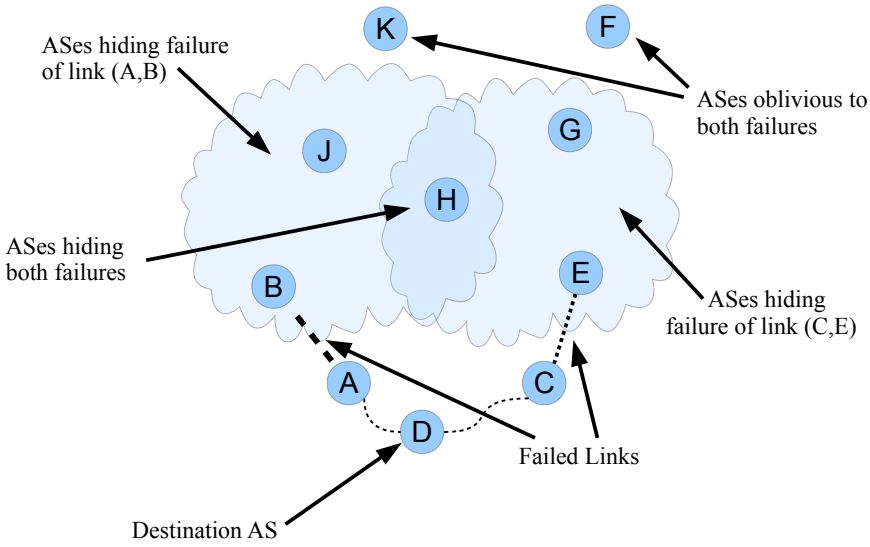


Figure 2.2: An illustration of hiding bubbles.

2.2 Hiding Route Updates

2.2.1 Overview

YAMR's hiding technique is a set of distributed mechanisms that can be applied to YPC, BGP, or other path-vector routing protocols to confine the effects of a link failure to a small neighborhood around the failure. The basic idea is for hiding ASes not to propagate the link failure information to their neighbors if they can safely reroute around the failure. For example, in Figure 2.1, if link (B, C) fails, B can reroute around this failure by deflecting all traffic onto $[B, D, C]$ without telling A that path $[B, C]$ has failed. We call B a *hiding* AS, path $[B, D, C]$ a *deflection* path, and path $[B, C]$ (the failed path being hidden) a *lame* path.

In the example above, B is able to completely hide the failure so that all other ASes remain oblivious to it. However, in general topologies and policies, B might be able to hide the failure only from a subset of its neighbors. The reason being that the deflection path that B selects might be non-exportable – B's export filters would not allow this path to be advertised to the other neighbors. In such a case, B

withdraws the failed path from the neighbors for whom it can't hide the failure. These neighbors then recursively try to hide the withdrawal from their neighbors using the same mechanism. This process continues until a group of ASes around the failure is collectively able to hide it. In the worst case, this group of ASes can encompass the entire Internet if the failure cannot be hidden. In other words, a single failure is hidden by a dynamically determined bubble of hiding ASes (see Figure 2.2).

Figure 2.2 illustrates a case of two simultaneous failures. D is the destination and links (A, B) and (C, E) fail. B attempts to hide the failure of (A, B) but does not fully succeed and the failure news spread to ASes J and H. Similarly, AS E attempts to hide the failure of (E, C) but has to withdraw the path from some neighbors. Ultimately, the set of ASes E, G, and H are able to contain the failure. As the figure illustrates, H is aware of both failures, whereas K and F are not aware of any failure. Thanks to the densely connected nature of the Internet, as we show later, usually just a few ASes can completely contain the failure.

When hiding is combined with YPC to produce the full YAMR protocol, the following results hold:

Theorem 3 *If there are no dispute wheels, YAMR always converges.*

Theorem 4 *In the converged state, YAMR has no forwarding loops or dead ends. Moreover, if ASes follow next-hop policies, all forwarding paths are policy-compliant.*

Theorem 5 *Assuming canonical conditions, for each AS A, if there is a policy-compliant path from A to the destination, A has a policy-compliant path to the destination in YAMR.*

Theorem 6 *When a failed link recovers, all hiding caused by it stops and routing returns to normal.*

As we mentioned earlier, hiding can also be applied to BGP with the theorems above (with obvious modifications) continuing to hold. These theorems are proved in Appendix A. Next, we present the four mechanisms that comprise hiding.

Algorithm 2: YAMR default path selection.

```
/* set the default path to null. If we don't find one below,
/* we won't have a default path and will be disconnected */
p_d := null
/* while we have at least one default path from the neighbors */
while U_d is non-empty do
    /* select the best default path from other default paths */
    p_d := best_A(U_d)
    if p_d is lame then
        /* If the path we picked is lame, try to find a deflection path for it */
        p_d.defl := best_non_lame_A()
        if p_d.defl is null then
            /* If we could not pick a deflection path, delete the default path*/
            /* so we don't pick it again and try one more time */
            delete p_d from RIB_IN
            continue
        end
    end
    /* If default path was not lame or we found a deflection path, */
    /* we are done. */
    break
end
```

2.2.2 Hiding Path Selection

The fundamental mechanism of hiding is to pretend that a withdrawn path is available. Note that a link failure to a neighboring AS can be treated as a path withdrawal even though no actual withdrawal message is received. In our subsequent description of hiding we will only talk about path withdrawals but presume both withdrawals and link failures.

When a path currently in the AS A's RIB_IN is withdrawn from A, A does not delete it from the RIB_IN as it would in BGP. Instead, A marks the path as *lame*

and calls the path selection algorithm. Path selection algorithm (algorithm 1 or the regular BGP path selection if hiding is applied to BGP) is run not caring whether any path is lame or not. If, as the outcome of path selection, A selects a lame path, it tries to choose a deflection path for it from the set of other default and alternate paths. If there is no suitable deflection path, the lame path is deleted from the RIB_IN and no hiding occurs.

YAMR's selection of the default path is presented in Algorithm 2. Alternate path selection follows an analogous algorithm. Here are the definitions of the notation used in the algorithm.

- p_L denotes an L-labeled path, with p_d denoting the default path.
- $best_A(U)$ denotes A's most preferred path from the set U
- U_d denotes the set of default paths A knows from its neighbors
- $best\_non\_lame_A()$ denotes A's most preferred non-lame path from the set of all paths that A knows about.
- $p_L.defl$ denotes the deflection path associated with a lame path p_L

Path selection ensures that once it is done, each lame path in the RIB_LOCAL (i.e. any selected lame path) has a deflection path associated with it. As in BGP, after the RIB_LOCAL has been updated, YAMR announces the changes to its neighbors. Export filters and actions are applied to non-lame paths in exactly the same way as in BGP. However, for lame paths, export filters are applied to the corresponding deflection paths and export actions are applied to the lame path.

This curious distinction is due to the fact that A will be advertising the lame path but forwarding along the deflection path. Since A will be advertising the lame path, A wants to apply the export actions to it before sending it to the neighbor. In the common case, the lame path will be the one that A has selected and advertised before the failure. So, after applying the export actions, the resulting path will be the same as the one A has previously sent and A won't be sending it again. This is the point where failure propagation stops.

A applies export filters to the deflection path because A will be forwarding along it and wants to make sure that the data path is compliant with its policies.

If export filters allow the deflection path to be advertised to a neighbor, the lame path is actually advertised. Otherwise, a withdrawal message is sent to the neighbor.

2.2.3 Hiding Forwarding

Forwarding in YAMR is the same as in YPC except that the forwarding entries for lame paths are built based on the corresponding deflection paths. If the lame path's label is different from the deflection path's label, the labels of packets forwarded along the deflection path are replaced with the deflection path's label.

2.2.4 Tokens Overview

In the two previous sections, we described that YAMR advertises lame paths, but forwards on deflection paths. When there is a single failure, this lie is harmless, but with multiple failures and multiple ASes hiding, lying can cause forwarding loops or leave an AS unnecessarily disconnected. These two problems originate from the following two fundamental issues.

Consider an AS B that advertises a path p to its neighbor A. Because of hiding, the actual forwarding path p' can be different from p. The difference between p and p' leads to problems in two cases:

1. If A is in p' but not in p, A can select p and create a forwarding loop.

2. If A is in p but not in p', A will not accept p and can become unnecessarily disconnected if p is the only path it was offered.

Hiding solves these two problems by introducing a new message type we call a *token*. There are two types of tokens. *Loop detection tokens* solve the first problem, while *disconnection preventing tokens* solve the second. We call the messages *tokens*, because their processing resembles passing a physical token - no AS remembers any state for any token, and when a token is received, it is either dropped or passed on to a single other AS.

At a high level, the goal of both token mechanisms is to cause some AS to stop hiding. Informally, hiding tries to hide as much as possible, but it recognizes that sometimes too much hiding can cause problems. The cure for these problems is to reduce the level of hiding in the network by asking some AS(es) to stop hiding. In the distributed environment of the network, the AS that is hurt by a problem is usually not the AS that is causing the problem. Moreover, the AS that is hurt by a problem might not know that it is hurt just by looking at its local state. In the loop detection mechanism, an AS sends a token when there is a possibility that its packets can be entering a loop. If indeed so, the token is guaranteed to find a responsible AS and make it stop hiding. In the disconnection preventing token mechanism, the AS creates a token when it sees that it is possibly unnecessarily disconnected. The purpose of the token is to find a responsible AS and to ask it to stop hiding. In the next section, we describe both mechanisms in details.

Hiding has a convenient property that allows us to utilize simple mechanisms to fix problems. The property is that any AS can decide to stop hiding at any time without any coordination with other ASes or any impact on the "correctness" (loop freeness and absence of unnecessary disconnection). This property allows us to use simple healing mechanisms rather than some complicated distributed coordination schemes. By healing mechanisms we mean that hiding proceeds without any distributed coordination between ASes. Instead, ASes sometimes send out token messages to heal a possible problem. The healing action is always to ask some AS to stop hiding. If the problem is not fixed when one AS stops hiding, some other AS will send another token.

The down side of this simple approach is that it might be overly-precocious and cause too many ASes to stop hiding (hence unnecessarily increasing churn). A possible alternative would be to use some distributed algorithm to compute the minimal set of ASes that need to stop hiding to fix the problem. While we did not develop such a mechanism, we feel that it will likely cause more control messaging overhead than it saves. From our experiments we saw that tokens do not cause many unnecessary hiding terminations.

2.2.5 Loop Detection Tokens

The loop detection tokens are quite simple. They are created in a single case - when a deflection path for a particular lame path changes. In other words, let AS A have a lame path P and the deflection path P' associated with P. When A goes through the path selection algorithm (that can be initiated because of a new path announcement from a neighbor, for example) it can again select P' or some other path P'' as the deflection path for P. If P' is selected again, no loop detection token is necessary. If another path P'' is selected, a loop detection token is generated. The generated token is forwarded along the newly picked deflection path.

Each loop detection token T contains two pieces of data: a destination prefix p and a list S of (B, N) pairs, where B is a label and N is an AS number. Each AS that forwards T appends to S a pair (B, N) containing the label of the path on which the token is forwarded and its own AS number. Let L denote the label in the last pair of S.

When a token arrives at AS A, A first looks up the path P along which it would forward a packet destined to p with label L. Path P can be either a regular path or a deflection path. Let Q be the label this packet would have when leaving A (recall that the packet's label can change if it is different from the label of the deflection path that the packet is forwarded onto). Then, A checks how many times (Q, A) is present in S (see algorithm 3 for the pseudocode of the following). If twice, A drops T. If none, A forwards T along P. If once, A checks if P is lame. If so, A deletes P from the RIB_IN, thus ceasing to hide it, and drops T. If P is not lame, A forwards T along P.

This token processing can seem somewhat complicated, but there is simple intuition behind it. The loop detection tokens are designed to break stabilized loops (when no AS in the loop is changing its forwarding table). If A sees the token for the first time (case 0 in algorithm 3), there is no indication of a loop and A simply forwards it along. If A see it the second time and its path is not lame, A cannot break the loop and so it forwards the token along as it will reach a hiding AS at some point. If A's path is lame, the token has served its purpose - A stops hiding. If A receives

Algorithm 3: Loop Detection Token Handling.

```
switch number of times (Q, A) is present in S do
    case 0
        A appends (Q, A) to S;
        A forwards T along P;
    case 1
        A checks if P is lame;
        if if P is lame then
            A deletes P from the RIB_IN, thus ceasing to hide it;
            A drops T;
        else
            A appends (Q, A) to S;
            A forwards T along P;
        end
    case 2
        A drops T;
end
```

the token for the third time, it means that the network was actively changing while the token was in-flight. It can safely drop the token because someone in the changing network either already sent another token or the original cause of this token's creation is gone.

Note that loop detection tokens keep track of $(label, AS\_number)$ pairs rather than simply AS_numbers because it is not yet a loop if a packet arrives to the same AS multiple times with different labels. While unusual, this behavior is possible and valid in the presence of hiding.

2.2.6 Disconnection Preventing Tokens

The disconnection token mechanism requires a small change to the path selection process: ASes must not use sender side loop detection and the import filters must

Algorithm 4: Disconnection Preventing Token Handling.

```
if if P is lame then
    A deletes P from the RIB_IN, thus ceasing to hide it;
    A drops T;
else
    if if S contains (Q, A) then
        A drops T;
    else
        A appends (Q, A) to S;
        A forwards T along P;
    end
end
```

not filter out loopy paths allowing them to be inserted in RIB_INs. Sender side loop detection is a BGP mechanism where AS A that uses a path going through its neighbor B does not advertise this path to B because B would not be able to accept it anyway. The second change is to modify the import filters that would normally filter out loopy paths to allow them. Thus, loopy paths are inserted into the RIB_IN alongside the regular paths. The loopy paths in the RIB_IN are never selected into RIB_LOCAL but act as a signal that the AS might be unnecessarily disconnected. Next, we describe the creation and processing of disconnection preventing tokens.

An AS A creates a preventing disconnection token when its path selection cannot pick a default path (because there are no candidates) but its RIB_IN contains at least one loopy path. A sends the created token along the best loopy path (in terms of A's path preference function) and schedules a timer to retransmit the token if the condition persists at the timer expiration. Disconnection preventing tokens require a retry timer because the timing of network events can cause tokens to be lost without fixing the problem. There might be another mechanism that does not require retransmission but we could not find one.

A disconnection preventing token contains the same data as the loop detection token and this data is updated the same way. Only the processing is different. As

before, let token T contain the destination prefix p and a list S of (B, N) pairs, where B is a label and N is an AS number. Also, let L be the label in the last pair of S. As with loop detection tokens, A first looks up P (the path on which a packet destined to p with label L would have been forwarded) and Q (the label that the packet would have had when leaving A). If P is lame, A deletes P from the RIB_IN and drops T - the token has found a problem and healed it. Otherwise, A checks if S contains (Q, A). If so, A drops T. This case means that the network was actively changing while the token was in-flight. There is no reason to send the token anymore. If S does not contain (Q, A), A forwards T along P. See algorithm 4 for pseudocode.

2.2.7 Failed Link Propagation

The main intent of failure hiding is to eliminate churn from transient network events. For example, it would be unfortunate if a permanent link removal would be hidden for a long time. Also YAMR, as described so far, does not guarantee that when a failed link recovers all hiding caused by it stops and the network returns to pre-failure state. One would wish that the protocol itself intrinsically had these properties. However, because the dynamics of policy routing are surprisingly complex, we were unable to find a simple intrinsic modification to YAMR that would achieve these properties. Having had to resort to an additional mechanism, we opted for Failed LInk Propagation (FLIP) protocol similar to Root Cause Information in [44], [38], and Root Cause Notification in [55].

The basic idea of FLIP is to propagate the failed link(s) together with the withdrawal that it (they) caused. Each AS, for each destination prefix, maintains a list $W = [F_1, F_2, \ldots, F_3]$ of failed interdomain links that it learned from path withdrawals for this prefix. Recall that when AS A cannot hide a withdrawal for its neighboring AS B, A sends a withdrawal message to B. With FLIP, this withdrawal message also contains all the links in W.

Alongside each F_i, each AS stores the path through which it learned about this link failure. These paths are advertised together with F_is. If some AS learned about a failure of a link from multiple neighboring ASes, it chooses a single (arbitrary) path

to remember. These paths are needed to reliably remove the link failure records from all ASes when the link recovers.

The reader familiar with BGP should have already noticed that FLIP can essentially be obtained from BGP by removing all policies and replacing IP prefixes with link failures. In BGP, ASes announce a prefix, record the announcement propagation path, and use this path for forwarding. In FLIP, ASes detect failures of neighboring links and include them in withdrawal messages. The propagation paths of the failure information are recorded, but used only to ensure that when the link recovers this information is reliably removed. The "reliably removed" guarantee of FLIP is identical to the guarantee that when a prefix is withdrawn by the originating AS, all records of this prefix will soon be erased from all ASes in the Internet.

The only distinctive feature of FLIP is that link failures are propagated only in withdrawal messages. This crucial distinction ensures that only a few ASes that need to know about the failures actively participate in FLIP. In other words, only the ASes that participate in hiding learn about the failures. This fact keeps FLIP's overhead minimal.

How do we use FLIP? As we mentioned earlier, FLIP is used to make sure that only temporary failures are hidden and all hiding state is cleared when the failure recovers. However, the description of FLIP so far did not tie it to hiding in any way. Their interaction happens through a very simple rule - an AS can hide a withdrawal of a path $[A_1, A_2, \ldots, A_n]$ as long as it knows from FLIP that for some $1 \leq i \leq n-1$ link (A_i, A_{i+1}) has failed. Using this rule and FLIP we can summarize when an AS can hide and when it has to stop.

[**When to Hide**] When a failure occurs, FLIP disseminates the failure to some ASes thus *allowing* them to hide the failure. If another type of change occurs (e.g. link is permanently removed), no failure information is disseminated, thus ensuring that no AS is allowed to hide this change.

[**When to Stop Hiding**] Each AS is free to stop hiding at any time for whatever reason, but it is required to stop hiding a path when FLIP revokes all link failures relevant to the path. When a failed link recovers, FLIP is guaranteed to revoke the

failure information from all the ASes it originally disseminated this information to. Thus, when a failed link recovers, all ASes that were hiding this failure are guaranteed to stop hiding and return the network to the original state.

2.2.8 Discussion

Recalling the high level picture, YPC is able to efficiently construct a set of paths with provable static diversity guarantee, but incurs higher messaging overhead than BGP. To decrease the overhead, we developed a hiding technique that, as we will see in the next section, brings the churn level of YAMR below that of BGP. The surprising result, again to be discussed in the next section, is that hiding also substantially improves *resilience*. Hiding localizes the impact of any routing update and decreases the chance that the convergence process will interfere with any functioning paths.

However, hiding deprives YAMR of some of YPC's valuable properties. First, the set of YAMR paths might not be one-failure resistant because the set of paths might already be hiding failures (so another failure can cause all working paths to fail). Second, YAMR's advertised paths can be different from the forwarding paths. Because ASes cannot be sure about the paths beyond the first hop, they cannot implement policies beyond next-hop policies with 100% confidence. The question is whether the benefits (described in the next section) of YAMR's increased resilience and substantially lower churn (compared with YPC) are worth these two disadvantages.

2.3 Evaluation

Recall that YAMR is composed of the path construction algorithm YPC and a hiding technique. The hiding technique can be applied to BGP, forming what we call HBGP. To understand the contribution of each of these two components to various metrics, we run each experiment for four protocols: BGP, HBGP, YPC, and YAMR.

2.3.1 Methodology

To evaluate the behavior of our protocols, we implemented a message-level event driven simulator. For simplicity, we treated each AS as a single router. Undoubtedly, this simplifying assumption reduces the simulator's faithfulness, but we believe that it captures the interdomain dynamics fairly accurately. Besides the core protocol algorithms, it also includes important features like MRAI timers (with average value of 30 seconds), router processing delay, and message propagation delay. We used annotated the Internet-like topologies generated using [15].

Our basic experiment is the following. Given a topology and a multihomed stub AS, we make the AS announce a prefix, let the network converge, fail one of the provider links from this AS, and let the network reconverge. This basic experiment is repeated for all multihomed stub ASes and all of their provider links. We use a 1000 node topology for most metrics. To study scalability, we use topologies of sizes from 500 to 5000 in increments of 500.

We selected this experiment, which focuses on failures close to the edge, because internal failures are substantially less common and more amenable to recovery, even in BGP. Thus, the edge failures are the most interesting case, and are the dominant case in practice. We also note that this simulation is similar to the live deployment experiment of [65]. The study of a broader class of failure models is left for the future work.

2.3.2 Results

We present and discuss our results for reliability, churn, path stretch, and forwarding table size.

Reliability

We measure reliability by measuring the number of ASes that experience disconnectivity and the time this disconnectivity lasts for. We consider an AS to experience disconnectivity if there is ever a moment when none of the paths in its forwarding table are working.

We detect working paths in our simulator in the following way. Recall that our

simulator is event driven. After each event in the simulator that causes a change to some forwarding table, we send a packet from each AS to the destination prefix using all the labels in the source AS's forwarding table. The simulation is paused while these packets are in-flight, so no forwarding state changes. If a packet reaches the destination we consider the path that it was originally sent on as working.

Table 2.1 shows the average number of ASes that experience disconnectivity during the convergence process. Looking at the percentage of ASes that experience disconnectivity, we see that YPC is about 75 times more resilient than BGP. An alert reader would ask why does any ASes experience disconnectivity in YPC? Wasn't YPC provably one failure resilient. The answer is that YPC is one failure resilient in the static sense (recall the introduction). The unfortunate property of BGP-like path-vector protocols is that convergence process can break even working paths that did not contain the failed link. This property affects YPC as well. Immediately following any single link failure, all ASes indeed have a working path. However, as network starts to reconfigure, a few of these working paths can change creating transient loops and disconnectivity periods.

The remarkable fact is that when hiding is applied to YPC, these dynamic effects of convergence all but disappear - only 0.01 percent of ASes experience disconnectivity in YAMR. That is almost 1000 times better than in BGP. While hiding was originally developed to decrease churn, it actually helps reduce adverse convergence effects by more than an order of magnitude. The reason for this improvement is that hiding dramatically reduces the number of ASes participating in convergence hence lowering the probability that a working path will change and cause disconnectivity.

Comparing the disconnectivity numbers between BGP and HBGP, we see that hiding is much less effective when applied to BGP, giving only 7% improvement. The reason for hiding being less effective in BGP is because BGP has significantly fewer path choices for deflection paths. This causes the hiding bubbles to grow large.

The table also shows the average convergence times for each protocol. The convergence numbers have similar relations and point to the same underlying reasons as we described above. First, hiding reduces YPC's convergence time by almost 40 times, while reducing BGP's convergence time only by 30%. The reason again is the

	BGP	HBGP	YPC	YAMR
Percentage of Disconnected ASes	9.05	8.43	0.12	0.01
Average Convergence Time	23.8	16.7	44.9	1.16

Table 2.1: Average percentage of ASes experiencing transient disconnectivity (top row) and average convergence time in seconds (bottom row) following a single link failure in a 1000 node topology.

scarcity of deflection paths in BGP.

Note that YPC converges almost 90% slower than BGP does. This behavior is expected because convergence in YPC is a two-step process. First, the default paths converge. Once the default paths have converged, the alternate paths can safely do so. These two steps have a significant overlap though - while the default paths far from the failure are still changing, the alternate paths close to the failure might already have converged.

Because the simulation evaluations of other interdomain multipath routing proposals [74, 68, 51] were done with different methodologies, we have not been able to accurately compare YAMR's reliability with them; however, we can give a very rough comparison of YAMR to path splicing [51]. Recall that static reliability means that a routing protocol, at the time of the failure, has an alternate path that avoids the failure. This is an easier quantity to measure than what we studied in our dynamic simulations, since it ignores the fact that when a routing protocol converges it can disrupt functioning alternate paths. Nonetheless, it does provide some measure of reliability. Eyeballing Figure 7 in [51], we see that path splicing is able to improve static BGP reliability by about a factor of 15 with 5 forwarding entries per router per destination (that is, there are 15 times more unnecessary disconnections in BGP than there are in path splicing). YAMR, on the other hand, has no unnecessary disconnections when a single link fails, and even in our dynamic simulations, which allow routing recovery to disrupt alternate paths, it has almost 1000 times fewer unnecessary disconnections than BGP.

