Symbiotic Relationships in Internet Routing Overlays

Cristian Lumezanu, Randy Baden, Dave Levin, Neil Spring, Bobby Bhattacharjee
University of Maryland

Abstract

We propose to construct routing overlay networks using the following principle: that overlay edges should be based on mutual advantage between pairs of hosts. Upon this principle, we design, implement, and evaluate PeerWise, a latency-reducing overlay network. To show the feasibility of PeerWise, we must show first that mutual advantage exists in the Internet: perhaps contrary to expectation, that there are not only "haves" and "have nots" of low-latency connectivity. Second, we must provide a scalable means of finding promising edges and overlay routes; we seek embedding error in network coordinates to expose both shorter-than-default "detour" routes and longer-than-expected default routes.

We evaluate the cost of limiting PeerWise to mutually advantageous links, then build the intelligent components that put PeerWise into practice. We design and evaluate "virtual" network coordinates for destinations not participating in the overlay, neighbor selection algorithms to find promising relays, and relay selection algorithms to choose the neighbor to traverse for a good detour. Finally, we show that PeerWise is practical through a wide-area deployment and evaluation.

1 Introduction

We propose *mutual advantage* as a principle for the construction of routing overlay networks: overlay edges should exist only between hosts that benefit from each other's resources or position in the network. Hosts negotiate connections based strictly on mutual advantage, and overlay paths follow only these connections.

Several distributed protocols and applications use *mutual advantage* as part of their design. BitTorrent [5] peers that download the same file trade blocks the other is missing. In backup systems [7], nodes store replicas of files for each other. Autonomous systems in the Internet negotiate peer-to-peer agreements to provide low-cost connectivity to each other's customers [9].

Bringing *mutual advantage* into the design of routing overlays has several benefits. First, mutual advantage induces better cooperation among nodes. Incentives to participate become simpler, and long-lived, fair connections appear. Building systems grounded in incentives for cooperation makes them robust to misbehavior and selfishness [23, 29]. Second, users could freely discriminate among the connections that they allow and would have the ability to explicitly say how much service they want to contribute. Finally, mutual advantage avoids the tragedy of the commons in routing overlays, when only a few, well-connected nodes provide transit. It keeps the trades of connectivity fair, in contrast to file-sharing where universities are net providers of content [27].

In this paper, we present the design, implementation, and evaluation of PeerWise, a latency-reducing routing overlay based on *mutual advantage*. PeerWise scalably discovers detour routes: "indirect" one-hop paths that have lower round trip latency than the "direct" path.

In a previous paper [17], we presented ideas that support a mutually advantageous latency-reducing overlay: that mutual advantage is common in the context of Internet latencies and that embedding error in network coordinate systems, such as Vivaldi [8] or GNP [20], could be used to scalably discover detours. However, we did not evaluate the potential and limitations of mutual advantage, nor did we design or implement a system to take advantage of the existing detour routes. In this work, we show that a mutually advantageous latency-reducing overlay is feasible and efficient, and that detours toward popular destinations are available. We design, implement, and evaluate a system that finds these detours.

We describe our contributions next.

First, we use a measurement-driven simulation to show the potential of PeerWise (§4). We collect two latency data sets to find what fraction of detours exist subject to the mutual advantage requirement and, independently, can be found by embedding error. The mutual advantage requirement reduces the number of destinations reachable via detour by approximately half, yet even popular websites, using content distribution services such as Akamai, are reachable by PeerWise-found detours. Only 5% of potential detours are missed by embedding error.

We next describe the design of PeerWise in two main parts: mechanism (§5) and policies (§6). We implement a *virtual network coordinate* approach to find coordinates for the destinations that do not participate directly in the overlay. *Neighbor tracking* determines the set of nodes that are more likely to offer detours by remembering those neighbors with high embedding error in the coordinate space. *Pairwise negotiation* establishes connections

promising mutual benefit while the *maintenance* component ensures that each node receives approximately as much benefit as it provides.

The second part of the design focuses on the decisions that each PeerWise node makes. We evaluate neighbor selection and relay selection algorithms. We show that coordinates can be used to choose among detours. Our environment is quite different from previous work on latency prediction using coordinates. Instead of focusing on source-to-destination, we must choose a source-to-relay-to-destination path based on a relay coordinate known to have high embedding error and a destination coordinate that may be stale or inaccurate.

Finally, we describe the implementation of PeerWise and its evaluation on PlanetLab (§7). We show that PeerWise nodes find detours to popular destinations, that these detours are stable, and that they offer significant latency reductions. Most detours last for a long time and are obtained using only one mutually advantageous peering. We then show how PeerWise detours translate into real life and whether user applications can benefit.

2 Related Work

Routing overlays, such as RON [2], Detour [28], SOSR [11], and OverQoS [31], promise to provide more-reliable or faster paths through the Internet. They forward packets along links in self-constructed meshes and make routing decisions without support from routers or ASes. RON [2] builds a fully connected mesh and monitors all edges. When the direct path between two nodes fails or has performance problems, communication is established through the other overlay nodes. Nakao *et al.* [19, 18] use static AS-level topology and geographical distance information to eliminate redundant overlay edges and improve scalability. Gummadi *et al.* show that all-to-all measurements are not necessary to find reliable paths: routing through a randomly chosen intermediary node is enough [11]. Similarly, we show that faster-than-default paths can be discovered with limited information: network coordinates and latencies to a few other nodes are sufficient.

Various file-swarming systems [5, 15, 30] apply tit-for-tat-like schemes to induce cooperation among peers. Tit-for-tat applies when there is a *mutual interest* among peers, which is common in file swarming; for any pair of peers, one may have blocks the other does not. We show that, perhaps surprisingly, mutual interest is common in low-latency routing in the Internet as well, and that locating nodes of mutual interest can be done in a decentralized fashion.

The requirements imposed by PeerWise on who can connect to whom are reminiscent of the bilateral connection game (BCG) [6], a special case of network formation game. In BCG, a link between two nodes is established only with the consent of both nodes, similarly to PeerWise. However, nodes construct links that minimize the cost of reaching other participating nodes, whereas in PeerWise, nodes create peerings that offer detours to destinations that do not necessarily participate.

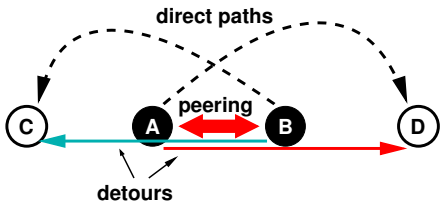


Figure 1: Obtaining faster paths with PeerWise: A discovers a detour to D through B; B also finds that it can reach C faster if it traverses A; A and B create a mutually advantageous peering that they both use to reach their destinations more quickly.

3 PeerWise Philosophy

In this section, we present an overview of PeerWise. We outline the two properties on which PeerWise is based: that mutual advantage is common in the Internet latency space and that network coordinate systems can help indicate detour routes. A previous paper [17] describes these properties in more detail. We then argue that it has the potential to be applied to a wide range of applications.

3.1 Overview

The key idea of PeerWise is that two nodes can cooperate to obtain faster end-to-end paths without either being compelled to offer more service than they receive. Peers negotiate and establish pairwise connections to each other based strictly on mutual advantage. Figure 1 shows an example. The default Internet path between two nodes is the *direct path*. A shorter, alternate path having one intermediate hop is a *detour*, using terminology from Detour [28]. Node A discovers a faster path to D via B. However, B will not help A unless A provides a detour in exchange. Since there is a shorter path from B to C going through A, A and B can help each other communicate faster with their intended destinations.

Mutual Advantage Each participant in overlay networks contributes resources in exchange for the resources of others. Unfortunately, free access and unrestricted demand may lead to over-utilization of certain resources, especially those of well-provisioned nodes. This tragedy of the commons occurs because the benefits of using common resources accrue to individuals, while the costs of exploitation are shared by the resource providers.

