Towards Internet-wide Multipath Routing

Jiayue He Dept. of EE, Princeton University
Email: jhe@princeton.edu Jennifer Rexford Dept. of CS, Princeton University
Email: jrex@cs.princeton.edu

Abstract

Today's Internet would be more efficient and robust if routers could flexibly divide traffic over multiple paths. Often, having just a few alternate paths is sufficient for customizing paths for different applications, improving security, reacting to failures and balancing load. Yet alternate paths are often unavailable, due to two barriers. First, in a network the size of the Internet, moving to flexible multipath routing can impose significant computational and storage overhead on the routers. Second, the independent networks that comprise the Internet will not relinquish control over the flow of traffic without appropriate incentives. In this paper, we survey flexible multipath routing techniques which are both scalable and incentive compatible. Techniques covered include: multihoming, tagging, tunneling, and extensions to existing Internet routing protocols.

Keywords: Internet, multipath routing, multihoming, tunneling, scalability.

I. Introduction

Most currently deployed routing protocols select only a single forwarding path for each source-destination pair. In this paper, we explore techniques which allow *flexible* division of traffic amongst multiple paths. Application requirements dictate the granularity of division, *e.g.* by prefixes, destination host, a Transmission Control Protocol (TCP) flow or a single packet. We start by examining the motivations and challenges surrounding flexible multipath routing.

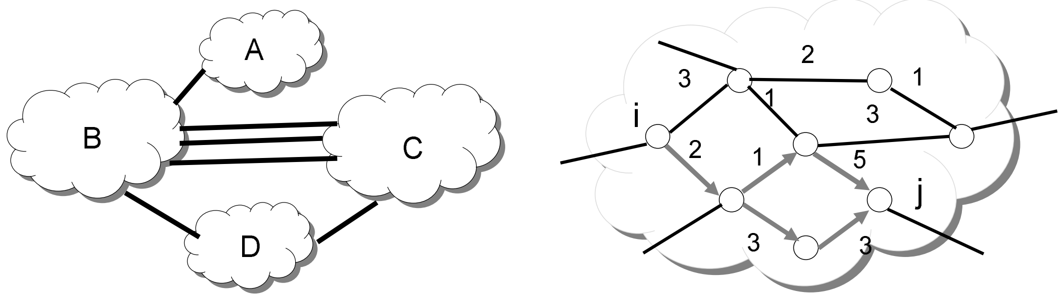
A. Motivation for Flexible Multipath Routing

Flexible multipath routing is useful in a variety of contexts as illustrated by the following examples:

- **Customizing for applications:** Different applications have different needs. Some applications (*e.g.* Voice over IP or online gaming) want low delay and loss, whereas others (*e.g.* file sharing) want high throughput. If multiple paths exist, then a low-delay path can be chosen for Voice over IP, while a high throughput path can be chosen for file sharing. In addition, an application can access more bandwidth by using multiple paths simultaneously.
- **Improving security:** Multiple paths can defend security risks in two ways. If a path performs poorly due to an adversary along the path is dropping packets, then the existence of alternate paths allows traffic to be moved onto a path that does not contain the adversary [1]. As an alternative to encryption, a sender can spread packets

belonging to the same data stream along different paths, so that it would be more difficult for an adversary to see the entire data stream [2].

- **Reacting to failures:** When no alternate paths exist, recovering from a failure could be a slow process involving several rounds of message passing between routers to establish an alternate path. If multiple paths exists, then the traffic can quickly switch onto a different path when a link or router fails along one of the paths.
- **Balancing load:** Responding to network congestion by moving traffic to a less congested path was tried in the early Internet, but resulted in routing oscillations [3], [4]. The routing oscillations can disappear if the traffic is dynamically balanced amongst multiple paths [5]. In fact, by just having *any* two paths and flexible splitting between them, protocols can be easily tuned to efficiently utilize network resources [6]. In addition, having two choices for load balancing yields a large reduction in the maximum load over a single choice [7].



(a) Topology between 4 ASes: B and C are ISPs A and D are enterprise networks

(b) Topology inside AS C

Fig. 1. Sample inter-AS topology, with a close-up on one AS.

