Abstract

Title of dissertation	Using Internet Geometry to Improve End-to-End Communication Performance
	Cristian Lumezanu, Doctor of Philosophy, 2009
Directed by	Professor Neil Spring Department of Computer Science

The Internet has been designed as a best-effort communication medium between its users, providing connectivity but optimizing little else. It does not guarantee good paths between two users: packets may take longer or more congested routes than necessary, they may be delayed by slow reaction to failures, there may even be no path between users. To obtain better paths, users can form routing overlay networks, which improve the performance of packet delivery by forwarding packets along links in self-constructed graphs. Routing overlays delegate the task of selecting paths to users, who can choose among a diversity of routes which are more reliable, less loaded, shorter or have higher bandwidth than those chosen by the underlying infrastructure. Although they offer improved communication performance, existing routing overlay networks are neither scalable nor fair: the cost of measuring and computing path performance metrics between participants is high (which limits the number of participants) and they lack robustness to misbehavior and selfishness (which could discourage the participation of nodes that are more

likely to offer than to receive service).

In this dissertation, I focus on finding low-latency paths using routing overlay networks. I support the following thesis: *it is possible to make end-to-end communication between Internet users simultaneously faster, scalable, and fair, by relying solely on inherent properties of the Internet latency space.* To prove this thesis, I take two complementary approaches. First, I perform an extensive measurement study in which I analyze, using real latency data sets, properties of the Internet latency space: the existence of triangle inequality violations (TIVs) (which expose *detour paths*: "indirect" one-hop paths that have lower round-trip latency than the "direct" default paths), the interaction between TIVs and network coordinate systems (which leads to scalable detour discovery), and the presence of mutual advantage (which makes fairness possible). Then, using the results of the measurement study, I design and build PeerWise, the first routing overlay network that reduces end-to-end latency between its participants and is both scalable and fair. I evaluate PeerWise using simulation and through a wide-area deployment on the PlanetLab testbed.

Using Internet Geometry to Improve End-to-End Communication Performance

by

Cristian Lumezanu

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2009

Advisory Committee:

Professor Neil Spring, Chair
Professor Bobby Bhattacharjee
Professor Samir Khuller
Professor A. Udaya Shankar
Professor Mark Shayman

To Mihaela, of course.

Acknowledgments

My journey into computer science began twelve years ago, almost to the day. Along the way, many people offered me their trust, friendship and advice. I am here because of them.

I learned from Neil the most: from thinking about research problems (big), to writing papers (no more "respectively" or "optimal"), to presenting my work. He would always proofread every piece of writing I would give him, send me to expensive conferences, and never let me slip away from the right path. He made me the researcher that I am now. Bobby trusted me unwaveringly (even when I myself did not). He allowed me to use my hands in a noble way and significantly decreased the frequency of contractions (the grammatical kind) from my life (and still has not collected).

I had the privilege of working and interacting with many other great researchers and professors. Katherine Guo was a wonderful mentor and provided me with even greater (and never redundant) moral support. I published my first paper with Sumeer Bhola and Mark Astley, who showed me the good sides of working in a research lab. Nick Feamster appeared at the right moment in my career. Adam Porter and Jeff Foster showed me that software engineering and programming languages can be a lot of fun. Irina Athanasiu encouraged me to come to Maryland, quickly dismissing my fears of catching the impostor syndrome. Samir Khuller, Udaya Shankar and Mark Shayman kindly agreed to serve on my thesis committee and listen to me ramble on about fast, fair and scalable routing overlay networks.

For the last five years, the lab was the perfect environment to work, learn, and have fun. Dave was my brother in arms, always ready to discuss new research ideas, resurrect old ones, eat, teach me the intricate ways of the English language, or watch clips from "The Office". I doubt I could have found a better friend, here or at any other school. Randy helped me with the PeerWise project, but more importantly, showed me that, to fight any carmichael in life, you start with a smile.

Adam has always tirelessly answered my questions about cycling or American football. If only he would defeat resistance and realize which is the real football. Rob was always ready to have conversations on the most diverse topics, from TCP congestion control to the different types of poker. Aaron is never afraid to be wrong. Justin, Katrina, Bo, Ruggero and Vijay have not spent nearly enough time in the lab, but they offered me first-hand examples of success.

It has been difficult enough to find the right words to acknowledge the people that helped me grow professionally. It is almost impossible to describe the gratitude that I feel towards my family. My parents, sister, aunt, and grandparents teach me everyday the meaning of love and help me constantly grow as a person. Above all, my wife has never left my side. This thesis would not have been possible without her support. I dedicate it to her and to that little person that she is carrying.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet has been designed as a best-effort communication medium for its users, limited to a most basic role: providing connectivity [15]. It is composed of thousands of interconnected and independently administered networks operated by Internet Service Providers (ISPs). ISPs cooperate and compete with each other to route traffic between users. They select paths based on their own cost, policies, past performance or even which route they learn first, and do not allow users to express choice in the routes taken by their packets.

Not surprisingly, ISPs do not guarantee good paths between users. Oftentimes, packets may take longer [86] or more congested [5] routes than necessary, they may be delayed by slow reaction to failures [41], there may even be no paths between users [29]. With thousands of users joining the network every day [33] and new applications being constantly deployed, the diversity of interests and requirements becomes increasingly important. Finding *any* path is not sufficient anymore; users need paths that are better tuned to their needs.

Several solutions have been proposed to overcome the inefficiencies of Internet

routing and improve end-to-end communication. Their arsenal includes diverse strategies such as upgrading [96], modifying [107], or completely redesigning [108] current routing protocols. However, one of the most attractive means for users to obtain better paths does not require any changes to the infrastructure: routing overlay networks.

Routing overlay networks are virtual networks of nodes and logical links (or edges) built on top of the existing routing infrastructure. Internet users join routing overlays and forward traffic between themselves along the virtual links. Although packets still traverse the underlying routing infrastructure, their path is constrained by the logical edges. Routing overlays improve the performance and robustness of packet delivery by delegating the task of selecting paths to users, who can choose among a diversity of routes which are more reliable [29], less loaded [5], shorter [86] or have higher bandwidth [31] than those selected by the ISPs.