Pairwise peerings based on mutual benefit offer users an effective way to resolve the tragedy of the commons, as they can freely discriminate among the connections they allow. However, such decentralized policy may be costly: if nodes accept only peerings that are mutually

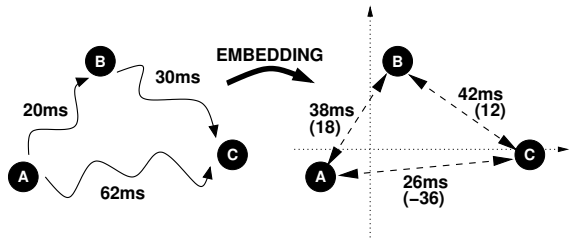


Figure 2: Embedding three points that form a TIV into a metric space introduces inaccuracies. The numbers in parentheses represent embedding errors.

advantageous, but mutual advantage is rare, the benefit of the overlay is lost. In Section 4, we show that mutual benefit is common and that a majority of nodes are in a position to both provide and receive service.

Network Coordinates Embedding Error Measuring and distributing the all-pairs latencies required to find detours would limit the scalability of a latency-reducing overlay. Instead, PeerWise detects triangle inequality violations (TIVs) and uses them to predict good detours.

Three nodes in the Internet form a TIV when the RTT between two of them (the *long side* of the TIV) is greater than the sum of the RTTs to the third node (the *short sides* of the TIV). The left side of Figure 2 shows an example TIV. Pairs of nodes that are long sides in TIVs may benefit from detours; pairs that are short sides may be part of detours.

To find TIVs scalably, PeerWise uses network coordinates. A network coordinate system associates nodes with points in a metric space such that the distance between the points estimates the real latency between nodes. Since TIVs are not allowed in metric spaces by definition, this embedding may result in high errors on the edges of the triangle (see Figure 2). The error for the long side of the TIV will be very negative, or the error for the sum of short sides of the TIV will be very positive. Thus, a pair of nodes with a negative estimation error has a higher chance of benefiting from a shorter path; conversely, when the nodes have a large estimation error between them, they are more likely to be part of a shorter path for another node.

3.2 Where does PeerWise apply?

We expect PeerWise has the most utility for latency-sensitive traffic such as HTTP HEAD requests that check for updates to a cached file before rendering, XML-RPC requests for rapid updates of existing content such as train status or sports scores, voice traffic relayed to bypass firewalls, and online games such as first-person shooters, whose playability hinges on low-latency updates among players [3]. Existing overlay networks could benefit from using PeerWise as a latency-reducing substrate by guiding PeerWise's neighbor- and relay-selection algorithms to better suit the application's needs.

Because PeerWise focuses on reducing latency, it can find and use the low-latency paths that may not support high-bandwidth use—that is, the low-latency paths that the default routing, likely tuned for high-bandwidth, misses. Going through a peer is likely to traverse another access link that might have low bandwidth. This means that bandwidth-intensive applications, such as video streaming, are unlikely to benefit from latency reduction with PeerWise.

4 Limitations of Mutual Advantage

We assess the potential performance of a mutually advantageous latency-reducing overlay. Because we restrict detour paths to mutually advantageous peerings, we would not expect PeerWise to find the shortest detours or find detours to all destinations. We simulate using two latency data sets to show that nodes can find shorter paths to the majority of destinations for which a shorter detour exists, despite the requirement of mutual advantage. We find that mutually advantageous detours exist even for popular destinations hosted on many prefixes.

4.1 Collected Data Sets

We collected two real-world latency data sets and computed all one-hop detours between each pair of nodes.

PW-King Data Set The first data set, PW-King, contains RTTs between 1,953 DNS servers of hosts in the Gnutella network. The list of hosts was gathered by Dabek *et al.* for the Vivaldi [8] project. We use King [10] to measure all-to-all latencies between the servers. King uses recursive DNS queries to estimate the propagation delay between two hosts as the delay between their authoritative name servers. The 1,953 servers were chosen for being in the same subnet as their hosts so that better-connected DNS servers would not influence the estimates of inter-client latencies. For each pair of nodes, we kept the median of all latencies measured at random intervals for a week in February 2008. Of the 1,953 servers, we removed 238 that appeared to experience high load during the measurement, as described by Dabek *et al.* [8]. A heavily-loaded DNS server can cause King to underestimate latencies to other nodes, which can lead to false triangle inequality violations.

Popular Destinations Data Set The second data set, PL-Dest, contains RTTs from 389 PlanetLab nodes to 500 popular web servers, measured in January 2008. We selected the servers based on a ranking by the Alexa Internet Company [1] using expected and measured client access. For faster content delivery, many of the websites have multiple IP addresses; users in different geographic regions see different IPs for the same server. To gather the IP addresses associated with a website, as visible from PlanetLab, we performed DNS lookups on each of the 500 names from the 389 PlanetLab nodes. We obtained 2932 distinct IP addresses in 796 /24 prefixes. We

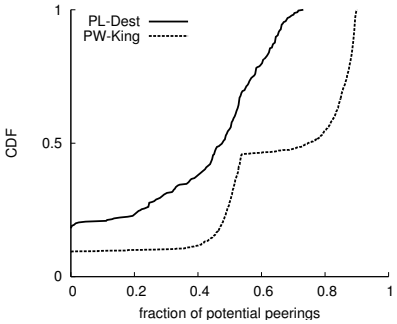


Figure 3: Distribution of the fraction of nodes with which a potential peering exists. In PL-Dest, 18% of nodes have no potential peerings; in PW-King, 50% of nodes have potential peerings with at least 75% of the other nodes.

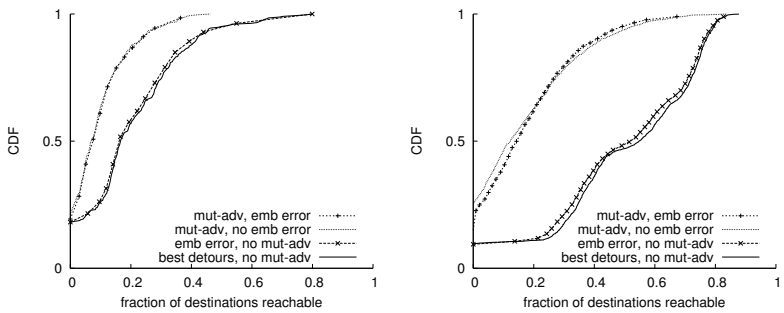


Figure 4: Distribution of the fraction of destinations reachable through mutually advantageous peerings for PL-Dest data set (left), and PW-King data set (right). In PL-Dest, few destinations can be reached by detour at all, some sources need no detours, and approximately half of the detours that could be used are lost by the mutual advantage restriction. In PW-King, all nodes have many detours available, and mutual advantage is less costly. In both, embedding error finds nearly all detours.

probed each prefix and each PlanetLab node from every PlanetLab node at random times over a week. We used the median RTT values to represent the link.

The latency collection process can produce incorrect data that may bias our results. We removed 52 servers from the final data set because we could not measure any RTT to them. Further, several PlanetLab nodes had very low latencies (< 1 ms) to most destinations. These latencies are likely caused by connection-tracking firewalls or "transparent" proxies near the PlanetLab nodes that generate spoofed responses as if from the destination. We removed those nodes from the data set since they would artificially overstate the potential of PeerWise. Our final latency matrix contains RTT values from 325 PlanetLab nodes to 718 prefixes corresponding to 448 websites.

The PW-King and PL-Dest data sets illustrate two scenarios in which PeerWise can be useful. Latency reduction on PW-King shows the potential benefit to applications a set of peers may run, such as a distributed multiplayer network game or VoIP application. On PL-Dest, reduced latency shows benefit for users accessing popular servers that would not participate in PeerWise.

4.2 Methodology

We built a simulation prototype of PeerWise to study how well it finds detours with mutual advantage and embedding error. To find network coordinates for nodes, we use Vivaldi [8]. We allow each node to communicate with all other nodes, to better study mutual advantage in isolation. When requesting detours for its destinations, a node starts with the neighbor that has the highest embedding error [17]. We evaluate alternative relay selection methods in Section 6.2.