R-BGP [44] is another promising approach, achieving both perfect static and

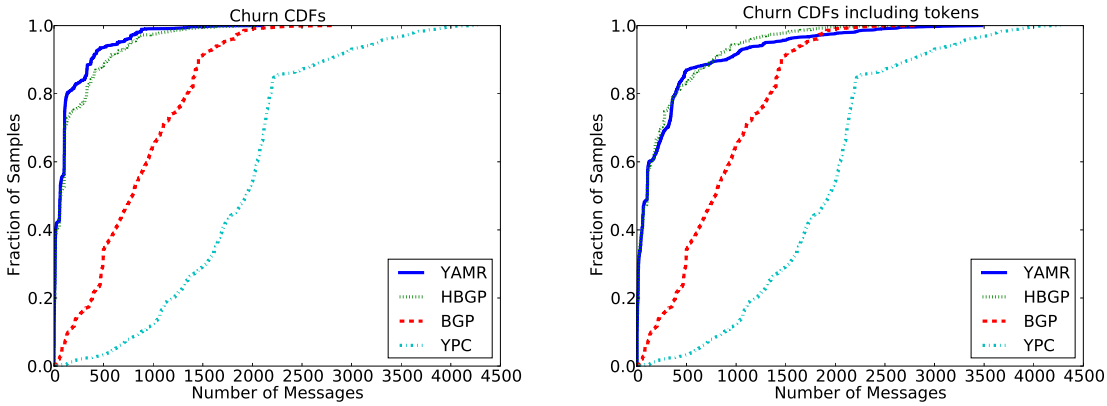


Figure 2.3: CDFs of number of messages following a link event. On the left side, only update messages are included. On the right side, all messages are included. The averages are BGP: 829, YPC: 1828, HBGP: 178 and 249, YAMR: 134 and 286.

dynamic reliability when a single link fails. However, it is not a canonical multipath algorithm because it does not make multiple paths available to the users; it only invokes them upon network-detected failure. It also does not provide these guarantees under arbitrary policies.

Churn

Figure 2.3 shows the CDFs of the number of messages following a link event. Link events include both link failures and link recoveries. We present two graphs with and without tokens because token processing is a much lighter operation than update message processing and because separating them shows how many updates hiding eliminated and how many extra messages it introduced.

In both graphs, YAMR and HBGP significantly outperform YPC and BGP, reinforcing the conclusion that hiding is effective in reducing the messaging overhead. It is interesting to note that if tokens are ignored, YAMR has less control messages than HBGP. This observation again confirms that hiding works best when there are plenty of candidates to pick for deflection paths. When tokens are included, YAMR performs somewhat worse than HBGP, especially in the higher percentiles. The reason is that higher percentiles correspond to cases where the network is sparsely connected around the failure and hiding cannot effectively hide it. In these cases, YAMR tries

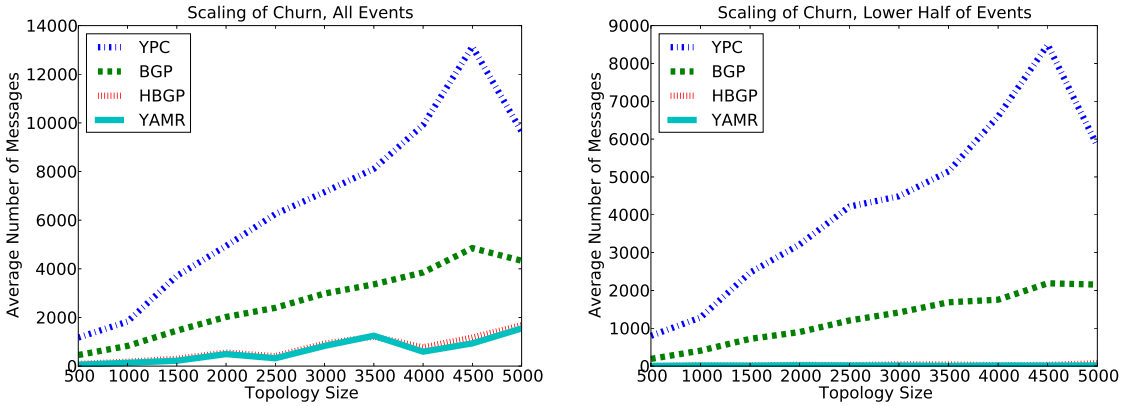


Figure 2.4: Average number of messages following a link event versus topology size. On the left side, all link events are included. On the right side, only half of the events with lowest number of messages are included, separately for each protocol.

to hide much longer than BGP. Each time it tries to hide, it sends a token to check for problems. HBGP gives up faster and hence sends less token messages than YAMR.

If tokens are ignored, YAMR reduces the message overhead by a factor of 6.2 compared to BGP, and by a factor of 2.9 if tokens are counted. This result is quite significant because YAMR maintains about 3 times more paths than BGP. YAMR's performance greatly depends on the connectivity density of the network - the denser the network, the easier it is to hide, the lower the churn. If the recent trend of more interconnections at the edges continues, YAMR's performance will continuously improve with time.

Note that compared to the protocols without hiding (BGP and YPC), the protocols with hiding (HBGP and YAMR) perform relatively better in the lower percentiles than in the higher percentiles. For example, at 20^{th} percentile, YAMR has only 5 messages while BGP has 388 messages. For every failure, BGP requires many messages to converge. YAMR, in contrast, is more bimodal: when the failure occurs in a densely connected portion of the network, it recovers with very few messages, but if connectivity is sparse then recovery is expensive (sometimes more so than BGP).

Scalability

Figure 2.4 shows the number of control messages of the four protocols for different

topology sizes. The most important observation is that all four protocols' churn scales linearly with the size of the network (the left graph in figure 2.4). However, if only the events that cause lower than median number of messages are counted (the right graph in figure 2.4), we see that protocols with hiding have a constant average number of messages. This shows that when the network is densely connected and hiding is able to hide, it does so in a manner independent of the size of the network!

Path Stretch

After each of our basic experiments with a single provider link failure, we measured the average path stretch for all 4 protocols and found that the average path stretch was negligible. For example, the average path stretch of YAMR was only 1.02. The reason is that in the Internet-like topologies, it is almost always possible to find an alternate path that has the same length as the shortest one.

Forwarding Table Size

Let F be the average number of forwarding entries per router per destination. As noted in Section 2.1.3, the pessimal F is $1+k$, where k is the average path length. The pessimal value is reached when all alternate paths are constructed and they are all different. In our 1000 node topology the average path length is 2.86, so the pessimal F is 3.86, but in our simulations $F = 2.21$, 43% less than the pessimal value. If this holds for the Internet, F for the Internet would be roughly 2.62. So, the forwarding table would need to grow by a small multiple of the current table size.

2.4 Conclusion

YAMR starts of with the familiar BGP, fortifies it with a provable resilience against a single interdomain link failure, gives users multiple paths to chose from for each destination, and reduces churn with an automatic failure localization technique.

The YAMR's path construction method is surprisingly efficient and is guaranteed to find a policy-compliant path, if one exists, avoiding any given interdomain link. But the increase in churn, however small, is node desirable given that the Internet's scalability is already a challenge. The failure localization technique cuts the YAMR's churn level well below that of BGP and as a side effect greatly improves convergence

time and dynamic availability. Furthermore, failure localization is applicable to any path-vector protocol and can be of independent interest.

While YAMR made the familiar path-vector into an efficient multipath protocol and taught us that maintaining more can cost less if done right, it feel like a maintenance release to an aging product. We believe that future interdomain routing should be more flexible, extensible, and at a higher abstraction level. The following chapter presents pathlet routing - a flexible, extensible, and surprisingly elegant interdomain routing protocol.

Chapter 3

Pathlet Routing

In this chapter, we start by describing the pathlet routing protocol and a possible design for disseminating pathlets throughout the Internet. We then switch to the expressiveness of pathlet routing, our implementation, and evaluation. Finally, we conclude with a discussion of pathlet routing and its comparison to YAMR.

3.1 The Pathlet Routing Protocol

We start off with an example of pathlet routing with the hope of getting the reader familiar with its abstract concepts. We then introduce the protocol more formally and talk about pathlet dissemination throughout the Internet.

3.1.1 Example

The figure 3.1 illustrates a few important aspects of pathlet routing. Pathlet routing is built on two core concepts: a *vnode* and a *pathlet*. For now, vnode is just an abstract entity that AS creates to represent its policies. In this example, ASes have trivial policies and each of them creates a single vnode. Pathlet are just sequences of vnodes. Each AS can create pathlets that start at its vnodes. By creating a pathlet, an AS announces to the world that it is willing to route along it.

Pathlet routing is concerned about delivering packets from one vnode to another

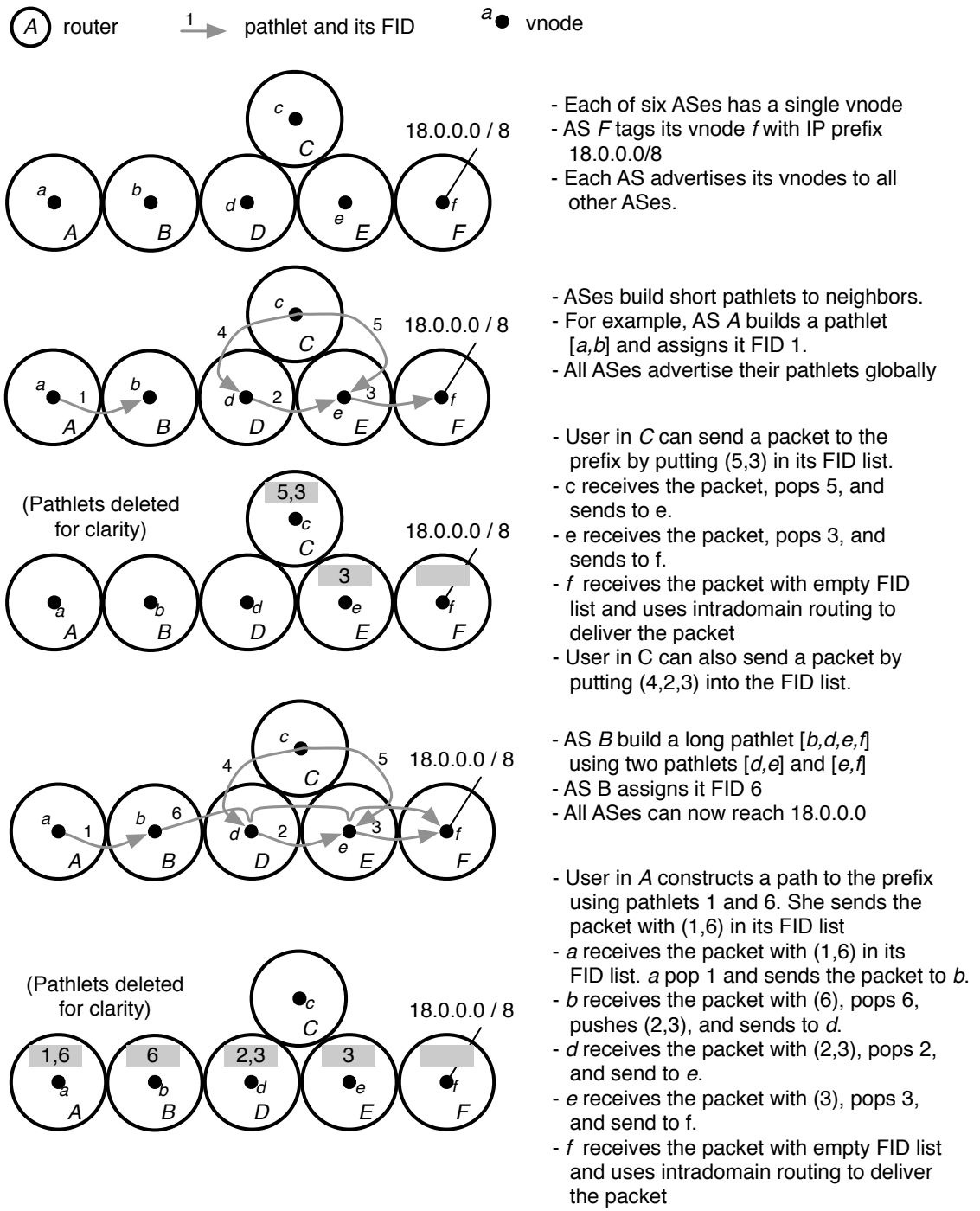


Figure 3.1: An illustration of pathlet routing.

vnode. If pathlet routing is to be used in today's Internet, vnodes need to be tagged with IP prefixes so that the sender knows to which vnode its packets should arrive. Once the vnodes are created and tagged with prefixes (if any), they are globally advertised.

When ASes learn about other vnodes around them, they can start creating pathlets. In the figure, all ASes except B create at least one short pathlet. AS C creates two pathlets. Once the short pathlets are created and advertised, users in AS C can reach 18.0.0.0/8 network by concatenating two pathlets $[c, e]$ and $[e, f]$. Each pathlet has a forwarding identifier (FID) that together with the first vnode uniquely identify it.

To send a packet along the $[c, e, f] = [c, e] + [e, f]$ path, user in C puts two FIDs into its packet header: 5 and 3. When the packet arrives to vnode c (using some intradomain routing system), it checks its forwarding table and find that 5 is the FID of pathlet $[c, e]$. The forwarding entry instructs the vnode to forward the packet to vnode e after stripping off 5 from the FID list in the header. Vnode e acts analogously when the packet arrives to it. When the packet arrives to vnode f it finds the FID list empty and routes on the destination address using its intradomain routing system.

If the physical connection between ASes C and E fails, the user can immediately divert its packets by routing them along the $[c, d, e, f] = [c, d] + [d, e] + [e, f]$ path. All she needs to do is to set the FID list of its packets to $(4, 2, 3)$.

In our example, AS B chooses not to construct short pathlets. Instead, it uses pathlets $[d, e]$ and $[e, f]$ to construct a long pathlet $[b, d, e, f]$. It assigns an FID of 6 to this pathlet and advertises it.

Users in AS A can now reach 18.0.0.0/8. They need to place $(1, 6)$ into the FID list of their packets. Each packet is processed as we described above except when it reaches vnode b. Because pathlet 6 is built from two other pathlets, the forwarding entry for it in vnode b instructs the vnode to not only pop 6 from the packet's FID list, but also push 2 and 3. The bottom left illustration in figure 3.1 shows how FID lists look when the packets arrive at each vnode.

Next, we describe *vnodes* and *pathlets* in more details.

3.1.2 Vnodes and Pathlets

As we mentioned earlier, vnodes are virtual entities that AS creates to represent the routes it wants to allow. The simplest case, which we have seen in example 3.1, is for each AS to have a single vnode. In fact, with one vnode per AS and only one-hop pathlets, pathlet routing can emulate source routing.

A vnode has an identifier of the form $v = (ASN, v)$ where ASN is the AS number and v is the AS's local ID for the vnode. This naming scheme insures that each AS has an independent namespace for its vnodes.

A pathlet P is simply a sequence of vnodes $[v_1, v_2, \ldots, v_n]$. P is identified by a *forwarding identifier* (FID), denoted by P_{fid}, that is an opaque, variable-length sequence of bits. Each AS is free to assign any FIDs to its pathlets without any coordination with other ASes. The only requirement that it needs to satisfy is:

- Packets arriving to vnode v_1 with an FID list that starts with P_{fid} will reach vnode v_n with P_{fid} popped from the FID list.

Pathlet routing does not specify the FID encoding scheme. Thus, each AS can choose a scheme that works best for it given the number of pathlets it has and the popularities of these pathlets. For example, to minimize the average packet header overhead most popular pathlets should be assigned short FIDs.

In our implementation, we use the following encoding scheme for variable-length FIDs. FIDs that begin with 0, 10, 110, 1110, and 1111 bit sequences have total length of 4, 8, 16, 24, and 32 bits, respectively. Following the initial bit sequences, are the actual values identifying the pathlets. We used a special encoding for FIDs of length 4 because most ASes in our simulations had less than $2^3 = 8$ pathlets, so their FIDs would fit into 4 bits.

Vnodes are mapped to routers in a many-to-many fashion. That is one router can host multiple vnodes and one vnode can live on multiple routers. The mapping question is essentially a question about realizing a virtual configuration on a physical network. While this is a very important and non-trivial question, we keep it outside of the scope of this thesis as it is largely independent of pathlet routing and would probably require another thesis.

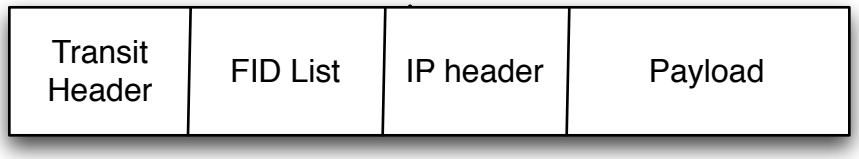


Figure 3.2: A high level view of a pathlet header.

Besides being used as nodes in pathlets, each vnode has an associated forwarding table. The physical instantiation of the forwarding table is part of the logical-to-physical mapping question we mentioned above, but we specify the forwarding table semantics that pathlet routing depends on.

The forwarding table F at vnode v has one entry per pathlet P that starts at v. The P_{fid} is used as the index into F. Thus, pathlets that start at v must have unique FIDs. This is the only requirement for FID uniqueness. So, a single AS can have multiple pathlets that are identified with the same FID, as longs as they all start at different vnodes. This fact can help reduce FID sizes.

The entry at v_1 indexed by P_{fid} and corresponding to pathlet P contains the following information:

- **Next Hop Rule** Some directives on how to reach the next vnode v_2. The exact nature of the directives are dependent on the implementation of the logical-to-physical mapping, but they can include (1) directly submitting the packet to v_2 if v_2 is resident on the same router, (2) sending the packet out of an interface, (3) tunneling the packet using an MPLS label, and so on.

- **Push FIDs** A list of FIDs that must be pushed to the front of the packet's FID list after popping P_{fid}. For one-hop pathlets, this list is empty. for pathlets built from other pathlets, it will contain the FIDs of these other pathlets. In example, 3.1, the forwarding table at vnode b for pathlet 6 has $(2, 3)$ as the push FIDs.

Figure 3.2 shows a high level view of a packet header in pathlet routing with IP as

the intradomain routing system. The first part of the header, we call a *transit header*, represents the headers of technologies that are used to deliver packets between vnodes. It can be an L2 Ethernet header, an MPLS header, or an IP encapsulation header. Following that is the FID list - the main field of pathlet routing in the header. This part of the header contains the list of pathlet ids along which the packet is going to travel. After the FID list is the IP header. Once the packet reaches the last vnode, its FID list is empty and the packet is forwarded using intradomain IP routing.

3.1.3 Pathlet construction

After defining the vnodes and designating some of them as ingress vnodes (we define ingress vnodes in the next section) for its neighbors, an AS announces the ingress vnodes to the corresponding neighbors.

Then, pathlet construction proceeds asynchronously. At any point in time, each AS knows (1) its own vnodes, (2) ingress vnodes exposed by its neighbors, (3) the pathlets it constructed (if any), as well as (4) some set of pathlets constructed by other ASes. Using this information, the AS can construct pathlets of two types:

- **One-hop pathlets** are pathlets consisting of two vnodes $[v_1, v_2]$. v_1 must be a vnode that the AS created itself. v_2 can be either a self-created vnode or a foreign ingress vnode.
- **Compound pathlets** are pathlets of the form $[v_1, P_1, P_2, \ldots, P_n]$ that consist of one self-constructed vnode v_1 and a sequence of pathlets $P_1, P_2, \ldots, P_n$.

For example, in Fig. 3.1, most of the pathlets are one-hop pathlet, while pathlet 6 of AS B is a compound pathlet - $[b, [d, e], [e, f]]$.

3.1.4 Packet forwarding

Recall that each router can have multiple forwarding tables: one for each vnode it hosts. Each forwarding table is an exact-match lookup table keyed by the pathlet FIDs that begin at this vnode. The FID is mapped to the next hop rule and the list of push FIDs.

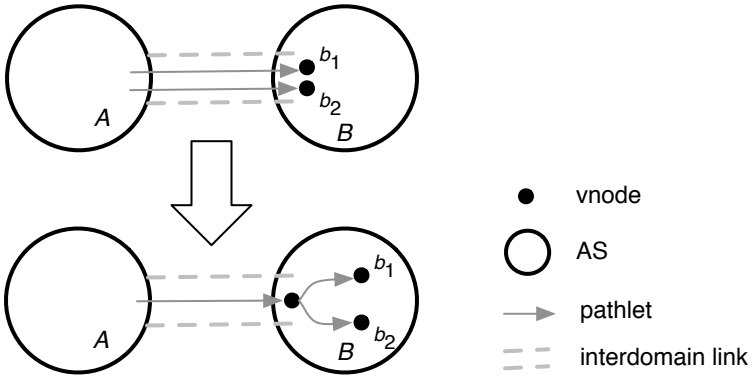


Figure 3.3: A transformation to convert a configuration with multiple ingress vnodes to a configuration with a single ingress vnode.

When a packet arrives at a router, the router first determines which vnode the packet should be given to, if any. An AS is free to chose the mechanism to achieve this[1]. Pathlet routing only specifies how vnode identification is done when packets cross domain boundaries - using the notion of an *ingress vnode*.

AS A designates one vnode to be the **ingress** vnode for each of its interdomain links. The same vnode can be used as ingress vnode for multiple links, but one link cannot have more than one ingress vnode. When a packet arrives to a router on an interdomain link, it is given to the ingress vnode designated for this link. The ingress vnodes are communicated across domains as part of pathlet routing control protocol.

A curious reader might wonder why we limited the number of ingress vnodes to one per link. The reason is that the design with a single ingress vnode is as powerful of the design that uses multiple ingress vnodes. The multi-ingress configuration can be converted to a single-ingress configuration using the trick illustrated in figure 3.3

Having separate ingress vnodes for different neighbors allows the AS to specify policies that depend on the previous hop AS. In section 3.3.1, we will see an example where ingress vnodes represent classes of neighbors: customers, peer, and providers.

Once the router determines the vnode to receive the packet, the packet is sub-

[1]Possible mechanisms to solve the vnode identification question for intradomain case include using the notion of ingress vnodes for intradomain links, mapping MPLS labels to vnodes, explicitly tagging the packets with the next vnode it should be delivered to.

mitted to the vnode's forwarding table. The vnode first checks if the FID list in the packet header is empty. If so, the vnode checks that the destination address in the packet header belongs to one of the prefixes that the vnode is tagged with. If so, the packet is forwarded to the destination address using the AS's intradomain routing. If the destination address does not belong to any prefix attached to the vnode, the packet is dropped.

If the FID list is not empty, the top FID is used to find the forwarding entry in the vnode's forwarding table. If the entry is not found, the packet is dropped. If the entry is found, the top FID is popped from the FID list. The *push FIDs* from the entry are pushed onto the FID list in the header and the packet is forwarded using the *next hop rule* from the entry.

Note that the sender cannot violate AS's policies because it cannot name a path that is not policy-compliant. If he specifies an FID that does not exist, the packet will be dropped because there won't be any forwarding entry for this invalid FID. Any path that the sender constructs from valid FIDs is a policy-compliant path by definition.

3.1.5 Route selection

The pathlet dissemination mechanism to be discussed in section 3.2 provides each AS with a set of pathlets enough to reach all destinations in the Internet. ASes can use any other external mechanism to learn more pathlets if they desire. For example, Google can provide a pathlet database service for others to query.

Given a set of pathlets, a sender or someone acting on its behalf (a gateway router, a DNS-like pathlet server provided by the AS, etc), can construct a path to a destination in the following way. The sender creates a directed graph with all the vnodes it knows as the nodes and all the pathlets as the edges (pathlet $[v_1, \ldots, v_n]$ results in a directed edge $v_1 \rightarrow v_n$). Then, it can run a shortest path algorithm such as Dijkstra to find the shortest path from itself to a vnode tagged with the destination's prefix.

If the pathlets are tagged with pathlet metadata such as the average latency of

the pathlet, or the average available bandwidth, or the average loss rate, the sender can use this information to optimize his utility function.

While the ability to select paths annotated with their properties is immensely valuable, it requires having the pathlet map of the whole Internet. There are several reasons why we don't see it as an important problem. First, the pathlet map of the whole Internet is not much more than what BGP disseminates today as we will show in section 3.5. If all ASes adopt local transit policies (to be introduced later), the Internet map is just a small multiple of the number of ASes in the Internet. Second, the pathlet Internet map can be much more stable than the BGP routes. Recall that pathlets are virtual path segments. An AS should change or withdraw a pathlet only when its network is partitioned or it is changing its policies. These events are more rare than arbitrary path element failures that result in BGP updates. Third, if path computation nevertheless turns out to be more expensive than what some senders are willing to incur, centralized path computation services can be deployed. A DNS-like service can compute a path in response to a user's query or some pathlet proxy can insert FID list into the packets flowing through it.

Pathlet routing does not limit the options of who (and how) computes the path and inserts the FIDs into the packet header. Users can utilize a myriad of previous work on path selection including: learning path properties based on availability or performance observations [75, 7], commercial route selection products [6]; a path monitoring service [72]; third-party path selection services [47].