In this dissertation, I present PeerWise, a latency-reducing routing overlay network. PeerWise finds detour routes between its participants. A detour is an "indirect" one-hop path that has lower round-trip latency than the "direct," ISP-provided path. Latency, the time required for a packet to travel from one end to the other and back, is an important metric for measuring the quality of a path. The performance of many distributed applications depends on finding low latency paths. For example, the playability of online games, such as first-person

shooters, hinges on low-latency updates between players [10, 9, 70]. Services that provide rapidly changing content, such as sports scores, real-time bus status or stock quotes depend on propagating the content as fast as possible to users [56]. VOIP applications that need to bypass firewalls could benefit from relaying traffic on low latency paths [92].

Unlike existing routing overlays [5, 86, 29, 97], PeerWise transcends its simple role of finding better paths and provides both scalability and fairness. Scalability means that the process of finding detours performs similarly when the number of users increases, while fairness encourages participation of honest users and discourages freeloaders and adversaries. Next, I discuss the three main goals of PeerWise, low-latency paths, scalability, and fairness, and address the challenges in achieving them.

1.1 PeerWise Goals and Challenges

I discuss the three main goals of PeerWise along with the challenges that each of them brings. In the next section, I describe how properties of the Internet latency space help address the challenges.

Low-latency paths

The paths provided by PeerWise must have lower latency than those offered by the underlying routing infrastructure.

This is the most simple and basic requirement of PeerWise. Before designing a system that offers paths that are shorter than those provided by ISPs, several questions need answers. Do lower-latency paths even exist? Of course, we know that Internet routing is not based on latency (more on this in Chapter 2), therefore we can assume that it yields paths that are not shortest. But how many are there? And how much shorter are they than the corresponding direct paths? Can many nodes benefit from them or are they available to only a chosen few? Can we design an entire system based on their existence?

Scalability

PeerWise must be scalable: its ability to provide lower-latency paths should not suffer as the number of participants increases.

That there exist detours for the paths provided by ISPs leads to a logical question: How do we find them? A simple way of finding detours from node S to node D is to measure the latency on all possible paths between S and D. However, the probing and monitoring would severely limit scalability and consume resources even when not needed. Many of the measured paths are useless: they

have higher latencies than the default path between S and D. Existing routing overlays are limited to small number of nodes and use all-to-all probing to discover better paths does not pose any significant resource constraints [5]. However, when one increases the number of participants to the order of hundreds, the challenge is how to discover detours while minimizing the number of measurements.

Fairness

PeerWise must be fair to its participants: everybody should be compelled to provide nearly as much service as they receive.

The Internet is a heterogeneous communication medium, filled with freeloaders, adversaries, and miscreants. The heterogeneity of connectivity means that well-connected nodes are more likely to provide service, while poorly connected nodes are more likely to request service [85]. This asymmetry could discourage the well-connected nodes from joining—they have little to gain and would pay highly. The presence of freeloaders and adversaries implies that routing overlay designs should include an incentive mechanism: a means by which nodes compel each other to provide nearly as much service as they receive. Without such incentive in PeerWise, the overlay nodes that are intermediate hops in detours may provide transit for others without benefiting from any detours. Existing routing overlays were designed with cooperative, selfless participants in mind. They focus exclu-

sively on performance and do not consider fairness among nodes. My challenge is to construct PeerWise as a fair routing overlay, robust to misbehavior and malice, but still offer participants the benefits of improved end-to-end communication.

1.2 From Internet Geometry to PeerWise

Geometry is the study of the properties and relationships between points, lines and figures in space. Most commonly, geometry studies metric spaces, which are spaces where a distance function is defined between any two points. The well-known two-dimensional Euclidean space is an example of metric space.

I define the *Internet geometry* to be the study of properties and relationships between nodes in the Internet. In particular, Internet geometry deals with the Internet latency space, the space defined by the nodes in the Internet with the latency as the distance between them. I use Internet geometry to derive the techniques and mechanisms that allow PeerWise to provide lower-latency paths, while being scalable and fair.

Low-latency paths

Triangle inequality violations expose detours.

An important characteristic of the Internet latency space is the existence of triangle inequality violations (TIVs). There exist triples of nodes such that the

latency between two of them is larger than the sum of latencies from these two nodes to the third. Triangle inequality violations are a natural and persistent consequence of routing in the Internet [111], as I describe in Chapter 2, and they become an excellent opportunity to expose detours.

To grasp the effect of TIVs on latency reduction and their suitability as a building block for PeerWise, I found it necessary to better understand TIVs. In Chapters 3 and 4, I examine TIVs by collecting and analyzing latency measurements between thousands of Internet hosts over the course of several weeks. My measurement study shows that TIVs are important for latency reduction: they are a prevalent and lasting feature of the Internet, they offer significant latency reduction, and many users can benefit from them.

Scalability

Network coordinates predict detours.

Network coordinates [69, 22] associate nodes in the Internet with coordinates in a metric space such that the coordinates of a node reflect its location in the Internet. With little or no measurement, one can then estimate the real latency between two nodes as the distance between their coordinates in space.

Because TIVs are not allowed in metric spaces, estimated distances between two nodes in a TIV may differ greatly from the real latencies. In particular, when the

estimated distance is much smaller than the real distance, the nodes have a higher chance of benefitting from a detour; conversely, when the estimated distance is much larger than the real distance, the nodes are more likely to be part of detours. In Chapter 5, I prove the relationship between TIVs and network coordinates and show that, by using the simple rules, described above, the number of measurements required to discover detours is greatly reduced.

Fairness

Mutual advantage provides fairness.

The final missing piece is fairness. I introduce mutual advantage as a fundamental design principle in PeerWise. Overlay edges exist only between nodes that can help each other find detours: each node is an intermediate hop on a detour of the other. Mutual advantage is possible due to another property of the Internet latency space: that there exist pairs of nodes in the Internet such that each benefits from a TIV where the other is the intermediate node.

Before applying the principle of mutual advantage in the design of PeerWise, I wanted to study its potential and limitations. In Chapter 6, I show, using collected real-world latencies, that many pairs of nodes have mutually-advantageous relationships in the Internet latency space, and that mutual advantage reduces the number of detours that a node can find by only half.

