Pathlet Routing

P. Brighten Godfrey[†], Igor Ganichev[‡], Scott Shenker[‡§], and Ion Stoica[‡*]
[†]University of Illinois at Urbana-Champaign [‡]UC Berkeley [§]ICSI
pbg@illinois.edu, {igor,shenker,istoica}@cs.berkeley.edu

ABSTRACT

We present a new routing protocol, pathlet routing, in which networks advertise fragments of paths, called pathlets, that sources concatenate into end-to-end source routes. Intuitively, the pathlet is a highly flexible building block, capturing policy constraints as well as enabling an exponentially large number of path choices. In particular, we show that pathlet routing can emulate the policies of BGP, source routing, and several recent multipath proposals.

This flexibility lets us address two major challenges for Internet routing: scalability and source-controlled routing. When a router's routing policy has only "local" constraints, it can be represented using a small number of pathlets, leading to very small forwarding tables and many choices of routes for senders. Crucially, pathlet routing does not impose a global requirement on what style of policy is used, but rather allows multiple styles to coexist. The protocol thus supports complex routing policies while enabling and incentivizing the adoption of policies that yield small forwarding plane state and a high degree of path choice.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Packet-switching networks; C.2.2 [**Network Protocols**]: Routing Protocols; C.2.6 [**Internetworking**]: Routers

General Terms

Design, Experimentation, Performance, Reliability

1. INTRODUCTION

Challenges for interdomain routing. Interdomain routing faces several fundamental challenges. One is *scalability*: routers running the Internet's interdomain routing protocol, Border Gateway Protocol (BGP) [25], require state that scales linearly in the number of IP prefixes advertised in the Internet. This is particularly a concern in the data plane where the router stores the routing table, or forwarding information base (FIB). Because it has to operate at high speeds and often uses SRAM rather than commodity DRAM, FIB memory is arguably more constrained and expensive than other resources in a router [22]. Moreover, the number of IP prefixes is increasing at an increasing rate [15], leading to the need for expensive hardware and upgrades. The Internet Architecture Board Workshop on Routing and Addressing recently identified FIB growth as one of the key concerns for future scalability of the routing system [22].

A second challenge for interdomain routing is to provide *multipath routing*, in which a packet's source (an end host or edge router) selects its path from among multiple options. For network users, multipath routing is a solution to two important deficiencies of BGP: poor reliability [1, 14, 17] and suboptimal path quality, in terms of metrics such as latency, throughput, or loss rate [1, 27]. Sources can observe end-to-end failures and path quality and their effect on the particular application in use. If multiple paths are exposed, the end-hosts could react to these observations by switching paths much more quickly and in a more informed way than BGP's control plane, which takes minutes or tens of minutes to converge [19, 21]. For network providers, multipath routing represents a new service that can be sold. In fact, route control products exist today which dynamically select paths based on availability, performance, and cost for multi-homed edge networks [3]; exposing more flexibility in route selection could improve their effectiveness. Greater choice in routes may bring other benefits as well, such as enabling competition and encouraging "tussles" between different parties to be resolved within the protocol [6].

But providing multiple paths while respecting network owners' policies is nontrivial. BGP provides no multipath service; it selects a single path for each destination, which it installs in its FIB and advertises to its neighbors. Several multipath routing protocols have been proposed, but these have tradeoffs such as not supporting all of BGP's routing policies [32, 30], exposing only a limited number of additional paths [28], making it difficult to know which paths will be used [31, 23], or increase the size of the FIB [28, 31, 23], which would exacerbate the scalability challenge.

Our contributions. This paper addresses the challenges of scalability and multipath routing with a novel protocol called *pathlet routing.* In pathlet routing, each network advertises *pathlets*—fragments of paths represented as sequences of virtual nodes (vnodes) along which the network

*The first and fourth authors were supported in part by a Cisco Collaborative Research Initiative grant.

SIGCOMM'09, August 17–21, 2009, Barcelona, Spain.

is willing to route. A sender concatenates its selection of pathlets into a full end-to-end source route.

From this architecture come three key ideas. First, *pathlets and vnodes are highly flexible building blocks*, able to express many kinds of policies as well as enabling a dramatic amount of path choice in a clean protocol. Intuitively, like path vector routing (i.e. BGP), pathlets can be used to constrain how a packet transits an autonomous system (AS) and where it goes after it leaves. But like source routing, pathlets may be concatenated in exponentially many ways. In fact, we show that pathlet routing's data plane can emulate the routing policies of BGP, loose and strict source routing, and three recent multipath proposals: NIRA [30], MIRO [28], and LISP [9]. We are not aware of any protocol which can emulate pathlet routing's policies, although there are several that pathlet routing cannot emulate [32, 31, 23].

The second key idea is that *an AS whose policies have only "local" constraints can represent its policies using a small number of pathlets, leading to small FIBs and many allowed paths.* We suggest a new class of policies of this type, local transit (LT) policies, that allow networks to control the portions of routes which transit across their own networks, but otherwise expose the full flexibility of the Internet's AS-level routes to sources. A special case of LT policies are valley-free routes, the common export policy used in BGP. We show that this kind of LT policy has a large amount of path choice which significantly improves reliability, and has FIBs whose size scales with the number of neighbors of a router, rather than with the number of destinations—thus reducing the average number of FIB entries for the Internet AS topology by more than $10,000\times$ compared with BGP.

The third key idea is that *pathlet routing does not impose a global requirement on what style of policy is used.* It cleanly allows multiple styles to coexist, for example with BGP-style policies at some ASes, and LT-style policies at others. A convenient consequence of our architecture is that *regardless of what the other ASes choose, the LT routers obtain the entire benefit of small FIBs*, and part of the improved path choice. Intuitively, a router needs space in its forwarding table only for the pathlets that it constructs.

We confirm these results in experiments with an implementation of pathlet routing. Our implementation also shows that while our protocol can have greater messaging and control plane memory overhead than path vector protocols like BGP, the overhead is small in Internet-like topologies.

Thus, pathlet routing supports complex BGP-style policies while enabling the adoption of policies that yield small forwarding plane state and a high degree of path choice.

Paper outline. We introduce the core of the protocol in Sec. 2, and the scheme for disseminating pathlets in Sec. 3. Sec. 4 discusses new uses of pathlet routing, including LT and mixed policies. In Sec. 5 we show pathlet routing can emulate the policies of several other protocols. Sec. 6 describes and evaluates our implementation of pathlet routing. We discuss related work in Sec. 7 and conclude in Sec. 8.

2. THE PATHLET ROUTING PROTOCOL

This section begins with a simple example (§2.1). We then describe the core pathlet routing protocol: its building blocks of vnodes and pathlets (§2.2), how pathlets are built (§2.3), how packets are forwarded (§2.4), and how the sender picks its route from the pathlets it knows (§2.5).

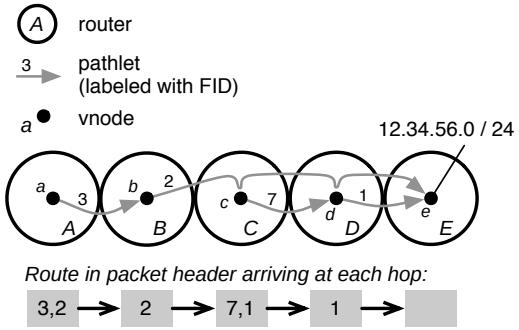


Figure 1: A pathlet routing example.

Besides this core protocol, we must specify how pathlets are disseminated throughout the network. This is a largely separable component of the design, because (unlike BGP) route policy is enforced by what pathlets are *constructed*, rather than by how they are *announced*. For now, the reader may assume "telepathic" routers which know every constructed pathlet. We will present out dissemination protocol in Section 3.