In addition to the many potential benefits of multipath routing, today's Internet topology already contains path diversity[1]. While only one end-to-end path is used today, a measurement study found that 30% to 80% of the time, an alternate path with lower loss or smaller delay exists [8]. Path diversity exists end-to-end due to multiple connections between Autonomous Systems (ASes) and multiple paths inside a single AS. Each AS is a collection of routers and links under the control of one entity. There are two types of ASes: Internet Service Providers (ISPs) provides connectivity to other ASes, while stub networks only provides connectivity to their own users and services. A single AS is often connected to many other ASes and neighboring ISPs often connect to each other in multiple locations, as in Figure 1a. In addition, some stub networks connects to more than one ISP, *e.g.* AS D is connected to two ISPs in Figure 1a. Most of the end-to-end paths traverse one or more large ISPs, which has internal path diversity. In a measurement study of a large ISP, it was found that almost 90% of pairs of Point-of-Presences (PoPs) have at least four link-disjoint paths between them [9].

[1]In this paper we focus on the path diversity at the network layer. There is a large body of research on path diversity at the physical layer which are important for reliability and security, but the shared risks at the physical layer are not visible to the IP routing system.

B. Challenges: Scale and Control

Unfortunately much of the existing path diversity in the Internet topology is not currently exploited, due to challenges in scalability and incentive compatibility. Multipath routing can be difficult to scale due to the extra overhead in both the *control plane* and *data plane* of the routers. In the control plane, routers run routing protocols to compute the forwarding table and disseminate information between routers. In the data plane, routers map incoming data packets to an outgoing link based on the forwarding table. There are two types of overhead in moving to flexible multipath routing:

- *Control-plane overhead:* First, exchanging path-level or link-level information will consume extra bandwidth and processing resources. Second, storage overhead at each router grows with the number of paths. Third, computing multiple paths requires more computational power.
- *Data-plane overhead:* Often, an extra header or label is added to a packet to control (directly or indirectly) the path it will follow. The forwarding tables at the source (though not necessarily the intermediate routers) will have multiple entries for each destination, thus consuming more memory.

The ultimate flexibility would be for sources to see the entire Internet-wide topology and utilize all paths to each destination. This will create a large scaling problem for the source, however, since there are over 25,000 ASes in the Internet and many more paths. Even if the scalability challenges were surmountable, ISPs are unlikely to give control of how traffic flows through their networks to others. Luckily, scale and control are well-aligned goals. When an AS hides information for better control, the other ASes receive less information and the system as a whole is more scalable.

In this survey, we focus on multipath routing schemes which are scalable without sacrificing much control. We start by reviewing how Internet routing works today in Section II, with an eye towards existing limitations. Section III covers a range of solutions for flexible forwarding such as tunneling. Section IV discusses techniques for a single AS to achieve multipath routing, without requiring cooperation from other ASes. The impact of a single AS on end-to-end path performance is limited, however, and more end-to-end paths will be available if ASes cooperated. Section V discusses techniques which only require a modest amount of cooperation between ASes. Finally, we conclude in Section VI.

II. Internet Routing Today

In this section, we will introduce the key Internet routing protocols used today. To exchange reachability information between ASes, routers use the Border-Gateway Protocol (BGP). BGP is a *path-vector* protocol, where routing decisions are made based on local policies. Inside an AS, routers communicate using an Interior Gateway Protocol (IGP). Most ISPs run *link-state* protocols that perform shortest path routing based on configurable link weights. The link weights in IGP and the policies in BGP are configured by human operators to satisfy management objectives.

A. Interdomain: Path Vector Protocol and Multihoming

Figure 1a represents an AS-level topology, where each cloud is an AS and each link represents a physical connection, as well as the existence of a business relationship between two ASes. In a path-vector protocol, the *entire routing path* is exchanged between neighbors. Edge routers in each AS learn multiple paths to reach a particular destination and store all of them in a routing table. From the list of paths, a router then applies a set of policies to select a *single active route*. This active route may or may not be advertised to a neighboring AS depending on the business relationship. Using a path vector protocol for BGP is good for both control and scale. A path vector protocol allows a flexible set of policies to be implemented locally, so that each AS can control both their outgoing traffic and the amount of transit traffic through them. For example, as an enterprise network, AS D in Figure 1a would not want to serve as a transit AS between B and C. In addition, since there are over 25,000 ASes, sending all received paths to all neighbors would be unscalable.

On the other hand, today's BGP has some limitations as a single-path protocol. Since only the active path is advertised, customer ASes are blocked from seeing alternate paths, including ones they might prefer. By only using the active path, each AS does not have fine-grained control, and can only balance traffic at the prefix level. Still, extending BGP to a multipath protocol is challenging because if an AS propagates more than one path, it would effectively be handing off some of the control over its own network to other ASes. In general, finding the correct incentives for ASes to propagate multiple paths is an open challenge.