For each pair of nodes in our data sets, we find all one-hop detours. We define a *good detour* as a detour that provides at least 10 ms and 10% latency reduction over the direct path. We consider only good detours. This cutoff helps avoid impractical or dubious detours due to measurement error. In the PL-Dest data set, we may find detours by server name: The detour path may end at a different IP address associated with the same name.

4.3 Mutual Advantage

How much mutual advantage exists in our data sets? We define a *potential peering* to exist between two nodes that can provide a detour to each other, for at least one destination, as between A and B in Figure 1. The number of potential peerings for a node represents the number of neighbors with which the node can construct mutually advantageous peerings. In Figure 3, we show a cumulative distribution of the fraction of nodes for which a potential peering exists. Each point represents a node, and its placement on the x-axis what fraction of the other nodes it shares a potential peering with. At least 50% of the nodes in either data set have have potential peerings with at least 50% of the rest of the nodes. The figure also shows that there is more mutual advantage in the PW-King data set than in PL-Dest.

Next, we show that mutual advantage sacrifices few detours. We study the fraction of destinations that each node can reach more quickly via mutually advantageous peerings in Figure 4. Each graph considers four cases to isolate the two main potential performance sacrifices: the requirement of mutual advantage (that could make detours unavailable) and relay choice by positive embedding error (that might not find them despite being possible). The solid line represents an unconstrained detour overlay. Considering mutual advantage eliminates over half of the potential destinations for many nodes. For some, mutual advantage eliminates *all* detours; trivially, these are the nodes that cannot provide service to others. Choosing among either set (constrained to mutual

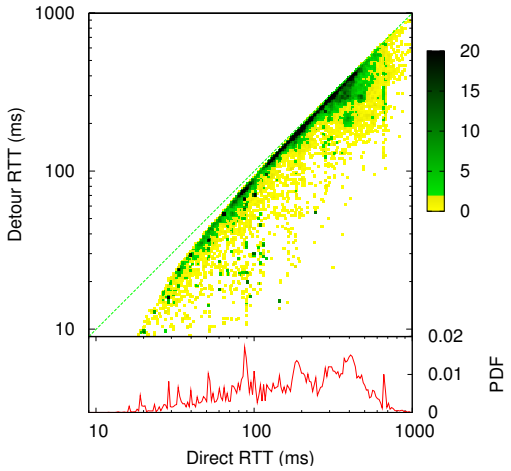


Figure 5: PL-Dest: When a detour exists, density plot of detour path RTT versus the direct path RTT (top, the color of each point represents the number of TIVs with the corresponding direct and detour RTTs), and PDF of direct path RTTs (bottom).

advantage or not) via embedding error between source and relay sacrifices very few detours (the corresponding lines are almost indistinguishable from each other in Figure 4). Mutual advantage does not impact the latency reduction to the destinations that are still reachable: only at most 12% of the median latency reduction is lost due to the requirement of mutual advantage.

4.4 Detours to Nearby Destinations

The destinations in PL-Dest include both regionally and globally popular websites. We expect that a regional website serves its pages from within the region of interest, so the direct path latencies to the destination from PeerWise nodes in that region should be small. Since the PlanetLab nodes are globally diverse, some "detours" may be for destinations unpopular in that node's region. For example, detours to popular websites in China may be less useful for nodes in Europe or North America. In Figure 5, we show that latency reduction is not limited to distant destinations. Because our rule to define a "good" detour requires at least 10 ms of reduction, few very short paths are featured. However, mutually-advantageous detours are found for direct paths too short to cross the Atlantic or Pacific oceans (< 100 ms).

4.5 Multiple-IP Websites

For faster content delivery, around 20% of the popular websites in the PL-Dest data set are served from geographically distributed locations. User requests are transparently directed to the geographically (or administratively) nearest IP address.

Using the PL-Dest data set, we compute how many nodes can find detours to each of the 448 websites and plot it against the total number of /24 prefixes of each website. Figure 6 presents the results. Each point in the plot is associated with one server name. Most websites with IP addresses in at least two prefixes can be reached faster from at least one PlanetLab node. We divide the plot into six regions and describe each in the accompanying table.

Figure 6 shows that PeerWise has the potential to be effective in reducing latency to most popular websites, even when they employ other latency-reducing techniques such as mirroring or DNS redirection.

4.6 Simulation Limitations

First, our pairwise peerings are established expecting that each destination will be accessed as often as any other. Clearly, not all destinations are equally popular, but we cannot estimate how often peers will use the peering. Our evaluation might favor VoIP applications where the endpoints are well distributed and no endpoint is orders of magnitude more popular than the others. In Section 7, we experiment with different access patterns, including random and zipf, to try to apply likely relative popularity models to traffic.

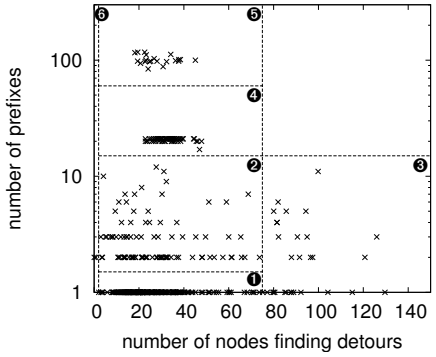
Second, the latencies between DNS servers or PlanetLab nodes may underestimate the latencies between end-hosts in the Internet. Although the latency matrix between DNS servers and PlanetLab hosts may represent the locations of hosts in the coordinate space, these data sets may not represent the latencies seen by such hosts.

Third, using PlanetLab nodes to reach popular destinations may raise questions about the validity of our evaluation. Connecting to a commercial site via a PlanetLab relay may reveal detours that would not be discovered had the relay been on the commercial network. However, Abilene and NLR, research networks that are part of Internet2, use wavelengths on fiber leased from other providers along rights-of-way shared with commercial networks. We believe that this sharing prevents research networks from providing an unfair advantage in latency reduction. We have even observed detours between PlanetLab nodes—routing within the academic network is not so latency-optimal as to prevent detours.

Finally, we do not model the bandwidth of the connection. Even though mutual latency reductions lead to a pairwise peering, limited bandwidth may prevent it from helping. As described in Section 3, we expect to use PeerWise only with latency-sensitive applications that do not require high bandwidth.

5 Design, Part I: Mechanisms

We present next the design of the PeerWise routing overlay network. In this section, we focus on the key features of PeerWise: detour detection using network coordinates for scalability, neighbor tracking for improving efficiency, and pairwise negotiations for fairness. In



	Region	Sites	Notes
❶	$p = 1$ $0 < c \leq 75$	239 (53%)	Under-provisioned, PeerWise expected to be useful.
❷	$1 < p \leq 12$ $0 < c \leq 75$	95 (21%)	Have many prefixes, PeerWise finds detours.
❸	$0 < p \leq 12$ $c > 75$	29 (6%)	Regional websites, most from China, many nodes find detours to them; perhaps not designed for global access.
❹	$12 < p \leq 60$ $0 < c \leq 100$	62 (14%)	All www.google.* websites; which IP address is chosen depends on the source, not the country suffix.
❺	$p > 60$ $0 < c \leq 75$	21 (6%)	Akamai-type destinations; finding a detour to any replica hosting center provides a detour to all sites hosted there.
❻	$p \geq 1$ $c = 0$	2 (0%)	PeerWise finds no detours to these multi-prefix sites; includes www.it168.com and www.sohu.com.

Figure 6: Detours to mirrored websites: The figure presents number of nodes that find detours (c) versus number of prefixes for each website (p). The table describes the six regions in the figure.

Section 6, we describe and evaluate the policies of each PeerWise node. We present the implementation and evaluation details in Section 7.