2.1 Example

Before defining the protocol in detail, we give a simple example to illustrate the pathlet mechanism. Consider the topology in Fig. 1 with routers A, B, C, D, E, each of which has one vnode (a, b, c, d, e, respectively). Initially, the routers learn the vnodes of their neighbors. They can then construct one-hop pathlets to their neighbors, as A, C, and D have done. A pathlet is given a forwarding identifier (FID) which identifies it in the routing table of its first vnode. For example, entry 7 in the routing table of vnode c instructs the router to forward the packet to D.

A sender determines its destination by finding a vnode tagged with the appropriate IP prefix, such as e in the example. It specifies a route as a list of FIDs in the packet header. A packet starting at c with route $(7, 1)$ will have its first FID popped off and will be forwarded to d with route (1) in the header, whereupon the next FID is popped off and the packet is forwarded to e with the empty route $()$, indicating that it has reached its destination.

After one-hop pathlets are constructed, multihop pathlets can be built using them. Here, B builds a pathlet $b \rightarrow c \rightarrow d \rightarrow e$. It picks FID 2, and sets the forwarding table instructions to push $(7, 1)$ onto the front of the packet's route and then forward it to C. A packet beginning at a can now reach e using route $(3, 2)$, as shown at the bottom of Fig. 1.

Note that the routers in this example have used different styles of routing policy. A, C, and D have policies that are "local", i.e., they depend only on their neighbors. B has a BGP-like policy which depends on the destination: it allows transit from B to C only when the ultimate destination is E. We will see these two styles again in Sec. 4.

In the rest of Sec. 2, we give a more detailed description of the pathlet routing protocol.

2.2 Building blocks

Pathlet routing can be seen as source routing over a virtual topology whose nodes are vnodes and whose edges are pathlets. We describe these two building blocks next.

Vnodes. A vnode is a virtual node created by an AS to represent the structure of routes within its own network. Vnodes can be used in many ways (some of which we will see later), but the simplest case is that each router has one vnode.

An AS creates a vnode by setting up a routing table for the vnode at one or more routers. Initially, each router is configured with at least one vnode. When a router X opens a control plane connection to a neighbor Y, it designates an **ingress vnode** for Y: it sends Y a globally-unique vnode identifier v, indicating that every packet X receives from Y will be directed to v. The ingress vnode can be different for each of X's neighbors, thus allowing X to control routes based on a packet's previous hop. For example, in Sec. 4.1 we will construct vnodes representing ingress from neighbors grouped into classes according to customer, provider, and peer business relationships.

Routers learn most vnodes implicitly, when they are mentioned as part of a pathlet's path. Additionally, a vnode can be tagged with a destination IP prefix, as in e in Fig. 1. Tag announcements are handled with the same dissemination algorithm as pathlets (see Sec. 3).

Pathlets. A pathlet represents a sequence of vnodes $v_1 \to \cdots \to v_n$ along which the AS X that originated the announcement is willing to route. The first vnode v_1 is in X, but the others may be in X or other ASes.

A pathlet is identified by a **forwarding identifier**, or **FID**, f. To routers other than the pathlet's creator, f is simply an opaque variable-length string of bits with the semantics that if a packet arrives at v_1 and the packet's source route begins with f, then it will be forwarded along the path $v_1 \to \cdots \to v_n$, arriving at v_n with f popped off the front of the source route. The sender of a packet will place a series of FIDs in the packet to indicate the full route.

To the router or routers on which v_1 is instantiated, f is used as an index into the vnode's forwarding table. Thus, f must uniquely identify the pathlet among all pathlets beginning at v_1—but importantly, f need not be globally unique like the identifiers in IP source routing, or even unique within an AS. The result is very compact FIDs. For example, in Fig. 1, it would actually have been possible to give all pathlets the same FID since they originate at different vnodes.

Our implementation uses the following variable-length FID encoding scheme. The first bits of the FID indicate its length: FIDs that begin with 0, 10, 110, 1110, and 1111 have total lengths of 4, 8, 16, 24, and 32 bits, respectively. The remaining bits contain the unique identifier. In our LT policies, most vnodes originate ≤ 8 pathlets, so these use the short 4-bit FIDs. Other encoding schemes are possible, and an AS can unilaterally pick its own scheme, since other routers simply view the FID as an opaque string of bits.

2.3 Pathlet construction

In the "base case" of pathlet construction, a pathlet connects two vnodes, $v_1 \to v_2$. In the general case, after X has learned or constructed some pathlets, it can concatenate these to build longer pathlets which take the form $v_1 \to P_1 \to \cdots \to P_k$, where each P_i is an existing pathlet. The router gives the pathlet a FID f, and in the forwarding table for v_1, it associates f with:

- a **next hop rule** to reach v_2: instructions for forwarding packets to the next vnode in the pathlet's path. Examples include transferring the packet to another vnode at the same router; sending it on a certain outgoing interface; or tunneling it across an intradomain routing protocol like OSPF to an egress point.
- the **remaining FIDs**: the list of FIDs necessary to route the packet from the second vnode to the final vnode in the pathlet. If the pathlet has just one hop, this list is empty. Otherwise it is a list $r_1, \ldots, r_k$, where each r_i is the FID for pathlet P_i.

Readers nostalgic for LISP might think of these two fields as the `CAR` and `CDR` of the pathlet's forwarding information.

For example, in Fig. 1, most of the pathlets are the "base case" one-hop pathlet, which can be constructed immediately after each router X opens the control plane connections to its neighbors and learns their ingress vnodes for X. After the one-hop pathlets are disseminated through the network (according to the protocol of Sec. 3), router B can create the "general case" pathlet $b \to c \to d \to e$ by concatenating the two pathlets $c \to d$ and $d \to e$. In the forwarding table for vnode b, it maps the FID 2 to the remaining FIDs ("push $(7, 1)$ onto the front of the route") and the next hop rule ("send the packet to router C").

2.4 Packet forwarding

Each router has multiple forwarding tables: one for each of its vnodes. The table for each vnode v is an exact-match lookup table with one entry for each pathlet beginning at v. The entry maps the pathlet's FID to the next hop rule and the remaining FIDs, as described above.

When a packet is sent from router Y to X, it is interpreted as arriving at the specific vnode v which is X's ingress vnode for Y. It is safe to assume that the previous hop cannot be spoofed: X knows on which of its interfaces it received the packet, and the interface would be associated with a specific neighboring AS.

The router X then inspects the route in the packet header. If it is empty, the packet is delivered locally. Otherwise the route is a sequence of FIDs $(f_1, f_2, \ldots, f_n)$ of the pathlets forming the route to its destination. Initially, this is set by the sender to be the selected route. The router checks for f_1 in the forwarding table for the ingress vnode v. If no entry is found, the packet is malformed and is dropped. Otherwise, the table maps f_1 to a next hop rule and the remaining FIDs $r_1, \ldots, r_k$ for this pathlet. It pops off the packet's first FID f_1 and pushes the remaining FIDs, resulting in a route of $r_1, \ldots, r_k, f_2, \ldots, f_n$. Finally, it forwards the packet to the next vnode, according to the next hop rule.

Fig. 1 gives a packet forwarding example. At a, the first FID (3) is popped off and used to look up the next-hop rule and remaining FIDs. There are no remaining FIDs because it is a one-hop pathlet, so the packet is forwarded to b according to the next-hop rule. At b, the next FID (2) is popped. Here the remaining FIDs $(7, 1)$ are pushed onto the route since this is a multihop pathlet, and then it is forwarded to c according to the next-hop rule. At c and d FIDs are popped off the route, none are pushed on, and the packet is forwarded. Finally the packet arrives at e with the empty route, where it is delivered.