3.1.6 Pathlet Routing Privacy

In pathlet routing, ASes globally announce their internal vnodes as well as the pathlets they construct. A reader might be worried that ASes will have to sacrifice their privacy when adopting pathlet routing. While we did not carry out a rigorous analysis of pathlet routing's privacy implications, we offer a few arguments that pathlet routing does not expose much more than what BGP does.

First, the map of vnodes and pathlets that the AS announces is not a physical map of its network, so the AS should not be worried about revealing its physical

network topology. For example, as we will see later, valley-free policies are always encoded in the same way, independent of the actual physical topology.

Second, even though BGP does not explicitly reveal AS's policies, it reveals the route decisions made using these policies. Many researchers ([26], [16], [63], [2]) have studied and successfully inferred the ASes' business relationships. In other words, ASes implicitly reveal a good extend of their policies even today.

3.2 Pathlet Dissemination

While pathlet dissemination is an independent aspect of pathlet routing - pathlet routing would work fine if AS operators called each other on the phone and described their pathlets - a base pathlet dissemination protocol should be standardized to ensure basic connectivity. In other words, each AS should be guaranteed to have enough pathlets to reach all destinations. After the basic connectivity is provided, it can be used to implement other pathlet dissemination schemes.

3.2.1 Design Motivation

We propose a particular pathlet dissemination algorithm that we evaluate in section 3.5. Before describing it, let us consider a couple of naive designs.

The simplest possible scheme can be for each AS to propagate all the pathlets it knows to all of its neighbors. This scheme would work fine, except that pathlet routing does not bound the maximal number of pathlets an AS can construct. If many ASes decide to create millions of pathlets, this approach will fail.

The obvious fix is to allow ASes to propagate only a subset of known pathlets to their neighbors. This solves the scalability problem, but introduces another one. Let's say that AS A created a pathlet P and advertised it to AS B, which then advertised it further to AS C. Also, assume that no other AS advertised P to C. Now, consider what happens if A withdraws P and the router in C that advertised P to A fails. A won't hear the withdrawal message and can potentially think that P is a valid pathlet for an arbitrary long time.

While this problem can be solved in several different ways (such as periodically checking the liveness of all the pathlets), we chose a simple proven mechanism.

3.2.2 Dissemination Algorithm

As ironic as it may seem, we chose a path-vector based dissemination protocol. As pathlets are advertised, ASes remember the channel through which they received the advertisement. The channel is just a sequence of ASes leading to the AS that created the pathlet.

Drawing a parallel with BGP, pathlets are analogous to prefixes and the dissemination channel to the ASPATH attribute of the BGP update messages. The major difference being that in BGP, ASPATH attribute is used as the data plane path of the packets, while in pathlet routing, the dissemination channel is needed only to ensure the liveness of the pathlet.

If the channel is live, the AS can be sure that it will receive the pathlet withdrawal message if the source AS were to withdraw the pathlet. If the channel is broken - some element along the channel fails – the AS can be sure to receive an updated channel (if one exists) or a pathlet withdrawal if no other channels are available. Thus, this mechanism is guaranteed to clear all stale pathlets from the entire Internet. The downside is that a working pathlet can be withdrawn from some ASes. This is not a serious problem because, as we will see later, our dissemination protocol guarantees that all ASes will be able to reach all destinations (given that a pathlet path exists).

The main disadvantages of path vector based dissemination are (1) that each AS must remember and advertise not only the pathlet but also the dissemination channel, leading to higher memory and bandwidth costs, and (2) the messaging overhead from changes in the dissemination channel.

The extra memory costs are negligible given the current costs of DRAM memory. The messaging overhead is more problematic, but there is a vast space for possible optimizations. The simplest optimization is to realize that unlike in BGP, ASes never need change the dissemination channel to a more preferred one because all channels are equally good. Furthermore, ASes can use path stability heuristics discussed in [29]

to reduce the number of updates from path failures. Lastly, it is interesting to realize that the channel actually does not have to be a path (it can be a set) and the update conditions can be relaxed. We did not investigate any of these theoretically interesting optimizations because our results in section 3.5 showed that messaging overhead is not much worse than that of BGP.

Our path-vector based dissemination protocol allows each AS to advertise any subset of pathlets it knows to any neighbor. In other words, the AS is free to choose what to advertise much like it is free to choose any export filter in today's BGP. The question then becomes which pathlets to choose.

Suppose v is A's ingress vnode for neighbor B (all traffic from B to A will be treated as arriving v). Then, A advertises the following subset S of all the pathlets it knows.

$$S = \left\{ \begin{array}{c} \text{Pathlets which form a shortest path} \\ \text{tree from } v \text{ to all destination} \\ \text{vnodes reachable from } v \end{array} \right\} \bigcup \left\{ \begin{array}{l} \text{Up to } limit(A) \text{ of pathlets} \\ \text{originating from AS } A \text{ and} \\ \text{reachable from } v, \text{ for all A} \end{array} \right\}$$

where $limit(A)$ is a function that determines the maximum number of pathlets originating from AS A that should be advertised. In our experiments, we used $limit(A) = 10 + number\_of\_neighbors(A)$. The pathlet weights for the purposes of shortest path tree computation are set to the number of vnodes in these pathlets.

The left side of the union is necessary and sufficient to ensure that B can actually reach all the destinations that it can possibly reach. We added the right side to provide a level of redundancy to the minimal tree of the left side.

3.2.3 Discussion

We chose to disseminate the set S from among many possible alternatives for the following reasons. First, because the basis connectivity is a must, we must include some tree spanning all destinations (similar to the left side of the union). We chose to include the shortest path tree because it favors shorter pathlets. Shorter pathlets are better because they result in relatively higher path diversity since they can be mixed and matched more than longer pathlets. Second, advertising only the minimal tree

can cause B to be disconnected from some destinations after a single pathlet failure. To avoid this problem we chose to advertise a few more pathlets per originating AS. We advertise more pathlets from ASes that have more neighbors hoping that these extra pathlets can significantly increase the path diversity.

There is certainly some algorithm that can maximize the achieved path diversity for the given number of extra pathlets. However, we found that this simple linear algorithm performs well enough in our experiments. For example, for most of the ASes using local transit policies, it advertised all of their pathlets providing very high path diversity.

3.3 Policy Implementation in Pathlet Routing

In this section, we begin by introducing local transit policies. Then, we describe how different styles of policies can be implemented in pathlet routing and how they can coexist at the same time.

3.3.1 Local Transit Policies

Pathlet routing enables a natural class of policies that we call Local Transit (LT) policies. We say that an AS A follows local transit policies if A's willingness to carry traffic along a path P depends only on the point P enters A and the point P leaves A.

Local transit policies should come across as one of the most natural classes of policies. Intuitively, local transit policies allow ASes to control exactly what they care most about - the way traffic flows through their networks.

Today, most ASes enter into bilateral agreements only with ASes they directly interconnect with. If an AS does not have any route influencing agreements with remote ASes it is essentially following local transit policies. We say "essentially" because the export filter of these ASes probably looks only at ingress and egress points to decide if the path is allowed - this is the definition of local transit. However, BGP forcing each AS to choose and advertise only a single path makes the other

paths effectively disallowed.

The common "valley-free" export filter in BGP ([25]) is a special case of local transit policies. To implement valley-free export policy, ASes label each of their neighbors as a customer, a peer, or a provider. Then, a path through neighbor A can be advertised to a neighbor B if and only if at least one of A and B is a customer.

As natural as local transit policies are, we are not aware of any multipath routing proposal that makes all policy-compliant paths under local transit policies usable in the data plane. NIRA ([72]) enables the valley-free special case of local transit policies.

Advantages. Local transit policies have a number of benefits compared to BGP. First, since LT policies result in short pathlets. These pathlets can be combined into an exponential number of different paths. A large number of usable paths leads to higher user utilities, better ISP services market, and the possibility of deploying critical applications over the Internet.

Second benefit is that LT policies require very little forwarding state. Forwarding state is proportional to the number of pathlets that the AS constructs (and is independent from the pathlets other ASes construct). As we will see shortly, local transit policies can be implemented with a very small number of pathlets, hence reducing the forwarding state requirement to just a few entries. Having few required forwarding entries in routers will lower the pressure to upgrade the routers as the Internet grows. ASes can also fill the remaining entries with forwarding state to implement other value-added services.

The third benefit is a bit speculative. Since vnodes and pathlets are a form of abstraction (or virtualization), they create a vast innovation space for the AS in actually implementing the pathlet routing guarantees. Just like the layers below IP have seen great innovation, the algorithms to implement pathlet routing on the physical network are likely to break new grounds. This hope is particularly true for local transit policies because with short local pathlets, the AS has complete control over how the pathlets are implemented.

The fourth benefit is related to the previous point. Short LT pathlets are likely to

be withdrawn less frequently than BGP paths. A local pathlet needs to be withdrawn only when the network between its ingress and the egress points is partitioned. This should happen less frequently than a failure of some physical element along the complete BGP path. Thus, LT pathlets should be more stable resulting in fewer control messages as well as new functional abilities like pathlet performance monitoring, per pathlet billing, etc.

Disadvantages. The fundamental disadvantage of local transit policies is that ASes cannot implement policies based on the destination address. A particular example of why this can be a disadvantage is the following. Consider an AS A, a destination prefix d, and three neighboring ASes: B and C customers of A, and D a provider of A. If it is possible to reach d from A through either B or D, with local transit policies A cannot enforce that C always sends packets destined to d along $[C, A, B, \dots]$ and never along $[C, A, D, \dots]$. In other words, AS's customers will be able to use any path through the AS, not necessarily the cheapest one.

A related disadvantage is that an AS following LT policies might find it harder to perform traffic engineering. Some external event (e.g. some monitoring service rates the AS on top) can suddenly draw crowds of senders to switch to using this AS for their transit. Comparing to today, since users have next to no choice over their paths, the amount of traffic traversing a given link is considerably stable and predictable. This allows the ASes to precompute link weights to achieve a relatively smooth distribution of traffic on their networks. With local transit policies a more dynamic mechanism might be necessary.

Another disadvantage is that headers in packets traversing LT domain are likely to be higher than for other styles of policies. For example, a packet traversing n valley-free LT domains will have $2n$ FIDs in its header. This can be seen as a disadvantage, but our experiments show that the FID list in the worst case is expected to be less than a single IPv6 address (see section 3.5).

One particular aspect of routing that ASes are giving up when switching to LT policies from BGP is some control over the quality of the whole path to the destination. With BGP, it is possible for ASes to pick shorter paths as well as paths

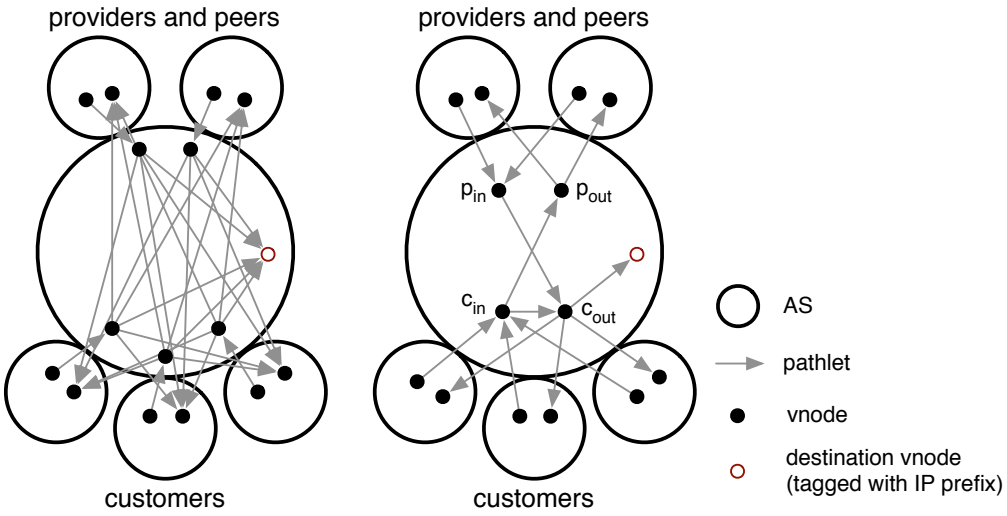


Figure 3.4: Illustration of two ways to implement local transit policies: the naive way (left), and the class-based way (right). The figures depict the valley-free special case of LT policies.

which they believe to have higher bandwidth or smaller loss rate. ASes might want to do this to provide better experience for their customers. With LT policies, this responsibility falls into the hands of the users.

After discussing the cons and pros of local transit policies, we would like to underline the fact that whether ASes choose to use local transit policies or to stay with the BGP-style policies they can do so within pathlet routing and independently of other ASes (more on this in the following sections).

Implementation.

A naive implementation of LT policies in pathlet routing can be readily read off from the definition of LT policies. First, recall that AS A exposes an ingress vnode denoted by $I_A(B)$ to each neighbor B. To implement an LT policy, AS A simply needs to construct a pathlet from vnode $I_A(X)$ to vnode $I_Y(A)$ if and only if A is willing to carry traffic from neighbor X to neighbor Y.

This construction is very simple, but it results in the number of pathlets growing as a square of the number of neighboring ASes. This number can be prohibitively large for well-interconnected ISPs that have thousands of neighbors.

Luckily, we can use the abstraction power of vnodes to implement local transit policies more compactly. If a number of neighbors have similar business relations with the AS, they can be grouped together into a class. All neighbors in a class are treated the same for the routing purposes. The AS then creates two vnodes for each class k: vnode k_{in} that represents an ingress from a member of this class, and a vnode k_{out} that represents an egress to a member of this class.

The k_{in} vnode is exposed as the ingress vnode to all the neighbors in class k. Then, the AS constructs pathlets from the k_{out} vnode to all the ingress vnodes of the neighbors in class k. These two steps "put" the neighbors into their classes. Next, the AS just needs to appropriately connect the classes. For each pair of classes k and m (k can equal m), the AS constructs a pathlet from k_{in} to m_{out} if it is willing to route from members of k to members of m.

The class-based implementation has a number of pathlets proportional to the square of the number of classes and is linear in the number of neighbors. In the case of valley-free LT policies, the number of pathlets is just $4 + number\_of\_neighbors$.

Figure 3.4 depicts these two ways of implementing LT policies on a concrete example of valley-free policies. Note that peers and providers can actually be put into one class and are effectively equivalent for ASes that follow LT policies. They were distinguished in BGP because ASes could prefer a peer path to a provider path. With LT policies, ASes can only decide what is *allowed* not what is *preferred*. The latter is in the hands of the users.

3.3.2 BGP-Style Policies

In this section, we describe how BGP-style policies can be implemented in pathlet routing. We start by describing how BGP in its purest form can be emulated by pathlet routing. Then, we describe how to implement a better version of BGP where export filters are enforced at the data plane (as opposed to relying on neighbors not to know about a path, as is done in BGP today). Finally, we describe how to utilize the idea of classes in the implementation of BGP for the case of valley-free policies.

Before we begin describing the implementation of BGP variants, we would like to

draw readers' attention to the BGP path selection process in pathlet routing. Recall that in ordinary BGP, an AS gets at most one path announcement from each of its neighbors. Then, the BGP path selection process chooses the best one from these paths. In pathlet routing, neighboring ASes can be following LT or other kinds of policies. Thus, the AS will likely receive a bag of pathlets from each of its neighbors rather than a single path.

The goal of path selection process then is to find the best path given all the pathlets the AS learned from its neighbors. In this context, it can be impossible to enumerate all possible paths (as there can be exponentially many of them) and find the best one by comparing them. A solution would be to construct a graph from all the known pathlets, assign a weight to each edge, and run a shortest path algorithm on the resulting graph. For example, small weights can be assigned to customer pathlets, larger ones to peers, and largest ones to providers. If an AS does not want to route through some other AS, it can assign an infinite weight to its pathlets.

BGP as-is. In today's BGP, if an export filter at AS A blocks a path from being advertised to a neighbor B, B can still use this path if it nevertheless sends a packet to A. In other words, export filter is enforced simply by not telling about the path's existence.

We can implement the same rather weak enforcement strategy in pathlets by having each AS create a single vnode (as in figure 3.1). The pathlets are built using the path selection process described above. All pathlets start at the only vnode and extend all the way to destination vnodes. Once a pathlet is constructed, the AS runs its export filter and advertises this pathlet only to neighbors allowed by the export filter. Figure 3.5 illustrates how a domain emulating the as-is BGP constructs a pathlet using LT pathlets of other domains.

BGP with enforced export policy. Using pathlet routing we can enforce the BGP export policy in the data plane so that it cannot be violated. Figure 3.6 illustrates the mechanism and we will describe it by walking through the figure.

Consider AS A with three neighboring ASes B, C, and D. A exposes ingress vnodes $I_A(B)$, $I_A(C)$, and $I_A(D)$ to these neighbors respectively. For each destination

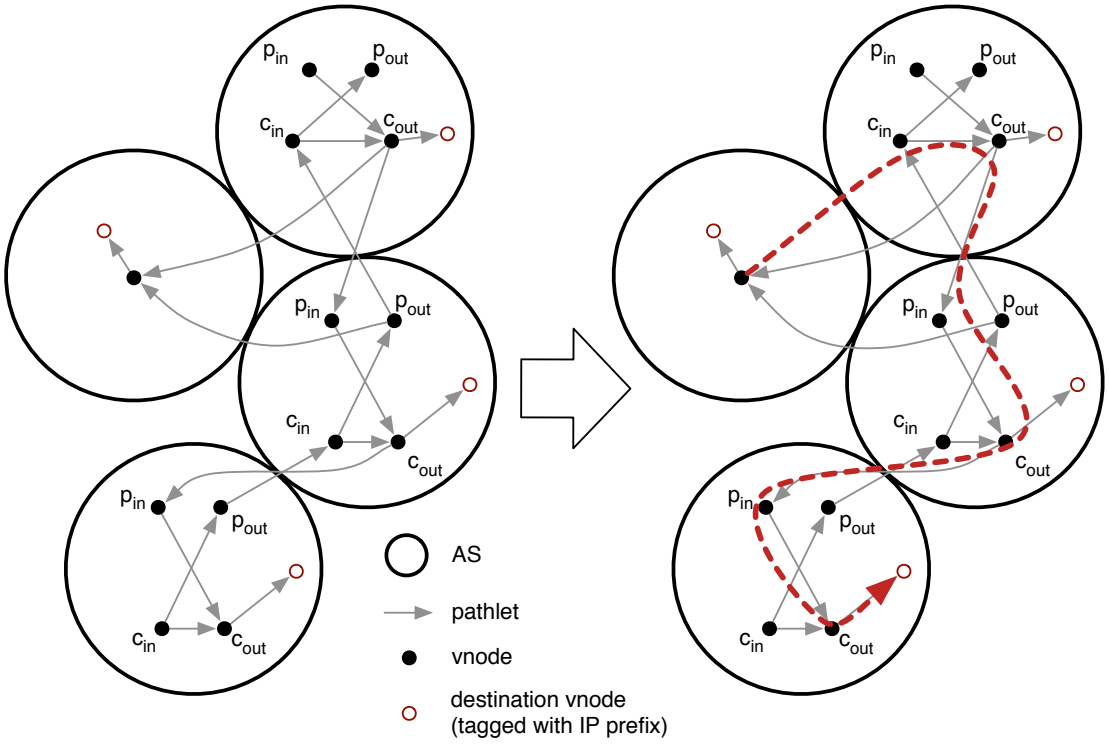


Figure 3.5: A BGP-style domain with a single vnode building a long pathlet across three LT domains.

d, A creates a vnode v_d. In the figure, A created v_{d1} and v_{d2} for destinations d_1 and d_2. Once these vnodes are created, A chooses the paths from the corresponding vnodes to the destinations. In the figure, these paths are depicted with a dashed red line. Finally, export filters are implemented by creating pathlets from ingress vnodes to the v_d vnodes. In the figure, there are no pathlets from ingress vnode exposed to B (i.e. $I_A(B)$) to the v_d vnodes. Thus, B cannot use any of these paths. C on the other hand can use both paths, and D can use on the path to d_2.

This scheme is general enough to implement any policy that BGP can implement, but it can require $O(\delta N)$ pathlets, where δ is the number of neighbors and N is the number of destinations.

Using vnodes to represent classes (similarly to the transformation shown in figure 3.4), we can group ingress vnodes and reduce the worst-case number of pathlets to

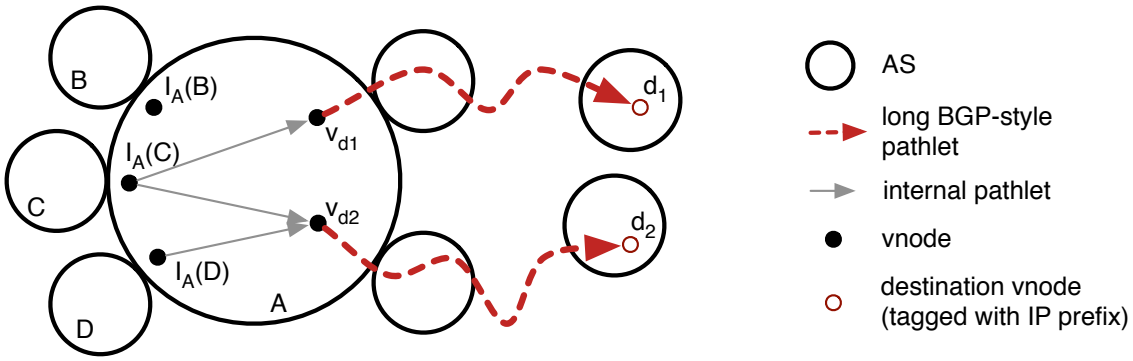


Figure 3.6: An illustration of a general mechanism to implement BGP-style policies with export policy enforcement.

$O(NK + \delta)$, where K is the number of classes.

In the special case of valley-free export policies, the representation can be even shorter. Since there are only two classes (customers and peers/providers) the v_d vnodes are not necessary - paths can start either from p_{out} or from c_{out}. Figure 3.7 shows an illustration of using classes to represent valley-free export policy at a BGP-style AS. The path that starts at c_{out} is usable by all ASes since c_{out} can be reached by all classes (customers as well as peers/providers). On the other hand, the path that starts at p_{out} is reachable only by customers so all customers and no one else can use this path.

3.4 Policy Expressiveness

In the previous section, we have seen several concrete policies implemented in pathlet routing. Moreover, pathlet routing is general enough to support different ASes adopting different policies simultaneously. The reader might wonder about the extent and the limits of pathlet routing expressiveness as well as what is the intuition behind its expressive power. In this section we briefly give the intuition, describe the main limitation, and list a few results for policy expressiveness.

The first part of the main intuitive reason to expressive power is that pathlet routing is built on top of purely abstract concepts - vnodes and pathlets. Pathlet

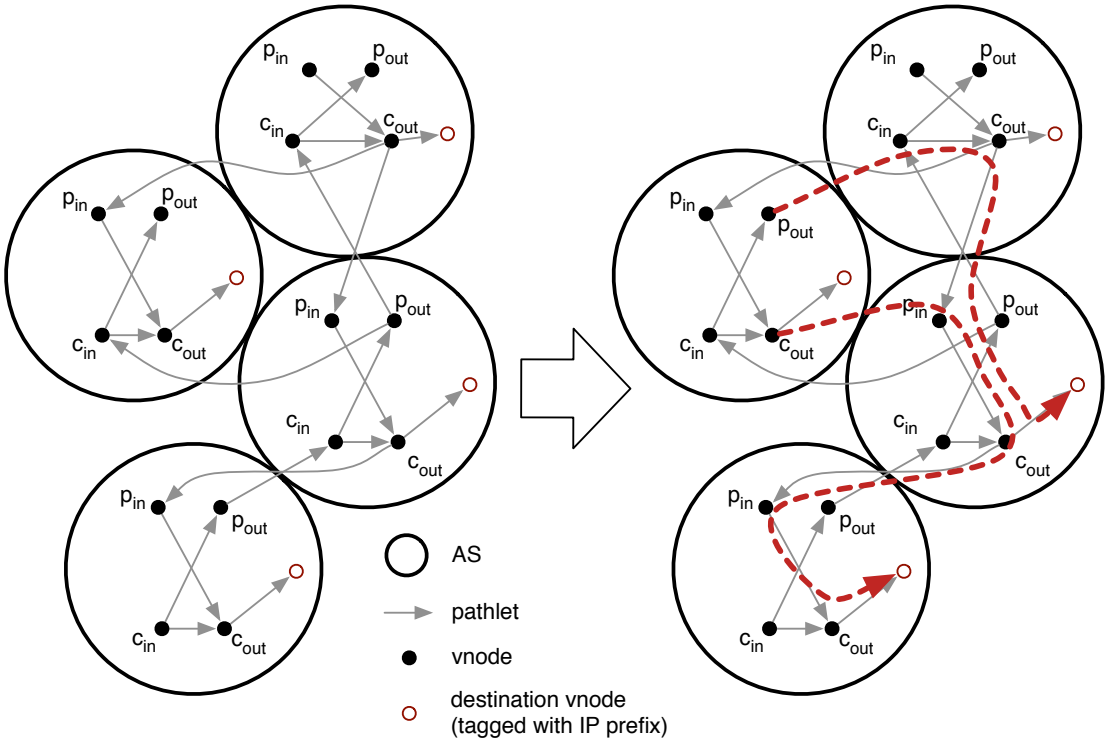


Figure 3.7: A BGP-style domain implementing valley-free export policy using vnodes that represent classes.

routing is completely oblivious to the meanings that are attached to vnodes. This allows us to create vnodes that represent anything from a neighbor ingress point, an egress point for all the neighbors in a given class, to a remote destination (vnodes v_d in figure 3.6).