5.1 Virtual Network Coordinates

Every PeerWise node must compute its own network coordinate before searching for detours. We use Vivaldi [8] for network coordinates. Every node maintains a set of neighbors that it probes periodically. It uses the round trip time and the network coordinate of these neighbors to update its own coordinate. After each probe, the node computes the coordinate that minimizes the squared estimation error to all of its neighbors. To help the system converge quickly, nodes with uncertain coordinates can move farther with each measurement. Figure 7(a) shows the coordinate computation process.

A node in PeerWise must learn the coordinates of destinations to discover long or short sides of a TIV. However, if a destination is not participating in the overlay, it will not provide its own network coordinate. We therefore extend Vivaldi to allow a node to compute a virtual network coordinate for any non-participating host. We refer to non-participating Internet nodes as hosts and to PeerWise participants simply as nodes.

To generate virtual network coordinates for non-participating hosts in Vivaldi, a participating node chooses to become temporarily responsible for that host. The node runs Vivaldi on behalf of the host with one minor adjustment. Since the host is not participating in the system, it cannot manage its own neighbor set or actively gather the round trip times needed to compute the coordinate. Instead, the participating node uses its own neighbor set as the neighbor set for the host, and requests that those neighbors measure the latency to the host, as shown in Figure 7(b). Our extensions are similar to those recently described by Ledlie *et al.* [13].

Requiring all nodes to compute virtual coordinates for all non-participating destinations would limit the scalability of PeerWise. We include a gossip mechanism to disseminate the calculated coordinates throughout the

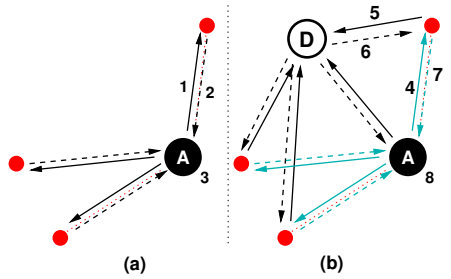


Figure 7: (a) *Computing network coordinates for a PeerWise node*: A measures RTT to its neighbors and asks for their coordinates (1); after it receives the replies (2) it computes the coordinate that minimizes the squared estimation error (3); (b) *Computing network coordinates for a non-PeerWise node D*: A asks each of its neighbors (4) to measure RTTs to D (5,6); after it receives the replies from the neighbors (7), A runs the network coordinate algorithm on behalf of D (8).

system. At fixed intervals (10s in our experiments), each node picks one of its neighbors at random, then selects a random destination and sends to the neighbor the IP address, name and virtual coordinate of the destination.

A node decides to take responsibility for a destination to which it wants to find a detour when the destination's coordinate does not yet exist, becomes too old (1 day in our experiments), or becomes unstable (where stability depends on the embedding error to other nodes). Any node can generate coordinates independently; this decentralization may allow simultaneous, redundant work. Rather than try to enforce a single consistent view of the coordinate, we allow any of these coordinates to be considered valid estimates. When a node receives a new virtual coordinate through the gossip protocol, it uses that new coordinate only if it is more stable and it was updated by the node responsible for it.

Virtual network coordinates are useful if a host is popular. If the host is not popular, a node trying to discover a detour to that host will need to compute its virtual coordinate. Since this requires that the node's neighbors

measure the round trip time to the host, the node would know all three sides of the triangle, so it would trivially discover TIVs. However, if the node knows the virtual coordinate of a host already (because the host is popular and its coordinate has been gossiped), it will only know the two adjacent sides of the triangle, and it will be able to make predictions about the third side between the neighbor and the destination. We evaluate these predictions in Section 6.3.

5.2 Neighbor Tracking

The success of our protocol depends on the ability of nodes to find other nodes to establish pairwise peerings. There are many possible relays for a node, any of which may have high embedding error with respect to the node. Recall that high embedding error for a pair of nodes indicates a higher probability that the pair is part of a detour. We use neighbor tracking to find the nodes that are more likely to offer detours. With neighbor tracking, a PeerWise node remembers extra neighbors and learns about good potential relays from its neighbors or from nearby (in latency) nodes. The *neighbors* in this section are not *relays*; they are only candidates for becoming so.

When joining PeerWise, a node bootstraps its potential neighbor set from a known PeerWise node and uses it to compute its network coordinate. Once the network coordinate is stable, the node asks its neighbors about their own neighbors, remembering those nodes with high embedding error. For example, in Figure 8, A asks for the neighbor set of B, formed of B_1, B_2 and B_3. Node A then computes the embedding error from itself to each of B_1, B_2 and B_3 and adds those nodes to which the error is most positive to its neighbor list. These nodes are the most likely to form a short side of a TIV with A.

For scalability, we limit the number of neighbors of each node. Neighbors with higher potential to offer the best detours replace less-efficient neighbors. We consider and evaluate different methods for ranking potential neighbors in Section 6.1. Because PeerWise allows a node to exchange information about neighbors with neighbors, we expect each node to have ample choices.

5.3 Pairwise Negotiation

PeerWise nodes negotiate with their neighbors to request or advertise alternate routes. As discussed in Section 3, a detour to a destination is likely to exist if the estimated distance to the destination is much smaller than the measured latency. In this case, a node asks its neighbors with high embedding errors whether they can offer a faster path (Figure 9(c)). Nodes are not limited to this simple strategy. In Section 6.2, we evaluate different policies for choosing relays and deciding whether to request detours for a destination.

Actively requesting detours may be inefficient, especially if the connection to the destination is short-lived.

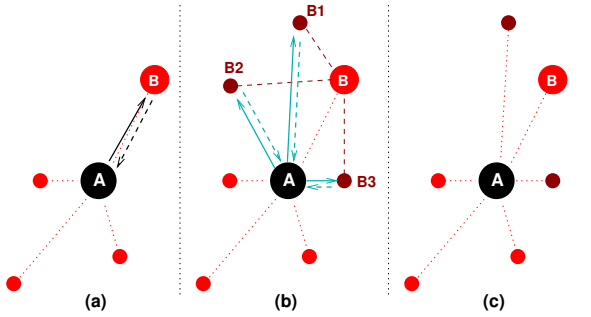


Figure 8: Neighbor Tracking. (a) A chooses the neighbor to which it has the highest embedding error and requests its neighbor set; (b) A measures RTTs to each of the nodes received from B; (c) A adds to its neighbor set those nodes to which it has a positive embedding error.

In addition, the time to find a detour may dominate the latency reduction achieved. To encourage fast detour discovery, PeerWise nodes also proactively advertise paths to popular destinations. For example, in Figure 9(d), node A observes that the link to node D, which may or may not be running PeerWise, has a high estimation error. This means that AD may be a short side in a TIV. A advertises D on all other potential short sides (*i.e.*, to all neighbors to which it has a high estimation error).

Finding detours is not enough: PeerWise is based on mutual agreements between nodes. A sender node can use a detour only if the relay that offers it also finds value in the sender. When requesting a detour from a neighbor, a PeerWise node includes a list of possible destinations to which it has high embedding error. The path to these destinations is more likely to be part of a detour for another node, as described in Section 3. Requests for detours are accepted only when both the sender and the receiver find mutual advantage in forwarding each other's traffic.

5.4 Maintenance

Each PeerWise node maintains two tables: a peering table and a negotiation table. The peering table tracks established, mutually advantageous peering relationships. The negotiation table is an antechamber for the peering table and tracks the nodes with which no peering has been established, but which are candidates for mutually beneficial peerings. Once a peering is established, the peer moves from the negotiation table to the peering table. An entry in either table is associated with a node i in the system and contains i's IP address, network coordinate, and a history of round trip times to i. The peering table adds the SLA and the utilization of the peering.