Note that it is impossible for a sender to "cheat" and use a source route that violates the routers' policies, even if it somehow gains knowledge of every pathlet. This is simply because there are no forwarding entries for invalid routes.

2.5 Route selection

Each router learns a set of pathlets, via control plane mechanisms to be discussed later. It can select from among these a route for each packet that it sends. A simple way to do this is to build a graph in which each vnode is a node, and each pathlet $v_1 \rightarrow \ldots \rightarrow v_n$ is a single edge $v_1 \rightarrow v_n$ (perhaps given a cost equal to the number of ASes through which the pathlet travels). Then, similar to link state routing, run a shortest path algorithm on this graph to produce a sequence of edges (i.e., pathlets) to each destination. After the router has made its path selection, it places the sequence of FIDs associated with the chosen pathlets into the packet header, and sends it.

Note that this algorithm requires a map of the entire Internet, as learned by the control plane. However, as we will see (§6), it is not dramatically more state than BGP disseminates. And memory use in this algorithm is not as critical as in the FIB, for several reasons. First, it is performed by edge routers which do not need as high packet processing speeds as core routers, so the map can be stored in slower (larger, cheaper) memory. Second, an edge router can cache routes for the (presumably small) set of destinations to which it is sending at any one point in time, so that the route computation is needed only for the first packet in each connection, or the caching could be offloaded to the senders. In fact, it would be feasible for many end-hosts (rather that edge routers) to choose routes themselves, but the protocol between end-hosts and their gateway routers is beyond the scope of this work.[1]

Routers have the freedom to make more intelligent choices than the simple shortest path algorithm presented above, through information learned outside of the pathlet routing protocol. Options include learning path properties based on observations of performance or availability [32, 4]; commercial route selection products [3]; each network operating a route monitoring service for its users [30]; third-party route selection services [20], or even a global "Internet weathermap" service that would assist end-hosts in their route selection.

3. PATHLET DISSEMINATION

We motivate our dissemination algorithm with two straw man proposals (§3.1). We then describe our chosen dissemination algorithm (§3.2), the choice of which pathlets to disseminate (§3.3), and an extension where nonadjacent routers exchange pathlets (§3.4).

Note that the algorithm we present here and evaluate in Sec. 6 should be seen only as one feasible design. The choice of dissemination algorithm largely does not affect the rest of our design, and could easily be replaced with another approach.

3.1 Design rationale

Straw man #1. Suppose we simply broadcast all pathlets to the entire network using a link state algorithm. This is in fact *not* a policy concern: if pathlet routing is used as intended, then policy is enforced in the data plane, rather than through obscurity in the control plane. If a certain route is not allowed, then the appropriate forwarding table entries do not exist, so no sequence of bits placed in the packet header can cause a packet to traverse the route.

Instead, the problem with this approach is simply that there may be too many pathlets. Certain cases would be manageable, such as if all ASes use the Local Transit policies that we propose; but there is no fundamental constraint on how many pathlets an AS constructs.

Straw man #2. Suppose now that we use a standard broadcasting algorithm, except any router may decide to propagate only a *subset* of its known pathlets. This solves the scalability problem. However, it leads to a subtle issue when there are multiple simultaneous failures: it is possible for a router to learn of a pathlet which later fails, but not be notified of the failure because the channel over which it would have learned about the pathlet's state has *also* failed. We omit an example due to space constraints. The key point is that it would be possible for a router to continue to use a pathlet for an arbitrarily long time after it has failed.

3.2 Pathlet dissemination algorithm

We choose a path vector algorithm to disseminate pathlets, much as BGP notifies the Internet of the existence of IP prefixes. We use a standard, bare-bones path vector protocol, with a pathlet announcement containing the pathlet's FID and its sequence of vnode identifiers.

Path vector has two important properties. First, it solves the scalability problem from the first straw man, by allowing routers to propagate an arbitrary subset of their known pathlets (much as BGP allows routers to choose which routes to export). Second, it guarantees that if a pathlet fails, it will eventually be withdrawn from all routers that learned of it (intuitively because path vector remembers the dissemination channel).

This choice of path vector might seem ironic, given that our goal is to improve on BGP's path vector routing; but we use path vector only to make pathlets known, not to pick packets' routes. In particular, the dissemination path attribute (corresponding to BGP's ASPATH) is used only for detection of loops in the dissemination channel, *not* as a path along which to route data packets.

The main disadvantage of path vector is that a router must remember up to $O(\delta\ell)$ state for each pathlet rather than $O(1)$ state as in a broadcasting algorithm, where δ is the number of neighbors and ℓ is the average path length. However, we have a great deal of flexibility to optimize the overhead of the algorithm, compared with BGP's use of path vector. The simplest optimization is that we never need to switch to a "more preferred" dissemination path, since they are all equally acceptable; see our experimental results concerning message rates in Sec. 6.3. Other optimizations to reduce both messaging and memory overhead would be feasible, as we discuss in Sec. 8.

3.3 Which pathlets are disseminated?

Since we use a path vector dissemination protocol, each router can choose exactly which subset of its known pathlets to announce to its neighbors, just as in BGP's export filter. The rule we choose is as follows.

Suppose v is router X's ingress vnode for neighbor Y. From among all the pathlets that X knows (both those

[1]For many end-hosts, it would be feasible to learn pathlets from the gateway router and run the same algorithm presented here. Alternately, the gateway router could give the end-host several choices without detailing what those routes are, as in [31, 23]; or path selection could be performed entirely by routers.

it constructed and those it learned from neighbors), it announces a subset to Y, as follows:

1. Announce pathlets which form a shortest path tree from v to all destination vnodes reachable from v.
2. Announce any additional pathlets that are reachable from v, up to $limit(\delta)$ pathlets originating at each AS with δ AS-level neighbors.

Step 1 ensures that sufficient pathlets are supplied in order that Y can reach every destination reachable from v. Step 2 adds desirable redundancy. In our implementation, we use $limit(\delta) = 10 + \delta$, so that higher-degree ASes can announce more pathlets. But this particular choice is not critical since Step 1 handles reachability.

3.4 Extensions

Information dissemination in distributed systems can generally be described as either "push" or "pull". The above dissemination algorithm, like BGP, pulls routing state. But an easy extension of our protocol allows any router to initiate a control plane connection with any other (perhaps non-adjacent) router, and *pull* certain pathlets, such as those relevant to a specified destination. This extension can be used to emulate control-plane features of three recent multipath routing proposals: MIRO, NIRA, and LISP (§5). It is also similar to BGP's multihop option, where an eBGP session connects to non-adjacent routers.

4. NEW USES OF PATHLET ROUTING

In this section, we discuss three practical examples of what one can do conveniently with pathlet routing, but not with BGP or most other protocols: **Local Transit policies** (Section 4.1) allow networks to control one of the most important aspects of routing policy—the portions of routes which transit across their own networks—while permitting very small forwarding tables and a large degree of route choice for senders. Pathlet routing cleanly supports **mixed policies** (Section 4.2), for example with some ASes using BGP-style policies and some using LT policies. The LT adopters still have small forwarding tables, and some of the benefit of additional route choice. Finally, pathlet routing may be a more flexible way of offering multiple **types of service** along the same physical path (Section 4.3).