One exception is multihoming, where a *stub* AS pay to be connected to more than one ISP. If an AS has no customer networks, then it is referred to as a stub AS. The use of multihoming has seen a dramatic increase in recent years for two main reasons. First, as more and more enterprises rely heavily on the Internet for their transactions, availability is of primary importance. Second, multihoming can be used to drive down the cost of Internet access. For example, the multihomed network can use a cheaper ISP for general traffic and a more expensive but better ISP for performance-sensitive traffic [10]. In Figure 1a, AS D is multihomed to AS B and AS C. Despite having two upstream routes, AS D can only load balance between the two at the prefix level, and only forwards on a single path. So multi-homing provides additional paths to customer networks, but not fine-grained control over them.

B. Intra-domain: Link State Protocol

Unlike the interdomain case, each AS has full control of its internal network. In addition, an AS typically has just ten or hundreds of routers, much fewer than the number of ASes in the AS-level topology. Inside a single AS, each router is configured with a static integer weight on each of its outgoing links, as shown in Figure 1b. The routers flood the link weights throughout the network and compute shortest paths as the sum of the weights using Dijkstra's algorithm. Each router uses this information to construct a table that drives the forwarding of each IP packet to the next hop in its path to the destination. Link-state protocols have several advantages. First, routing is based only on a single link metric, *i.e.* link weights. Second, to reduce message passing overhead, routers only disseminate information when the topology changes. Finally, by flooding the link weight information, each router has a complete view of the topology and associated link weights.

On the other hand, even though each router can see the whole topology, the existing path diversity is under exploited [9]. Even when alternative paths have been computed, packets towards a destination are forwarded on a single path. At most, the routers keep track of *all* shortest paths when running Dijkstra's algorithm, and then install multiple next-hops in a forwarding table entry when multiple outgoing links lie along a shortest path. In Figure 1b, we see that router i has two shortest paths to reach router j. In today's IGPs, the traffic would be divided evenly between the two paths. Even this limited version of multipath routing is useful for reducing transient disruptions. If multiple paths between a source-destination pair exists, transient loss of connectivity can be avoided when one path fails. For fast reaction to failures, some operators tune the link weights to create equal-cost multipaths [11].

Multiple shortest paths can be useful for reacting to failures and load balancing, but cannot customize paths for different applications. An existing option for operators to customize paths inside their own network is the Constrained Shortest Path First (CSPF) protocol, an extension of the shortest path protocol. The path computed using CSPF is a shortest path fulfilling a *set* of constraints. CSPF simply runs shortest path algorithm after pruning those links that violate a given set of constraints. A constraint could be minimum bandwidth required per link, end-to-end delay or maximum number of link traversed. CSPF can be useful for a range of applications, *e.g.* reserving bandwidth for a file transfer or ensuring low delay for a voice-over-IP call.

III. Towards Flexible Forwarding

The most prevalent forwarding mechanism in the Internet today is *destination-based hop-by-hop forwarding*. Each router identifies the outgoing link based on the destination address from the IP packet header, then match with an entry in the *local* forwarding table based on the longest prefix. For example, in Figure 1b, an internal router will forward a packet towards j, independent of where the packet came from. Destination-based hop-by-hop forwarding keeps the forwarding table small, but cannot implement all forwarding policies. For example, in Figure 1a, if AS A wanted to reach D via (B, C), but B wanted to reach D directly, then A is forced to use path (A, B, D). Even when the forwarding table contains multiple next hops for the same destination, common practice would divide the traffic evenly amongst the multiple paths.

In this section, we describe alternatives schemes which forward traffic *flexibly* over multiple paths. This is useful for customizing paths for different applications. In order to decide which path should carry a packet, a packet is first classified, and then mapped to corresponding path(s).

- **Packet Classification:** Packets are classified based on the requirements of the application [12]. For example, the application may want low delay, high throughput or a secure path. The application could be defined by a prefix, a destination or a TCP flow (source address, destination address and port number). Packets within the same flow are normally classified in the same way, and can be marked using the Type of Service bits in the IP header.
- **Mapping Packets to Paths:** The edge routers can measure (or infer) path properties, and also classify the paths to correspond to the classes for the packets. By looking at the IP header, flows can be mapped to path(s) in the same class. In Section III-A, we discuss how traffic can be forced to flow on an alternate path.

If multiple paths are available for packets in a particular class, then further benefits are possible by using multiple paths simultaneously. The amount of traffic to place on each path can be calculated to satisfy traffic management objectives. For example, the objective could be to adapt to traffic dynamics such as network congestion. The percentage of traffic on each path is commonly referred to as *splitting percentage*, which can be dynamically adapted. In Section III-B, we survey the pros and cons of splitting traffic at different granularities.