The SLA specifies the benefit that each node is expected to receive and offer through the peering. We allow different measures for the mutual benefit of a connection as long as the peering nodes both agree upon them. Two nodes can form a peering and agree that each of them

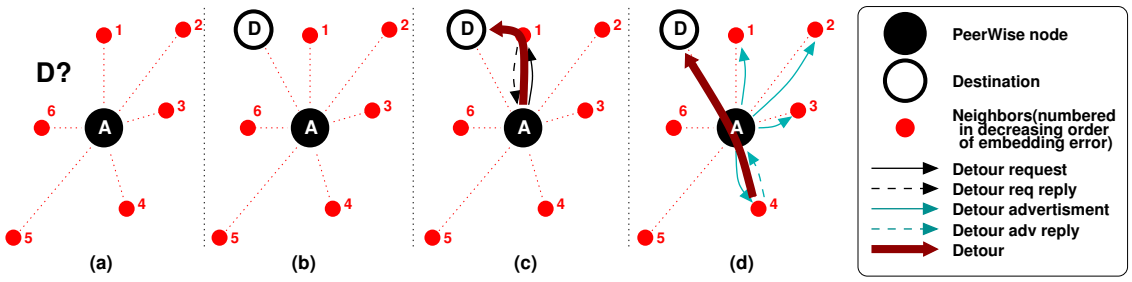


Figure 9: Detour Requests and Advertisements. (a) A wants to connect to destination D; (b) A discovers the network coordinate of D using Vivaldi or Virtual Vivaldi; (c) A requests a detour to D from the neighbor to which it has the highest embedding error; (d) A advertises its path to D to all neighbors that have positive embedding error to A.

uses the other for the same number of detours. Alternatively, they may decide that their benefit is measured in the average latency reduction obtained through each other. For example, in Figure 1, nodes A and B may establish an SLA that promises an average latency reduction of 30 ms from A to D and from B to C. In addition, two peers may establish an imbalanced peering, in which one peer benefits more than the other, if both consider the agreement to be *fair*.

Peerings may become imbalanced in time. This happens because latencies change due to failures or congestion, because peers do not respect the agreement, or because they have different connection rates to their destinations. PeerWise nodes renegotiate existing peerings to account for latency changes and to find the best detours available, as we describe in Section 7. However, we do not monitor the byte-level usage of a peering. Our focus is on finding and taking advantage of mutual latency reductions. In a previous paper [14], we describe a monitoring and accounting mechanism that ensures long-lived and mutually advantageous peerings, even when nodes are selfish or traffic demands differ.

6 Design, Part II: Policies

PeerWise is designed to be a scalable overlay for finding low-latency detours. For scalability, each node must *choose* which neighbors to maintain peerings with, *choose* among neighbors to find a relay, and *predict* whether to seek a relay for a destination.

PeerWise nodes must *learn*. Nodes compute coordinates for new destinations to help other nodes predict detours. Newly used relay paths can be instrumented so that they can be dropped if the prediction of their utility was incorrect or preserved if their utility is clear. Finally, nodes must remember a recent destinations so that a neighbor set can be customized to the likely traffic stream. Learned behavior will depend on practical deployment: for example, how frequently nodes return to the same latency-sensitive destination. In fact, as a destination is contacted again and again, PeerWise might lower its standards for a "good" detour to provide improved application performance, or try reaching the destination via relays that are not obvious candidates. In this section, we make no assumptions about the utility of learned information, and instead focus on establishing a broad base of PeerWise connections for reaching all destinations.

To study neighbor and relay selection algorithms, we collected latency measurements and coordinates for 262 PlanetLab nodes and the 448 popular web servers. We considered only the PlanetLab nodes responsive at the time of the measurement. To gather this PL-Dest-Pyxida data set, we used Pyxida [24], an implementation of the Vivaldi coordinate system. To compute coordinates for the web servers, we extended Pyxida with our virtual coordinate algorithm. Every 30 seconds, for 18 hours on January 14, 2008, we took a snapshot containing RTT measurements and coordinates (virtual and non-virtual). We use only a subset of this data: median latency over the past 10 measurements, and network coordinates, all observed after Pyxida ran for two hours (to converge).

6.1 Choosing Neighbors

Each PeerWise node must be able to decide whether a new node would offer better detours than existing neighbors. A new neighbor may provide relays toward a region of coordinate space or directly to known destinations. Deciding upon future mutual advantage is a prediction of future accesses and future performance. In this section, we evaluate the ability of a PeerWise node to predict, from coordinates and measurement, whether a neighbor will contribute.

If nodes were to contact only a few, known destinations, choosing neighbors would be simple: replace a neighbor if the new one provides a better path to an interesting destination. However, we do not expect access patterns to be nearly so predictable. Instead, we wish to determine, when a new neighbor arrives, whether it is likely to provide a shortcut to a useful region in coordinate space.

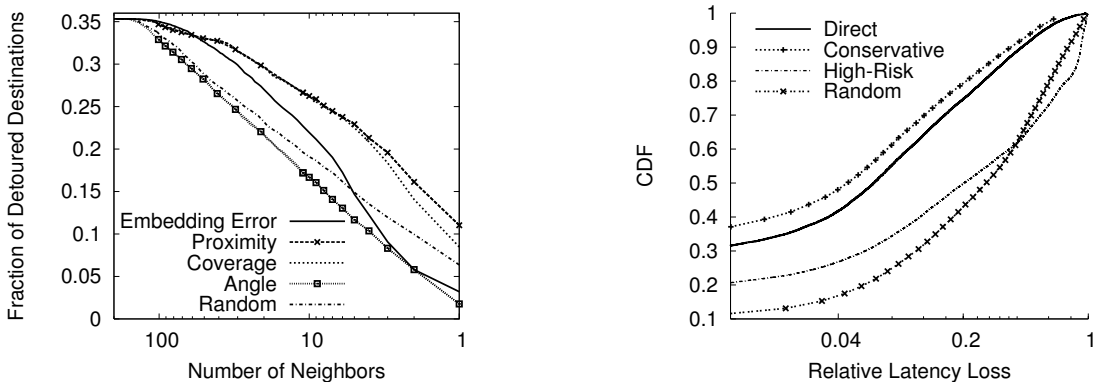


Figure 10: **(left)** *Neighbor selection algorithms*. As the number of legitimate neighbors is restricted, coverage, proximity and embedding error (for 32 or more neighbors) algorithms preserve the most detours. **(right)** *Relay selection algorithms*. Best detours are found through relays selected using the direct and conservative algorithms.

We consider a few traffic-independent neighbor selection policies, expecting that a combination of schemes would perform best. We separate them into two classes: *value* schemes are likely to provide the best detours, but may overlap; *diversity* schemes prefer relays that are different from those already chosen.

Value schemes include *embedding error* and *proximity*. *Embedding error* prefers neighbors with the largest positive error in the embedding of the source to potential neighbor edge: these nodes are likely to traverse the most coordinate distance with the lowest latency. *Proximity* prefers neighbors with the smallest absolute latency between the source and a potential neighbor.

By choosing the best neighbors exclusively, a node may miss diversity. *Coverage* uses the relay's coordinate and latency to determine the region in coordinate space that that relay covers. We split the space with a 2^4-tree structure (for scalability) and prefer neighbors that minimize the expected detour latency to every point in space. *Angle* prefers neighbors in different directions in the coordinate space. For all pairs of potential neighbors, a node computes the angle between the line segments from itself to the neighbors, and selects the neighbors with the largest angles. *Random* chooses neighbors at random to provide a point of comparison.

In Figure 10(left), we compare these neighbor selection algorithms. We vary how many neighbors a node can have from 1 to 200. At each step, we add a new neighbor based on one of the five schemes. Proximity and coverage perform the best, but embedding error also performs well with 32 or more neighbors. We choose proximity as our primary neighbor selection metric because it performs similarly to coverage and is easier to use.