4.1 Local Transit Policies

Definition. A network X has a **local transit (LT) policy** when X's willingness to carry traffic along some route depends only on the portion of the route that crosses X's network. In other words, under an LT policy the permissibility and cost of some path, according to X, is a function only of the ingress and egress points of the path in X. An alternate definition is that the policy can be constructed using pathlets which never extend beyond X's neighboring vnodes—hence, the policy is local. (Note that LT policies concern only how traffic may *transit* X's network; X may still have arbitrary preferences on the paths taken by traffic that X *sends*.)

An important example of LT policies is **valley-freeness** [11], a common export policy in BGP today. Valley-free routes can be defined as follows: each neighbor is labeled as a customer, provider, or peer; a BGP route is exported to a neighbor X if and only if either the next hop of the route is a customer or X is a customer. This is a function only of the ingress and egress points, and hence is an LT policy.

To the best of our knowledge, other multipath routing protocols [8, 28, 30, 18, 23, 32] cannot implement local transit policies, though some [30] can implement special cases.

Advantages, disadvantages, incentives. LT policies are an example of policies in which constraints are local. In general, this locality leads to two advantages over BGP-style policies. First, the pathlets can be combined in potentially an exponentially large number of ways, leading to improved performance and reliability for senders. Second, they can be represented with a small number of pathlets, leading to small forwarding tables. (We will demonstrate both advantages in Sec. 6.)

The primary disadvantage is that policies cannot depend on a route's destination. Such policies are used in BGP, where ASes select and advertise their most preferred route on a per-destination basis. Thus, ASes would end up permitting routes other than the one for each destination that is cheapest. A more minor issue is that LT policies use slightly more control plane state than BGP (as we will see in Sec. 6), but this is not as constrained a resource as data plane memory.

We argue that for a large fraction of ASes, the advantages of LT policies may outweigh the disadvantages. LT-capable routers require less data plane memory and hence less cost, and are much less susceptible to scalability problems as the Internet grows; and providing more path choice is a service that customers may be willing to pay for. Providing more paths may also attract more traffic and hence more revenue. Moreover, the disadvantage of policy locality is limited since the common BGP export policy of enforcing valley-free routes is an LT policy. At a higher level, we note that offering multipath network services at a constant price *regardless of the path* may be a feasible business model for the same reasons that today's transit services have a fixed rate *regardless of the destination.*

Ultimately, whether ASes choose to use LT policies depends on business decisions and payment mechanisms, but pathlet routing makes it feasible at a technical level. Furthermore, as we will see in Section 4.2, this is a decision that can be made by each AS individually.

Implementation. Figure 2 depicts two ways of implementing LT policies in pathlet routing. The simplest way uses one ingress vnode for each neighbor. Then, the AS constructs a pathlet between each ingress-egress pair for which it wants to allow transit service. However, this results in $O(\delta^2)$ pathlets for a network with δ neighbors.

Fortunately, we can use our vnode abstraction for a much more compact representation. We assign each neighbor to a "class", an abstract group of neighbors which are equivalent from a policy standpoint. Examples are business relationships (customer, provider, peer) or potentially geographic regions. We have vnodes representing ingress and egress for each class (rather than each neighbor), and then construct the appropriate pathlets between these vnodes.

This reduces the number of pathlets from $O(\delta^2)$ to $O(c^2 + \delta)$, where c is the number of classes. We argue that the number of classes would typically be small. For example, in the case of valley-free routes, we need only 2 classes, customer and provider/peer, for a total of $4 + \delta$ pathlets per AS. This is depicted in Fig. 2.

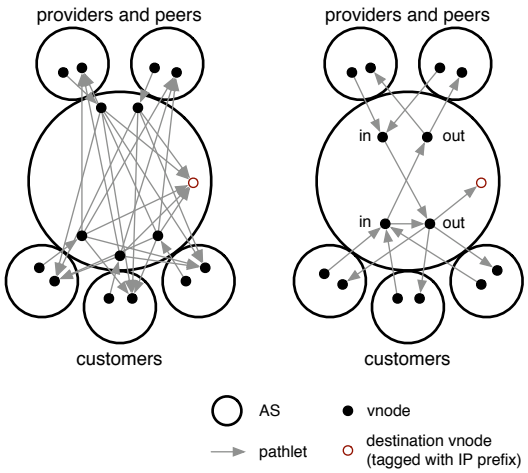


Figure 2: Two ways to implement a local transit policy are to connect the appropriate ingress-egress pairs (left), or to group neighbors into classes and connect the appropriate classes (right). Here we show the vnodes and pathlets in one AS to permit valley-free routes.

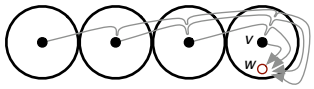
One difficulty arises with class-based LT policies. If an AS is internally partitioned, it may not be able to represent reachability as a class-based LT policy, which implicitly assumes that if the provider class can reach the customer class, then *all* providers can reach *all* customers. A solution is for the AS to advertise two sets of LT policies, one on each side of the partition, in the rare case that it becomes internally partitioned. Alternately, the AS can simply continue announcing the pathlets; sources will realize that some pathlets have failed and will switch to a different route, assuming another is available.

4.2 Mixed Policies

In this section we describe how pathlet routing supports mixed policies, in particular with some ASes using traditional BGP-style policies, and some using LT-style. Mixed policies are important because it allows ASes to make independent choices about their policies. Some may require restrictive BGP style policies; others may use LT-style policies, giving up some control but getting the benefit of small forwarding tables and providing many possible paths. It is unlikely that either choice would be preferred by *all* ASes.

We require no new protocol mechanisms to support mixed policies; routers run the algorithms we have already described in Sections 2 and 3, constructing the pathlets appropriately to match their routing policy. However, since we believe this is an important way of using pathlet routing, we illustrate the process here.

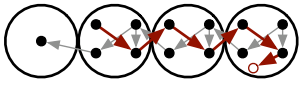
Emulating BGP. As a prelude, we illustrate the non-mixed case when all ASes use BGP-style policies. To emulate BGP, each AS constructs one vnode v from which all its pathlets originate, and which is its ingress vnode for all neighbors. If it owns an IP prefix, then it has a second vnode w tagged with the prefix, from which no pathlets depart. It then constructs a pathlet from v to every destination it can reach, along its most-preferred available path. If the destination is at w, this is a one-hop pathlet; otherwise it is a multihop pathlet which adds one hop to one of the pathlets it learned from a neighbor. This is depicted below in a topology with a single IP prefix destination:



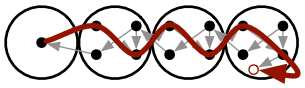
To mimic BGP's export rules, in which routes are exported to only certain neighbors, there are two options. First, the router may simply not announce certain pathlets to its neighbors. This mimics BGP "too closely", since a neighbor could still use a prohibited path if, for example, it managed to guess the pathlet's FID. (Similarly, in BGP, a neighbor could send a packet even when there is no announced route for the destination IP.) A solution which is better than BGP is to enforce policy in the forwarding tables themselves. For the common valley-free export policy, this can be done with a small constant number of additional vnodes and pathlets, similar to the LT construction of Fig. 2; we omit the details.

A BGP-policy router in a mixed setting. In the previous example all routers used BGP-style policies. Now consider a "mixed" network, where the leftmost router in the previous example continues using BGP-style policies, but the others use LT policies. It runs the same algorithm as before, building a pathlet to each destination. However, it may now have an exponentially large set of possible paths to each destination, rather than one path through each neighbor. This necessitates a generalization of the BGP decision process to select the best path without constructing each option explicitly. For example, it could run a least-cost path algorithm, with large costs assigned to its provider links, less cost on its peering links, and lowest cost on its customer links.