Building and evaluating PeerWise

The properties of the Internet latency space described above inspire the design of PeerWise. PeerWise has three main components: network coordinate system, neighbor tracking and pairwise negotiation, described in detail in Chapter 7. Each participant must compute its *network coordinate* before searching for detours. One challenge was how to scalably discover detours to non-participating nodes, such as web servers. Because these destinations do not maintain network coordinates, I could not readily predict detours for them. With colleagues, I implemented a virtual coordinate system through which a PeerWise node can become responsible and compute coordinates for any host in the Internet. The *neighbor tracking* component identifies other participants in PeerWise that are more likely to provide good detours. Finally, *pairwise negotiation* establishes connections based on mutual advantage. In Chapter 8, I describe the implementation and deployment of PeerWise on the PlanetLab testbed. I show that nodes quickly find detours to popular destinations, that these detours are stable and that they offer significant latency reductions. I then confirm that user-level applications such as web transfers can benefit from the network-level detours of PeerWise.

1.3 Thesis and Contributions

In this dissertation, I support the following thesis: *It is possible to make end-to-end communication between Internet users simultaneously faster, scalable, and fair, by relying solely on inherent properties of the Internet latency space.* To prove this thesis, I take two complementary approaches. First, I perform an extensive measurement study in which I analyze, using real latency data sets, properties of the Internet latency space: the existence of triangle inequality violations, the interaction between TIVs and network coordinates, and the surprising presence of mutual advantage. Then, using the results of the study, I design and build PeerWise, a routing overlay network that reduces end-to-end latency between its participants and is both scalable and fair.

I bring the following contributions:

Measurement study on triangle inequality violations. I present the first extensive study on triangle inequality violations in the Internet and on their suitability for detour routing. For this, I collected six latency data sets of various sizes and at various times. I show that TIVs are not measurement illusions but real properties of Internet latencies. Although the number of TIVs varies with time, aggregating measurements using medians provides a conservative data set for evaluating detour routing. Further, I demonstrate that, although the number of TIVs

is small, they account for significant detours, benefitting many. Finally, I explore the negative consequences of TIVs and present their interaction with policy routing and BGP, the interdomain routing protocol: using the detours provided by TIVs will likely violate routing policies of ISPs. However, the extent of these violations is not as high as previously believed, and I show that finding short routes that abide by routing policies is possible.

Scalable detour detection. I study the interaction between triangle inequality violations and network coordinates and develop a scalable technique to detect TIVs with few measurements. My technique relies on measuring the difference between the real latency between a pair of nodes and the distance estimated with network coordinates. If the difference is positive, it is more likely that the pair of nodes is part of a detour; if the difference is negative, the pair of nodes is likely in need of a detour.

Mutual advantage in overlay routing. I introduce mutual advantage as a novel principle in the construction of overlay networks: overlay edges should exist only between nodes that benefit form each other's position or resources in the network. I examine the potential of mutual advantage in the context of detour routing and show, perhaps contrary to expectation, that there are not only "haves" and "have nots" of low-latency connectivity. I show that such a simple, locally-

enforced mechanism is sufficient to provide detours in the Internet.

PeerWise. I design and build PeerWise, the first scalable and fair latency-reducing routing overlay network. By providing its participants with paths that are shorter than those offered by ISPs, while protecting them against freeriding and misbehaving, PeerWise offers a practical and flexible alternative to Internet routing, and to existing latency-reducing overlays such as Detour [86].

1.4 Roadmap

This dissertation is organized as follows. In Chapter 2, I present details of how routing is done in the Internet, with emphasis on the mechanisms and policies that lead to triangle inequality violations and routing overlay networks. Chapter 3 offers new insight into the dynamic properties of triangle inequality violations and how they affect the design of PeerWise. I study the positive and negative effects of using triangle inequality violations for latency reduction in Chapter 4. I focus on how to discover TIVs in Chapter 5, and present a scalable detour discovery technique based on network coordinates. In Chapter 6, I discuss how to achieve fairness in PeerWise and introduce mutual advantage as a novel principle for the construction of overlay networks. I also discuss the the limitations of using mutual advantage in detour discovery. I present the design of PeerWise in Chapter 7 and

results from its deployment on the PlanetLab testbed in Chapter 8. I conclude with an overview of future work in Chapter 9.

Chapter 2

Background and Related Work

Two simple concepts are at the foundation of PeerWise: triangle inequality violations (TIVs) and overlay routing. That TIVs exist in the Internet represents an excellent opportunity to obtain shorter end-to-end paths. With overlay routing, users can take advantage of this opportunity by overriding the default Internet routing and sending traffic along the detours provided by TIVs.

TIVs and overlay routing are both a consequence of Internet routing. TIVs occur naturally because routing is not based on latency, while routing overlay networks are an artificial effect of routing not taking into account user performance. In this section, I present an overview of routing in the Internet, with emphasis on those policies and mechanisms that lead to TIVs and routing overlays. I then describe both triangle inequality violations and routing overlays and review related research.

2.1 Routing in the Internet

In this section, I present an overview of routing. This is not an exhaustive presentation of Internet routing; for treatises on the subject I refer the reader to the books by Peterson *et al.* [75] and Ross *et al.* [39].

The Internet is a large network composed of thousands of smaller networks, called autonomous systems (AS), operated by Internet Service Providers (ISPs). ASes compete and cooperate with each other to control routing and ensure connectivity between users in the Internet. ASes enter business relationships with each other. Some provide Internet service to others in exchange for money: they have a provider-customer relationship (p2c). Others negotiate to exchange traffic without exchanging money as a way of avoiding sending traffic through a provider: they have a peer relationship (p2p) [25, 95].

Each AS is composed of tens to thousands of routers. Routers are the devices that direct traffic between users. Routers build routing tables that contain collected information on all the best paths to all the destinations they know how to reach. They use routing protocols to advertise and receive route information to and from other routers, and to compose paths between users.

The goal of routing is connectivity: find a usable path to the destination. This path is built with information from routing protocols. The path computation is next-hop based. If a router does not have direct connection to the destination, it

looks in its routing table, finds another router that is "closer" to that destination, and forwards traffic to it. Each routing protocol may have a different definition for "closer": some may choose the router with the lowest number of hops to the destination (as in RIP [64]), others the one with the least-cost path based on a metric defined by the network operator (as in OSPF [66]). Routing protocols allow operators to express preference for certain paths to shape the flow of traffic. Some paths may be preferred because they are lightly-used, cheaper or more reliable. Operators express these policies in the configuration of routing protocols.