The second part is that we are free to build a pathlet connecting any two vnodes. In other words, we can "connect" any policy-relevant meanings. Overall, pathlet routing gives us a freedom to build any directed graph - which is a very versatile tool.

The main limitation of pathlet routing is that it does not support policies that depend on the route a packet took before reaching the AS. Once a packet arrives at a vnode, it can take any pathlet starting at this vnode, independent of its previous hops. One can think of extensions to pathlet routing that can add support for such policies, but the core of pathlet routing does not support them cleanly.

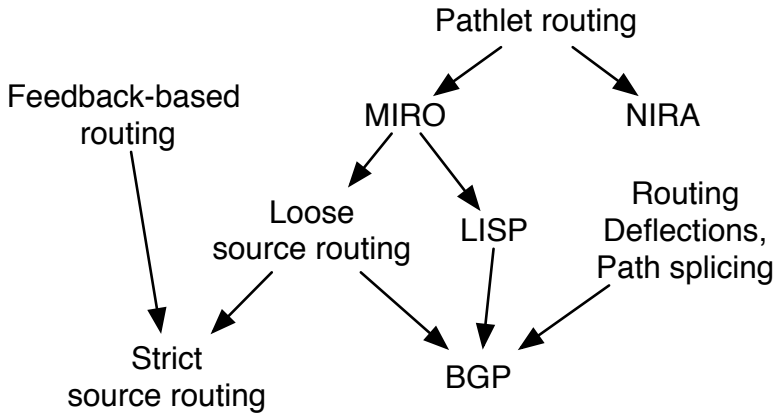


Figure 3.8: Policy expressiveness of different routing protocols. $P \rightarrow Q$ means that P can emulate the routing policies of Q. Furthermore, the $\rightarrow$ relation is transitive.

Paper [28] proposes a formal definition of policy emulation to be able to compare policy expressiveness of different routing protocols. Figure 3.8 depicts the main findings of the analysis based on the policy emulation framework. As the figure shows, pathlet routing is able to emulate the policies of a wide range of previous proposals, while no proposal can emulate pathlet routing to the best of our knowledge. We refer the readers to [28] for details.

3.5 Experimental evaluation

We implemented pathlet routing in a software router and evaluated it in a cluster of server machines. We describe our implementation in section 3.5.1, the evaluation scenarios in section 3.5.2, and the results in section 3.5.3.

3.5.1 Implementation

We implemented pathlet routing as a user-space software router, whose components are depicted in figure 3.9. Implementation is 8000 semicolon lines of code and uses standard asynchronous I/O, graph theory, and networking libraries. Each software router represents a whole AS. While this simplification does not allow us to test

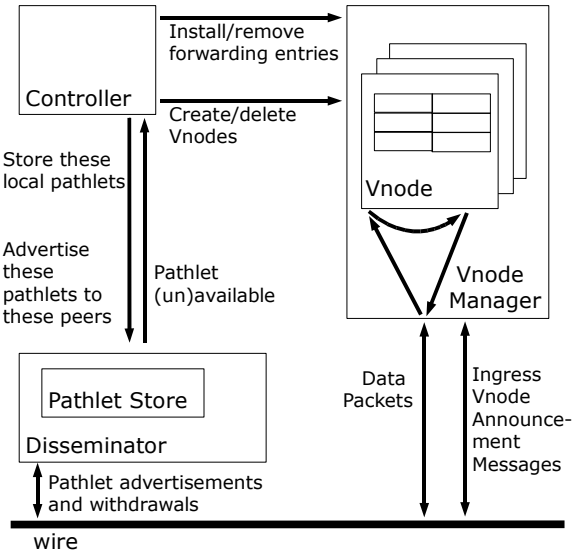


Figure 3.9: Structure of the software pathlet router.

intradomain performance (and scenarios like domain partitioning), it is sufficient to understand the interdomain performance characteristics.

Each router runs as a separate process and connects to its neighbors using TCP connections. For simplicity, the TCP connection is used for both data and control traffic. Using TCP connections to emulate interdomain links, allows us to easily simulate link failures by dropping the connection.

Our router contains three main modules: a vnode manager, a disseminator, and a controller. Through the implementation we found that it is possible to shield the core policy module (the controller) from the details of pathlet dissemination and vnode management. This finding reinforced our understanding that pathlet dissemination and data plane mechanisms – are independent of the interdomain routing policies. Furthermore, this independence will be handy in practice because ASes will likely want to tune their policies without touching other parts of the code.

Next, we describe the three modules.

The **vnode manager** is responsible for managing vnodes together with their forwarding tables. The controller calls this module to create or delete vnodes as well as

to designate a vnode as the ingress vnode for a neighbor. When a vnode is designated as an ingress vnode, this module announces the designation to the corresponding neighbor. On the other hand, when it receives an ingress vnode advertisement from a neighbor, it notifies the controller.

Vnode manager is the module through which data packets flow. It installs entries into vnodes' forwarding tables upon pathlet creation by the controller module. At the data plane level, it is responsible for directing incoming data packets to the right vnodes, performing the forwarding table lookups, executing the instructions in the found entries, and sending the packets out on the right links.

The **disseminator** stores the pathlets and sends and receives pathlet announcements and withdrawals. When a new pathlet becomes available or a pathlet is withdrawn, it notifies the controller. When the controller decides to advertise or withdraw a pathlet from a particular neighbor, it calls the disseminator to execute the decision.

The disseminator is responsible for keeping track of the dissemination channels, updating them when current ones fail, and communicating the changes to the neighbors. Because our dissemination protocol is based on path vectors, we used BGP tricks including the "Minimum Route Advertisement Interval" (MRAI) timer to decrease the convergence time and the dissemination churn. All of the dissemination details are hidden from the controller. So, another dissemination mechanism can be substituted without any changes to the controller.

The **controller** module encapsulates the policy logic. The vnode manager and the disseminator are general and oblivious to the AS's policies. They only implement the policy decisions of the controller. The controller implements the policy by constructing and deleting pathlets and vnodes and by deciding which pathlets to announce to which neighbors. In our implementation, one makes a router an LT router or a BGP-style router by picking the corresponding controller.

In a production quality implementation, the controller should provide a high-level policy specification language that would allow operators to specify the policies declaratively. The policies would then to translated by the controller to what vnodes and pathlets should be created. There are examples of policy languages in research [11] as well as in IETF [9], but we suspect that a compiler into pathlet primitives

would need to be significantly more complex and leave it out of this thesis.

3.5.2 Evaluation scenarios

[**Policies**] We ran experiments with three different mixes of policies adopted by ASes.

1. **LT policies**, where all ASes adopt valley-free local transit policies.
2. **Path Vector-like (PV) policies**, where all ASes use BGP-style policies (i.e. create long pathlets) with valley-free export filters.
3. **Mixed policies**, where a random half of ASes uses LT policies and the rest uses PV.

The LT policies that we experiment with are the valley-free policies shown in figure 3.4. The PV policies mimic the common BGP decision process of preferring routes through customers as a first choice, through peers as a second choice, and providers last. We then break ties based on path length and router ID.

[**Topologies**] We used two types of topologies in our experiments.

1. **Internet-like** topologies annotated with customer-peer-provider relationships.
2. **Random graphs** generated using the $G(n, m)$ model (random connected graphs with n nodes and m random edges).

The Internet-like topologies were generated using the algorithm of [14]. Most of our experiment on the Internet-like graphs were performed on graphs with 400 nodes. To give the reader a feeling for how our Internet-like graphs look, here are some statistics about one of them. It has 400 nodes and 748 edges. Seven of the nodes are core nodes with degrees 133, 125, 87, 52, 47, 35, and 32. There are 339 stub ASes. 155 of which are single-homed, 151 are dual-homed, and 33 have more than two providers. The remaining 54 ASes are in-between the stubs and the core and have both providers and customers.

The random graphs we generated have the same number of nodes and edges as the Internet-like graphs. The random graphs have no business relationship annotations

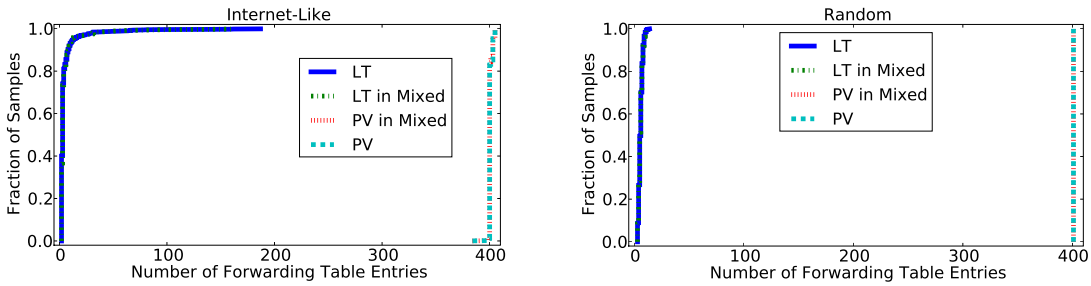


Figure 3.10: Forwarding table (FIB) size for the Internet-like topology (left) and the random graph (right).

on the edges. ASes simply prefer shortest paths and are willing to export any path to any neighbor. In other words, all paths in the random graphs are policy-compliant.

[**Event patterns**] We performed experiments and present the results of several event scenarios.

1. The convergence process from a clean start until each AS can reach all other ASes.

2. The static state of the network after this convergence.

3. A sequence of failures and recoveries of all interdomain links one at a time.

In all of our experiments each AS advertises a single destination prefix. The link failures and recoveries were separated by an 8 second interval making the complete experiment run for 3.6 hours.

[**Metrics**] In our experiments, we recorded the following metrics.

1. End-to-end data plane connectivity.

2. Packet header size.

3. Forwarding table size.

4. Control plane memory size.

5. Number of control plane messages.

Each of our experiments was repeated 3 times with a new topology in each round. Each CDF graph in this section includes all the data from the 3 experiments combined. So, a given point on a CDF graph essentially says that "given a random graph of the certain class, this metric value is expected to be at this percentile in the given scenario."

3.5.3 Results

In this section, we present the results of our experiments. Because in all of our experiments ASes were implemented as single routers, we use the words "router" and "AS" interchangeably.

Forwarding plane memory. Figure 3.10 shows a CDF of the number of forwarding table entries at each router. This figure confirms our understanding that the forwarding table size for LT routers scales with the number neighboring ASes, whereas the forwarding table size for PV routers scales with the size of the network.

In the Internet-like topologies, we see that a few routers have a rather large number of forwarding entries. This happens because core ASes in these topologies have a very high degree of connectivity. In the random graphs, the degree of connectivity varies much less with the most connected node having only about 20 neighbors.

Figure 3.10 also confirms another fact - the size of forwarding table in LT routers is independent of the style of policies their neighbors follow. The "LT" and "LT in Mixed" lines overlap in both graphs. The number of forwarding table entries for LT nodes in the "LT" case is 5.19 and is 5.23 in the "LT in mixed" case. The small difference in this numbers is due to random sampling of LT nodes for the mixed case. In comparison, the PV nodes averaged at 400.5 entries.

We also analyzed an AS-level topology of the Internet generated by CAIDA [10] from Jan 5, 2009. Using LT policies in this topology results in a maximum of $2,264$ and a mean of only 8.48 pathlets to represent an AS. In comparison, BGP FIBs would need to store between $132,158$ and $275,779$ entries for the currently announced IP

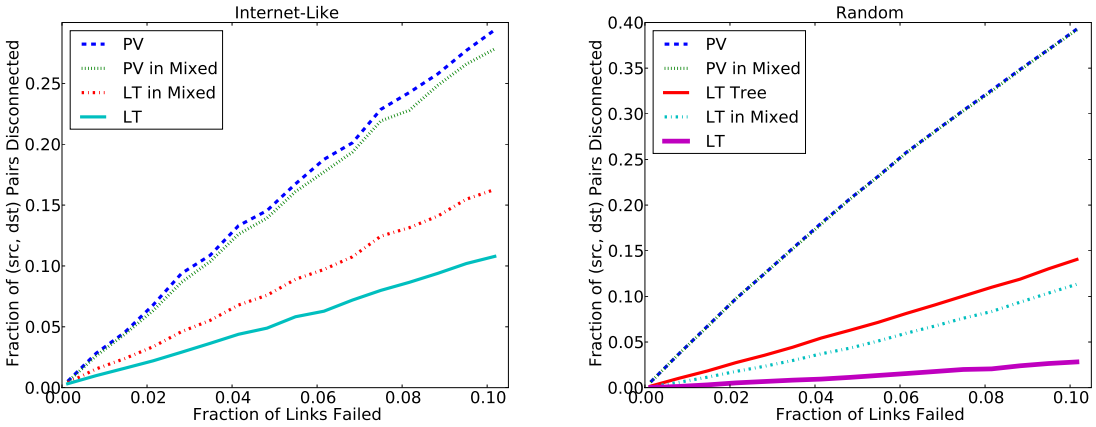


Figure 3.11: Probability of disconnection for a varying number of link failures in the Internet-like topology (left) and the random graph (right)

prefixes, depending on aggregation [5]. Thus, in this case, LT policies offer more than a $15,000\times$ reduction in forwarding state relative to BGP.

Route availability. Recall that one of the reason for developing interdomain multipath routing was to improve the Internet's availability. Here we evaluate how much pathlet routing is able to contribute on this front.

We measure the availability achieved in each of our scenarios as follows. We bootstrap the scenario and let the network converge to a stable state. Once converged, we dump all the vnode forwarding tables and record the pathlets known to each router. After all this information is collected we load it into a program and perform the following computation hundreds of times until the results look smooth.

We pick a random set of interdomain links and fail them. Then, for each pair of ASes (X, Y), we check if they are still connected in the vnode-pathlet graph. If so, then X can send packets to Y after these failures and we consider them connected.

This connectivity measurement is the same as the *static* availability we discussed for YAMR. Note that for pathlet routing with local transit policies, the static availability always equals dynamic availability because there is *no* convergence process after a link failure. An LT AS simply withdraws the failed pathlet from the Internet.

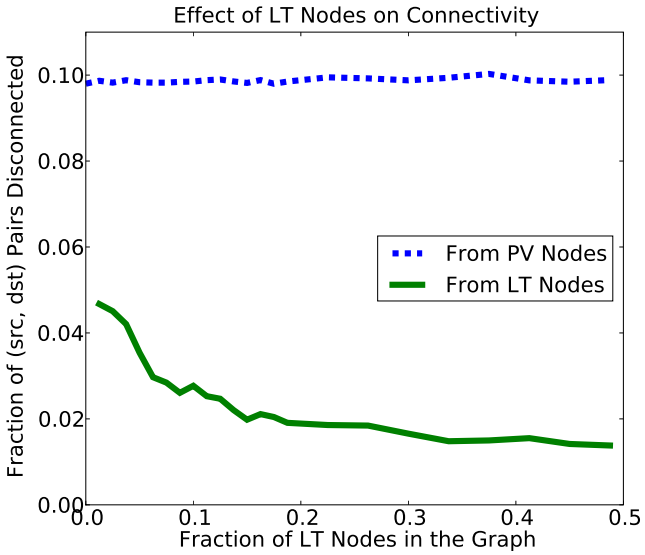


Figure 3.12: Probability of disconnection as a function of the number of LT nodes.

This withdrawal is entirely decoupled from all other LT pathlets and cannot "break" any working path.

In treating this computation as a measure of availability, we ignore the question of how the sender can find a working path. Previous work [51] has shown that a random search works surprisingly well. One can try further heuristics like choosing the most disjoined path from the failed one. If the sender wants to achieve ultimate reliability, she can send several copies of her packets simultaneously on different paths.

The availability results are shown in figure 3.11. The top line corresponding to all ASes adopting path-vector policies, essentially shows the current availability of BGP. The "PV in Mixed" line shows the availability seen by path-vector nodes in the mixed scenario. It is a bit different from the "PV" line in the Internet-like graph due to random distribution of LT and PV nodes in a topology with high variance of node degrees. Had we run this experiment on 30 as opposed to 3 topologies, these lines would have coincided as they do for the random graph.

The availability of LT nodes is different between the all-LT and the mixed scenarios. This difference comes from the fact that when PV nodes are introduced, they

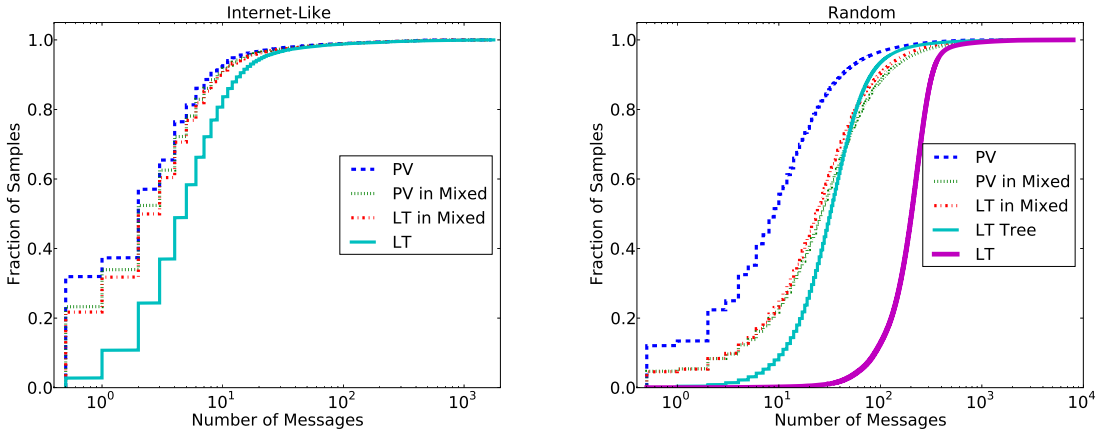


Figure 3.13: CDF of the number of messages received by a router following a link state change, for the Internet-like topology (left) and the random graph (right).

reduce all the paths that were available through them to a single one that they choose. Figure 3.12 shows how availability changes as the function of the fraction of LT nodes in the network. This figure comes from a series of experiments on 5 random graphs with 80 nodes and 150 edges. For all experiments, the number of failed links was fixed at 5 and the number of LT nodes was varied from 1 to 40.

The most important observation from these graphs is that local transit policies greatly improve the route availability. Moreover, even in the mixed scenario, LT nodes are able to greatly improve their own availability. The improvement that LT nodes see in the random graph is larger than in the Internet-like graph because random graph has more policy-compliant paths than the Internet-like graph, where only valley-free paths are allowed.

It is equally important to look at the difference between the Internet-like topology and the random graph from a different angle. In all cases, we use pathlet routing. The availability numbers change depending on the policy style ASes choose: PV is most restrictive, valley-free LT takes the middle ground, and LT without any policies gives the ultimate availability. The point to notice is that pathlet routing is able to support all of these policy styles (even when mixed) and realize their potential

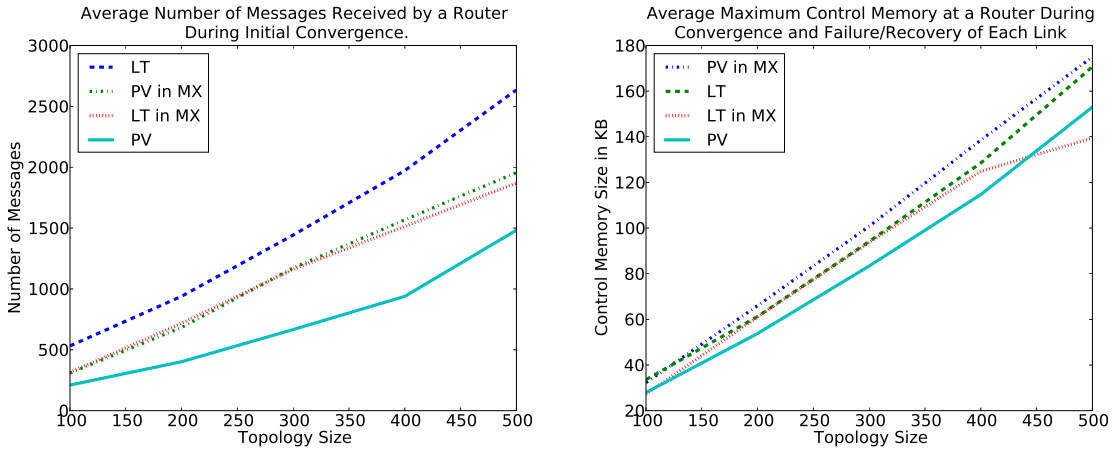


Figure 3.14: Scaling of messaging and control memory in the Internet-like graph, for 100, 200, 300, 400, and 500 nodes.

for path availability. In other words, pathlet routing does not artificially restrict the number of paths usable in the data plane the same way as BGP does.

To see this point more clearly, note that the "LT" line in the Internet-like graph actually represents the *best possible availability* because *all* of the policy-compliant paths are usable in the data plane.

The plot for the random graph in figure 3.11 shows one extra line called "LT Tree". This line represents the availability achieved by the all-LT scenario when each router advertises only the shortest path tree of pathlets to its neighbors. This line illustrates the fact that our availability figures depend on the number of pathlets known to the routers from our dissemination algorithm. In a practical deployment, if some router does not have enough pathlets advertised to it through pathlet dissemination, it can always pull more pathlets itself.

The "LT Tree" scenario achieves a worse availability level than regular "LT" but it requires less control messages as we will see next.

Control plane messages. Before we look at the figure, let us build some intuition for the number of control plane messages in different scenarios. Consider the Internet-like topology. Let N be the number of nodes in the topology and $N_c(X)$ be the number

of nodes in X's customer cone[2]. Let Q be the average number of pathlets per AS in the LT scenario.

An LT router X should be announcing at most QN pathlets to each of its customers because customers can reach all of the ASes through X but there are some pathlets which customers cannot use. X should be announcing at most $QN_c(X)$ pathlets to each of its peers and providers because both can reach only the ASes in X's customer cone.

A PV router Y should be announcing N pathlets to each of its customers because any customer can reach every other AS through Y via a single pathlet. Finally, Y should be announcing $N_c(Y)$ pathlets to each of its peers and providers because every one of them can reach only the customers of Y and only via a single pathlet.

From the analysis in the two paragraphs above, we see that LT routers are expected to announce about Q times more pathlets to their neighbors than PV routers. Figure 3.13 shows the experimental results. It presents the CDFs of the number of messages received by a router following a link failure or recovery event for the Internet-like and random topologies.

In our experiments, Q is equal to 5.19, but LT scenario results in only 1.69 times more messages than the PV scenario. There are two reasons explaining this fact:

1. As we already noted above, some pathlets are not reachable even by customers and they are not advertised. See section 3.2.2.

2. When a link fails or recovers in the LT scenario, only two pathlets need to be advertised or withdrawn. However, when a link fails in the PV scenario, the AS next to the link might have to withdraw all the pathlets that used this link. In addition, this AS will construct new pathlets and announce them as well.

In the random graph topology, the first factor is not true. Not only does every router learn all pathlets, but it learns them from all of its neighbors. The result is that LT has 10.3 times as many messages as PV.

[2]customer cone of X is the set of all ASes that are reachable from X using only provider→customer links.

Policies	Mean (bytes)	Max (bytes)
LT	125,706	835,900
Mixed	120,982	765,104
PV	112,656	519,994

Table 3.1: Mean and Max control plane memory of a router in different scenarios.

This overhead can be reduced by making ASes disseminate less pathlets. In particular, ASes can advertise only the shortest path tree sufficient to reach all destinations (see section 3.2.2). The "LT Tree" line in figures 3.13 and 3.11 is based on this dissemination algorithm. Disseminating less pathlets reduces the availability but improves the control messaging overhead by a factor of 4.6 bringing it to just 2.23 times more than that of PV in the random topologies.