A. *Forwarding on Alternate Paths*

Tunneling is a widely available alternative to destination-based hop-by-hop forwarding that offers much more flexibility. At a high level, tunneling establishes a *logical* link between two routers (or hosts). Forwarding packets over a tunnel usually involve "pushing" a header (or label) at the tunnel ingress and "popping" the header (or label) at the tunnel egress, in a process called *encapsulation*. For example, in Figure 2, a packet going from B to F could be encapsulated to ensure it travels through E. At B, an extra header would be "pushed" on the packet to ensure it travels towards E. Once the packet reaches router E, the extra header would be "popped" from the packet, then E would forward the packet to F hop-by-hop. Encapsulation can be implemented through IP-in-IP tunnels or MultiProtocol Label Switching (MPLS). MPLS is a label-based forwarding mechanism. In essence, each router would have a label-based forwarding table. A packet can be encapsulated by adding a label onto it. In either case, encapsulation requires packets to carry an extra label or an extra IP header. In the case of MPLS, each router also stores the label-based forwarding table, although a label-based look-up is simpler than matching the longest prefix of the destination address.

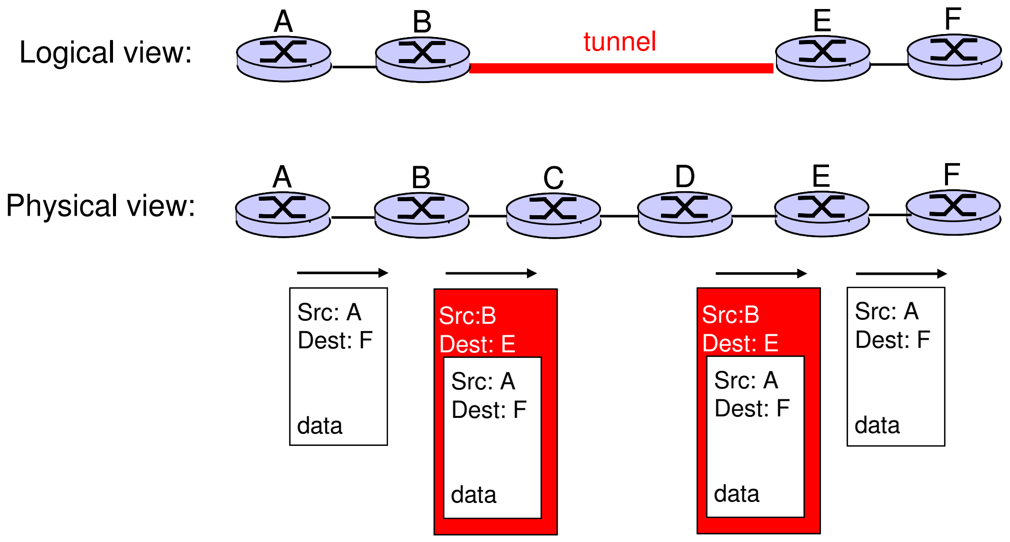


Fig. 2. Illustration of how a tunnel works.

The path between the tunnel ingress to tunnel egress can depend on the underlying routing protocol, or the entire path can be specified explicitly. Encapsulation alone is often sufficient for most application needs such as directing a packet to a particular egress point or through a particular AS. When the intermediate nodes (routers or ASes) only depend on the underlying protocol, it has the added advantage of adapting to topology changes automatically. For example, in Figure 2, B could forward the packet towards E hop-by-hop. This implies if the link from C to D fails, the encapsulated packets would *transparently* switch to another path. Still, by only specifying the endpoints of the tunnel, it is difficult to satisfy certain applications needs, *e.g.* end-to-end bandwidth requirement. So for those specialized applications, explicit routing is used instead.

Explicit routing specifies every router (or AS) along the path. For example, in Figure 2, the path sequence (A, B, C, D, E, F) could have been an explicit path for certain packets traveling from A to F. Then A would know to forward to router B next based on the path information in the packet header. Explicit routing can be implemented by MPLS as a combination of Constrained Shortest Path First (CSPF) and Resource Reservation Protocol (RSVP). CSPF selects the path using a variety of metrics, while RSVP is the signaling protocol used to set-up the path. RSVP establishes a hop-by-hop chain of labels to represent the path and it reserves bandwidth along the path by signaling in advance. At one end of the path, a label would be pushed onto the packet based on information from the packet header such as source address, destination address and port numbers. Each intermediate router would do a label look-up to find the outgoing label and outgoing link. In this case, the outgoing label would be signaled by RSVP ahead of time. While the packets themselves just carry a label, explicit routing can impose significant overhead on the routers as it requires a label-based forwarding table and signaling to establish the path.