I separate the discussion on routing into two parts: inter-domain routing and intra-domain routing.

2.1.1 Inter-domain routing

Inter-domain routing protocols carry routing information between autonomous systems. The main inter-domain routing protocol is BGP (Border Gateway Protocol) [83].

BGP is a path vector protocol. The term "path vector" comes from the fact that BGP routing information carries a sequence of AS numbers, which indicates the path a route has traversed. Each BGP router maintains a table of IP networks, or prefixes, and the AS path to them. For each prefix, a BGP router receives many path advertisements. It has to select a single "best" path to advertise further to

its own neighbors—other BGP routers to which it is directly connected.

The local policies for selecting and exporting BGP routes depend on the business relationships between ASes. Common practice is for customers to not advertise routes learned from a provider to peers or other providers. This prevents the customer from being used as transit between two of its providers. Similarly, routes learned from peers or customers are advertised to other customers and not to providers or other peers. This order is established by business interests: delivering packets through customers creates revenue, delivering packets through providers costs money, while delivering packets through peers typically costs nothing. To break ties, routes with fewer ASes are chosen. By abiding to these policies, packets may take a more circuitous policy-compliant path instead of the shorter or less congested policy-violating path. I present an example of how routing policies can lead to more circuitous paths in Figure 2.1.

Once the next-hop AS to a destination is chosen, the AS has to decide the router-level path to that neighboring AS. Commonly, ASes have more than one peering point to each other. A peering point is a place where routers from many ASes interconnect to exchange traffic. To which peering point to send traffic then? The most common practice of choosing the path is called early-exit (or hot-potato [27]): the AS tries to send traffic to the closest peering point, to minimize the cost incurred on its own network to route packets. The decision of selecting the

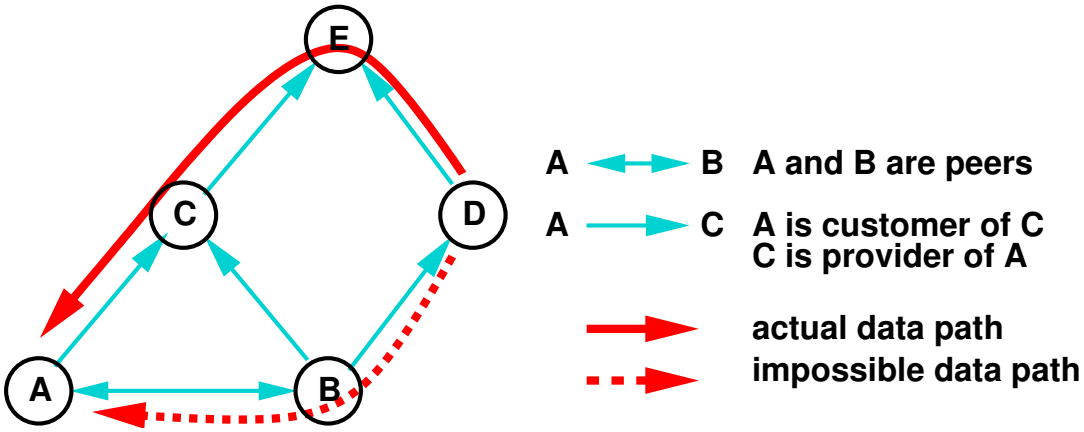


Figure 2.1: Policies for selecting and exporting BGP routes can lead to circuitous paths. A, B, C, D and E are ASes. B has a peer relationship with A and is customer of both C and D. B does not advertise to D its routes to A or C, therefore D discovers only the path (D, E, C, A) to A, instead of the more direct, and perhaps shorter or less loaded, (D, B, A).

peering point depends only on local performance and is oblivious to the latency to the destination. In Figure 2.2, I show how early-exit routing inflates end-to-end paths.

2.1.2 Intra-domain routing

Intra-domain routing protocols carry routing information inside autonomous systems. Intra-domain routing is typically based on finding shortest paths with respect to configured weights assigned to each link.

The most common intra-domain routing protocols are RIP, OSPF and IS-IS. RIP is a distance-vector protocol. In distance-vector routing, each link is assigned a cost, and the path chosen between two nodes has the lowest total cost. Each

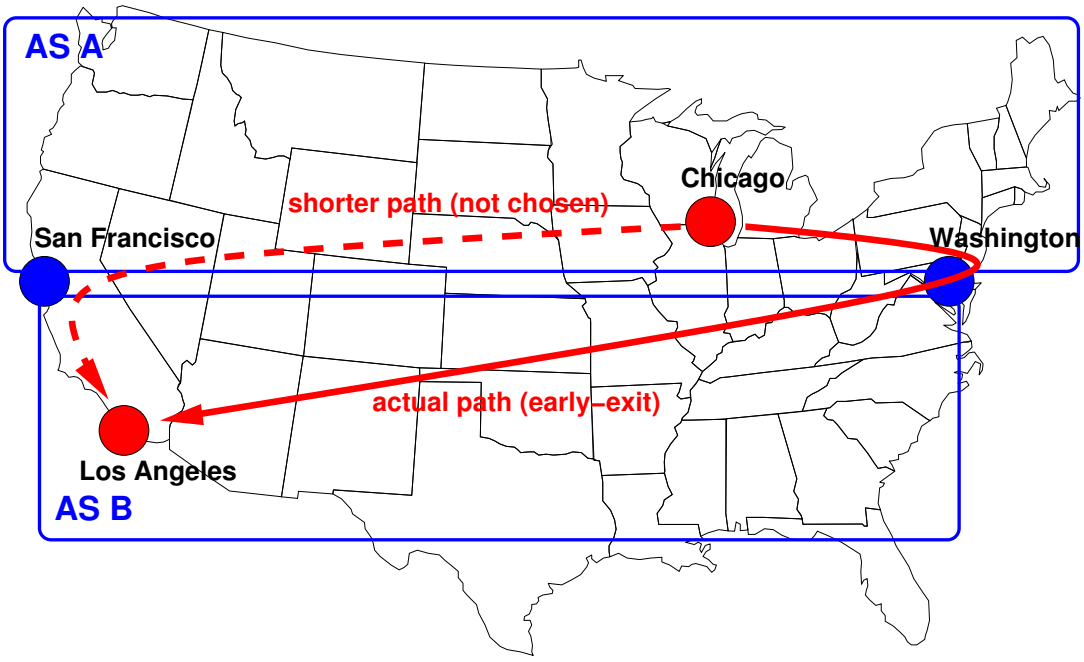


Figure 2.2: Early-exit routing leads to inflated paths. Hypothetical ASes A and B peer in San Francisco and Washington, DC. When a user from Chicago, in AS A, wants to communicate with a user in Los Angeles, in AS B, the path chosen by AS A goes through the peering point in Washington (since it is closer to Chicago). A more direct, and potentially shorter path, between Chicago and Los Angeles, would traverse the San Francisco peering point.