Figure 3.14 plots the messaging cost for initial convergence as a function of N. It is evident that both LT and PV scale linearly with the size of the network. This behavior is expected because both use the same path-vector based dissemination algorithm and announce a similar number of pathlets across a given link. Over a given link, PV announces one pathlet per destination (on the order of N total) and LT announces at most all the LT pathlets in the network (on the order of $N \times \delta$, where δ is the node degree).

Finally, it should be possible to reduce the control plane messaging overhead using optimizations mentioned in section 3.2.2.

Control plane memory. Asymptotically, if the number of neighbors of an AS is δ, a PV router's state is $O(N\delta)$ because it will receive at most N advertisements from each of its δ neighbors. In LT, there are a total of $O(N\delta)$ pathlets in the Internet. An LT router, in the worst case, can receive all of these pathlets from all of its neighbors resulting in $O(N\delta^2)$ of state. Thus, control memory grows linearly with the number of ASes in the Internet in both cases.

The asymptotic analysis is confirmed by the right graph of figure 3.14. It shows the mean (over routers and over three trials) of the maximum (over time for each of trial) of the control plane state at each router. A trial consists of allowing the

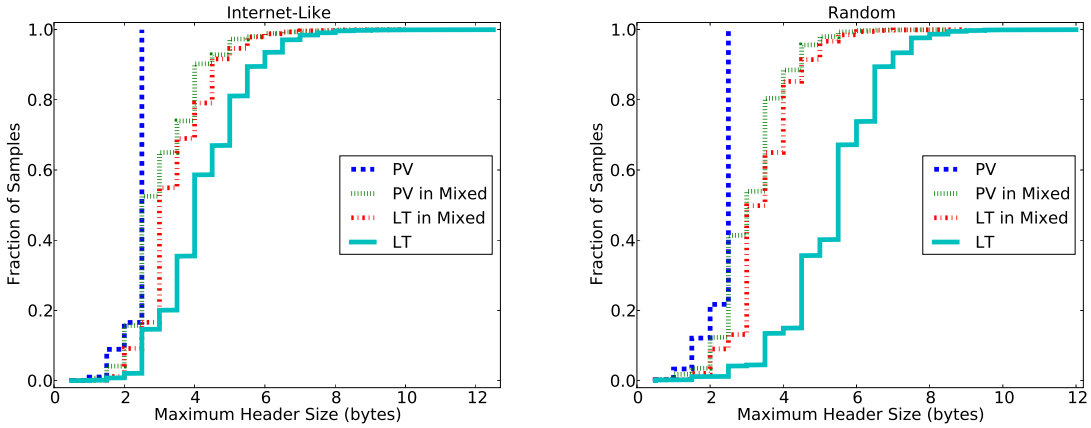


Figure 3.15: CDF of the size of the route field in the packet header, for the Internet-like topology (left) and the random graph (right).

network to converge and then failing and recovering each interdomain link one at a time. Table 3.1 shows the numbers for a 400 node topology.

Also note that because we disseminate only reachable pathlets, LT never actually reaches the worst case of being worse than PV by a factor of δ.

Header size. Because pathlet routing headers include the source route, their header size is not constant and can grow with the length of the path. During our experiments we sent packets between all pairs of ASes in the network along the shortest path (least number of pathlets) and recorded the packets' header sizes (by header size we mean the size of the FID list). Because FIDs can be popped and pushed along the way, we report the maximal header size for each packet. Figure 3.15 shows the CDFs of the maximal header sizes.

Packets in a PV only scenario always have exactly one FID in their FID lists, which is overwritten at each hop. Thus, the "PV" line in the graphs is the left-most one. In the mixed scenario, packet's headers change similarly to the headers of the packet sent from AS A to F in figure 3.1. Thus, the "PV in Mixed" and "LT in Mixed" lines are the middle ones. Finally, headers in the LT-only scenario are the largest ones because they contain the most FIDs. Nevertheless, average LT header

length for the Internet-like topology is only 4.21 bytes and less than 6 bytes in the random graph (random graph has larger headers because the average path length is higher). The maximal header length in all the cases is less than 12 bytes.

Using the fact that header length scales with the path length, we can extrapolate the numbers that we got from our experiments to the expected numbers for the Internet. In our Internet-like graph, the mean path length is 2.96. The mean AS-level path length of the Internet on January 22, 2009 is estimated to be 3.77 [10]. Thus, the average header length for the Internet is expected to be $\frac{3.77}{2.96} \times 4.21 = 5.36$ and the maximal header length is expected to be 15.9, which is less than a single IPv6 address.

Users can obviously construct paths of nearly arbitrary length, which would result in large header sizes. It does not pose a problem to pathlet routing because the users constructing long paths will themselves bare the cost (assuming that routers cannot be DoS'ed by such packets)

3.6 Conclusion

Pathlet routing is a flexible and extensible routing architecture that is able to efficiently support a wide class of policy styles. Among the various styles it is able to support, we have proposed and argued for a class of local transit policies. Local transit policies control the traffic between the ingress and the egress points, reduce the forwarding tables to a just few entries, and offer exponential number of paths to the users.

The building blocks of vnodes and pathlets bring the interdomain routing to the right level of abstraction. The language of vnodes and pathlets succinctly represents the (exponential) set of allowed paths, abstracts the details of the paths' implementation, and allows ASes to easily offer extra services at the vnode or pathlet granularity. In the next chapter, we will see how the right level of abstraction makes pathlet routing a forerunner for the interdomain routing in an evolvable Internet architecture.

Chapter 4

Framework for Evolvable Internet

In this chapter we take a slight side step and look at how pathlet routing can be a good candidate for interdomain routing of an evolvable Internet architecture. Going through this exercise, will let us underline some generally valuable principals that come together in pathlet routing.

4.1 Introduction

The popular attempt to rethink the Internet architecture has borne numerous fruits that addressed a variety of important challenges facing the status quo. Architectural proposals suggested how to improve the security (e.g. [3, 57, 66, 70]), how to make the network more data-oriented (e.g. [37, 40, 23]), how to better support mobility and middleboxes (e.g. [8, 56]), as well as how to improve the network's availability (e.g. [51, 74, 73]) and scalability (e.g. [61, 28, 19, 73]). However, it is sad to note that none of these great proposals is deployed.

We believe that a large part of the reason is that the current Internet architecture is not designed for evolution. One familiar manifestation of this fact is that the Internet Protocol (IP), with its header level details, permeates so many layers of networking from the applications' interface through which IP addresses are passed as 32bit integers to the Border Gateway Protocol (BGP) where reachability information is disseminated and aggregated in terms of IP prefixes. The depth of this entanglement

can be seen in the enormity of effort that IPv6 deployment is requiring.

We argue that any future Internet architecture must be evolvable. A static architecture can address all the existing problems, but as time has shown it is rather hard to predict future challenges. Thus, a long lasting architecture must permit the introduction of new architectural components to address arising challenges.

We find it mentally easier to call our evolvable architecture a *Framework* for Internet Innovation (FII). As the name framework correctly suggests, we envision that multiple architectures will coexist within the framework. Being a framework, the goal of FII is actually to decide as little as possible. We explicitly want to identify the minimal set of interfaces that need to be fixed to enable diverse architectures to evolve, coexist, and cooperate.

A detailed analysis of this question is presented in [41]. In this thesis, we concentrate on the role of interdomain routing in FII.

4.2 Architectural Anchors

Each architectural component has a fundamental set of entities, a *span*, that are dependent on its detailed specification. For example, DNS spans the name resolution servers, the host stacks, and some applications. TCP spans the host stacks and some middle boxes. IP spans virtually everything from router hardware all the way to the applications.

An evolvable framework should minimize the spans of architectural components, thereby facilitating changes to these components. However, some components have an inherently large span. We call these components architectural anchors. One anchor is the interface between the applications and the network because changing that interface would require recompiling all existing applications. Another anchor is the interdomain routing.

Interdomain routing is an anchor because it is responsible for basic end-to-end connectivity. Changing interdomain routing would require all domains in the Internet to adopt the new scheme. Without adopting the new scheme a domain won't be able to talk to its neighbors and communicate the basic reachability information.

Another candidate for an anchor can be a universal packet protocol (the seat taken by IP today). We believe that it is not an anchor because it does not have to exist. As we will see, interdomain routing can provide the basic reachability without requiring a single universal packet protocol.

Readers might also feel that hardware can be an anchor because it has to understand the packet protocol to forward at hardware speeds. However, recent Software-Defined Networking (SDN) approach transfers protocol understanding from the routers to the controller, leaving routers as dump lookup-and-forward boxes. It is our hope that SDN can make router hardware general enough to handle packet format changes with a possible firmware update.

Since architectural anchors are inherently hard to change, their interfaces must be as *abstract* and *extensible* as possible. By *abstract* we mean that the interface must be at the highest possible level, concealing as many implementation details as possible. By *extensible* we mean that anchors should be able to absorb new functionality so that upgraded components are able to utilize this functionality without breaking compatibility with the old ones. Making the anchor interfaces abstract and extensible minimizes the chances that they will need to change.

4.3 Pathlet Routing in FII

In this section, we argue that pathlet routing is extensible, abstract, and is a generally good fit for interdomain routing in FII.

4.3.1 Abstraction of Pathlet Routing

In some sense, pathlet routing is the epitome of abstraction. Vnodes are tied neither to the intradomain routing architecture nor to its addressing scheme. For example, domains can use regular IP, or a recently proposed AIP ([3]) to improve their security, or can even be flat L2 domains. Vnodes can be used to represent the domain's policy independently of the architecture they use. In the pathlet description, we mentioned that vnodes can be tagged with IP addresses that are reachable from

them. With a trivial extension, we can make vnodes taggable with AIP or MAC addresses. Pathlet routing simply carries these tags along leaving it up to the users to interpret them.

Pathlets are specified purely in terms of these vnodes and are independent of the technology used to implement then. By constructing a pathlet the AS is communicating to the rest of the internet merely that it is willing to carry traffic from the first vnode to the last vnode. It specifies neither the locations of these vnodes, nor the technology that will be used to carry the traffic. For example, carrier pigeons are a perfectly valid technology to implement a pathlet.

4.3.2 Extensibility of Pathlet Routing

As we argued above, vnodes and pathlet have a great policy expressiveness power. Here, we argue that they are also a great abstraction for ISPs to offer QoS and other extra services.

Specification and announcement. Recall that destination addresses were carried in pathlet routing as tags on vnodes. This tagging idea can be extended to arbitrary metadata attached to either vnodes or pathlets. For example, a domain can decide to offer low loss rate connection service for Internet conferencing applications. In the world of pathlet routing, specifying this new service and announcing it is straightforward. If this type of service is not entirely new, there is probably an existing pathlet tag[1] that is understood by popular videoconferencing applications. The domain then tags the pathlets on which it would like to offer the new service. As these pathlets are advertised, users who are interested in this service can pick them and send their videoconference data along them.

Tagging pathlets is most appropriate when the service is offered along the path. Likewise, domains can tag vnodes with metadata to announce services that are not associated with a path segment. For example, a domain might offer a virus checking service and tag a specific vnode with it. Users who would like their incoming packets

[1] we refer to pieces of metadata as tags for simplicity. In reality it would probably be a structured container similar to Google Protocol Buffers or JSON objects.

to be scanned for viruses can choose pathlets that go through this vnode.

Composability Another advantage of extensibility through pathlet or vnode tagging is that users can take advantage of new services as soon as they are offered by a few or even just one domain. Users are able to utilize the services offered by a few domains because pathlet routing allows them to pick the path that goes through the domains offering the service. Some services like virus detection need to appear at a single point along the path. The benefit of other services like Early Congestion Notification (ECN) grows proportionally with the portion of the path that offers them. Yet other services like a hypothetical reliable delivery service are only useful when they are present on the whole path. For all of these services, users in pathlet routing can pick the path that maximizes the service's benefit. On the other side, domains that don't implement a specific service can carry all packets without even being aware of what services these packets use in other networks.

To illustrate this point, let us imagine that the current Internet architecture is augmented with a repository that contains all the services offered by all the ISPs. Moreover, let us imagine that this repository is rather full with many (but not all) ISPs offering a variety of services. Furthermore, imagine that there exists an adequate mechanism to signal the network which services are sought for each packet. Unfortunately, even in this magical world, users won't be able to extract maximum benefit from the available services. The path that the network has chosen for them might not go through the services they desire, or only a fraction of the path might contain the service that requires presence on the complete path. Thus, being able to compose the path is critical for a successful extensible protocol.

The immediate usability of new services should encourage innovation because domains can start collecting revenue as soon as the service is rolled out.

4.3.3 Implementation Freedom

Pathlet routing offers domains the freedom to implement pathlet routing in any technology they choose. Let us update the figure 3.2 that showed a pathlet packet in the IP world to figure 4.1 that shows a pathlet packet in the FII world. The difference

Figure 4.1: A high level view of a pathlet header in FII.

between these figures is in change of "IP Header" into "Destination Domain Header". This change emphasizes that pathlet routing is actually oblivious to the technology used in the destination domain. The FID list is the only part of the header that pathlet routing needs to deliver the packet to the destination vnode. At that vnode, the FID list is empty and the destination domain uses the "Destination Domain Header" to deliver the packet to the final destination.

Likewise, pathlet routing is oblivious to the technologies used to carry the packet between intermediate vnodes. Transit domains can utilize the technology of their choice, be it some L2 protocol like Ethernet, or a tunneling mechanism like MPLS, or an optical link. The "Transit Header" is the placeholder for any transit domain's technology.

Another subtler piece of implementation flexibility is that the bits in the pathlet header, the FIDs, are completely opaque to everyone but the domain that originally constructed this pathlet. This enables fast innovation and service rollout compared for example to differentiated services bits (until recently known as Type of Service (see [53])) that are defined in the IP packet header. The latter requires a global agreement, changes once in a decade or two, and eats up the bits in every single packet, even in those that are not interested in differentiated services. In pathlet routing, the domain is free to decide on the signals that it wants to see in the packet headers. The whole loop from domains deciding on the signals to the users employing these signals in their packet headers can be on the order of minutes, versus several decades in the status quo.

4.3.4 Discussion

Stepping aside from concrete details, let us review the grand picture. Innovation happens when there are little barriers. Besides the technical ones, there is a barrier of human coordination, which gets increasingly higher when the parties represent different competing organizations. In the context of the Internet, this observation suggests that to facilitate innovation we must widen the ability of autonomous domains to innovate independently of other domains.

In recent years, ISPs have successfully adopted MPLS for traffic engineering and QoS purposes ([62, 67]) showing that ISPs are willing and capable to innovate when no widespread agreement with other ISPs is necessary. Hence the interdomain routing protocol in FII must allow each domain to innovate on the widest plane possible.

Pathlet routing can meet this requirement because domains can deploy any intradomain routing architecture and offer new service all without coordinating with other domains[2]. Pathlet routing even allows the two ends of a communication to be residing in domains with different architectures[3]. Hence we believe that pathlet routing is a good candidate for an interdomain routing protocol in FII. In the next section, we present our proof of concept pathlet-based FII implementation.

4.4 Implementation

In this section, we report on a skeleton implementation of FII's interfaces. While our implementation barely scratches the surface of what a fully functional prototype would include, it captures most aspects of the information flow within the framework. Since the purpose of our implementation is to evaluate FII's ability to support innovation, not to evaluate its performance or scalability, we feel this degree of implementation is sufficient.

[2]Domains must obviously agree on the peering technology for the shared links, but they can utilize any architecture internally.

[3]Using the "Destination Domain Header". See figure 4.1

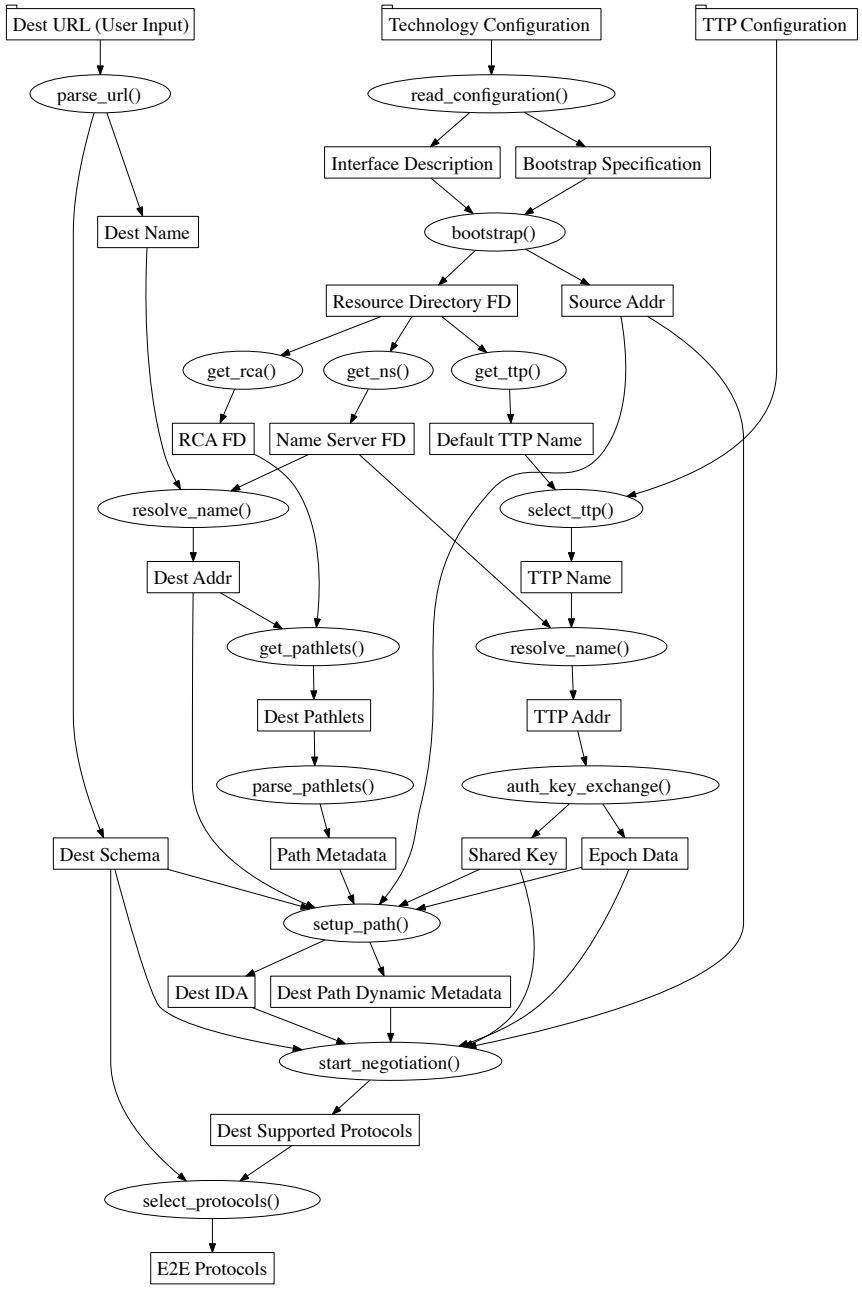


Figure 4.2: Information flow in FII from the perspective of a client host. Rectangles represent pieces of information and ovals represent functions that combine information to yield new information.

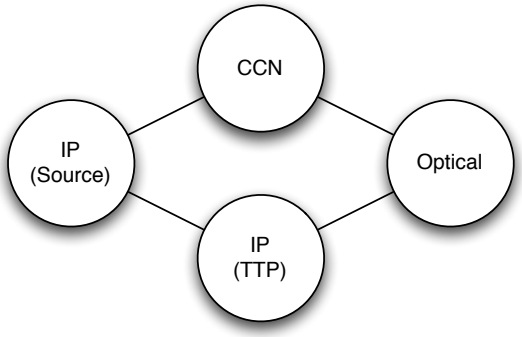


Figure 4.3: The domain topology used in the experiment.

4.4.1 Implementation Details

Our implementation focuses on the core FII interfaces, information flow, and the main features of implemented architectures. In particular, we do not implement the detailed mechanisms *behind* various interfaces. For example, we don't implement recursive DNS queries, just the name resolution request and reply between a host and a server. In the case of CCN [37], we implement the main features of name registration and name-based routing with simple mechanisms behind them. Also, note that because no prior architecture was designed with FII in mind, we necessarily altered their designs to fit them within FII.

The information flow in our working implementation is very similar to the information flow depicted in Figure 4.2. This figure shows the information flow that a fully general FII implementation would go through at a client host from the bootstrap to sending the first packet for a named resource. It shows a few components of FII that we did not describe in this thesis including Trusted Third Party (TTP), Route Computation Agent (RCA), network API schemas, protocol negotiation, etc. Please refer to [41] for details. We included the complete figure to give a visual sense of our implementation and of some of the features that pathlets can work with.

To capture information flow more explicitly, we implemented FII entities (hosts, routers, RCAs, name servers, etc.) in separate processes communicating via protocols defined using Google protocol buffers [1]. We chose to use protocol buffers for all of

our messages to focus on the content and structure of headers rather than their byte-level formatting. Communication between processes happens over a topology whose links are implemented as TCP connections. Our experiment described below contains a total of 22 communicating processes.

4.4.2 Experiment Setup

We setup an experiment with 4 domains as shown in figure 4.3. The "IP (Source)" domain is an IP domain that contains the host initiating communication. The "IP (TTP)" domain is another IP domain that hosts a trusted third party server. The "CCN" domain runs a content-centric architecture where routing is done on names. Finally, the "Optical" domain is a domain that has deployed a hypothetical all-optical architecture.

The hypothetical all-optical architecture differs from other architectures in that pathlets are setup on-demand (a dedicated lambda might need to be setup along some path for a high bandwidth data transfer). The all-optical architecture with the path setup feature acts as yet another illustration of abstractness and extensibility of pathlet routing. The path setup functionality can also be used on pathlets that require senders to obtain a capability before using the pathlet.

In our implementation, to perform path setup, a host reads pathlet metadata indicating the entities (middleboxes or routers) that must be contacted to establish the path. By sending a path setup request to these parties, the host causes path establishment and receives the information necessary to construct the "Destination Domain Header" (see figure 4.1).

In the experiment, we send data transfer requests from the source host to hosts in each of the three other domains, all within the same experiment. As the experiment is running, we collect packet traces in all four domains. We then classify the packets in the traces into one of 7 categories (e.g. bootstrap, key exchange, naming, etc.) and plot all the packets for each domain and each category on a separate horizontal line in figure 4.4. Above the graph we denote the five phases of communication in the experiment.

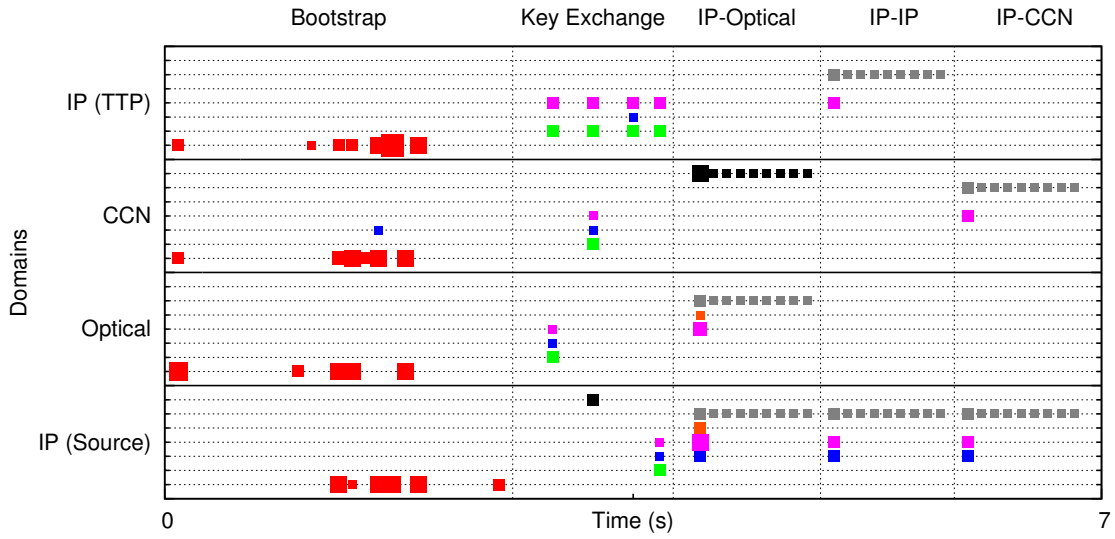


Figure 4.4: Packet traces for the 4-domain experiment. For each of the domains (listed along the y-axis), we categorize packets into seven types, each of which is a row (from bottom to top) for each domain: 1) bootstrap, 2) key exchange, 3) naming, 4) pathlet data, 5) path setup, 6) end-to-end data transfer, and 7) interdomain transit. A dot appears if packet(s) were observed of that type, in that domain, at that time. Dot size depicts packet count. We note the communication phases above the graph.