The resulting pathlet is also slightly different: instead of adding one hop to a multihop pathlet, it is built by concatenating multiple short pathlets from the LT nodes, highlighted below:



The result is a single long pathlet:



An LT-policy router in a mixed setting. Due to locality, an LT router's pathlets do not depend on pathlets constructed elsewhere. Therefore, it has the same number of pathlets originating from its vnodes, and hence the same small routing tables regardless of what other routers do.

However, it does have to deal with disseminating other routers' pathlets in the control plane. If an LT node has δ neighbors using BGP-style policies, then it receives $O(\delta n)$ pathlets for a network of size n, and *all of these are reachable from its ingress vnodes.* The router would therefore be happy to disseminate them all. In fact, a mixed setting of BGP-style and LT policy routers could result in $O(n^2)$ pathlets usable by LT nodes. However, our dissemination-limiting mechanisms from Sec. 3.3 will take effect, and only a small number of these will be propagated.

4.3 Quality of Service

We discuss one final novel use of pathlet routing. Our protocol's abstraction of the topology results in a convenient mechanism to set up multiple paths along the same physical path. In particular, an AS can construct multiple "parallel" pathlets, over the same sequence of vnodes. Packets can then be treated differently depending on which pathlet is used. For example, an AS could provide better or worse throughput or delay, or more non-traditional services like reliable packet delivery across that pathlet rather than a full end-to-end route, similar to new services proposed by Ford and Iyengar [10]. To inform sources of these different types of service, pathlet announcements can be tagged with labels, or sources could discover performance by measurement or by a third-party routing appraisal service, as discussed in Section 2.5.

This approach is more flexible and extensible than other approaches for several reasons. First, the service is selected for a specific segment of a path, rather than for the entire path. Thus, different combinations of services are possible and an AS which is not involved in providing a particular type of service does not need to support that service or even be aware of what services a packet is using in other ASes. Second, the bits in the packet header are not tied to a small number of predetermined meanings (as in, for example, IP's Type of Service bits). Instead, the header bits (FIDs) are tied to a much more extensible control plane announcement by the AS providing the service.

One drawback is that the amount of routing state increases as more types of service are offered. However, in terms of forwarding plane state, this only affects the ASes that are offering the service; and if an AS is using policies like Local Transit policies, the amount of forwarding plane state is quite small to begin with.

5. POLICY EXPRESSIVENESS ANALYSIS

In this section, we develop a framework for comparing routing protocols. We show that pathlet routing's data plane can emulate any policy expressible in the data plane of several existing protocols using a similar amount of state, even though these protocols have substantially different forwarding architectures. Our goal is to support the claim that pathlet routing is highly expressive, but we also believe this framework is of independent interest since it takes a step towards making sense of the large number of proposed protocols in this area.

We begin by discussing intuition for pathlet routing's flexibility (§5.1). We then we define policy emulation (§5.2) and discuss the protocols pathlet routing can emulate (§5.3) and those it cannot (§5.4), highlighting some limitations of our protocol. These results are summarized in Figure 3.

5.1 Intuition

Why does pathlet routing appear to be so flexible, generalizing multiple other routing protocols? One piece of the intuition is that many protocols incorporate some variant of tunneling. Our pathlet construct elevates the tunnel to be a "first-class object" that can be advertised, concatenated, and source-routed. We will repeatedly use pathlets to emulate tunnel-like features of other protocols. However, this is only part of the story since we also utilize vnodes.

At a higher level, the following intuition is simple yet powerful. One of the most flexible and general abstractions is the graph. Pathlet routing allows routing policies to be specified as an *arbitrary virtual graph*, whose nodes are vnodes and whose directed edges are pathlets.

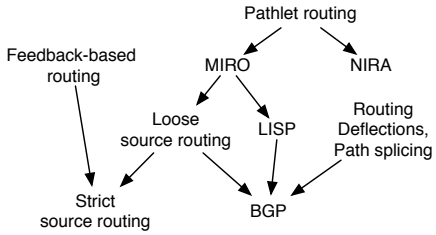


Figure 3: Relative policy expressiveness of the data planes of routing protocols. $P \to Q$ indicates P can emulate the routing policies of Q.

5.2 Definition of policy emulation

Our analysis focuses on the data plane: the information carried in a packet and in the forwarding table, and forwarding operations. We found that considering only the data plane gives a much cleaner and more well-defined way of reasoning about protocols, compared with modeling both the data and control planes. We will also discuss control plane differences between the protocols, leaving rigorous analysis of their control planes to future work.

A **configuration** of a protocol is defined by an arrangement of forwarding table state at each router. Given a configuration c_1 from protocol P and a configuration c_2 from protocol Q, we say that c_1 **covers** c_2 when

- every end-to-end route which is *allowed* in c_1 is also allowed in c_2, and every end-to-end route which is *prohibited* in c_1 is also prohibited in c_2; and
- for each router i, $|c_1(i)| = O(|c_2(i)|)$, where $|c_j(i)|$ denotes the amount of forwarding state for router i in configuration j.

Finally, we say that protocol P can **emulate** Q if for every configuration c_2 of Q, there is a configuration c_1 of P such that c_1 covers c_2. In other words, if P can emulate Q, then P can match every possible outcome of Q in terms of allowed paths, prohibited paths, and forwarding table size.

This definition is limited. We have chosen not to incorporate aspects of policy like the price of a route, visibility of routes, misbehavior outside the rules of the protocol, or game-theoretic aspects like which outcomes might actually arise under selfish behavior, or the control plane as discussed above. However, we believe that policy emulation provides a way to begin reasoning about the relative strengths of different protocols.

5.3 Protocols pathlet routing can emulate

Due to space constraints, we only briefly describe some of the protocols and the emulation relationships between them.

BGP [25]. We outlined how pathlet routing can emulate BGP (in both the data an control planes) in Section 4.2.

NIRA [30] offers more choice to sources than BGP, while simultaneously reducing control plane state. IP addresses are assigned so they encode a specific path between a router and the "core" of the Internet or a peering point. A source

and destination IP address together specify a full end-to-end route. The most challenging part of emulating NIRA is that an AS can allow or prohibit routes based on the *upstream* hops as far back as the core providers. In contrast, in the usages of pathlet routing we have seen so far, policies depend only on the immediately prior hop and any downstream hops. However, it is possible to construct vnodes to encode the packet's upstream hops, using the same amount of state as NIRA. This requires some coordination of vnode names in the control plane between neighbors, but no data plane changes.

The basic NIRA protocol is limited to valley-free routes, so it cannot emulate the other protocols we consider. An extension including a source-routing option is described in [29] but is not analyzed here.

Locator/ID Separation Protocol (LISP) [9] maps a topology-independent endpoint ID into an egress tunnel router. BGP is used to tunnel a packet to its destination's egress tunnel router, which delivers it to the final destination using the endpoint ID. This arrangement can reduce forwarding state since most routers need only know about egresses, rather than all endpoints. Pathlet routing can emulate LISP by concatenating a pathlet representing the tunnel and a pathlet representing the remainder of the route.

IP strict source routing [8], IP loose source routing [8] and **MIRO [28].** Pathlet routing can emulate these protocols in a straightforward manner. The common thread in these protocols and LISP is routing via waypoints or tunnels, which we find can be emulated with MIRO's data plane. (Note, however, that MIRO's data and control planes were intended for a significantly different usage scenario in which most paths are set up via standard BGP and a relatively small number of additional tunnels are constructed.) We omit a full description due to space constraints.