6.2 Choosing Relays

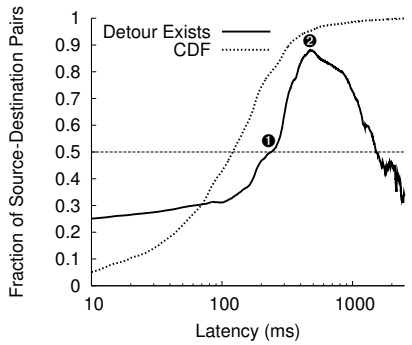
Neighbor selection determines the set of neighbors that may provide a detour path. With relay selection, a node attempts to discover quickly the neighbor that offers the best detour to a specific destination. Like server-selection problems solved by network coordinates, relay selection seeks the shortest combination of the direct path to the relay and the predicted path between relay and destination. Over time, this performance can be measured, but to minimize latency, detour performance should be predicted. At the very least, we hope to reduce the number of relays that we need to simultaneously contact to find a good detour when contacting a destination for the first time.

We consider the following policies for choosing relays for a destination. *Direct prediction* adds the measured source-to-relay latency to the estimated relay-to-destination distance in coordinate space, then chooses the relay with the lowest sum. Because latency measurements may be more reliable than coordinates, we evaluated a *conservative prediction*, which adds the source-to-relay latency measurement again to increase its influence in the prediction. This is based on the expectation that coordinates are inaccurate and seeks greater likelihood of a good detour in preference to the best detour at the top of the list. A *high-risk* scheme chooses the neighbor with the highest embedding error. Finally, *random* provides a baseline.

We select 32 neighbors for each node using the proximity-based algorithm and evaluate the four relay-selection algorithms. In Figure 10(right), we show the quality of predictions made using these algorithms in terms of relative performance lost compared to the best choice. The conservative approach performs best: approximately 80% of the detours chosen are only 20% longer than the best detour between the same pair of nodes.

6.3 Deciding Whether to Relay

Deciding whether to use a detour depends on a prediction of whether it will improve application performance.



❶	
CDF	0.80
Frac	0.50
ms	236

❷	
CDF	0.95
Frac	0.88
ms	474

Figure 11: As the latency to a destination increases, so does the probability that there is a detour.

This has two components: whether the traffic is sensitive to latency and whether a known neighbor is likely to provide a detour path. We evaluate the latter. Whether traffic is latency sensitive can be crudely inferred by ports, by commercial packet scheduling products, or by application-based proxies that can differentiate classes of traffic. In this section, we assume that the traffic is latency sensitive and attempt to predict whether to relay.

The decision of whether to relay depends first on whether virtual coordinates for the relay are available and recent. If there are no coordinates available for the destination, a node may choose to seek a relay by probing. If there are coordinates for the new destination, it may speculatively use a predicted relay, collect more information, or go directly to the destination without probing.

6.3.1 If the destination has no coordinates

If the destination lacks coordinates, the node should forward the packet directly, and if the destination is somewhat distant, *i.e.*, latency is long enough that a good detour is possible, the node may trigger latency probing from neighbors. The latency measurements by neighbors will, first, allow coordinates to be estimated and, second, provide direct latency measurements of the potential detour paths. Conveniently, if a detour path is available, the node may learn about it before the end of the second round trip (by starting the latency probing as soon as 10 ms have elapsed in the first contact).

The distance to the destination may be an indicator of whether the destination has a detour. In Figure 11, we show how often a destination has a relay within the neighbor set, given that the latency to the destination is above some value. For 95% of the edges, as the latency increases, so does the probability of a detour for the edge. The plot suggests that, after sending a probe to the destination, the longer a node waits to receive a response, the more likely it is that a detour exists for that destination. For 15% of destinations (between 236 ms and 1054 ms of latency), there is more than a 50% chance that a detour exists. We expect that actual node behavior, in terms of when to seek out a detour, will be application dependent.

	Correct decision		**Incorrect decision**	
	w/o probing	with probing	w/o probing	with probing
Detour exists	7.3%	11.1%	16.6%	12.8%
Detour absent	55.8%	57.3%	20.3%	18.8%
Total	63.1%	68.4%	36.9%	31.6%

Table 1: Using coordinates alone or coordinates with a latency probe to the destination, nodes can predict whether to use PeerWise. Probing the destination slightly increases the probability of making a correct decision.

For instance, a node may *always* try to find a detour for frequently contacted destinations.

6.3.2 If the destination has coordinates

If the destination has known coordinates that have been gossiped, a node can decide before sending the first packet: is there likely to be a detour among its neighbors? Assuming that all coordinates are accurate, except for the measured latencies to neighbors, the node can find a shortcut without direct contact to the destination.

For certain uses of PeerWise, getting the relay right before contacting a destination is useful. If the destination will be reached with a TCP connection, the first choice can stick: the source address on the SYN packet is fixed, and the connection cannot be easily migrated to a relay. For interactive applications over long TCP connections—shell, game, chat, perhaps voice—this decision may be important.

We show that, most of the time, when the coordinates of the destination are known, a node makes the correct decision on whether to use PeerWise. We define a correct decision as finding a good relay (within 25% of the best latency reduction) when a detour exists, or not attempting to find one when a detour does not exist. All other decisions of a node (*i.e.*, attempting to find a relay when a detour does not exist or finding a bad relay) are considered incorrect. We summarize all possible situations in Table 1. We used the *proximity* policy for neighbor selection and the *conservative* policy for relay selection. Using coordinates alone, nodes make a correct decision 63.1% of the time. The prediction accuracy improves to 68.4% if the latency to the destination is known. We consider the frequency of correct and incorrect decisions to be acceptable; a more ambitious node might try to discover detours more often at the expense of making more mistakes.

7 Implementation and Evaluation

We implement PeerWise and run it under real network conditions on PlanetLab. In this section, we briefly describe our implementation, then show that this implementation can *quickly* find mutually advantageous de-

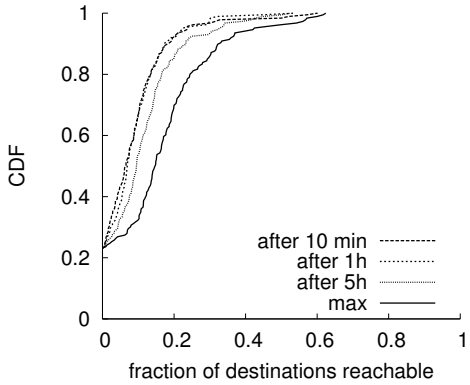


Figure 12: Fraction of the popular destinations reachable through mutually advantageous detours from PlanetLab.

tours that offer *significant* and *continuous* latency reduction. We then confirm that PeerWise detours can speed short web transfers in practice.

7.1 Implementation

We divide the functionality of PeerWise into two parts: the network coordinate system and a stand-alone daemon that includes all other components described in Section 5. We use Pyxida [24] for computing coordinates, since it is the only network coordinate system implementation we are aware of that is tested extensively under realistic network conditions [12]. Pyxida is written in Java and uses the Vivaldi algorithm [8] to compute coordinates for nodes. Each Pyxida node maintains a variable number of neighbors, updated constantly, and probes them at regular intervals. We augmented Pyxida to compute virtual coordinates for hosts that do not participate as described in Section 5.1.

We wrote the PeerWise daemon in approximately 3,000 lines of Ruby. The daemon listens for connections from other PeerWise nodes, and negotiates, establishes, and maintains mutually advantageous peerings. It communicates with Pyxida regularly, using RPC over TCP, to update the measured latencies and coordinates of the current set of neighbors as well as of the destinations that are currently served. By relying on the latency measurement and coordinate computation performed by Pyxida, we minimize the communication overhead. On the average, every node consumes less than 1KB/s (including Pyxida traffic).

7.2 Finding Detours

We ran PeerWise on 189 PlanetLab nodes, chosen for their stability, in September 2008. We focus on what detours PeerWise can find, where a detour is determined by the pings not by actual transfers. We express mutual advantage between two nodes as the number of detours that each offers the other. We experimented with three scenarios:

- **All-dest:** Each node tries to find detours to all 500 popular websites (described in Section 4) to which it can measure an RTT.
- **Rand-dest:** Each node tries to find detours to a random subset of the 500 websites.
- **Zipf-dest:** The popularity of destinations follows a Zipf distribution.