In the first phase, all entities perform ordinary bootstrap operations, except CCN, which also registers a name. In the key exchange phase, all end hosts and their routers perform key exchange with the TTP. Since the CCN domain and the TTP's domain are not directly connected, they use the other IP domain as transit in this phase. The next three phases involve data retrieval. The first of those, from the source host to a host in the all-optical domain requires path setup, and subsequently more pathlet-related messages than other data retrieval requests. This data retrieval uses the CCN domain as transit.

4.4.3 Discussion

The main point of this experiment is that the abstract and extensible nature of pathlet routing enabled domains with different architectures to communicate with each other. Moreover, interdomain transfers successfully utilized transit domains without knowing the architecture they deployed.

The path setup feature included in this experiment was implemented using pathlet metadata. We found that using metadata as extensibility vehicle to be natural and straightforward.

4.5 Conclusion

Designing for evolution can make facing unforeseeable future challenges less painful. Inspired by this goal, we tried to reason about the minimal set of architectural components that need to be fixed to glue other components together. Interdomain routing appears to be among the must-be-fixed components as it necessarily spans all domains and requires global agreement.

Our analysis argued that pathlet routing is a great candidate for the interdomain routing protocol. Its building blocks – vnode and pathlets – are able to support evolution and diversity not only in policy styles but also in domain technologies. As our experiment demonstrates, domains deploying different architectures are able to communicate and carry each other's traffic with the help of pathlet routing's abstractions.

Chapter 5

Conclusion

This thesis presented two interdomain multipath routing protocols: YAMR and Pathlet Routing. Both improve the Internet's availability, give users choice over their paths, and improve certain dimensions of the Internet's scalability, but that is where the similarities end.

YAMR keeps the default paths the same as in BGP and constructs alternate paths that avoid links on the default ones. Constructing paths in YAMR requires more messages that BGP, but YAMR offers a novel hiding technique that is able to localize failures to small neighborhoods around them. Hiding is fully automatic, safe, and preserves next-hop policies. If the network around the failure is densely connected, hiding is able to contain the failure with a small number of messages that is independent of the size of the network. Together with hiding YAMR keeps the churn level well below that of BGP and improves BGP's availability but almost 3 orders of magnitude.

Pathlet routing is a departure from familiar approaches to routing and an embracement of abstraction. Pathlet routing defines abstract constructs of vnodes and pathlets that encode domains' policies in a directed graph with the crucial property that any path in this graph is policy-compliant. Vnodes and pathlets are expressive enough to cover a wide range of policy styles from newly proposed local transit policies to the familiar BGP-style policies. Domains that follow local transit policies see their forwarding table state drop to nearly zero, while providing exponential path

choice to the users. The vnodes and pathlets are also convenient concepts for defining QoS and other services on top of them. Finally, the abstract nature of pathlet routing makes it a great candidate for an evolvable Internet architecture.

YAMR is a testament that even decades old *designs* can often be revamped into surprisingly well-performing constructions. Pathlet routing on the other hand, is a testament and even decades old *problems* can hide surprisingly elegant novel solutions. Whatever the case maybe, it's worth a second look.

5.1 Limitations and Future Work

Since the routing system takes one of the center seats in networking, a complete evaluation requires great effort and depth. Below we describe some of the limitations of our work and possible future directions.

Anycast and multicast for pathlet routing. Being a kind of source routing, pathlet routing inherits the source routing's difficulties with anycast and multicast. It is theoretically possible to encode a whole distribution tree in the pathlet header. However, this approach does not scale well. A more promising approach is to adopt the trick of [58] to encode the multicast tree branches using a bloom filter. In fact, this approach should be applicable to pathlet routing in a cleaner fashion than to BGP because pathlets are a natural branching unit. The branching points can probably be placed at special vnodes whose forwarding rules will differ slightly (i.e. they will send multiple packets with different FID lists for each incoming packet) from the regular vnode forwarding. The details of this approach have to be worked out and evaluated.

Optimizing Pathlet Dissemination. As we pointed out in the pathlet discussion, our path-vector based dissemination scheme can be optimized by using the fact that path-vectors are needed only to ensure the liveness of the dissemination channel. Investigating this question can lead to interesting theoretical results.

Deeper study of offering services through pathlet metadata. We have implemented a simple case of utilizing pathlet metadata in our FII implementation. However, devil is usually in the details. A deeper study of how an AS can offer a

practical service through pathlet metadata would be valuable. In particular, how can differently parametrized versions of a service be offered without exploding the number of advertised pathlets.

Consistency benefits of pathlet routing. Recent work [39] has shown that lack of consistency in routing causes major problems and that other useful mechanisms can be built on top of consistent routing. It also illustrated that it is challenging to make BGP consistent. Pathlet routing can be made consistent with high probability fairly easily, by requiring that ASes do not reuse FIDs until the dissemination mechanism has withdrawn the pathlet that previously used that FID. We have not evaluated how this local requirement affects various metrics and what realistic benefits can be drawn from it, but we believe that consistency has important practical implications that may be interesting to combine with pathlet routing.

More faithful implementation of FII. FII argues for a microkernel approach to Internet architecture. However, even the microkernel of the Internet is amazingly large. In our implementation we necessarily had to take many shortcuts. A down-to-the-wire implementation of FII in undoubtedly necessary to understand exactly what types of changes FII can facilitate and where it falls short.

Bibliography

[1] Google protocol buffers. http://code.google.com/p/protobuf/.

[2] Inferred as relationships dataset, May 2008. http://www.caida.org/data/active/as-relationships/index.xml.

[3] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable internet protocol (aip). In *SIGCOMM*, pages 339–350, 2008.

[4] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proc. 18th ACM SOSP*, October 2001.

[5] Routing table report. http://thyme.apnic.net/ap-data/2009/01/05/0400/mail-global.

[6] Avaya. Converged network analyzer. http://www.avaya.com/master-usa/en-us/resource/assets/whitepapers/ef-lb2687.pdf.

[7] B. Awerbuch, D. Holmer, H. Rubens, and R. Kleinberg. Provably competitive adaptive routing. In *INFOCOM*, 2005.

[8] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the Internet. In *Proc. of ACM SIGCOMM*, 2004.

[9] L. Blunk, J. Damas, F. Parent, and A. Robachevsky. Routing Policy Specification Language next generation (RPSLng). RFC 4012 (Proposed Standard), March 2005.

[10] CAIDA AS ranking. http://as-rank.caida.org/.

[11] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. In *SIGCOMM*, pages 1–12, 2007.

[12] David Clark, Scott Shenker, and Aaron Falk. GENI Research Plan. Technical Report GDD-06-28, April 2007.

[13] David Clark, John Wroclawski, Karen Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's Internet. In *SIGCOMM*, 2002.

[14] X. Dimitropoulos, D. Krioukov, A. Vahdat, and G. Riley. Graph annotations in modeling complex network topologies. *ACM Transactions on Modeling and Computer Simulation (to appear)*, 2009.

[15] Xenofontas Dimitropoulos, Dmitri Krioukov, Amin Vahat, and George Riley. Graph annotations in modeling complex network topologies. In *NSDI*, August 2007.

[16] Xenofontas A. Dimitropoulos, Dmitri V. Krioukov, Marina Fomenkov, Bradley Huffaker, Young Hyun, Kimberly C. Claffy, and George F. Riley. As relationships: inference and validation. *Computer Communication Review*, 37(1):29–40, 2007.

[17] Z. Duan, J. Chandrashekar, J. Krasky, K. Xu, and Z.-L. Zhang. Damping BGP route flaps. In *IEEE International Performance Computing and Communications Conference*, 2004.

[18] Jon Postel (ed.). DARPA internet program protocol specification. In *RFC791*, September 1981.

[19] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID separation protocol (LISP). In *Internet-Draft*, March 2009.

[20] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of autonomy for the expressiveness of policy routing. In *SIGCOMM*, pages 25–36, New York, NY, USA, 2005. ACM.

[21] Joan Feigenbaum, Christos H. Papadimitriou, Rahul Sami, and Scott Shenker. A bgp-based mechanism for lowest-cost routing. *Distributed Computing*, 18(1):61–72, 2005.

[22] Joan Feigenbaum, Rahul Sami, and Scott Shenker. Mechanism design for policy routing. *Distributed Computing*, 18(4):293–305, 2006.

[23] Michael J. Freedman, Matvey Arye, Prem Gopalan, Steven Y. Ko, Erik Nordstrom, Jennifer Rexford, and David Shue. Service-centric networking with SCAFFOLD. Technical Report TR-885-10, Princeton Computer Science Department, September 2010.

[24] Igor Ganichev, Bin Dai, Brighten Godfrey, and Scott Shenker. Yamr: Yet another multipath routing protocol. *Computer Communication Review*, 40(5):13–19, 2010.

[25] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, December 2001.

[26] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, 2001.

[27] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. In *SIGMETRICS*, pages 307–317, 2000.

[28] Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In Pablo Rodriguez, Ernst W. Biersack, Konstantina Papagiannaki, and Luigi Rizzo, editors, *SIGCOMM*, pages 111–122. ACM, 2009.

[29] P. Brighten Godfrey, Matthew Caesar, Ian Haken, Scott Shenker, and Ion Stoica. Stable Internet route selection. In *NANOG 40*, June 2007.

[30] Mohamed G. Gouda. *Elements of network protocol design.* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[31] T. Griffin and B. Premore. An experimental analysis of BGP convergence time. In *ICNP*, 2001.

[32] Timothy Griffin, F. Bruce Shepherd, and Gordon T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002.

[33] K. Gummadi, H. Madhyastha, S. Gribble, H. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *OSDI*, 2004.

[34] P. Krishna Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall. Improving the reliability of internet paths with one-hop source routing. In *OSDI*, pages 183–198, 2004.

[35] G. Huston. Damping BGP, 2007. http://www.potaroo.net/presentations/2007-12-02-dampbgp.pdf.

[36] Geoff Huston. BGP routing table analysis reports, 2009. http://bgp.potaroo.net/.

[37] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking Named Content. In *Proc. of ACM CoNEXT*, 2009.

[38] Luo Jiazeng, Xie Junqing, Hao Ruibing, and Li Xing. An approach to accelerate convergence for path vector protocol. In *Globecom*, pages 2390 – 2394, 2002.

[39] J. John, E. Katz-Bassett, A. Krishnamurthy, and T. Anderson. Consensus routing: The Internet as a distributed system. In *NSDI*, 2008.

[40] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-Oriented (and Beyond) Network Architecture. In *Proc. of ACM SIGCOMM*, 2007.

[41] Teemu Koponen, Scott Shenker, Hari Balakrishnan, Nick Feamster, Igor Ganichev, Ali Ghodsi, Brighten Godfrey, Nick McKeown, Guru M. Parulkar, Barath Raghavan, Jennifer Rexford, Somaya Arianfar, and Dmitriy Kuptsov. Architecting for innovation. *Computer Communication Review*, 41(3):24–36, 2011.

[42] N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?! it must be BGP. In *Computer Communication Review*, 2007.

[43] N. Kushman, S. Kandula, D. Katabi, and B. Maggs. R-BGP: Staying connected in a connected world. In *NSDI*, 2007.

[44] Nate Kushman, Srikanth Kandula, Dina Katabi, and Bruce M. Maggs. R-bgp: Staying connected in a connected world. In *NSDI*, 2007.

[45] C. Labovitz and A. Ahuja. Experimental study of internet stability and wide-area backbone failures. In *Fault-Tolerant Computing Symposium (FTCS)*, 1999.

[46] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. In *ACM SIGCOMM*, 2000.

[47] Karthik Lakshminarayanan, Ion Stoica, Scott Shenker, and Jennifer Rexford. Routing as a service. Technical Report UCB/EECS-2006-19, UC Berkeley, February 2006.

[48] Z. M. Mao, R. Bush, T. Griffin, and M. Roughan. BGP beacons. In *IMC*, 2003.

[49] Nick McKeown and Bernd Girod. Clean-slate design for the internet. Whitepaper, April 2006.

[50] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), September 2007.

[51] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. In *SIGCOMM*, pages 27–38, 2008.

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

[53] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260.

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

[55] Dan Pei, Matt Azuma, Daniel Massey, and Lixia Zhang. Bgp-rcn: improving bgp convergence through root cause notification. *Computer Networks*, 48(2):175–194, 2005.

[56] Lucian Popa, Ion Stoica, and Sylvia Ratnasamy. Rule-based forwarding (RBF): Improving the Internet's flexibility and security. In *Proc. Workshop on Hot Topics in Networking*, 2009.

[57] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. In *ACM SIGCOMM*, 2004.

[58] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting IP multicast. In *Proc. ACM SIGCOMM*, 2006.

[59] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (BGP-4). In *RFC4271*, January 2006.

[60] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. In *RFC3031*, January 2001.

[61] Ankit Singla, Brighten Godfrey, Kevin R. Fall, Gianluca Iannaccone, and Sylvia Ratnasamy. Scalable routing on flat names. In *CoNEXT*, page 20, 2010.

[62] Neil T. Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. In *SIGCOMM*, pages 133–145, 2002.

[63] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *INFOCOM*, 2002.

[64] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. In *ACM SIGCOMM*, 2006.

[65] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. In *ACM SIGCOMM*, 2006.

[66] Dan Wendlandt, Ioannis Avramopoulos, David Andersen, and Jennifer Rexford. Don't Secure Routing Protocols, Secure Data Delivery. In *Proc. Workshop on Hot Topics in Networking*, 2006.

[67] Xipeng Xiao, Alan Hannan, and Brook Bailey. Traffic engineering with mpls in the internet. *IEEE Network Magazine*, 14:28–33, 2000.

[68] Wen Xu and Jennifer Rexford. Miro: multi-path interdomain routing. In *SIGCOMM*, pages 171–182, 2006.

[69] Wen Xu and Jennifer Rexford. MIRO: Multi-path Interdomain ROuting. In *SIGCOMM*, 2006.

[70] Abraham Yaar, Adrian Perrig, and Dawn Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[71] Xiaowei Yang. NIRA: A new internet routing architecture. Technical Report MIT-CSAIL-TR-2004-064, Massachusetts Institute of Technology, October 2004.

[72] Xiaowei Yang, David Clark, and Arthur Berger. NIRA: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 15(4):775–788, 2007.

[73] Xiaowei Yang, David Clark, and Arthur W. Berger. Nira: a new inter-domain routing architecture. *IEEE/ACM Trans. Netw.*, 15(4):775–788, 2007.

[74] Xiaowei Yang and David Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, pages 159–170, 2006.

[75] Dapeng Zhu, Mark Gritter, and David Cheriton. Feedback based routing. *Computer Communication Review (CCR)*, 33(1):71–76, 2003.

Appendix A

Proofs for Chapter 2

A.1 Preliminaries

First, we define and discuss several preliminaries.

A.1.1 Policy Assumptions

The proofs of some of our results require two assumptions about ASes' routing policies. We assume that ASes follow *next-hop* and *widest-advertisement* policies, which we define below. Neither of these policy classes are new ([22], [44]). We adopt the definition of former class without change, but clarify the definition of the latter.

We say that an AS follows *next-hop* policies if its export filter is based solely on the destination and on the path's next-hop AS. In other words, all paths for a given destination from a given peer[1] are announced to the same set of peers. We say that AS A's policies are *widest-advertisement* policies if the following is true for all destination prefixes p (all paths are towards p). If A is willing to advertise a path from peer B to peer C, then whenever B advertises a path to A, A has to be willing to advertise its path to C. If A uses a path through C it does not need to advertise it back to C, but the export filter has to allow A to advertise the path back to C.

[1] We use the word "peer" in appendix to mean a neighbor without implying the peering business relationship.

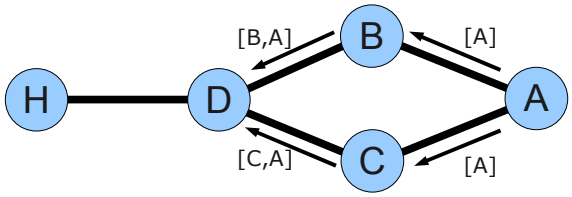


Figure A.1: A announces a prefix and sends it with path $[A]$ to B and C. Both B and C choose this path and send paths $[B, A]$ and $[C, A]$ to D, respectively. D prefers path $[D, B, A]$ over path $[D, C, A]$. D's export filter allows path $[D, C, A]$ but path $[D, B, A]$ is blocked. Therefore, D does not send anything to H, which becomes disconnected even though the path $[H, D, C, A]$ is working and policy-compliant.

Finally, if the AS hosting a destination prefix follows widest-advertisement policies, it has to advertise the destination prefix to all of its peers.

We make a specific set of assumptions that include valley-free customer-peer-provider policies. Our results most probably hold under a larger set of assumptions, but we do not attempt to find the precise class of policies under which our results hold. In fact, even for BGP the class of policies that are guaranteed to give each AS a path when there is a policy compliant path is unknown to the best of our knowledge. Figure A.1 is an example where BGP leaves an AS disconnected when there is a working policy-compliant path.

Definition 1 *A path* $P = [A_n, A_{n-1}, \ldots, A_0]$ *is a* policy-compliant *path if for each* $0 \leq i \leq n-1$, A_i *is willing to advertise path* $[A_i, A_{i-1}, \ldots, A_0]$ *to* A_{i+1}.

In fact, by contemplating figure A.1 and the widest-advertisement assumption, it seems that some version of the widest-advertisement assumption is required to ensure that nodes with policy-compliant paths will not be disconnected in BGP.

A.1.2 Ordering of Peers

The next-hop and widest-advertisement policies imply a useful categorization of peers that we define and proof here. Next-hop and widest-advertisement policies are assumed throughout this section.

Fix a destination prefix, i.e. everything in this section will refer to paths to a single destination prefix. Let $p_1, p_2, \ldots, p_k$ be the peers of an AS A. For each p_i, let s_i denote the set of peers to which A would be willing to advertise a path learned from p_i. The set s_i is determined solely by the export filter and can contain p_i.

Partition peers into equivalence classes based on the equality of corresponding sets s_i and order classes in decreasing order of the size of s_i (for now, break ties arbitrarily. We will show that there is actually no ties.). Denote the peers in class j by $p_{j,1}, p_{j,2}, \ldots, p_{j,k_j}$. Denote the set s_i corresponding to the class j by S_j and let there be a total of c classes.

Lemma 1 *For any two classes $1 \leq i < j \leq c$, $S_j \subset S_i$ (S_j is a strict subset of S_i).*

Proof: Pick two peers $p_{i,m}$ and $p_{j,n}$ from classes i and j, respectively. Consider a case when A gets advertisements only from $p_{i,m}$ and $p_{j,n}$. There are two possibilities: either A chooses a path through $p_{i,m}$ or a path through $p_{j,n}$.

If A chooses a path through $p_{i,m}$, by the assumption of widest-advertisement policies, A has to advertise the path it chose (the path through $p_{i,m}$) to all of S_j. Because, the set of peers to which A advertises a path through $p_{i,m}$ is by definition S_i, S_j has to be a subset of S_i. Furthermore, S_j cannot have the same number of elements as S_i because then the two sets would be equal and would not represent two different classes. Thus, $S_j \subset S_i$

If A chooses a path through $p_{j,m}$, by the symmetric argument to the one above, we get that S_i has to be a strict subset of S_j, which is impossible because $|S_i| > |S_j|$ by the choice of ordering. Therefore, A cannot prefer a path through $p_{j,m}$ if it follows widest-advertisement and next-hop policies. The impossibility of this case implies the next lemma. ■

Lemma 2 *For any two classes $1 \leq i < j \leq c$ and any two peers $p_{i,m}$ and $p_{j,n}$ from these classes, a path through $p_{i,m}$ is more preferred than a path through $p_{j,n}$. Further, we call class i a* more preferred class *than j.*

Proof: Follows from the proof of the previous lemma. ■

A.2 YPC Convergence

In this section, we prove theorem 2. We use the framework and results presented in [32]. Unfamiliar readers should read this work if they desire to rigorously understand our arguments. Otherwise, the basic ideas should be clear.

First, we extend the SPVP definition of [32] to YPC and call it SYPC. Because the extension is an obvious one, we present only the salient differences and omit the details. The `rib` and `rib_in` of SYPC contain not a single path as in SPVP but multiple paths - one per label. Each message still contains a single path, but it also contains a label for this path. The same policies are applied to all paths, independent of their labels. The path selection process is the same as described in the YPC design above. We restate it in the next paragraph.

When a node processes a message with a default path, it updates the default path entry of the peer's `rib_in`. Then, it chooses the best default path from all the available default paths in `rib_in`s. If the best path has changed, it is sent to all the peers and the path selection is run for all the labels on the new default path. When a node processes a message with an alternate path, it inserts this path into the `rib_in` and runs path selection on the path's label. Path selection for an alternate label is the following process. The node chooses the best path from all the default and like-labeled paths in `rib_in`s. If this path is different from the current path with the same label in the `rib`, the `rib` is updated and the path is sent to the peers.

Proof of Theorem 2: First of all, notice that the processing of alternate paths results in some extra state, messages, and processing, but does not impact the dynamics and processing of the default paths. In particular, given an instance of SYPC and an analogous instance of SPVP (with the same graph, the same policies, equivalent initial configuration, and equivalent activation sequence), they will have exactly the same evolution (same state changes and messages). Therefore, because we know that SPVP converges in the absence of dispute wheels, we can conclude that the default paths of SYPC converge in the same conditions. Moreover, because we know that SPVP always converges to a unique configuration, the default paths of SYPC converge to a unique configuration. This argument is a restatement of the fact that

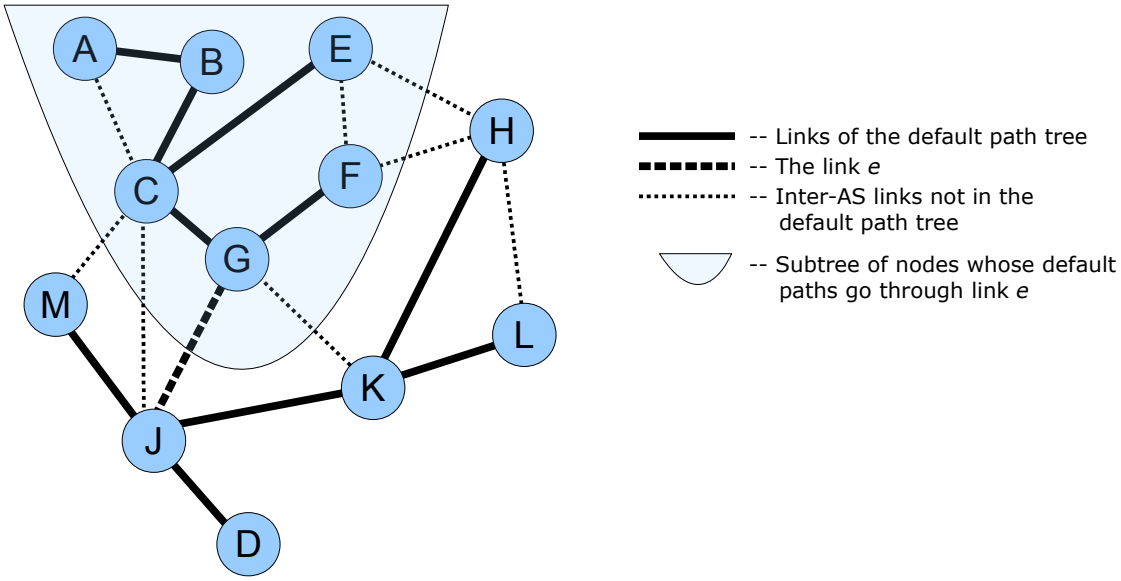


Figure A.2: An illustration of *e-brother*. Node D is the destination. The subtree of nodes in the shaded area together with thick edges is the T^e. All the nodes together with thick edges is the default path tree T. The only change to the policies in e-brother is that nodes outside of T^e that have peers in T^e (nodes M, J, K, H), don't accept any paths from these peers. The only structural change is that link e ((G, J)) is removed. These changes cannot introduce a dispute wheel and they preserve the widest-advertisement and next-hop policies.

the default paths of YPC are constructed in exactly the same way as default paths of BGP. Next, we show that alternate paths also converge.

Intuitively, after the default paths have converged, alternate paths for each label behave just like regular BGP path on a subgraph (the subgraph of nodes whose default paths go through the edge corresponding to the label) of the whole graph. The reasons for this similarity are that the processing of each label is isolated from the processing of any other label (i.e. there is no interdependence between labels), and that the nodes don't change their interest in any label (because the default paths don't change). Therefore, intuitively, all alternate paths should converge. To show this fact more rigorously, for each label e, we define an instance of SPVP that evolves

in the same way as the e-labeled paths in SYPC. We call such an instance of SPVP an *e-brother* of SYPC (see figure A.2 for an illustration).