5.4 Protocols pathlet routing cannot emulate

Feedback Based Routing (FBR) [32] is source routing at the AS level, with each link tagged with an access control rule. A rule either whitelists or blacklists a packet based on prefixes of its source or destination IP address. Pathlet routing cannot emulate FBR for two reasons. First, it is difficult for pathlet routing to represent policies based on upstream hops. Essentially, the only way to carry utilizable information about a packet's origin is to encode the origin into the vnodes the packet follows, which would increase the number of necessary vnodes and hence the amount of state by a factor of n relative to FBR, where n is the number of nodes in the network. (Note that we solved a similar problem in emulating NIRA, but NIRA's solution used the same amount of state as ours.) The second problem in emulating FBR is that FBR has both blacklisting and whitelisting, while pathlet routing effectively has only whitelisting. FBR can therefore represent some policies efficiently for which pathlet routing would require significantly more state.

But FBR cannot emulate pathlet routing either. For example, controlling access based only on source and destination address ignores intermediate hops which can be taken into account by pathlet routing (and BGP).

Routing deflections [31] and **interdomain path splicing [23].** In these protocols, each hop can permit multiple alternate paths to the destination. The packet includes a "tag" [31] or "splicing bits" [23] to specify which alternative is used at each step. The selected path is essentially a pseudorandom function of the tag.

Pathlet routing cannot emulate these protocols due to the handling of tags. In [31] the tag has too few bits to represent the set of all possible routes that the routers intended to permit, so the *de facto* set of allowed routes is a pseudorandom subset of those—an effect which to the best of our knowledge pathlet routing cannot reproduce. One mode of operation in [23] behaves the same way. A second mode of operation in [23] uses a list of source-route bits to explicitly select the next hop at each router; pathlet routing *would be able* to emulate this version.

But there is also a problem in the control plane. Even if pathlet routing can match the allowed paths in the data plane, actually *using* the paths would require the senders to know what FIDs to put in the packet header. Because [31, 23] allow per-destination policies as in BGP, this can result in a large amount of control plane state in pathlet routing. Routing deflections and path splicing avoid this control plane state by not propagating alternate path information; the tradeoff is that the senders and ASes in those protocols cannot identify which end-to-end paths are being used.

6. EXPERIMENTAL EVALUATION

We implemented pathlet routing as a custom software router, and evaluated it in a cluster environment. We describe the structure of our implementation in Sec. 6.1, the evaluation scenarios in Sec. 6.2, and the results in Sec. 6.3.

6.1 Implementation

We implemented pathlet routing as a user-space software router, depicted in Fig. 4. Each router runs as a separate process and connects to its neighbors using TCP connections, on which it sends both data and control traffic.

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, making it compact and easy to tune to the specific needs an AS might have. We describe these three modules briefly.

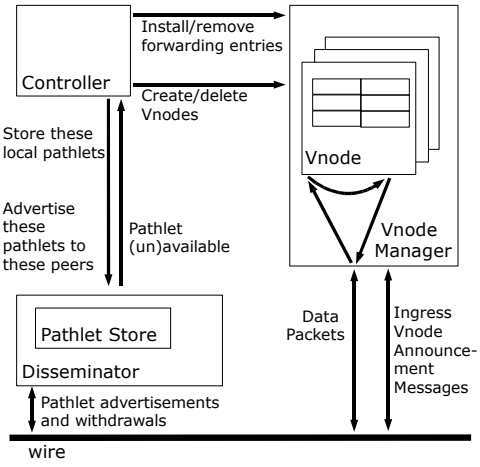


Figure 4: Structure of the software pathlet router.

The vnode manager is responsible for directing incoming data packets to vnodes, which store their forwarding tables and perform the lookup, as well as for sending the data packets out to the next hop. The vnode manager also sends and receives control messages that inform peers about the ingress vnodes.

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.

While the vnode manager and the disseminator are general and oblivious to the AS's policy, the controller encapsulates the policy logic. It implements the policy by constructing and deleting pathlets and vnodes and by deciding which pathlets to announce to which peers. In our implementation, one makes a router an LT router or BGP-style router by picking the corresponding controller.

6.2 Evaluation scenarios

Policies. We tested three types of policies: LT policies, Path Vector-like policies, and mixed policies, with 50% of nodes randomly chosen to use LT policies and the rest using PV.

The special case of LT policies that we test are valley-free routes, as in Fig. 2. The PV policies emulate BGP. Specifically, we 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, similar to BGP. We use the common BGP export policies of valley-free routes.

Topologies. We tested two types of topology. **Internet-like** topologies annotated with customer-provider-peer relationships were generated using the algorithm of [7]. Each AS is represented as a single router. **Random graphs** were generated using the $G(n, m)$ model, i.e., n nodes and m random edges, excluding disconnected graphs. In the random graph there are no business relationship annotations; ASes prefer shortest paths and all paths are exported. Unless otherwise stated, these graphs have 400 nodes and an average of 3.8 neighbors per node.

Event patterns. We show results for several cases: the initial convergence process; the state of the network after convergence; and a sequence of events in which each link fails and recovers, one at a time in uniform-random order, with 8 seconds between events for a total experiment length of 3.6 hours. We implemented link failures by dropping the TCP connection and link recovery by establishing a new TCP connection.

Metrics. We record connectivity, packet header size, forwarding table size, number of control plane messages, and control plane memory use. The CDFs of these metrics that we present are the result of three trials for each evaluation scenario, each with a fresh topology. We show all data points (routers or source-destination pairs) from all trials in a single CDF.

6.3 Results

Forwarding plane memory. Fig. 5 shows a CDF of the number of forwarding table entries at each router. The number of entries varies with the node degree for LT routers, and with the size of the network for PV nodes. As a result, the LT nodes have a mean of 5.19 entries in the all-LT case and 5.23 in the mixed case. PV averages 400.5 entries in the mixed and all-PV cases.

We also analyzed an AS-level topology of the Internet generated by CAIDA [5] 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 entries for the between $132,158$ and $275,779$ currently announced IP prefixes, depending on aggregation [2]. Thus, in this case LT policies offer more than a $15,000\times$ reduction in forwarding state relative to BGP.

Route availability. One of the principal advantages of multipath routing is higher availability in the case of failure, since the source can select a different path without waiting for the control plane to reconverge.

We measure how much LT policies improve availability as follows. We allow the network to converge, and then take a snapshot of the routers' forwarding tables and the pathlets they know. We then select a random set of links to be failed, and determine for each pair of routers (X, Y) whether there is a working $X \rightsquigarrow Y$ route in the data plane, using only the pathlets that X knows. If so, then X has a continuously working path to Y, *without waiting for the control plane to detect the failures and re-converge.* Thus, we are measuring how often we can avoid PV's transient disconnectivity. (This measurement ignores the algorithm X uses to find a working path among its options, which is outside the scope of this work. However, a simple strategy, like trying a maximally disjoint path after a failure, is likely to perform well.)

The results of this experiment are shown in Fig. 6. Relative to PV, which emulates the availability of BGP, LT has significantly improved availability. In fact, it has the *best possible availability* assuming ASes allow only valley-free routes, because every path allowed by policy is usable in the data plane. In the random graph topology, we see a bigger connectivity improvement in part because the possible paths are not constrained by valley-freeness.

In the mixed environment, the LT nodes obtain the majority of the improvement of a full deployment of LT nodes. To further illustrate this relationship, Fig. 6 shows how connectivity improves as a function of the fraction of nodes that use LT policies, in an 80-node random graph. Here the number of failed links is fixed at 5, and we report the mean of 5 trials.

In the random graph topology in Fig. 6, we also show availability when fewer pathlets are advertised: in particular, each router advertises only the shortest path tree (see Sec. 3.3). This worsens availability, but improves the number of control messages, as we describe next.