router computes and stores locally the cost of the best paths to each destination. It periodically disseminates this information to its neighbors, who can update their own routes if they are costlier. OSPF and IS-IS are link-state routing protocols. In link-state routing, nodes exchange partial maps of the network. Each node can reassemble the fragments into a global picture and choose paths from a complete topology.

To set policy, each network operator configures a weight (cost) for each link. The chosen path between two nodes has the lowest cost. Operators can decrease or increase the cost of each link to encourage or discourage traffic. For paths to be shortest in terms of latency, link weights must be consistent with actual delays.

2.2 Triangle inequality violations

In this section, I introduce triangle inequality violations (TIVs) in the Internet. TIVs are a direct and natural consequence Internet routing not being based on latency [111].

2.2.1 The triangle inequality

The triangle inequality states that for any triangle ABC, the length of a given side must be less than or equal to the sum of the other two sides:

$$d(A, C) \leq d(A, B) + d(B, C) \tag{2.1}$$

where $d(x, y)$ is the distance between x and y.

Now consider a space $\mathcal{S}$, formed all nodes in the Internet. I define the distance between two nodes in $\mathcal{S}$ as the round-trip time (or latency) between them ($d(x, y) = rtt(x, y), \forall x, y \in \mathcal{S}$). It has been shown [88, 111] that there exist triples of nodes in the Internet that do not satisfy the triangle inequality:

$$\exists \text{ A,B,C} \in \mathcal{S} \text{ s.t. } rtt(A, C) > rtt(A, B) + rtt(B, C) \tag{2.2}$$

I call a triple of nodes that violates the triangle inequality a *bad triangle*. In the bad triangle ABC, AC is the *long side* while AB and BC are the *short sides*. Alternatively, borrowing terminology from Detour [88, 86], I refer to the path (A,B,C) as the *detour path* and to the path (A,C) as the *direct path*. I define the *latency reduction* of a detour path as the difference between the latency on the direct path and the latency on the detour path.

That there exist TIVs in the Internet is an excellent opportunity for improving end-to-end latency. If three nodes are part of a TIV, the latency between two of them decreases when traffic is redirected through the third. In Figure 2.3, I show a real TIV in the Internet formed by nodes in College Park, MD, Seattle, WA, and New York, NY. Packets take 97 ms on the direct path between College Park, MD and Seattle, WA. However, the College Park user can reduce her latency to the Seattle node to 74 ms (by 23 ms) if it redirects all packets through the node in New York.

2.2.2 Research on TIVs

Savage *et al.* [88] are among the first to show the existence of triangle inequality violations in the Internet. They measure a large number of Internet paths between geographically diverse hosts and show that alternate paths of lower latency exist between more than 20% of the pairs of nodes in their data sets. The authors study

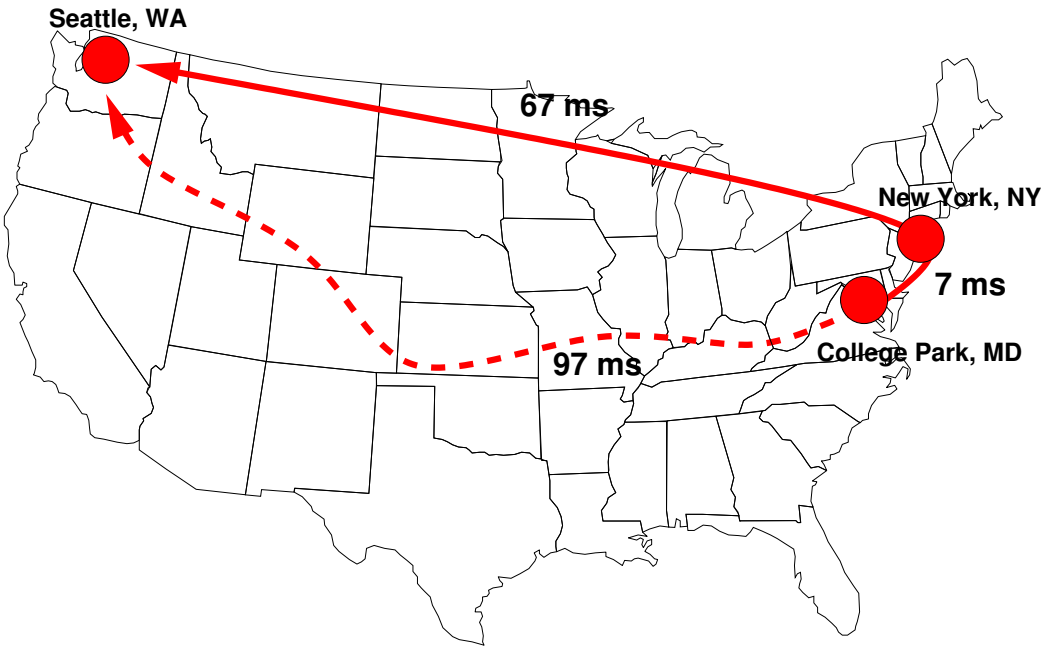


Figure 2.3: Example of triangle inequality violation formed by three nodes located in College Park, MD, New York, NY and Seattle, WA. The latency on the direct path between College Park and Seattle is 97 ms, using the detour through New York takes only 74 ms.

the origins of the TIVs and conclude that the availability of alternate paths does not depend on a few good or bad ASes.

Zheng *et al*. [111] use data collected between nodes in the PlanetLab and GREN research networks to argue that TIVs are not measurement artifacts, but a persistent, widespread and natural consequence of Internet routing policies. We confirm their findings that TIVs are caused by routing policies. We also study and quantify, using much larger latency and AS path data sets, the different policy decisions that may affect the formation of TIVs.