Consider the state S of SYPC after all default paths have converged. Let $G = (V, E)$ be the AS graph from the given instance of SYPC. Let the fixed destination prefix p be hosted by AS D. Consider the tree T of converged default paths to D in S. Let T^e denote the subtree of T consisting of the ASes whose default paths go through e. T^e is the tree of ASes that are interested in e-labeled paths. Next, we first define the state of ASes in the e-brother, then the state of edges, and finally, the permitted path sets.

For each AS A in T^e, its `rib` in e-brother contains the e-labeled path from A's `rib` in S. For each peer AS B of A, the $\texttt{rib\_in}_A(B)$ in e-brother contains the e-labeled path from $\texttt{rib\_in}_A(B)$ in S, if B is in T^e. If B is not in T^e, then $\texttt{rib\_in}_A(B)$ in e-brother contains the default path from $\texttt{rib\_in}_A(B)$ in S.

For each AS A not in T^e, its `rib` in e-brother contains the default path from A's `rib` in S. For each peer AS B of A, the $\texttt{rib\_in}_A(B)$ in e-brother is empty, if B is in T^e. If B is not in T^e, then $\texttt{rib\_in}_A(B)$ in e-brother contains the default path from $\texttt{rib\_in}_A(B)$ in S. If any of the paths is not available, the corresponding entry in e-brother is empty.

The links of e-brother are the same as the links of S except for link e, which is not present in e-brother. For two ASes A and B, the link from A to B contains the following based on whether A and/or B are in T^e:

- $A \in T^e$, $B \in T^e$: the link $A \to B$ in e-brother contains all the e-labeled paths in link $A \to B$ in S.
- $A \in T^e$, $B \notin T^e$: the link $A \to B$ in e-brother is empty.
- $A \notin T^e$, $B \in T^e$: the link $A \to B$ in e-brother contains all the default paths in link $A \to B$ in S (there is actually no such paths because all default paths have converged in S).
- $A \notin T^e$, $B \notin T^e$: the link $A \to B$ in e-brother contains all the default paths in link $A \to B$ in S (there is actually no such paths because all default paths have

converged in S).

Finally, we describe the permitted path sets and complete the definition of e-bother. The permitted path sets in e-brother are exactly the same as in S except that for each AS $A \notin T^e$ and all of A's peers $B \in T^e$, we exclude all paths through B from P^A. The reason for this exclusion is to prevent e-labeled paths from T^e from affecting default paths outside of T^e. This is a feature of YPC and the e-brother should simulate it.

Given the definition of e-brother, it is obvious that it will evolve in the same way as the e-labeled paths in S. Therefore, to show that e-labeled path converge in S, we need to show that e-brother converges. The e-brother is guaranteed to converge because it has the same given policies, which don't have dispute wheels. We only need to note that shrinking permitted path sets cannot introduce a dispute wheel because no new paths are allowed and no preferences between paths have changed. Moreover, since the e-brother converges to a unique final configuration, S will converge to the unique analogous final configuration. Because the argument above can be repeated for each edge e, the whole SYPC is guaranteed to converge to a unique final configuration.

■

A.3 YPC Path Diversity Guarantees

In this section, we proof theorem 1 that we also call a path diversity guarantee. Our path diversity guarantee for YPC follows from a feature of widest-advertisement policies. We first state the feature in a lemma below and proof that BGP (SPVP) has this feature. Then, we apply this feature to YPC.

Recall that in section A.1.2 we proved two lemmas (1, 2) partitioning AS's peers into classes. We now introduce two new notions: a *nicest possible class* and a *nice path*.

Definition 2 *Consider an instance Z of SPVP (or SYPC) with no dispute wheels and each AS following next-hop and widest-advertisement policies. For each AS A in Z, let C be a set of classes c such that there exists a path $p = [A, A_{n-1}, A_{n-2}, \ldots, A_0]$*

such that p is policy-compliant and $A_{n-1} \in c$. *Then, the* nicest possible class *for A is the most preferred class in C.*

Definition 3 *A path* $p = [A_n, A_{n-1}, \ldots, A_0]$ *is a* nice path *of* A_n *if p is policy-compliant and* A_{n-1} *is a peer in* A_n*'s nicest possible class.*

Lemma 3 *Let Z be a converged instance of SPVP with no dispute wheels and each AS following next-hop and widest-advertisement policies. Then, for each AS A, if there exists a policy-compliant path from A to the destination, then A is connected in Z through a nice path.*

Proof: Note that this lemma actually makes two separate points and it can be of independent interest because it is about BGP. First, each AS that can possibly be connected (there is a policy-compliant path from it to the destination) will be connected. Second, it will actually be connected through a nice path.

First, note that nodes that have no policy-compliant paths to the destination, will obviously be isolated and won't affect any of the formed paths. Thus, without loss of generality we can assume that there are no such nodes.

We first show the following sublemma. Let A be an AS that is not connected through a nice path in Z (from now on the specification "in Z" is assumed and we don't write it explicitly). Let $p = [A, A_{n-1}, A_{n-2}, \ldots, A_0]$ be a nice path of A. Then, at least one of $A_{n-1}, A_{n-2}, \ldots, A_1$ is not connected through a nice path.

Assume the contrary, that all of the intermediate nodes have a nice path. There are two possible cases: either A is disconnected or A is connected. If A is disconnected n has to be at least 2. Consider A_{n-1}. By assumption, A_{n-1} is connected through a nice path. Because A_{n-1} is willing to advertise a path through A_{n-2} to A and it is connected through a nice path, A_{n-1} has to be willing to advertise its current path to A. Thus, A has to be connected. This contradiction proves the sublemma in the case that A is disconnected. Next, we consider the case when A is connected.

In the case that A is connected, we show that the paths of $A_{n-1}, A_{n-2}, \ldots, A_0$ has to go through A, which is a contradiction because the path of A_0 is $[A_0]$. Let p_X^n and p_X^c denote a nice and the current path of X, respectively. Further, let $p(Y)$ denote

the suffix of path p starting at Y. First, note that since all A_i's have a nice path, they all must be connected. If $p^c_{A_{n-1}}$ does not go through A, A_{n-1} has to advertise $p^c_{A_{n-1}}$ to A because it is a nice path, because A_{n-1} is willing to advertise $p(A_{n-1})$ to A, and because we assume widest-advertisement policies. But then, A would have a nice path. Therefore, $p^c_{A_{n-1}}$ has to go through A.

Assume $p^c_{A_{n-2}}$ does not go through A. Because it does not go through A, it does not go through A_{n-1}. Then, A_{n-2} is advertising $p^c_{A_{n-2}}$ to A_{n-1}. Because $p^c_{A_{n-2}} \neq p^c_{A_{n-1}}(A_{n-2})$ (one goes through A and one does not) and because A_{n-1} current path is $p^c_{A_{n-1}}$, $\lambda^{A_{n-1}}([A_{n-1}]p^c_{A_{n-2}}) < \lambda^{A_{n-1}}(p^c_{A_{n-1}})$. Recall that because A's current path is not nice, $\lambda^A([A, A_{n-1}]p^c_{A_{n-2}}) > \lambda^A(p^c_A)$. These four paths and nodes A and A_{n-1} form a dispute wheel, which we assumed does not exist. Thus, $p^c_{A_{n-2}}$ goes through A.

The argument above can be repeated inductively. At the step for A_{n-i}, the following holds:

$$\lambda^{A_{n-i+1}}([A_{n-i+1}]p^c_{A_{n-i}}) < \lambda^{A_{n-i+1}}(p^c_{A_{n-i+1}})$$
$$\lambda^A(p^c_A) < \lambda^A([A, A_{n-1}, \ldots, A_{n-i+1}]p^c_{A_{n-i}})$$

and these four paths together with A and A_{n-i+1} form a dispute wheel. This finishes the proof of the sublemma. Next, we prove the lemma.

Assume the contrary, that there exists an AS A_0 whose path is not nice. Then, pick a nice path $p_1 = [A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ from A_0 to the destination. On this path, pick a node B_i, $0 \leq i \leq n-1$ such that

- B_i's path is not nice
- Path $[B_i, B_{i-1}, \ldots, B_0]$ is not a nice path of B_i.

Such B_i can be found in the following way. By the sublemma, there is an AS B_j whose path is not nice. If for this B_j, the path $[B_j, B_{j-1}, \ldots, B_0]$ is nice, we can apply the sublemma to B_j and its nice path $[B_j, B_{j-1}, \ldots, B_0]$, to find a node B_k, $k < j$, whose path is not nice. If the path $[B_k, B_{k-1}, \ldots, B_0]$ is a nice path of B_k, we can continue analogously. Because the original path $[A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ is finite and on each iteration it gets smaller, this process cannot continue forever. Therefore, there exists such a B_i. Rename it to A_1.

Define $A_2, A_3, \ldots$ analogously until some A_q is not the same as a previously found A_r. Without loss of generality, assume that $r = 0$ (because we could have started at A_r). Let p_i be the nice path of A_i we used to find A_{i+1}. Interpret the subscripts module q and let R_i be the prefix of the path p_i until and including A_{i+1}. Let Q_i be the suffix of the path p_{i-1} starting from and including A_i. Let $R = R_0, R_1, \ldots, R_{q-1}$, $Q = Q_0, Q_1, \ldots, Q_{q-1}$, and $A = A_0, A_1, \ldots, A_{q-1}$. Then, $W = (A, Q, R)$ is a dispute wheel. This contradiction, proves the lemma. ■

We are now ready to proof theorem 1.

Proof of Theorem 1: For the proof, we will use the e-brother defined in the proof of theorem 2. While constructing an instance of e-brother from an instance of SYPC involves many details, here we consider only converged instances of SYPC and e-brother and many of the details become irrelevant. The only details we need are the structural and policy changes. These changes are described in the caption of figure A.2.

In the proof of theorem 2, we have already noted that the structural and policy changes in e-brother cannot obviously introduce a dispute wheel, because they merely cut down on policy-compliant paths. We now show a lemma that if all ASes in an instance S of SYPC follow widest-advertisement and next-hop policies, then all ASes in S's e-brother follow widest-advertisement and next-hop policies.

At the high-level widest-advertisement policies say that if an AS A is willing to advertise some path to its peer B, than A has to advertise a paths to B under some conditions. Because the changes in e-brother are that some ASes never advertise any path to some other ASes (deletion of edge e is equivalent to the ASes at e ends not advertising anything to each other), the "if" clause of the definition never happens for these pair of ASes. For other pairs of ASes, e-brother does not change their interactions at all. Therefore, widest-advertisement policies are preserved in e-brother.

The preservation of next-hop policies is obvious. Given an AS A and its path p, in S, A would advertise p to a set of peers determined by p's next-hop. In e-brother, A advertises p to the same set of peers, possibly minus some peers to whom A does

not advertise anything. Thus, the set of peers to whom p is advertised in e-brother is also determined just by the next-hop of p. This completes the proof of the lemma. This lemma allows us to apply lemma 3 to e-brother.

First, we show that if BGP gives A a path p after failure of e, then there is a policy-compliant path from A to the destination in the e-brother. This is not immediately obvious because p itself might not be a policy-compliant path in e-brother. To show the existence of such a path q, we construct it based on p in the following way.

If $A \notin T^e$, then A's default path in e-brother does not go through e and is obviously a policy-compliant path. In this case, q is A's default path. If $A \in T^e$, let B be the last AS along p starting from the destination that is not in T^e. Let C be the AS after B. By definition, $C \in T^e$. Then, $q = st$ is a concatenation of two paths, where s is the prefix of p from A to B, and t is B's default path (there are actually no "alternate" paths in e-brother, we still use the qualifier "default" to indicate that the path is the same as the default path in SYPC). We only need to show that st is policy-compliant. Because we assumed next-hop policies, to show that st is policy-compliant we only need to show that B is willing to advertise t to C. By lemma 3, B's default path exists and, moreover, is a B's nice path (in SPVP, not only in e-brother). Therefore, B has to be willing to advertise t to C.

Thus, we have showed that there exists a policy-compliant path q in e-brother from A to the destination. By existence of q and lemma 3, node A will have a path p_A in the e-brother's converged state. Because paths in e-brother correspond one-to-one to the e-avoiding path of YPC, p_A is the e-avoiding path that A has in YPC before the failure of e, as desired. ■

A.4 Hiding Convergence

In the paper, we introduced the hiding technique and applied it to BGP and YPC. We now define a formal model for hiding and prove its convergence properties. Also, the description of hiding in the paper contained many details. The model we define and study here strips many of the details exposing the core of hiding. We hope that

the core hiding model can be of independent interest.

A.4.1 HPV Definiton

We first define the formal model for hiding, which we call Hiding Path Vector (HPV). HPV and the framework around it are based on SPVP [32], where from we borrow most of our definitions. We repeat them here for completeness of presentation.

The HPV algorithm is defined over an undirected graph $G = (V, E)$, where $V = \{0, 1, 2, \ldots, n\}$ is a set of vertices (also called *nodes* and E is a set of edges. Vertex 0 is a special destination vertex. Each edge in E represents two reliable FIFO message queues - one in each direction. For a vertex v, we denote the set of v'th neighboring vertices by $N(v)$.

Definition 4 Path $P = [u_k, u_{k-1}, \ldots, u_1, u_0]$ *is a sequence of nodes* $u_i \in V$ *such that* $(u_i, u_{i-1}) \in E$. *There is a special* empty path *denoted by* ϵ. *Path* P *is called* simple *if all of its nodes are pairwise different. Nodes* u_k, u_{k-1} *and* u_0 *are called the* first node, *the* next-hop, *and the* last node *of* P, *respectively. Paths* $P = [u_k, \ldots, u_0]$ *and* $Q = [v_m, \ldots, v_0]$ *can be* concatenated *into a path* $PQ = [u_k, \ldots, u_0, v_{m-1}, \ldots, v_0]$ *if* $u_0 = v_m$. *Concatenation with the empty path* ϵ *is the identity operation.*

Definition 5 *For each node* $v \in V$, *a set of simple paths* P^v *such that* $\epsilon \in P^v$ *is a set of* permitted paths *at node* v. *We also require that* $P^0 = \{[0]\}$. $\mathbb{P}$ *is the set of all sets* P^v.

Note that sets P^v let us model several different characteristics of BGP. First, we model both import and export filters using permitted path sets. Modeling of import filters is obvious - paths that are rejected by the import filters are not permitted. Modeling of export filters is done in the following way: the case when node u does not export path P to its neighbor v, is modeled by not including path P in P^v. Second, we model the loop detection of BGP by including only simple paths in permitted path sets.

Definition 6 *Each node* $v \in V$ *has a* ranking function λ^v *that assigns a non-negative number (preference) for each path in* P^v. *The higher the value of* $\lambda^v(P)$, *the higher is*

v'th preference for P. Furthermore, for all $v \in V$*, we require that* $\lambda^v(\epsilon) = 0$ *and for* $P \in P^v$*,* $P \neq \epsilon$*,* $\lambda^v(\epsilon) > 0$*. Finally, we assume that for* $P_1 \neq P_2$*,* $\lambda^v(P_1) \neq \lambda^v(P_2)$*. A weaker form of this injectivity assumption is sufficient for the proofs, but since this assumption is true for BGP, we might as well assume this strong version.* Λ *denotes the set of all* λ^v

Each node in HPV has two data structures: `rib` - corresponding to Loc-RIB in BGP specification, and `rib_in` - corresponding to the Adj-RIB-In in BGP specification. Like BGP, $\texttt{rib}(u)$ always contains the most preferred path among the paths in $\{[u,v]P : v \in N(u), P = \texttt{rib\_in}_u(v)\}$. We denote this most preferred path by $best(u)$. Unlike BGP, the $\texttt{rib\_in}_u(v)$ in can contain not only the last path received by u from v, but also the last permitted path (in P^u) received by u from v or the empty path ϵ. The case when $\texttt{rib\_in}_u(v)$ contains last path received by u from v corresponds to the regular BGP-like situation. The case when $\texttt{rib\_in}_u(v)$ contains the last permitted path received by u from v corresponds to the case when u is hiding. The case when $\texttt{rib\_in}_u(v)$ contains the empty path corresponds to the case when u was hiding, stopped, and have not yet received a permitted path from v.

As in [32] and [30], to model the distributed nature of HPV we introduce a notion of an *activation sequence*. Activation sequence specifies when which node does what. Different activation sequences model different execution orders of the true distributed version of HPV. HPV executes by processing one activator from the sequence at a time. We say that i'th activator is processed at time i. Each activator in an activation sequence is tuple $a = (u, v, t)$ where u is the node being activated, v is a neighbor of u towards which u is activated, and t is the type of the activator. HPV has two types of activators: a *regular activator* and a *revealing activator*, which are handled with algorithms 5 and 6. Each algorithm is run atomically.

Regular activator for u towards v is handled (algorithm 5) by first removing a pending message from v (if there is no message the activator results in a nop) containing a path P. If P is not a permitted path, the current path in the $\texttt{rib\_in}_u(v)$ is marked lame and P is discarded. If P is permitted, it is handled in the standard BGP fashion - it is put into $\texttt{rib\_in}_u(v)$; the current best path is computed; if the

Algorithm 5: Algorithm for node u when it is activated by a regular activator towards v

```
if there is a pending message m from v to u then
    remove m from the link queue
    P := path in m
    if P ∈ P^u then
        rib_in_u(v) := P
        if rib(u) ≠ best(u) then
            rib(u) := best(u)
            foreach w ∈ N(u) do
                send rib(u) to w
            end
        end
    else
        mark rib_in_u(v) as lame
    end
end
```

current best path is not in the $\texttt{rib}(u)$, $\texttt{rib}(u)$ is updated and the change is sent to the neighbors. Thus, regular activators are essentially the same as the activators in [32], with the only difference that a non-permitted path is not put into `rib_in`, which is marked as lame instead.

Revealing activator for u towards v is a nop if $\texttt{rib\_in}_u(v)$ is not lame. If $\texttt{rib\_in}_u(v)$ is lame, the lame path is deleted (by setting $\texttt{rib\_in}_u(v)$ to the empty path) and the standard BGP path selection is carried out. Revealing activator essentially brings the $\texttt{rib\_in}_u(v)$ into a state that is equivalent to the one SPVP would have brought it into.

We call an activation sequence *fair* if for each pair of neighboring nodes (u, v) it contains infinitely many regular activators $(u, v, "regular")$. In our presentation, all activation sequences are assumed to be fair.

In a truly distributed HPV, regular activators correspond to an arrival of a message

Algorithm 6: Algorithm for node u when it is activated by a revealing activator towards v

```
if rib_in_u(v) is lame then
    rib_in_u(v) = ε
    if rib(u) ≠ best(u) then
        rib(u) := best(u)
        foreach w ∈ N(u) do
            send rib(u) to w
        end
    end
end
```

from a neighbor, while revealing activators correspond to the node deciding to stop hiding. This decision can be caused by many factors such as a reception of a token (described earlier) or an expiration of a timer. These factors are intentionally left out of the model. The fact that our convergence result for HPV is valid for arbitrary activation sequences shows that it is safe for operators to stop hiding any path at any time.

The last piece that we need to formally talk about HPV is the state consistency. The state of HPV is said to be *consistent* if all of the following hold

1. For all $u \in V$, $\texttt{rib}(u) = best(u)$, i.e. $\texttt{rib}(u)$ contains the best possible path given the values of $\texttt{rib\_in}$'s.

2. For each pair (u, v) of neighboring nodes, if the link from v to u is not empty, the last message in this link contains the path in $\texttt{rib}(v)$.

3. For each pair (u, v) of neighboring nodes, if the link from v to u is empty, then one of the following is true

   (a) $\texttt{rib}(v) \in P^u$ and $\texttt{rib\_in}_u(v) = \texttt{rib}(v)$

   (b) $\texttt{rib}(v) \notin P^u$ and $\texttt{rib\_in}_u(v) = \epsilon$

(c) $\texttt{rib}(v) \notin P^u$ and $\texttt{rib\_in}_u(v)$ contains the last permitted path that v sent to u.

Throughout the paper we assume that the initial state is consistent. It is easy to check that any activation sequence (not necessarily fair) takes the system in consistent state into consistent state.

Thus, we have finished defining HPV. Its inputs are a graph, a collection of ranking functions, a collection of permitted path sets, an activation sequence, and an initial state. This 5-tuple of inputs defines an instance of HPV.

A.4.2 Dispute Wheels

Next, we restate the definition of a dispute wheel from [32].

Definition 7 *A sequence of nodes $U = u_0, u_1, \ldots, u_{k-1}$ together with two sequences of nonempty paths $Q = Q_0, Q_1, \ldots, Q_{k-1}$, $R = R_0, R_1, \ldots, R_{k-1}$ constitute a* dispute wheel *$W = (U, Q, R)$ if*

1. *R_i is a path from u_i to u_{i+1}*
2. *$Q_i \in P^{u_i}$*
3. *$R_i Q_{i+1} \in P^{u_i}$*
4. *$\lambda^{u_i}(Q_i) < \lambda^{u_i}(R_i Q_{i+1})$*

where all subscripts are to be interpreted modulo k. For an illustration of a dispute wheel see figure 9.a of [32].

A.4.3 HPV Convergence

Given an instance of HPV, we say that $\texttt{rib\_in}_u(v)$ converges if $\texttt{rib\_in}_u(v)$ does not change after some time t_0. We say that a node u converges if $\texttt{rib}(u)$ does not change after some time t_0. Finally, we say that HPV converges is all nodes converge. Next, we state the main theorem, discuss why we choose to prove this theorem, prove a number of lemmas, and finally prove the theorem.

Theorem 7 *An instance of HPV converges if it does not have a dispute wheel.*

Even through the analogous result for BGP convergence is not the sate of the art ([20]), it is well-known and relatively simple result. At the same time, it is powerful enough to guarantee convergence in at least two important classes of policies - the customer-peer-provider policies and the generalized shortest-path based policies. Finally, this result is particularly appealing because the proof exposes the effects of hiding on the dynamics of the model.

Let $value(u)$ be a set of paths that node u picks infinitely many times. C is the set of nodes that converge (whose $value()$ has a single path), R is the set of converged `rib_in`'s, and O is the set of oscillating nodes (whose $value()$ has at least two paths). It is obvious that O and C are disjoined and cover V.

Lemma 4 *Given nodes u and v, if $\texttt{rib\_in}_u(v) \notin R$, then $v \in O$.*

Proof: We prove the contrapositive - if $v \in C$, then $\texttt{rib\_in}_u(v) \in R$. By definition, because $v \in C$, $\texttt{rib}(v)$ does not change after some time t_0. Because $\texttt{rib}(v)$ does not change after t_0, v does not send any messages to u after t_0. Thus, the link from v to u is always empty after some time t_1 (because we always assume that the activation sequence is fair). Then, regular activators for u towards v in a nop after t_1 and the $\texttt{rib\_in}_u(v)$ cannot change during regular activator processing after t_1. Moreover, $\texttt{rib\_in}_u(v)$ can change at most once during a revealing activator processing after t_1. Thus, there is a time after which $\texttt{rib\_in}_u(v)$ does not change, i.e. $\texttt{rib\_in}_u(v) \in R$ as desired. ■

Lemma 5 *If $u_0 \in O$ and $P_0 = [u_0, u_1, \ldots, u_{k-1}, u_k = 0] \in values(u_0)$, then for some $0 \leq i \leq k-1$, $\texttt{rib\_in}_{u_i}(u_{i+1}) \in R$. In other words, there is a convergent `rib_in` along P_0.*

Proof: If $\texttt{rib\_in}_{u_0}(u_1) \in R$, we are done. Assume $\texttt{rib\_in}_{u_0}(u_1) \notin R$. Then, by lemma 4, $u_1 \in O$. We next show that $P_1 = [u_1, \ldots, u_{k-1}, u_k = 0] \in values(u_1)$

First, because $P_0 \in values(u_0)$ and $\texttt{rib\_in}_{u_0}(u_1) \notin R$, P_1 has to appear in (and disappear from) $\texttt{rib\_in}_{u_0}(u_1)$ infinitely many times. The only way for P_1 to appear in

$\texttt{rib\_in}_{u_0}(u_1)$ replacing another path is for u_0 to process a message containing P_1 from u_1. Therefore, u_1 sends P_1 infinitely many times, which implies that $P_1 \in values(u_1)$.