B. Flexible Splitting Amongst Multiple Paths

Some applications may wish to send traffic on multiple paths, we refer to percentage of traffic on paths as *splitting percentages*. For example, an application may send 40% of traffic on one path and 60% of traffic on another. To achieve a given splitting percentage, traffic can be switched onto different paths using four major techniques: round-robin, hashing, flow cache and flowlet cache [13]. Each technique has a different trade-off point in terms of overhead, splitting percentage accuracy and out-of-order packets.

A weighted **round-robin** will switch traffic at the granularity of packets. Since packets are small in size, round-robin can achieve very accurate splitting percentages dynamically. Round-robin also adds very little extra overhead on today's forwarding functions. The downside is that since different paths between the same source-destination pair often have different delays, some packets which belong to the same Transmission Control Protocol (TCP) flow could arrive out-of-order. This is problematic as TCP considers out-of-order packet delivery as a sign of network congestion, and consequently, TCP would slow down the transfer. If the paths have very similar delay, however, then weighted round-robin is a good choice due to its low overhead and accurate splitting percentages.

Hashing avoids more out-of order packets than round-robin, though hashing can lead to inaccuracies in the splitting percentages. The most common implementation today involves first dividing the hash space into weighted partitions corresponding to the outbound paths. Then packets are hashed based on their header information, and forwarded on corresponding path. Hashing ensures in-order delivery of most packets since a flow is likely to be mapped to a specific path for its entire duration. On the other hand, since flows vary drastically in their sizes and rates, it is difficult to realize accurate splitting percentages. Finally, if splitting percentages change or a path fails, a flow is likely to be hashed onto a different path than before, possibly still causing a few out-of-order packets during the transition.

The best way to avoid out-of-order packets is to implement a **flow cache**. A flow cache is a forwarding table that keeps track which path each active flow is on to ensure packets belonging to the same flow always go on the same path. A flow is defined by the following attributes in the packet header: source IP address, destination IP address, IP protocol number, source port and destination port. Another advantage of flow caching over hashing is that when new flows arrive, they can be placed on any path, which leads to better control of dynamic splitting percentages. Although dividing traffic at the granularity of flows is still less accurate than dividing traffic at the granularity of packets. The big downside is that a high speed link could easily carry tens of thousands concurrent flows [13], requiring a significant amount of additional memory in the router.

It is possible to reduce data-plane overhead and improve splitting ratios by dividing traffic at the granularity of packet-bursts, using a **flowlet cache** [13]. Also, if the time between two successive packets is larger than the maximum delay difference between multiple paths, then the second packet can be routed on any available path without reordering. Since flowlets only lasts for a packet burst, there are many fewer active flowlets than active flows. Therefore, the table required by flowlet cache is much smaller than flow cache [13]. In addition, flowlet switching always achieves within a few percent of the desired splitting percentage, for both static and dynamic splits, without reordering any packets. Overall, flowlet cache would be the best choice for most applications, although it is not yet implemented in routers today.

IV. Multipath Routing by a Single AS

Each AS has complete control over its own network, making it easier to adopt new techniques. In this section, we present techniques which can be adopted by a single AS: each AS can exploit its internal path diversity, and multihomed stubs can achieve fine-grained control over its end-to-end paths. In particular, we focus on light-weight schemes which avoid tunnels.

A. Intradomain: Non-shortest Paths within an AS

Each AS can select its own IGP, allowing it to change the protocol without requiring cooperation from others. In link-state protocols, since link weights and topology information are already flooded to all routers, no extra overhead is needed for information dissemination. One way to extend the link-state protocol for multipath is to compute K-shortest paths rather than just the shortest path. This is cumbersome for several reasons. To start with, computing the K-shortest paths is more computationally intensive than computing the shortest paths, more precisely, it is $O(N \log N + KN)$ for a network with N routers [14]. The forwarding-table size would also grow to reflect the increase in number of paths per destination. Perhaps the biggest overhead increase is in the data plane, where tunneling is required. If each router does destination-based hop-by-hop forwarding, then there is no guarantee packets will be traveling on the K-shortest paths from source to destination. In order to ensure that packets indeed travel on the K-shortest paths, it is necessary to create K tunnels. This is significantly more cumbersome than the current hop-by-hop forwarding.