Control plane messages. Fig. 7 shows the CDF of the number of messages received by a router following a link failure or recovery event.

Consider the Internet-like topology. LT has more objects being announced than PV: an average of 5.19 pathlets per node, vs. 1 per destination in the PV case. One might therefore expect $5.19\times$ as many messages. However, LT has only $1.69\times$ as many as PV. There are two factors that cause this. First, when a link recovers, PV needs to send many messages because its path preferences (selected by the BGP

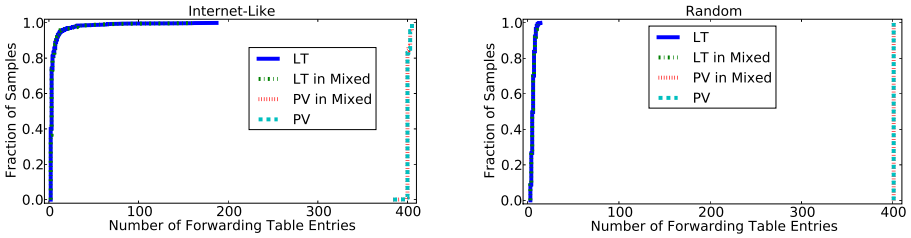


Figure 5: Forwarding table (FIB) size for the Internet-like topology (left) and the random graph (right).

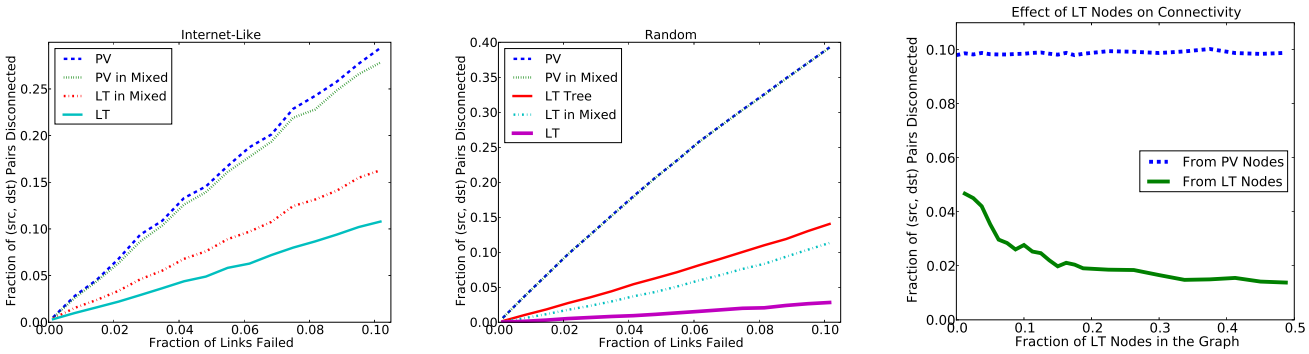


Figure 6: Probability of disconnection for a varying number of link failures in the Internet-like topology (left) and the random graph (center), and as a function of the number of LT nodes (right).

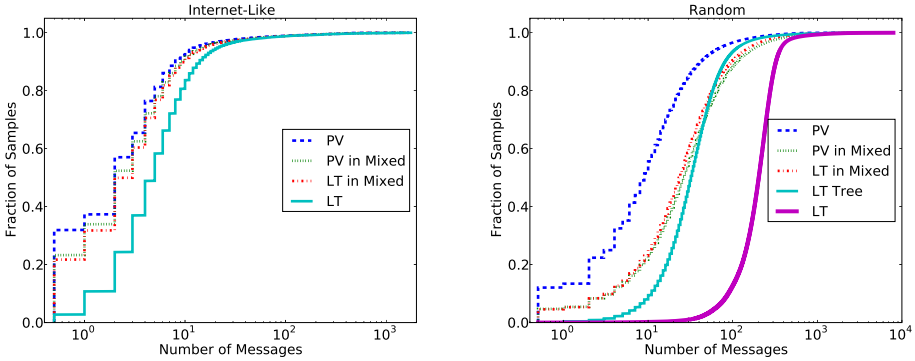


Figure 7: CDF of the number of messages received by a router following a link state change, for the Internet-like topology (left) and the random graph (right).

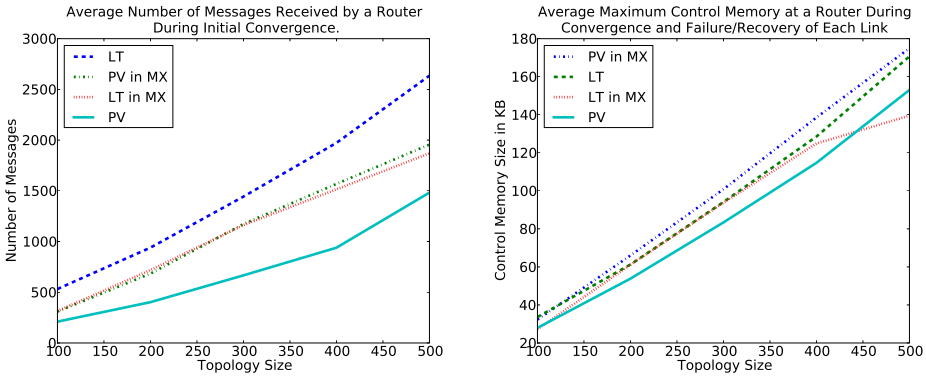


Figure 8: Scaling of messaging and control memory in the Internet-like graph, for 100, 200, 300, 400, and 500 nodes.

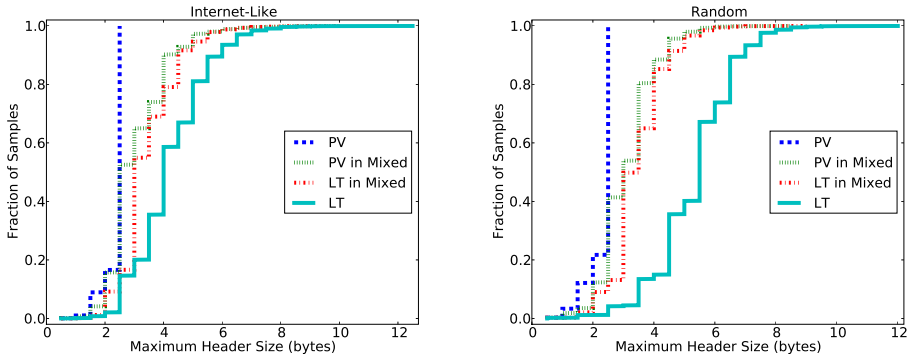


Figure 9: CDF of the size of the route field in the packet header, for the Internet-like topology (left) and the random graph (right).

decision process) cause it to switch back to paths using the recovered link. In contrast, in the LT case we need only advertise a single pathlet for the recovered link (see discussion in Sec. 3.2). Second, according to our dissemination algorithm (Sec. 3.3), many of those 5.19 pathlets are never disseminated to some parts of the network, because they are not reachable.

In the random graph topology, the second factor is not true. Not only does every router learn all pathlets, but it learns them from all of its neighbors. The result is 10.3× as many messages as PV. However, this can be reduced if nodes simply advertise fewer pathlets, which is safe as long as the advertised pathlets reach every destination. The line "LT Tree" in Fig. 7 shows the case when we advertise only a tree reaching all destinations (described in Sec. 3.3). This reduces the messaging overhead to just 2.23×. The tradeoff is that there are fewer choices of paths, and connectivity in the face of failures is worsened, as shown in Fig. 6.