Thus, we have showed that if $\texttt{rib\_in}_{u_0}(u_1) \notin R$, then $u_1 \in O$ and $P_1 \in values(u_1)$. Applying the same argument, we can show that if $\texttt{rib\_in}_{u_1}(u_2) \notin R$, then $u_2 \in O$ and $P_2 \in values(u_2)$. Continuing analogously, if no `rib_in` along the path converges, then $u_k = 0$ has to be oscillating, which is impossible. Thus, for some $0 \leq i \leq k-1$, $\texttt{rib\_in}_{u_i}(u_{i+1}) \in R$. ■

Proof of Theorem 7: We prove the contrapositive statement - if an instance of HPV does not converge, there is a dispute wheel. Since HPV does not converge, O is nonempty. Let $u \in O$ and $P \in values(u)$. Then, by lemma 5, there is a convergent `rib_in` along P. Let u_0 be the first node along P, whose `rib_in` from the downstream node is convergent. Then, $u_0 \in O$ because otherwise the node upstream of u_0 would have a convergent `rib_in` from u_0 (and hence u_0 would not be the first node with convergent downstream `rib_in`).

Let H_0 be a path starting at u_0 such that $P = [u, \ldots, u_0]H_0$. Because, the downstream `rib_in` from u_0 is convergent, the path H_0 is always available to u_0 after some point in time. Because H_0 is always available and u_0 is an oscillating node, H_0 has to be the least preferred path by u_0 from among all the paths in $values(u_0)$. Let $J_0 \in values(u_0)$ be a more preferred path than H_0.

Now, we have an oscillating node u_0 and a path $J_0 \in values(u_0)$. Using the same argument as we did for u and $P \in values(u)$, we can find an oscillating node u_1 and a path $J_2 \in values(u_1)$. Continuing in this fashion, for some value of k, a newly found u_k will be equal to an already found u_j. Notice that an oscillating node can have at most one path it picks infinitely many times that comes from a convergent `rib_in` - the least preferred path in $values()$ of this node. Therefore, $H_k = H_j$.

Now, lets rename u_i to a_{i-j} and H_i to Q_{i-j} for $i = j, j+1, \ldots, k-1$. Also, for $i = 0, 1, \ldots, k-j-1$, let R_i be such a path that $J_{j+i} = R_i H_{j+i+1}$. In other words, R_i is the prefix of J_{j+i} until and including u_{j+i+1}. Finally, let $m = k-j-1$. With these definitions, we now have a dispute wheel, $W = (A, Q, R)$, where $A = a_0, a_1, \ldots, a_m$, $Q = Q_0, Q_1, \ldots, Q_m$, $R = R_0, R_1, \ldots, R_m$. The first three conditions for the dispute

wheel in definition 7 are obviously satisfied by construction. The last condition holds because J_i was picked to be a more preferred path than H_i for $i = 0, 1, \ldots, k-1$, and because $H_k = H_j$. This completes the proof. ■

In this section, we defined a HPV and proved that it converges under any fair activation sequence as long as the policies don't contain dispute wheels. HPV can be viewed as an application of the hiding technique to SPVP. The hiding technique simply says that when a node's peer withdraws a path from it, the node can pretend that it continues to have the path until the peer announces another path or until the node decides to stop pretending. An intuitive reason why hiding does not affect convergence of SPVP is because hiding does not introduce anything new into the dynamics of SPVP - it simply *delays* the processing of a withdrawal. SPVP processes the withdrawal immediately, while HPV processes the withdrawal when it decides to (or never if the peer sends a new path before the node processes the withdrawal). In some sense, the fact that SPVP converges under any activation sequence means that SPVP's dynamics are invariant under the timing of events. Thus, it seems reasonable that if the timing of withdrawal processing is made variable (what hiding does to SPVP), SPVP will still converge.

In the next section, we show how the convergence result for HPV can be applied to YAMR to prove its convergence. To avoid boring the reader with pages of formal details, we present only the main constructs and arguments of the proof.

A.4.4 HSYPC Convergence

In the previous sections we defined SYPC to be the formal model for YPC. SYPC for YPC is what SPVP is for BGP. In this section, we talk about HSYPC, which is the result of applying the hiding technique to SYPC, just like HPV was the result of applying the hiding technique to SPVP. Throughout the discussion, we consider an instance of HSYPC, Z, with no dispute wheels and a consistent initial configuration. We assume that there are no link events and policy changes after some moment. Our goal is to prove that HSYPC converges. Note that HSYPC is a simplified model YAMR. Thus, in this section we come very close to proving that YAMR converges

(i.e. proving theorem 3). In the next section, we actually prove theorem 3.

First of all, the default paths of HSYPC are constructed in exactly the same way as the paths of HPV. Therefore, by theorem 7, the default paths of HSYPC converge.

After convergence of default paths, the dynamics of paths with a given label are completely independent from the dynamics of other labels. Therefore, it is sufficient to show that paths with a given label e converge.

Let S^e be the set of nodes whose (converged) default paths contain e. Let S be the set of all nodes whose default paths don't contain e. If Z contains nodes that don't have default paths, these nodes are completely isolated and we can assume there are no such nodes without loss of generality. Therefore, $S \cup S^e = V$, where V is the set of all nodes as usually. Using the notation from the previous section, we can note that $S \subseteq C$ and $O \subseteq S^e$.

Note that because of hiding the set S^e does not have to be a tree. However, similar arguments that we used in the previous section apply here as well. For the remainder of a section, when we talk about the path of a node u, we mean u's default path if $u \in S$ and u's e-labeled path if $u \in S^e$. Further, when we talk about $\texttt{rib\_in}_u(v)$, we mean the default path in $\texttt{rib\_in}_u(v)$ if $v \in S$ and the e-labeled path in $\texttt{rib\_in}_u(v)$ if $v \in S^e$. In other words, $\texttt{rib\_in}_u(v)$ contains the path that we are concerned with for node v.

Thus, for each node we have at most one path that we are concerned with and there is a single path in $\texttt{rib\_in}_u(v)$ for each connected ordered pair (u, v) of nodes that we are concerned with. The interdependence of these paths is not the same as the paths in HPV - the paths at different nodes affect each other as was described in the YAMR's design. However, the dynamics are sufficiently similar to HPV that we can use the same notation and the same arguments.

Lemma 6 *Given nodes u and v, if $\texttt{rib\_in}_u(v) \notin R$, then $v \in O$.*

Proof: The meaning of this lemma in the new context is somewhat different from the meaning of lemma 4. However, the unchanged proof of lemma 4 proves this lemma as well. ■

Lemma 7 *If $u_0 \in O$ and $P_0 = [u_0, u_1, \ldots, u_{k-1},$ $u_k = 0] \in values(u_0)$, then for some $0 \leq i \leq k-1$, $\texttt{rib\_in}_{u_i}(u_{i+1}) \in R$. In other words, there is a convergent* `rib_in` *along P_0.*

Proof: First of all, $u_0 \in S^e$ because $u_0 \in O$. If for any i, $0 \leq i \leq k$, $u_i \in S$, the statement is obvious. Thus, we can assume that for all i $u_i \in S^e$. Then, the proof of lemma 5 applies here unchanged. ■

At this point, we have repeat the proof of theorem 7 verbatim (but obviously using the new meanings for paths and `rib_in`s) to show that e-labeled paths converge. As mentioned earlier, repeating the same argument for all labels, show that HSYPC converges when there are no dispute wheels.

A.4.5 YAMR Convergence

In this section, we finally prove theorem 3. We restate it here in a more rigorous way.

Theorem 8 *Consider an instance Z of YAMR with consistent initial configuration, no dispute wheels, and all ASes following widest-advertisement and next-hop policies. Assume there are no policy changes and link events after time t. Then, there exists a time $t' > t$ after which no update messages are sent, no tokens are sent, and no state changes occur. In other words, YAMR converges completely.*

Proof: Consider an instance X of HSYPC that corresponds to Z. In the previous section, we showed that X converges. This implies that after some time t_0 no update messages are sent and no state changes occur Z. Therefore, we only need to show that no tokens are sent after some time.

Before showing that tokens cease to be sent, we need to realize that no state changes after t_0 implies that after t_0 no AS changes what it hides. In other words, no AS can decide to stop hiding some path after t_0 by definition of t_0. ASes have complete freedom deciding what to hide and when to stop hiding, but whatever they do, they cannot continue forever as was shown in the previous section. Thus, there exists a time, t_0, after which no hiding changes occurs. This logical point that gives

complete freedom to ASes and yet claims that all ASes stop after some point can be hard to understand. We further illustrate it with an example.

Consider a finite set of integers C. Then, in any infinite sequence S of integers from C, there exists an index after which S contains only integers that appear in it infinitely many times. Think of constructing a sequence S_0 that will contradict this statement. We have complete freedom to choose how many elements of C will appear in the sequence finitely many times. Further, we have complete freedom to choose how many times each of these elements will appear in S_0. Even further, we can put each occurrence of each of these elements as far in the sequence as we want. Yet, despite all these freedoms, there is an index after which there is none of our elements. The convergence of state in hiding has the same logic. ASes have a lot of freedom, but after some point they all stop. Next, we show that loop and disconnection tokens stop.

Recall that loop detection tokens are sent only at the end of path selection. Path selection can only be triggered by an update message or by a decision to stop hiding (because of a reception of another token or inability to find a deflection path). We know that no update messages are sent after t_0. We also argued above that no AS can decide to stop hiding after t_0. Therefore, no loop detection tokens can be sent after some time $t_1 > t_0$ when all the updates message sent before t_0 have been processed (no new ones can be sent after t_0).

Lastly, we show that disconnection tokens cannot be sent forever. Assume the contrary that there exists an AS A that sends disconnection tokens forever. In particular, A sends tokens after t_1 when all the state has converged and no update messages are sent. Because A sends disconnection tokens A must choose a loopy path. Because the state has converged and the control path is loopy, the forwarding path corresponding to it has to contain a hiding AS B. Upon reception of A's disconnection token B has to stop hiding, thus changing its state. This contradiction completes the proof. ■

A.5 Hiding Loop-Freeness

In this section, we prove theorem 4 that guarantees loop-freeness of YAMR (and HBGP as a special case).

As before, we fix a destination prefix. We say AS A *has an e-labeled path* if A's `rib` contains an e-labeled path (even if this path is lame). We also introduce the notion of label L's *forwarding path* starting at node A. Given a node A and a label L, the forwarding path of L starting at A is the path on which a packet leaving A with label L will travel. The label can change along the forwarding path. The forwarding path can contain a loop, in which case it is infinite. Next, we restate theorem 4 more rigorously and prove it.

Theorem 9 *Assume after some time t_0, there are no link events and policy changes for a sufficiently long time that the network converges at time t_1. Then, if an AS has an L-labeled path at time t_1, the forwarding path of L starting at the AS is finite and ends at the destination. In other words, if the AS sends a packet with label L the packet will reach the destination (ignoring practicalities like data corruption).*

Proof: Assume the contrary, that there is an AS that has a path for a label but whose forwarding path for this label does not reach the destination. Note that since the network is in the converged state at time t_1, the forwarding path cannot be finite and not end at the destination. If the forwarding path ends at a node X that is not the destination and the previous nodes is Y, then Y forwarding table is not consistent with X RIB, which is impossible in the converged state. Thus, the forwarding path has to be infinite and hence contain a loop.

Let a *supernode* (A, L) be a pair, where A is a node and L is a label. Intuitively, a supernode (A, L) is a node A and its path for L. It is useful because we can talk about *the* path of a supernode and *the* forwarding entry of the supernode without specifying the label. Using supernodes, we can represent the loop F in the forwarding path as a sequence of supernodes $[(A_0, L_0), \ldots, (A_{k-1}, L_{k-1}), (A_k, L_k) = (A_0, L_0)]$, where each supernode (A_i, L_i) represents the fact that packet sent on the forwarding path arrives at node A_i with label L_i. Whenever we use indices of supernodes in F, they are

to be interpreted module k. We call a supernode (A_i, L_i) a *hiding supernode* if the path for label L_i at node A_i is lame. We say that a supernode (A_i, L_i) sent a loop token we mean that node A_i sent a token along its L_i-labeled path. We say that a supernode (A_i, L_i) changed its forwarding state if the next-hop supernode of (A_i, L_i) has changed. Note that if the label changed the next-hop node can be the same.

Because the network state has converged and there is a loop F in the forwarding path, F has to contain a hiding supernode. Without loss of generality let (A_0, L_0) be the last hiding supernode in F that sent a loop token T, say at time t_2 (if there are multiple such supernodes, let (A_0, L_0) be an arbitrary one of them). Next, we show that there exists a supernode in F that changed its forwarding state after it processed T (not necessarily because of T).

Recall that a hiding supernode always sends a token after it changes its forwarding state. Because T is the last token (A_0, L_0) sent, it did not change its forwarding state after sending T and T was sent to (A_1, L_1). Now, if there is no supernode in F that changed its forwarding state after processing T, then the token must have traveled around F and must have come back to (A_0, L_0). (A_0, L_0) would then have changed its forwarding contrary to the fact above. Thus, there is a supernode that changed its forwarding after processing T, say at time t_3. Let (A_j, L_j) be the supernode that changed its forwarding last among all the supernodes in F, say at time t_4 (if there are multiple such supernodes, choose one randomly). Then, $t_4 \geq t_3 > t_2$ and hence there are no tokens sent by supernodes in F after t_4.

Because (A_j, L_j) did not send a token after updating its forwarding state at time t_4, it is not a hiding supernode. Therefore, it must have sent an update to (A_{j-1}, L_{j-1}) because no forwarding changes happen after t_4. Because (A_{j-1}, L_{j-1}) also did not send a token after receiving this update, it is not a hiding supernode and it must have sent an update to (A_{j-2}, L_{j-2}) (because no forwarding changes happen after t_4). We can continue this argument until we conclude that (A_0, L_0) is not a hiding supernode, which contradicts our choice of (A_0, L_0). This contradiction completes the proof. ■

A.6 Hiding Connectivity

In this section, we formally prove theorem 5. Note that it is enough to prove that an AS that can be connected is connected though its default path. Thus, it is enough to prove theorem 5 for HBGP. We start with some definitions:

- *control path* - a path that the AS has in its RIB or RIB_IN. Control path can be lame. Then, it has a deflection path. For an AS A, we denote, its control path by c_A. By $c_A(u)$ we denote the control path of A were it to choose the peer u's control path as best.

- *forwarding path* - a path that the packets actually travel from an AS. For an AS A, we denote, its forwarding path by f_A. By $f_A(u)$ we denote the forwarding path of A were it to forward through peer u. Note that $f_A(u)$ can be infinite if the packet comes back to A.

- Given a forwarding path p, the corresponding control paths is denoted by $ctrl(p)$. Given a control path p, the corresponding forwarding paths is denoted by $fwd(p)$

- *nice forwarding path* - like nice path but referring specifically to a forwarding path.

- *class(p)* - is a class of a path p, which is the class of the next-hop peer in p.

- *fine forwarding class* c of some AS A is defined as follows. Let U be the set of all peers of A that announce a path to A (including peers that announce a loopy path to A). Then c is the highest class in the set $\{class(f_A(u)) : u \in U,\ f_A(u)\ is\ finite\}$. If the set is empty, we say that c is null.

- *fine forwarding path* p is a path whose class is the fine forwarding class.

- A path advertised by a peer u to A is called *misaligned* if it contains A but the suffix of u starting at A is not the same as any of A's paths from which it could have originated. In the case of HBGP, there is a single originating path

- the path in the RIB. In the case of YAMR, a default path can only originate from another default path and an alternate path can originate from the default path and from a labeled path with the same label. We also say a *misaligned* advertisement if it contains a misaligned path. If a path is not misaligned, we say it is *aligned*.

Notice that for ASes that don't hide, the next-hop ASes for corresponding forwarding and control paths are the same. Therefore, a control path is nice if and only if the forwarding path is nice. Thus, we can simply talk about a *nice path* without specifying which we mean. Also, we can talk about the *next-hop* without specifying on which path the next-hop is.

Lemma 8 *Let Z be an instance of HBGP with no dispute wheels and all ASes following widest-advertisement and next-hop policies. Assume that there has been no policy changes and link events for a long enough time that Z has converged. Then, for each AS A, f_A is in the fine forwarding class of A. In particular, if f_A is null, the fine forwarding class is null.*

Proof: First, consider the case when f_A does not exist. If A does not forward anywhere, either no peer advertises a path to A or there is a path advertised to A that A cannot forward on (if there are multiple such paths, choose the most preferred one). In the first case, we have nothing to show. In the second case, the only reason A cannot forward on a path advertised to it is that this path is loopy. However, because A does not have any path in its RIB, this loopy path must be misaligned and A must be sending disconnection tokens along this path. This contradicts the assumption that Z has converged and concludes the case when f_A is null.

Consider the case when f_A exists. Assume the contrary that f_A is not in the fine forwarding class c of A and let the next-hop along f_A be B. If f_A is not in c, then class of f_A must necessarily be less preferred that c, by definition of c. Let f'_A be a forwarding path of A that is in c and let C be the next-hop along f'_A. Because f'_A is in the more preferred class than f_A, A would prefer any path through C to any path through B. However, since A does not prefer the path p_C advertised by

C (the control path whose corresponding forwarding path is f'_A), p_C must be loopy. Moreover, p_C has to be aligned. Indeed, if p_C were misaligned, it would be considered in A path selection, would be preferred over p_B (the control path whose corresponding forwarding path is f_A) and A would send a disconnection token contradicting our assumption that Z converged.

Because p_C goes through A but f'_A does not, there is an AS along f'_A that hides. Let D be the first AS along f'_A starting from A that hides. Then, D prefers the path $p_C(D)$ more than q - the control path corresponding to the suffix of f'_A starting at D. Now, we can identify a dispute wheel.

Let us rename some paths and nodes to illustrate the dispute wheel. Let A be u_0. Let D be u_1. Let the control path corresponding to f_A be Q_0. Let q be Q_1. Let $[A, C, \ldots, D]$, the prefix of f'_A until D be R_0. Let the prefix of $p_C(D)$ until and including A be R_1. Now, the dispute wheel is $W = (U = u_0, u_1, Q = Q_0, Q_1, R = R_0, R_1)$. Verifying that W is indeed a dispute wheel is straightforward. This contradiction finishes the proof of the lemma. ■

In the presence of hiding nodes, advertised control paths are different from the forwarding paths. Therefore, ASes don't have enough knowledge to choose the best available forwarding path. However, as the lemma 8 states, the disconnection token mechanism ensure that ASes end up with almost the best available paths. Even though this result is quite promising, under the general assumptions we have been using, the small imperfection of the mechanism can prevent some ASes from getting a nice forwarding path. To guarantee that each AS gets a nice forwarding path, we have to narrow the class of possible AS policies to those for which these imperfections cannot cause any harm. We call this class of policies *dispute circlet free* policies, or in other words, policies that don't have dispute circlets. Luckily the common customer-peer-provider policies don't have dispute circlets, if we assume that there are no cycles made entirely from provider-to-customer links.

The only difference between dispute circlets and dispute wheels is the path preference relation they use. For dispute wheels, the preference relation is the same relation that ASes use to rank paths based on the ranking function. For dispute circlets, the

preference relation R_A of an AS A is a general binary relation that is not necessarily a partial order defined as follows. Given the AS graph, the ranking functions of all ASes, an AS A, and two paths p_1 and p_2 from A to the destination, $(p_1, p_2) \in R_A$ if and only if $class(p_1) \leq class(p_2)$. If $(p_1, p_2) \in R_A$, we write it as $p_1 \succ^A p_2$.

If $\lambda^A(p_1) \geq \lambda^A(p_2)$, $class(p_1) \leq class(p_2)$ and $(p_1, p_2) \in R_A$. Therefore, all preference relations based on ranking function are included in R_A, from which it immediately follows that every dispute wheel is a dispute circlet.

Lemma 9 *If there are no dispute circlets, each AS in Z has a nice forwarding path.*

Proof: We first show the following sublemma. Let A be an AS that does not have a nice forwarding path in Z (from now on the specification "in Z" is assumed and we don't write it explicitly). Let $p = [A, A_{n-1}, A_{n-2}, \ldots, A_0]$ be a nice path from A. Then, at least one of $A_{n-1}, A_{n-2}, \ldots, A_1$ does not have a nice forwarding path.

Assume the contrary, that all of the intermediate nodes have nice forwarding paths. There are two possible cases: either A is disconnected or A is connected. If A is disconnected n has to be greater or equal to 2. Consider A_{n-1}. By assumption, A_{n-1} has a nice forwarding path. Because (1) A_{n-1} is willing to advertise a path through A_{n-2} to A (namely p), (2) A_{n-1} has a nice forwarding path, A_{n-1} has to be willing to advertise its control path to A. Therefore, the fine forwarding class of A is non-null. By lemma 8, A is connected. This contradiction proves the sublemma in the case that A is disconnected. Next, we consider the case when A is connected.

In the case that A is connected, we show that the forwarding paths of $A_{n-1}, A_{n-2}, \ldots, A_0$ has to go through A, which is a contradiction because the path of A_0 is $[A_0]$. Let p^n_X and p^c_X denote a nice and the current forwarding paths of X, respectively. Further, let $p(Y)$ denote the suffix of path p starting at Y. First, note that since all A_i's have a nice forwarding path, they all must be connected. Because A_{n-1} forwards through a nice peer, it has to advertise its control path to A. If $p^c_{A_{n-1}}$ does not go through A, A's fine forwarding class is equal to the nice forwarding class. By lemma 8, A's forwarding path is a nice forwarding path. This is a contradiction. Therefore, $p^c_{A_{n-1}}$ has to go through A.

Assume $p^c_{A_{n-2}}$ does not go through A. Because it does not go through A, it does not go through A_{n-1}. Then, A_{n-2} is advertising $p^c_{A_{n-2}}$ to A_{n-1}. Because $p^c_{A_{n-2}} \neq p^c_{A_{n-1}}(A_{n-2})$ (one goes through A and one does not) and because A_{n-1}'s current forwarding path is $p^c_{A_{n-1}}$, $[A_{n-1}]p^c_{A_{n-2}} \prec^{A_{n-1}} p^c_{A_{n-1}}$. Recall that because A's current forwarding path is not nice, $[A, A_{n-1}]p^c_{A_{n-2}} \succ^{A} p^c_A$. These four paths and nodes A and A_{n-1} form a dispute circlet, which we assumed does not exist. Thus, $p^c_{A_{n-2}}$ goes through A.

The argument above can be repeated inductively. At the step for A_{n-i}, the following holds:

$$[A_{n-i+1}]p^c_{A_{n-i}} \prec^{A_{n-i+1}} p^c_{A_{n-i+1}}$$
$$p^c_A \prec^{A} [A, A_{n-1}, \ldots, A_{n-i+1}]p^c_{A_{n-i}}$$

and these four paths together with A and A_{n-i+1} form a dispute circlet. This finishes the proof of the sublemma. Next, we prove the lemma.

Assume the contrary, that there exists an AS A_0 whose forwarding path is not nice. Then, pick a nice path $p_1 = [A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ from A_0 to the destination. On this path, pick a node B_i, $0 \leq i \leq n-1$ such that

- B_i's forwarding path is not nice
- Path $[B_i, B_{i-1}, \ldots, B_0]$ is not a nice forwarding path of B_i.

Such B_i can be found in the following way. By the sublemma, there is an AS B_j whose forwarding path is not nice. If for this B_j, the path $[B_j, B_{j-1}, \ldots, B_0]$ is nice, we can apply the sublemma to B_j and its nice path $[B_j, B_{j-1}, \ldots, B_0]$, to find a node B_k, $k < j$, whose forwarding path is not nice. If the path $[B_k, B_{k-1}, \ldots, B_0]$ is a nice forwarding path of B_k, we can continue analogously. Because the original path $[A_0, B_{n-1}, B_{n-2}, \ldots, B_0]$ is finite and on each iteration it gets smaller, this process cannot continue forever. Therefore, there exists such a B_i. Rename it to A_1.

Define $A_2, A_3, \ldots$ analogously until some A_q is not the same as a previously found A_r. Without loss of generality, assume that $r = 0$ (because we could have started at A_r). Let p_i be the nice path of A_i we used to find A_{i+1}. Interpret the subscripts

module q and let R_i be the prefix of the path p_i until and including A_{i+1}. Let Q_i be the suffix of the path p_{i-1} starting from and including A_i. Let $R = R_0, R_1, \ldots, R_{q-1}$, $Q = Q_0, Q_1, \ldots, Q_{q-1}$, and $A = A_0, A_1, \ldots, A_{q-1}$. Then, $W = (A, Q, R)$ is a dispute circlet. This contradiction, proves the lemma. ■

To show theorem 5, we simply apply lemma 9 to default path of YAMR.

A.7 Hiding Recovery

The Theorem 6 is obvious because of the failed link propagation mechanism. Recall that having a failed link information gives an AS a permission to hide the failure. When the link recovers, all failure information is guaranteed to be withdrawn from every AS that had it. When the failure information is withdrawn, all ASes stops hiding and routing returns to normal.