It is possible to retain hop-by-hop forwarding by forwarding packets along "downward paths" [15], [16], loop-free paths which makes forward progress toward the destination. Each router can make local forwarding decisions based on the cost of the shortest path through each of its neighbors [16]. For example, by forwarding to neighbor routers who have a shorter path to the destination, the routing is guaranteed to be loop-free [15]. To discourage the use of longer paths, diminishing proportions of the traffic would be directed on the longer paths. For example, in Figure 1b, i has two outgoing links along downward paths to j. The downward paths have costs $(8, 9)$, then less traffic would be placed on the path with cost 9 [16]. Under this scheme, there is no extra overhead associated with path computation, *i.e.*, each router will simple run Dijkstra's shortest path algorithm. The forwarding tables will store more next-hop routers than shortest path routing. There will be slightly more data-plane overhead in order to implement the splitting percentages, as explained in Section III-B. Still, each router can make local forwarding decisions without the use of tunnels.

Yet another approach is to run multiple instances of the link-state routing protocol [17], [18]. Instead of having a *single* link weight associated with each link, each link has a *vector* of weights. Each instance of the link-state protocol can just compute the shortest path and create a forwarding table for each topology. The vector of weights doesn't lead to the K shortest paths, but rather than shortest path relative to each of K sets of link weights. Each set of link weights can be tuned independently. This enables customizing paths for different applications; for example, one set could be tuned for high throughput and one set could be tuned for low delay. The link weights could even be specialized to handle different failure scenarios [17]. In the control-plane, if K routing instances are run simultaneously, the control-plane overhead would be exactly K times as much as shortest-path routing. Running multiple routing instances is more computationally intensive than forwarding on the downward paths, but can apply to a wider range of scenarios. In the data-plane, there are two ways to forward packets on the multiple topologies. The simpler (and more restrictive) way is for each packet to belong to a single topology [17]. Further benefits are possible when packets can switch between topologies based on network conditions [18].

B. Interdomain: Flexible Splitting by a Multihomed Stub

So far, we have described how to exploit path diversity inside a single AS. Next, we will examine how to exploit interdomain path diversity. Many routers learn multiple interdomain paths and could conceivably split traffic over them by installing multiple next-hops in the forwarding table. This is not done in practice due to the extra control-plane overhead. For transit ASes, edge routers would need to announce multiple paths to neighboring ASes, and the neighboring ASes would now need to store multiple paths. In addition, tunneling would be needed to direct packets on any non-default path, as explained in Section III.

For edge routers in a stub AS, however, there is no need to propagate any of the learnt paths as it has no customers. In addition, the router already stores multiple paths locally for reaching the same destination. Therefore, stub networks are natural places to deploy flexible splitting. Since applications are run at the edge, the stub network also has direct knowledge of the application requirements. Flexible splitting enables an AS to customize paths for different classes of traffic and balance load across multiple paths. Balancing load between multiple classes allows for efficient use of network resources and can avoid potential routing oscillations. For example, if all traffic is forwarded on the least-delay path, route oscillations can occur [19]. Luckily, flexible path selection adds very little extra overhead on the data plane for a stub AS, since choosing an outgoing link determines the entire path a packet will follow and no tunneling is required.

V. Multipath Routing with Inter-AS Cooperation

When multiple ASes cooperate, even more paths are available than when an AS acts alone. In addition to scalability, the challenge is to find incentives for inter-AS cooperation. In this section, we focus on schemes which access additional paths with only limited cooperation between ASes. Sources can *directly* encapsulate packets to be sent through a *deflection point*, an end host or edge router which lies on an alternate path. This only requires a bilateral agreement between two parties. Sources can also deflect packets *indirectly* via *tagging*, where a few opaque bits in the packet header are used to indicate dissatisfaction with the current path. Tagging requires more inter-AS cooperation than encapsulation since routers in participating ASes will need to be modified to recognize and process tags.

A. Encapsulation: Forwarding through a Deflection Point

Encapsulation can be used to explicitly force traffic onto an alternate path with better performance properties. A packet would be encapsulated to first arrive at the deflection point, then follow that deflection point's default path to destination [15]. Deflections can occur at the application layer or the network layer. At the application layer, end-hosts can form an overlay and deflect through each other [20]. Without control-plane information, choice of a deflection point is only guided by data-plane measurements. At the network layer, deflections can occur between edge routers based both control-plane and data-plane information [21].