Our discussion focuses on the **All-dest** experiment, but we summarize the results from **Rand-dest** and **Zipf-dest** in Table 2. Recall that the destinations are already very popular servers, many of which use content distribution. Therefore, **All-dest** is not a best case scenario.

We describe the behavior of each node next. Nodes start looking for detours, after their network coordinates have stabilized, by successively sending detour requests to their neighbors. We limit the number of neighbors of each node to 32 for scalability and use the *proximity* policy for selecting neighbors. We make sure that no two detour requests are simultaneous: a new request is sent only when a reply (either positive or negative) has arrived for a previous one or a timeout has occurred. Each request tries to find detours to as many destinations as possible. Requests are sent continuously, even to the nodes with which peerings have been established or to the nodes that, in the past, could not offer detours. In this way, we are constantly renegotiating the peerings and are always ready to adapt to changes in latency.

PeerWise relies on the latency measurements and coordinate computations performed by Pyxida. We update both every 10 minutes. To avoid instability due to varying latencies, the updated values for latencies represent moving medians across the last 10 samples collected.

We present results for the first 36 hours of the experiment, counting from the time when nodes start requesting detours. For ease of exposition and to study startup behavior, all nodes start requesting detours simultaneously. We show that most nodes find mutually advantageous detours and that these detours lead to significant and stable latency reductions.

7.2.1 PeerWise finds detours

For each node, we count the destinations that can be reached using a mutually advantageous detour for the duration of the experiment. Figure 12 shows the distribution of the fraction of reachable destinations. Focus only on the line labeled "max" for now. Each point corresponds to a node, and its projection on the horizontal axis represents the fraction of destinations for which the node finds detours. Around 25% of the nodes cannot find any detours, while most nodes find detours to at least 10% of the popular destinations. Our results are consistent with those of the evaluation in Section 4 (see Figure 4). For **Rand-dest** and **Zipf-dest**, fewer nodes (around 50%) are able to find detours at all. This is because the number of destinations is much smaller than in **All-dest**.

	Latency reduction (§ 7.2.3) relative (absolute)			Longevity (§ 7.2.4) % of (src,dest) pairs			Variability (§ 7.2.4) % of (src,dst) pairs		
	median	**10 percentile**	**90 percentile**	**≥0.9**	**0.5-0.9**	**<0.5**	**1**	**2-10**	**>10**
All-dest	26% (29ms)	12% (12ms)	63% (131ms)	54%	18%	28%	67%	2%	31%
Rand-dest	25% (33ms)	12% (13ms)	60% (115ms)	36%	19%	45%	51%	23%	26%
Zipf-dest	24% (27ms)	12% (13ms)	59% (76ms)	31%	31%	38%	48%	23%	29%

Table 2: Characteristics of PeerWise detours: latency reduction, longevity and variability.

7.2.2 PeerWise finds detours quickly

How quickly are the detours discovered? We compute the fraction of destinations to which a detour is discovered by PeerWise within the first 10 minutes, 1 hour and 5 hours. Figure 12 shows the results as cumulative distributions. Many detours are discovered within the first 10 minutes of the experiment and the majority after less than an hour. Fewer and fewer detours are discovered afterward. These are mostly the detours that appear due to varying latencies—they are discovered because PeerWise constantly adapts to new latencies and coordinates.

7.2.3 PeerWise offers significant latency reduction

The detours discovered by PeerWise would not be very useful if they offered minimal latency reductions compared to the direct paths. We show that this is not the case. Recall that we have set a threshold: we consider only those detours that offer reduction of more than 10 ms and 10% of the direct-path latency. Here we focus on the latency reductions negotiated by PeerWise. In Section 7.3, we show how these reductions hold when user traffic traverses the detour path.

We compute all latency reductions for each (source, destination) pair for which a detour exists, both as absolute (milliseconds) and relative (fraction of the direct path latency) values. We show the median, 10th and 90th percentiles in Table 2. The median latency reduction is 29 ms or 26% of the latency of the direct path. 10% of the pairs have a reduction of more than 131 ms. This is caused by unusually high direct-path latencies, possibly due to traffic shaping. By circumventing these slow links, PeerWise can offer significant latency reduction.

7.2.4 Longevity and variability

PeerWise nodes may offer continuous latency reduction to a destination using several peerings. For each (source, destination) pair, we evaluate how long PeerWise offers reduction and with how many different relays. Ideally, every destination will be reached continuously through the same peering. Long-lived reductions through the same peering offer nodes more choices in when to use the mutually advantageous connection.

We consider two metrics: *longevity* and *variability*. Longevity captures how PeerWise nodes maintain latency reduction once a detour is discovered. We define the longevity of a destination D from a node S as the fraction of time that PeerWise offers S a detour to D, after PeerWise first learns about a shorter path from S to D. A longevity of 1 for the pair (S, D) means that, after PeerWise discovers the first detour between S and D, it will always offer some detour between S and D. Variability represents the number of different relays that S uses to obtain continuous reduction to D. The lower the variability, the easier it is to maintain latency reduction.

Table 2 summarizes longevity and variability for all (source, destination) pairs for which PeerWise offers latency reduction. For **All-Dest**, more than half of the pairs have a longevity higher than 0.9. 67% of the pairs use only one relay. When fewer destinations are selected at random or using a Zipf distribution, the number of detours, their longevity, and variability are reduced. However, about half of the (source, destination) pairs still have longevity higher than 0.5 and variability of 1.

7.3 Using Detours

We show how the detours discovered by PeerWise translate in real life. Can user-level applications benefit from the network-level detours of PeerWise? From each PlanetLab node running PeerWise, we download the front page of each of the 500 popular websites to which a mutually-advantageous detour exists. We use *wget* to perform two transfers every time it is called: one using the direct path and one using the PeerWise detour. To make the web request follow the detour path, we install the *tinyproxy* HTTP proxy on every PlanetLab node that can be used as a relay. We run each transfer 100 times, alternating whether detour or direct comes first, and record the individual completion times.

We verify whether the detours promised by PeerWise are seen by the web transfers. For each (source, destination) pair with a detour in PeerWise, we compute the *wget* reduction ratio—the ratio between the median relay transfer time and the median direct transfer time—and plot it against the PeerWise reduction ratio—the latency reduction ratio promised by PeerWise. Figure 13(left) presents the results. For 58% of the pairs, the *wget* reduction is less than 1; web transfers take less time through the relay than through the direct path, as predicted by PeerWise. However, many PeerWise detours do not materialize for the *wget* transfers.

We explain the dissonance between the PeerWise view and the application view next. PeerWise detours are determined by network-level pings. On the other hand, the *wget* end-to-end latency includes server and proxy wait

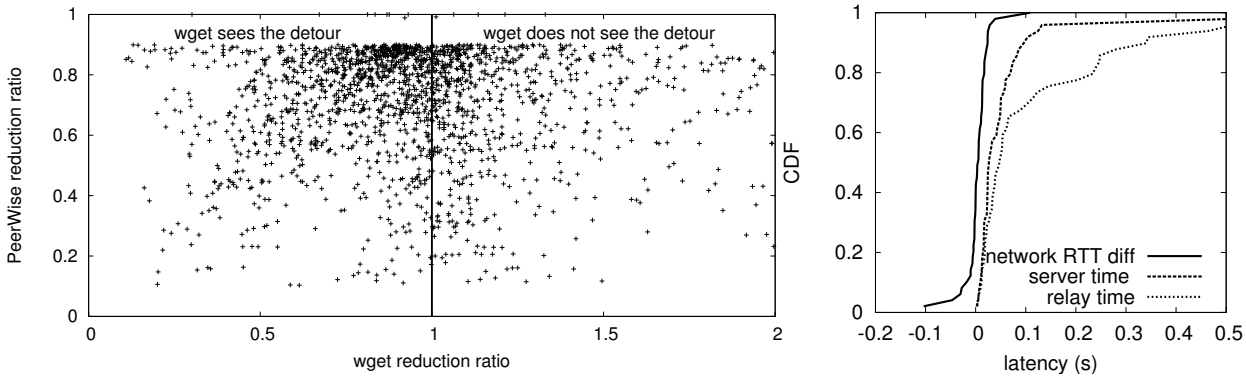


Figure 13: **(left)** *Wget* latency reduction versus PeerWise latency reduction: 58% of all PeerWise detours achieve latency reduction in real life. **(right)** Distributions of average server wait times, relay times, and difference between *wget* and PeerWise RTTs for all detour transfers. Relay times inflate application latencies the most.

times and thus may be larger than network latency. Further, PeerWise detours are based on medians of latencies gathered over long periods of time. Due to potential latency variations, these medians may differ from the RTTs at the time of the transfer.