Recently, triangle inequality violations have been studied in relationship to network coordinates [46,102]. I review this research in Chapter 5, when I introduce network coordinates.

2.3 Alternatives to Routing

ISPs do not guarantee good paths between users. For scalability, BGP does not maintain performance-based path metrics and exchanges only reachability information, in the form of full AS paths to each destination prefix. Therefore, as seen earlier in the section, it may select paths that are longer or more congested than necessary [88, 86, 93, 71]. The slow convergence of BGP may lead to packet delay, reordering, and even loss when failures occur [41].

Several research efforts attempt to improve the BGP end-to-end path selection

process either by proposing changes to BGP or by offering alternatives such as source routing and overlay networks.

2.3.1 Changes to BGP

Subramanian *et al.* propose HLP [96], a hybrid link-state and path-vector protocol, as an alternative to BGP. HLP achieves better scalability and convergence than BGP, by exposing common routing policies. The customer-provider relationships—which can be easily inferred from routing information [25, 23] anyway—are made public and used to create hierarchies that limit the propagation of routing updates. HLP uses link-state routing within a given hierarchy of ASes to improve connectivity and uses path-vector routing between hierarchies to preserve scalability.

R-BGP [40] adopts a different approach. Instead of improving path selection by minimizing the periods without connectivity between two users, Kushman *et al.* focus on protecting the data plane by offering the user a backup path. While waiting for the default routes to recover after failure, R-BGP forwards data on precomputed fail-over paths advertised by ASes before the fault occurred. R-BGP guarantees connectivity between two ASes as long as there is a policy compliant path between them.

Wiser [63] extends BGP with a simple mechanism that coordinates the route selection of each ISP with those of its neighbors. Each ISP shares information

by tagging advertised routes with costs derived from internal paths. Routes are computed based on a combination between local costs and the neighbor's costs. When there are several flows between two ISPs, each ISP can trade off a small increased cost in the path of some flows for a large performance benefit for other flows [62], resulting in a win-win situation.

MIRO [107] is a multi-path interdomain routing protocol that offers flexibility in choice of paths to both end-users and transit domains. In MIRO, routers learn about default routes using BGP but ASes are allowed to negotiate alternate paths when needed. MIRO retains the simplicity of BGP and improves its connectivity and failure resiliency by offering path diversity.

2.3.2 Source Routing

Source routing protocols [113, 35, 82, 108, 6] offer greater flexibility and diversity in path selection. They allow end-hosts to select the path to the destination by specifying intermediate hops—routers or ASes—to be traversed. However, for sources to compute efficient paths, knowledge of the network topology and path performance metrics is necessary, which can lead to scalability problems. Further, the intermediate hops have little control over the traffic they relay, which limits wide scale deployment.

For scalability, feedback-based routing [113] and NIRA [108] separate the topol-

ogy information propagation (*i.e.*, the existence of routes) from the route discovery. In feedback-based routing, transit routers propagate only structural information, while route computation is exclusively done at the edge routers based on end-to-end measurements and reachability information. NIRA propagates structural information only within provider-rooted addressing hierarchies. Each user discovers topology information on domains that provide transit service to her and retrieves on demand topology information from the destination's providers.

BANANAS [35] allows source-based multipath routing where the source needs not be an end-host, but any AS on the path of a packet. Intermediate ASes can control the path by announcing only a subset of all available destinations, similarly to BGP. Platypus [82] achieves policy compliant packet forwarding using network capabilities to securely verify source routing requests. Argyraki and Cheriton [6] propose WRAP, a variant of the original Loose Source Record Route Protocol [79]. WRAP controls the domain-level path of a packet by adding state inside the packet's header.

2.3.3 Overlay Networks

Overlay networks are virtual networks of nodes and logical links built on top of the existing routing infrastructure, with the purpose of adding new functionality. There are many types of overlays that arise to meet a range of purposes and needs:

file sharing [26,36,18], content distribution and caching [2,24,37], improved routing [5, 86, 29], enhanced security and privacy [98, 17, 101], multicast and streaming [7,8,32,14,13], ordered message delivery [59], implementation and experimentation of new technologies [74].

A routing overlay is an overlay that controls or modifies the path of data through the network. Routing overlay networks [5,86,29] improve the performance and robustness of packet delivery by delegating the task of selecting paths to users, who can choose among a diversity of routes which are more reliable, less loaded, shorter, or have higher bandwidth than those selected by the ISPs. Unlike other path selection methods, overlay networks do not require support from routers. Although packets still traverse the underlying routing infrastructure, their path is constrained by the logical edges of the overlay.

Several strategies are used to determine which nodes should peer and what links should be followed. RON [5] builds a fully connected mesh and monitors aggressively all existing edges. When the direct path between two nodes fails or has performance problems, communication is re-established through the other overlay nodes. Other approaches sacrifice unnecessary edges for scalability to define more sparse meshes: Nakao *et al.* [67, 68] employ topology information and geography-based distance prediction to build a mesh that is representative of the underlying physical network. Gummadi *et al.* present SOSR [29], a simple overlay network for

improving end-to-end path reliability. When failures occur, the source attempts to reach the destination using a small set of randomly chosen intermediary nodes, obtaining close to maximum possible benefit.

Many peer-to-peer applications build their own routing overlay networks to improve connectivity issues. Skype [92] redirects voice packets between two users that are behind NATs and cannot communicate directly through a third Skype user with public IP address. BitTorrent [18] creates connections between the users that have mutual interest in each other's data.

Chapter 3

Measuring Triangle Inequality Violations

Existing studies on triangle inequality violations show that TIVs are persistent and widespread [102, 22]. They are not measurement artifacts, but a natural consequence of the Internet routing structure [111]. However, all evidence about TIVs has been limited to aggregate latency data sets [22, 94, 109, 102, 106, 111] that combine measurements taken at different times over long periods. These data sets fail to capture the variations of triangle inequalities and may offer false illusions to applications that rely on TIVs or the lack thereof. For example, representing multiple measurements with their median values may reveal TIVs that are short-lived in reality, and thus not necessarily critical for latency reduction, or may miss long-lived TIVs that could be exploited by overlay routing.