Fig. 8 plots the messaging cost for initial convergence as a function of n, the number of nodes in the topology. LT and PV have similar scaling properties for this metric because both, like BGP, use a path vector dissemination algorithm to announce objects; PV announces one object per node and LT announces $O(\delta)$ (where δ is the number of neighbors), which is independent of n.

Control plane memory. Since control plane memory use is dependent on the implementation, we evaluate its scaling properties. Asymptotically, if the n nodes each have δ neighbors and the mean path length is ℓ, BGP and PV policies need to store $\Theta(n)$ announcements of mean size $\Theta(\ell)$ from up to δ neighbors, for a total of $\Theta(n\delta\ell)$ state in the worst case. In a full LT deployment there are $\Theta(\delta n)$ pathlets; each announcement has size $\Theta(\ell)$ and is learned from up to d neighbors, for a total of $\Theta(n\delta^2\ell)$ in the worst case.

Fig. 8 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 network to converge and then failing and recovering each link once. The results confirm that PV and LT policies scale similarly as a function of n. It is also apparent that with the Internet-like topologies, we don't reach the worst case in which LT is a factor δ worse than PV.

It may be possible to optimize our dissemination algorithm to reduce control state and messaging; see Sec. 8.

Header size. Fig. 9 shows the CDF over source-destination pairs (X, Y) of the number of bits in the route field of the packet header for the shortest $X \rightsquigarrow Y$ route. The number of bits may vary as the packet travels along the route; we report the maximum (which for LT policies occurs at the source).

Header length scales with the path length. In our Internet-like graph, the mean path length is 2.96 and headers average 4.21 bytes. An AS-level topology of the Internet mapped by CAIDA [5] from Jan 22 2009 has mean path length 3.77, so we could therefore extrapolate that mean header length would be about 27% greater in a full Internet topology, i.e., less than 6 bytes, assuming other characteristics of the topology remain constant. Even the maximum header length of 12.5 bytes in our implementation would not add prohibitive overhead to packet headers.

7. RELATED WORK

We have compared the policy expressiveness of a number of multipath routing protocols in Section 5. Here we compare other aspects of the protocols.

MIRO [28] uses BGP routes by default. Only those ASes which need an alternate path (say, to avoid routing through one particularly undesirable AS) need to set up a path. But this increases the latency of constructing a path, and increases forwarding plane state if there are many paths. MIRO is likely easier to deploy than pathlet routing in today's network of BGP routers.

NIRA [30] provides multiple paths and very small forwarding and control state as long as all routers have short paths to the "core"; but for arbitrary networks, forwarding state may be exponentially large. The scheme requires a potentially complicated assignment of IP addresses, and works primarily for Internet-like topologies with valley-free routing policies. Exceptions to these policies may be difficult.

Routing deflections [31] and path splicing [23] permit policies specified on a per-destination basis, while still providing many path choices with limited deviations from the primary paths. However, sources are not aware of what paths they are using, [31] has relatively limited choice of paths, and [23] can encounter forwarding loops.

IP's strict source routing and loose source routing [8] provide source-controlled routing but have limited control for policies. For this reason and due to security concerns unrelated to our setting, they are not widely used. They can also result in long headers because each hop is specified as an IP address, unlike our compact forwarding identifiers.

Feedback based routing [32] suggested source-based route quality monitoring that is likely to be a useful approach for pathlet routing.

Platypus [24] is similar to loose source routing except each waypoint can be used only by authorized sources to reach either any destination, or a specified IP prefix. Pathlet routing supports a different set of policies and enforces these using the presence or absence of forwarding tables, rather than cryptography.

R-BGP [18] adds a small number of backup paths that ensure continuous connectivity under a single failure, with relatively minimal changes to BGP. However, it somewhat increases forwarding plane state and is not a full multipath solution. For example, sources could not use alternate paths to improve path quality.

LISP [9] reduces forwarding state and provides multiple paths while remaining compatible with today's Internet. Although it can limit expansion of forwarding table size, LISP's forwarding tables would still scale with the size of the non-stub Internet, as opposed to scaling with the number of neighbors as in our LT policies.

MPLS [26] has tunnels and labels similar to our pathlets and FIDs. It also shares the high level design of having the source or ingress router map an IP address to a sequence of labels forming a source route. However the common use of these mechanisms is substantially different from pathlet routing: tunnels are not typically concatenated into new, longer tunnels, or inductively built by adding one hop at a time. To the best of our knowledge MPLS has not been adapted to an interdomain policy-aware routing.

Metarouting [13], like pathlet routing, generalizes routing protocols. It would be interesting to explore whether pathlet routing can be represented in the language of [13].

8. CONCLUSION

Pathlet routing offers a novel routing architecture. Through its building blocks of vnodes and pathlets, it supports complex BGP-style policies while enabling and incentivizing the adoption of policies that yield small forwarding plane state and a high degree of path choice. We next briefly discuss some limitations and future directions.

We suspect it is possible to optimize our path vector-based pathlet dissemination algorithm. The techniques of [16] may be very easy to apply in our setting to reduce control plane memory use from $O(\delta\ell)$ to $O(\ell)$ per pathlet, where δ is the number of neighbors and ℓ is the mean path length. Routers could also pick dissemination paths based on heuristics to predict stability, which for common failure patterns can significantly reduce the number of update messages [12]. The more radical approach of [30] could also be used to dramatically reduce state in Internet-like environments.

Traffic engineering is an important aspect of routing that we have not evaluated. One common technique—advertising different IP to different neighbors to control inbound traffic—is straightforward to do in our LT policies. But source-controlled routing would dramatically change the nature of traffic engineering, potentially making it more difficult for ISPs (since they have less control) and potentially making it easier (since sources can dynamically balance load).

Acknowledgements

We thank the authors of [7] for supplying the Internet-like topologies.

9. REFERENCES

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. 18th ACM SOSP*, October 2001.

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

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

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

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

[6] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. In *SIGCOMM*, 2002.

[7] 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.

[8] J. P. (ed.). DARPA internet program protocol specification. In *RFC791*, September 1981.

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

[10] B. Ford and J. Iyengar. Breaking up the transport logjam. In *HOTNETS*, 2008.

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

[12] P. B. Godfrey, M. Caesar, I. Haken, S. Shenker, and I. Stoica. Stable Internet route selection. In *NANOG 40*, June 2007.

[13] T. Griffin and J. Sobrinho. Metarouting. In *ACM SIGCOMM*, 2005.

[14] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proc. OSDI*, 2004.

[15] G. Huston. BGP routing table analysis reports, 2009. `http://bgp.potaroo.net/`.

[16] E. Karpilovsky and J. Rexford. Using forgetful routing to control BGP table size. In *CoNEXT*, 2006.

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

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

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

[20] K. Lakshminarayanan, I. Stoica, S. Shenker, and J. Rexford. Routing as a service. Technical Report UCB/EECS-2006-19, UC Berkeley, February 2006.

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

[22] D. Meyer, L. Zhang, and K. Fall. Report from the iab workshop on routing and addressing. In *RFC2439*, September 2007.

[23] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *ACM SIGCOMM*, 2008.

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

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

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

[27] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet routing and transport. In *IEEE Micro*, January 1999.

[28] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *SIGCOMM*, 2006.

[29] X. Yang. NIRA: a new Internet routing architecture. Technical Report Ph.D. Thesis, MIT-LCS-TR-967, Massachusetts Institute of Technology, September 2004.

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

[31] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *ACM SIGCOMM*, 2006.

[32] D. Zhu, M. Gritter, and D. Cheriton. Feedback based routing. *Computer Communication Review (CCR)*, 33(1):71–76, 2003.