The easiest way to access another path is by deflecting through another end host, which does not require cooperation from or coordination between ISPs. An *overlay* or logical topology can be established between end hosts using tunnels [20]. If a path with better performance is found through probing, packets will be deflected through another end host as seen in Figure 3. Similar to multihoming, each end host can measure end-to-end performance properties, but does not know the actual path to the destination. Each host does not need to advertise additional paths and only needs to maintain a few extra routing entries. To compensate for the lack of control-plane information, however, it is necessary to probe aggressively, which imposes significant measurement overhead on the network, particularly as the overlay grows in size. In fact, deflections through an overlay, does not scale beyond tens of end hosts due to the measurement overhead [20]. In addition, sending traffic through other end hosts consumes edge link bandwidth and potentially increases latency.

A more scalable and efficient approach is for ISPs to provide alternative paths [21]. As seen in Figure 3, the deflection point is an edge router inside an AS, rather than an end host. While this approach requires more

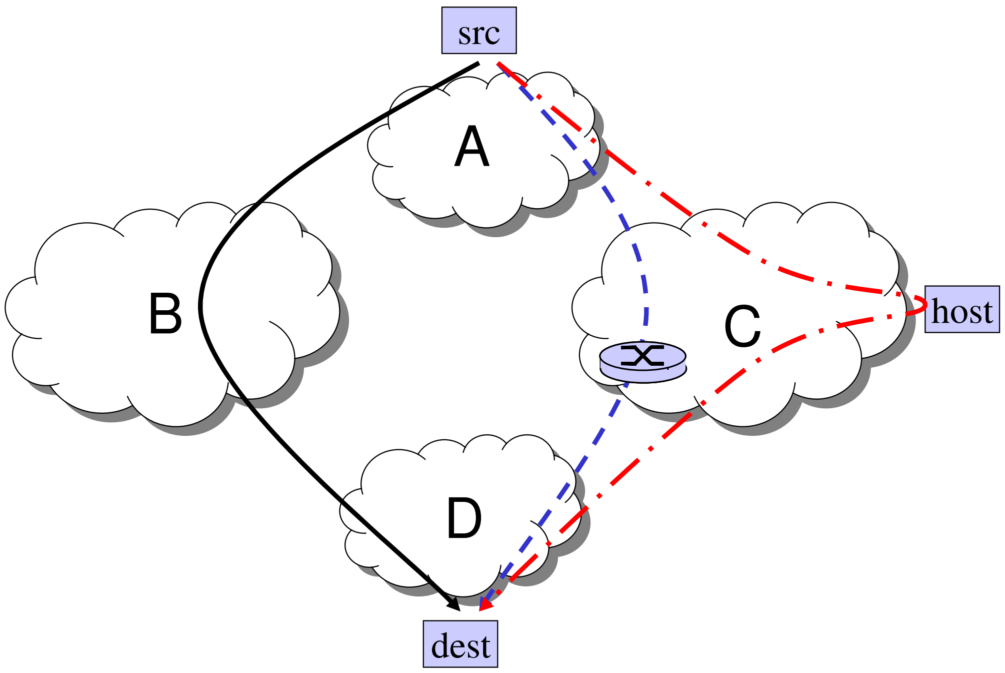


Fig. 3. The default path, shown in solid line is through AS B. Deflection through AS C is possible either with an overlay (dot-dash line) or through an ISP (dashed line).

cooperation from (and between) ISPs, it is incrementally deployable. To ensure scalability, an AS would only request an alternative path (perhaps with certain properties) from another AS if it is unhappy with its default path. For example in Figure 3, the source could request an alternative path from its provider AS A for reaching the destination D, AS A can then choose to export the alternative path (A, C, D) (possibly for a price). In this case, encapsulation would be used to deflect the packets through AS C. The amount of control-plane overhead is directly proportional to the portion of ASes unhappy with their default path. An AS unhappy with its default path will store extra routes, and its neighbor will need to advertise extra routes.

B. Tagging: Requesting an Alternate Path

An alternative to encapsulation is for end-hosts to *tag* their packets to request for an alternate path [18], [15]. Upon receiving a tagged packet, a router can choose to forward it on the default path or an alternate path based on the tag number. Alternative paths inside an ISP can be constructed by one of the methods described in Section IV-A. An alternate path could involve an different egress point and consequently a different AS-level path.

Tags can specify known alternative paths, but direct knowledge of multiple paths is not required [15]. Tagging without path visibility is effective when an end-to-end path is undesirable due to one particular segment of the path. For example, the path could contain a low capacity link, a high delay link or a point of congestion. In these cases, routing around the problem link or router does not require direct knowledge of the route. By trying out a few tag numbers, the source AS is likely to find a better path.