To quantify the factors that inflate the application latency, we instrument our experiment as follows. During the web transfers, we run *tcpdump* on every relay node and log all proxy traffic. Using the packet timestamps, we compute, for each detour transfer, the network latency (from the TCP connection setup), the time spent at the relay and the time waiting for the server. Figure 13(right) shows the distributions of average server time, relay time and of the difference between network latency at transfer time and latency promised by PeerWise. The time spent at the relay and at the server accounts for most of the inflation in application latency: half of the relays induce an additional average latency of at least 50 ms. PeerWise predicts the network part of the *wget* transfer time well.

All relays are PlanetLab nodes; PlanetLab does not always reflect the realities of the Internet. We believe that the slowness of PlanetLab is the main factor that contributes to the unusually high relay time for our transfers. To confirm, we set up *tinyproxy* on a computer with minimal load, located at University of Maryland and run web transfers through it. The average relay time for all transfers through the UMD proxy is 5ms, less than 95% of all PlanetLab relays. If we consider the hypothetical situation in which all PlanetLab relay times were replaced by the average UMD relay time—effectively minimizing the time spent by a transfer at the relay node—then 78% of our web transfers would see the detours promised by PeerWise. We conclude that PeerWise has the potential to improve application performance.

8 Discussion

We discuss some of the implications that wide adoption of PeerWise would have for both ISPs and users.

8.1 Implications for ISPs

Overlay networks violate routing polices. How then would inter-domain routing policy and traffic engineering practices coexist with widespread PeerWise deployment? Routing overlay networks enable rule violations: customers and peers provide transit, and selfish routing [25] can subvert traffic engineering decisions. We discuss each in turn.

Customers provide transit, which is forbidden in inter-domain routing [9]. Even when a detour AS path precisely matches the direct (because an overlay node lies within the address space of one of the autonomous systems of the path), the overlay node is still a customer and a customer still provides transit. Whether that customer has an autonomous system or instead pays a monthly fee for a residential connection hardly matters.

Overlay networks bypass traffic engineering decisions. It is unclear to what extent the excessively long latency paths are deliberately chosen by network administrators. One might worry that a successful deployment of PeerWise would hamper ISP efforts to shape traffic toward slower, but less utilized, links. PeerWise is not intended for high-bandwidth transfers. Its structure discourages bandwidth consumption: we intend to shave packet transmission latency and, because each pair of nodes must strive to maintain the fairness of the application-level SLA that connects them, they may not consume unnecessarily. Downloading a large file through PeerWise may not reduce the download time significantly, considering the many other bottlenecks in the network (loss, client-side queuing, server load, etc.) [21, 4, 22].

8.2 Implications for Users

Forwarding traffic *through* and *on behalf of* others raises issues of privacy and liability for PeerWise users. Although unencrypted traffic is "public" regardless of the path it takes, it is reasonable to assume that users would

be more reluctant to forward their traffic through other users than directly through the "faceless" ISPs. Another concern is being held liable for forwarding potentially illicit traffic on behalf of another user. While some such traffic may be straightforward to filter (and negotiate in a PeerWise SLA), say by mechanisms similar to parental controls, such an approach requires knowing "questionable" destinations ahead of time, and leads to both false negatives and positives. A more general mechanism for *non-repudiation*—a means of verifiably proving to authorities the source of forwarded traffic—may be more appropriate, but is beyond the scope of this paper.

A potential extension of PeerWise would be to limit one's neighbors to a set of trusted users, determined for example via friend-of-friend links in an online social network, similar to the *f2f* file store [16]. While such an extension may obviate the concerns of non-repudiation, it may exacerbate privacy concerns; users may be less inclined to forward private traffic through their friends.

Interestingly, PeerWise can assist in *securing* an end-user's traffic. Reis *et al.* demonstrated that some ISPs modify users' web pages in transit [26]. PeerWise could assist in routing around such ISPs, or perhaps in lending greater credence to a page's authenticity.

9 Conclusions

PeerWise is based on building overlay networks from *mutually advantageous peerings*; we show that such a simple, locally enforced mechanism is sufficient to provide detour routes in the Internet. Surprisingly, pairs of nodes can help each other: few nodes are so well positioned that they need no help, and few are so poorly positioned that they can help no one. Our evaluation of PeerWise on two sets of real world latencies and on PlanetLab shows that most nodes can find good detours, reducing latency by at least 10 ms and 10%. PeerWise finds detours to both regionally and globally popular destinations, as well as to websites that use other latency-reduction techniques such as mirroring or DNS redirection. Most detours are long-lived and stable and reflect well the performance of applications using them.

Acknowledgments

We are grateful to our shepherd, Venugopalan Ramasubramanian, and to the NSDI reviewers for their help in improving this paper. We also thank Peter Druschel, Bo Han, Jay Lorch, Harsha Madhyastha, Justin McCann, Larry Michele, Alan Mislove, Vivek Pai, and Angie Wu for their comments. This work was supported by NSF grants CNS-0435065, CNS-0643443 and CNS-0626629, and by a Microsoft Live Labs fellowship.

References

[1] Alexa. http://www.alexa.com/.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.

[3] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *SIGCOMM*, 2008.

[4] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *IEEE Infocom*, 2000.

[5] B. Cohen. Incentives build robustness in BitTorrent. In *P2PEcon*, 2003.

[6] J. Corbo and D. Parkes. The price of selfish behavior in bilateral network formation. In *PODC*, 2005.

[7] L. Cox and B. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *SOSP*, 2003.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, 2004.

[9] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[10] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *IMW*, 2002.

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

[12] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. In *NSDI*, 2007.

[13] J. Ledlie, M. Seltzer, and P. Pietzuch. Proxy network coordinates. Tech. rep., Imperial College London, 2008.

[14] D. Levin, R. Baden, C. Lumezanu, N. Spring, and B. Bhattacharjee. Motivating participation in Internet routing overlays. In *NetEcon*, 2008.

[15] D. Levin, R. Sherwood, and B. Bhattacharjee. Fair file swarming with FOX. In *IPTPS*, 2006.

[16] J. Li and F. Dabek. F2F: reliable storage in open networks. In *IPTPS*, 2006.

[17] C. Lumezanu, D. Levin, and N. Spring. PeerWise discovery and negotiation of faster paths. In *HotNets*, 2007.

[18] A. Nakao and L. Peterson. Scalable routing overlay networks. In *ACM SIGOPS Operating Systems Review*, 2006.

[19] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *SIGCOMM*, 2003.

[20] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, 2002.

[21] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM*, 1998.

[22] J. Padhye and S. Floyd. Identifying the TCP behavior of web servers. In *SIGCOMM*, 2001.

[23] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *NSDI*, 2007.

[24] Pyxida. http://pyxida.sourceforge.net/.

[25] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On selfish routing in Internet-like environments. In *SIGCOMM*, 2003.

[26] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting in-flight page changes with web tripwires. In *NSDI*, 2008.

[27] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An analysis of Internet content delivery systems. In *OSDI*, 2002.

[28] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A case for informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, 1999.

[29] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *ACM CCR*, 29(5):71–78, 1999.

[30] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *USENIX*, 2007.

[31] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *NSDI*, 2004.