The limitations of these data sets open crucial questions for the design of systems that exploit TIVs: Are TIVs stable or transient? Are they real or simply illusions caused by aggregating measurements taken at different times? Are they caused by queuing delay or load? And finally, is the performance of these systems affected by the way data is aggregated?

time	AB	BC	AC	TIV?
t1	103	40	135	no
t2	90	26	106	no
t3	135	25	139	no
med	103	26	135	yes

(a)

time	AB	BC	AC	TIV?
t1	78	47	140	yes
t2	98	15	135	yes
t3	100	50	166	yes
med	98	47	140	no

(b)

Figure 3.1: Median values can create the illusion of TIVs. Latencies for AB, BC and AC are measured several times. We show the values at $t1$, $t2$ and $t3$. The final data set is compiled from the medians: although at no timestep is there a TIV among A, B and C, with AC as the long side, the medians indicate otherwise (a); alternatively, even if each measurement indicates the presence of a TIV, the medians do not reflect it in the final data set (b).

In this chapter, I aim to offer new insight into the properties of triangle inequality violations in the Internet and how they affect the design of PeerWise. I collect new latency data sets of different sizes and at varying time granularities. I show that TIVs are real and not illusions of measurement. The number of TIVs varies with time and aggregating multiple measurements using medians *underestimates* the number of TIVs that existed at any point during the measurement. Aggregating measurements eliminates most of the TIVs that appear sporadically during the measurement, but it also misses many ($\geq$70%) of the TIVs that last longer than 5 hours.

3.1 Pitfalls of measuring TIVs

Existing evidence about TIVs is derived from aggregate all-to-all latency data sets that combine many measurements [102, 22, 43, 111]. The final latency between two nodes is obtained by taking the median [22] or the minimum [106, 111, 109] of measurements performed over long periods of time such as days or even weeks. Although these data sets are meant to reflect the real Internet latency space, they may fail to accurately depict the characteristics of TIVs. Consider an experiment that measures the latencies among nodes A, B and C at regular intervals and computes the final latency value for each pair as the median of the measured values. In Figure 3.1 we show values of latencies at three intervals, $t1$, $t2$, and $t3$, as well as the median. Although at no time during the measurement was there a triangle inequality violation among A, B and C, the medians indicate otherwise (Figure 3.1(b)). The opposite can also be true: the triple A, B and C violates the triangle inequality at every time step, but this is not reflected by the medians (Figure 3.1(c)).

Scenarios such as the ones above, although rare, reveal the potential pitfalls of reasoning about triangle inequality violations with aggregates of data. Some triangle inequality violations may appear when computed with median values for latency but may not be long-lived enough to be significant. Further, aggregates of data may not capture TIVs that, although do not appear continuously during the

data collection, may still be present for enough time to be useful for an overlay routing network.

3.2 New measurements

In this section, I describe the methodology for collecting latency data sets that are better suited for studying the properties of triangle inequality violations. I use the King tool [28] to estimate latencies between arbitrary nodes in the Internet.

3.2.1 King

King uses recursive DNS queries to estimate the latency between two nodes in the Internet. Given the IP addresses of two nodes, King computes the propagation delay between them as the delay between authoritative name servers for those addresses. Figure 3.2 shows an example. A user located at S tries to estimate the latency between nodes A and B. First, S measures the round-trip time to nsA, the closest recursive name server of A. Then, S asks nsA to recursively resolve a name served by a name server of B, nsB. The latency between nsA and nsB is obtained by subtracting the times taken to perform the two operations and represents an estimate of the latency between nodes A and B. For more details on the design and implementation of King, please refer to the paper by Gummadi *et al.* [28].

Using King to estimate all-to-all round-trip times between nodes in the Internet

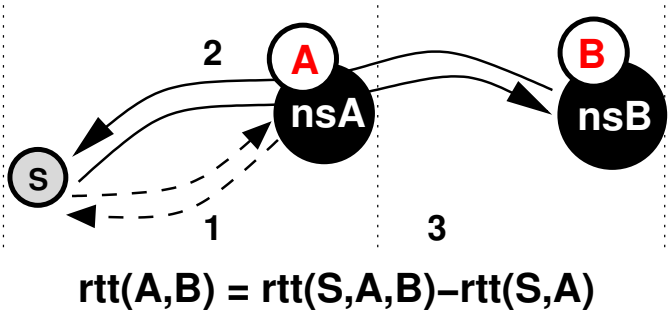


Figure 3.2: How King works: 1) S measures the RTT to the closest recursive name server of A, nsA, 2) S sends a recursive query through nsA for a domain resolved by a name server of B, nsB and measures its round-trip time, 3) the latency between A and B is estimated as the difference between the time taken to perform the previous two operations.

may introduce errors in the latency prediction:

- First, DNS servers have higher connectivity and may not necessarily be co-located with the nodes they represent in the measurement process. This may result in very optimistic predictions. To minimize the potential errors, I measure latencies only among name servers that are in the same subnet as their associated nodes.

- Second, heavily-loaded DNS servers may cause King to underestimate latencies between many nodes. In Figure 3.2, if the server nsA is loaded, it may delay significantly both the DNS reply to S and the recursive query to nsB. The estimation $rtt(A, B) = rtt(S, A, B) - rtt(S, A)$ may be unusually low, which can lead to false triangle inequality violations. In my measurements, I

limit the number of measurement probes to avoid unnecessarily loading the DNS servers. I also remove from the final data sets those nodes that experience very low latency ($\leq$1ms) to most other nodes, as described by Dabek *et al.* [22].

- Finally, anecdotal evidence suggests that characteristics of the King measurement process such as the location of the user that performs the measurement (node S in Figure 3.2), the sampling interval, or the time of the measurement may also influence the results. In Section 3.3.2.2, I study the impact of these factors and present evidence to the contrary.