Tagging is quite scalable in the control plane since intermediate ASes do not need to disseminate extra information or store AS-level paths. An intermediate AS can merely exploit the path diversity inside its own domain. There is little extra data-plane overhead, since the tag can use some rarely used bits in the existing IP header [15]. The only extra data-plane overhead is for an ISP to determine whether or not to process the received tag, and direct the packet onto an alternate path based on its tag number. Although tagging imposes less overhead than forwarding through a deflection point, it is unclear what incentives are required for an ISP to provide alternate paths.

VI. Conclusions

The ability to forward traffic on multiple paths would be useful for customizing paths for different applications, improving security, reacting to failures and balancing load. Yet today's routing protocols select a single forwarding path for each destination due to two barriers: scale and control. In this paper, we survey a variety of incrementally deployable techniques which achieve flexible multipath routing. We start with the additional data-plane support required for flexible forwarding. Then we examine how a single AS can achieve multipath routing by extending multihoming practices and link-state protocols. Finally, we describe how deflections can access additional end-to-end paths with minimal cooperation between ASes.

While this survey demonstrates it is possible to efficiently expose path diversity, there remain challenges to Internet-wide deployment. First, the incentives required for each AS to give up some of the control is not always clear. Second, designing a measurement infrastructure to support path classification is challenging. One reason is that measurements of path performance can be inherently inaccurate, for example, round-trip time estimation is a classic challenge. In addition, the inaccuracies can be even greater in a competitive environment where other ASes may treat probing packets differently than data packets. Last, but not least, the trade-off overhead and performance should be quantified to guide judicious choices. Overall, we believe exploring challenges surrounding flexible multipath routing is a fruitful avenue for future research.

References

[1] D. Wendlandt, I. Avramopoulos, D. G. Andersen, and J. Rexford, "Don't secure routing protocols, secure data delivery," in *Proc. SIGCOMM Workshop on Hot Topics in Networking*, November 2006.

[2] E. Gustafsson and G. Karlsson, "A literature survey on traffic dispersion," *IEEE Network Magazine*, vol. 11, pp. 28–36, March 1997.

[3] J. M. McQuillan and D. C. Walden, "The ARPA network design decision," *Computer Networks*, vol. 1, pp. 243–289, August 1977.

[4] D. Bertsekas, "Dynamic behavior of shortest-path routing algorithms for communication networks," *IEEE Trans. Automatic Control*, pp. 60–74, February 1982.

[5] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *Proc. ACM SIGCOMM*, August 2005.

[6] J. He, J. Rexford, and M. Chiang, "Don't Optimize Existing Protocols, Design Optimizable Protocols," in *ACM SIGCOMM Computer Communication Review*, October 2007.

[7] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, 2001.

[8] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," in *Proc. ACM SIGCOMM*, August 1999.

[9] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker, "Characterizing and measuring path diversity of Internet topologies," in *Proc. ACM SIGMETRICS*, June 2003.

[10] R.Gao, C. Dovrolis, and E.W.Zegura, "Avoiding oscillations due to intelligent route control systems," in *Proc. IEEE INFOCOM*, April 2006.

[11] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of IP restoration in a tier-1 backbone," *IEEE Network Magazine*, March 2004.

[12] I. Matta and A. U. Shankar, "Type-of-service routing in datagram delivery systems," *IEEE J. on Selected Areas in Communications*, vol. 13, pp. 1411–1425, October 1995.

[13] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic Load Balancing Without Packet Reordering," in *ACM SIGCOMM Computer Communication Review*, June 2007.

[14] D. Eppstein, "Finding k shortest paths," *SIAM J. Computing*, vol. 28, no. 2, pp. 652–673, 1998.

[15] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," in *Proc. ACM SIGCOMM*, August 2006.

[16] D. Xu, M. Chiang, and J. Rexford, "DEFT: Distributed exponentially-weighted flow splitting," in *Proc. IEEE INFOCOM*, May 2007.

[17] A. Kvalbein, T. Cicic, and S. Gjessing, "Post-failure routing performance with multiple routing configurations," in *Proc. IEEE INFOCOM*, May 2007.

[18] N. Feamster, M. Motiwala, and S. Vempala, "Path splicing with network slicing," May 2007. Georgia Tech Technical Report GT-CS-07-04.

[19] R. Keralapura, C.-N. Chuah, N. Taft, and G. Iannaccone, "Can coexisting overlays inadvertently step on each other," in *Proc. International Conference on Network Protocols*, November 2005.

[20] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proc. ACM SOSP*, October 2001.

[21] W. Xu and J. Rexford, "MIRO: Multi-path Interdomain ROuting," in *Proc. ACM SIGCOMM*, August 2006.