3.2.2 Data Sets

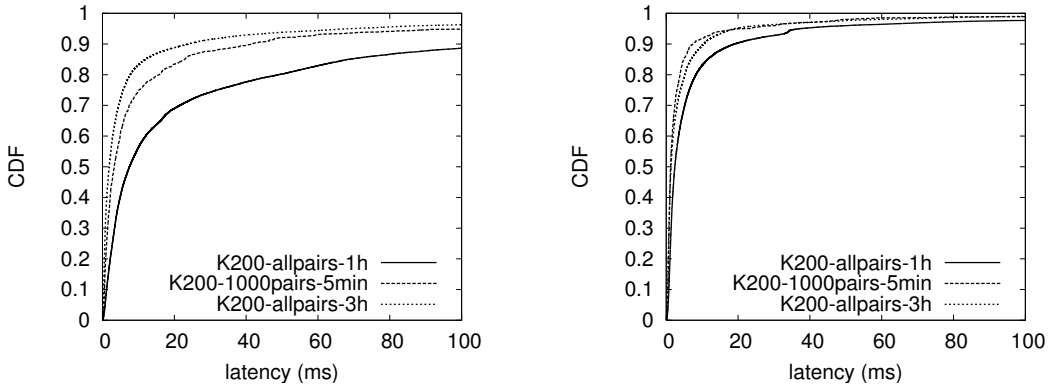
I collect three latency data sets at different time granularities. The IP addresses of the nodes in my measurements are of participants in the Gnutella network and are available through the Vivaldi project [22]. The chosen IPs share the same subnet with their authoritative name servers so that better-connected DNS servers would not influence the estimates of inter-client latencies.

I describe the properties of the data sets in Table 4.1. My goal is to collect data sets that are synchronous: all pairs of nodes are measured at least once within a predefined time interval. The size of the interval determines the granularity of the data set. I use three sampling intervals: 5 minutes, 1 hour, and 3 hours. At the

Data set	Nodes (Pairs)	Duration	Interval
K200-1000pairs-5min	200 (1000)	24h	5min
K200-allpairs-1h	200 (all)	44h	1h
K200-allpairs-3h	200 (all)	30h	3h

Table 3.1: Latency data sets. For each set I show: a) the name, b) the total number of nodes (and the number of pairs measured), c) the duration of the experiment, and d) the average interval between consecutive measurements of the same pair. All data sets were collected in the period March-April 2008.

beginning of each interval, I run King for all pairs of nodes in the data set from a computer at the University of Maryland. Each individual King measurement consists of four consecutive probes, out of which I keep the minimum value. Collecting latencies at smaller time granularities provides more accurate snapshots of the latency space. However, it also limits the number of pairs that I can measure accurately, without unnecessarily loading the DNS servers or the source computer. Thus, for the smaller granularities, I limit the scope of the measurement to 200 IP addresses (1000 pairs chosen at random for the 5 minute interval and all pairs for the 1 hour and 3 hour intervals). In Chapter 4, I describe the collection of a much larger data set (1715 IP addresses) with granularity of two days.



(a) (b)

Figure 3.3: Cumulative distributions for a) Standard deviation and b) Interquartile range for the data sets in Table 4.1

3.3 Latency variability

Latency variation on a path may lead to TIVs; conversely, if one perceives latencies to be varying (when the underlying path is stable), one may assert the existence of fake TIVs. In this section, I classify the causes of the recorded latency variations in my measurements. I show that the chances of inferring fake TIVs is small, and that most latency variation can be attributed to changes in load or changes in routing.

I begin with an overview of the measurements (which show that latencies do vary over time).

3.3.1 Measurements Vary Over Time

I study how end-to-end round-trip time varies for the duration of the measurement. I use two measures of variability: standard deviation (STD) and interquartile range (IQR). Standard deviation represents the variability of all data points equally, while interquartile range—the difference between the 75th and 25th percentiles—measures the variability of the 50% of points around the median. Figure 3.3 shows the STD and IQR for the three data sets. I make the following observations:

- All distributions have long tails; each data set has a few pairs of nodes that exhibit high variations in latency. 5% of the pairs in K200-1000pairs-5min and K200-allpairs-3h and 12% of the pairs in K200-allpairs-1h have standard deviations of more than 100ms.

- Second, in all data sets, less than 10% of the pairs have interquartile ranges of more than 40ms. Combined with the previous observation, this implies that the variability of the latency comes mainly from the more extreme values, rather than values closer to the median.

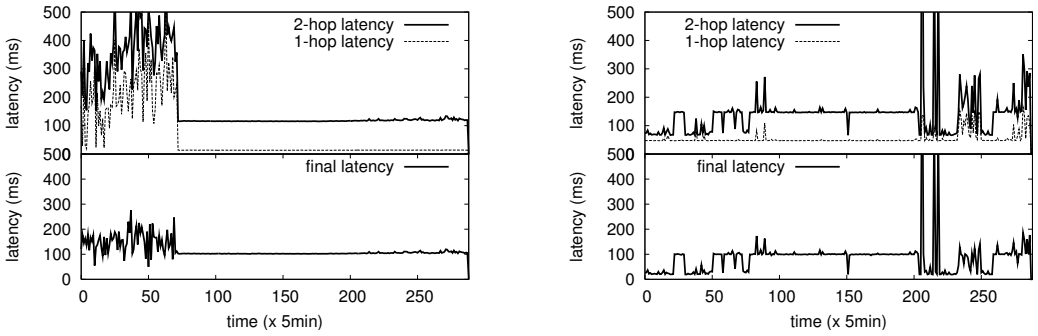
- Finally, the pairs in K200-allpairs-1h have higher standard deviations than the pairs in K200-allpairs-3h. This suggests that variability decreases with an increase in sampling interval. I confirm that this is true in Section 3.3.2.2.

3.3.2 Causes of Variations

Determining the exact cause that leads to each latency change is difficult. Instead, I classify the possible causes of variation into three categories: load-based, routing-based and measurement-based. Load-based causes refer to events such as queuing delay at the routers or transient load at the DNS servers involved in measurements. They are likely to manifest as short-duration spikes or oscillations [72]. The routing-based causes are path changes in the Internet determined by link or node failures or by routing changes. Although routes can also oscillate, their oscillations tend to have longer durations [41]. Thus, path changes are more likely to trigger longer-term changes in latencies. Measurement-based causes depend on the parameters of the measurement process. I consider three potential sources of variation: the sampling interval, the time at which we measure each sample and the location of the source of the measurement. Since I limited the number of pairs probed per sampling interval to avoid unnecessarily loading DNS servers, I do not consider load on name servers a measurement-based cause of variation.

3.3.2.1 Routing-based and Load-based causes

I focus first on the routing-based and load-based causes of variation. I select two pairs from the K200-1000pairs-5min data set and show their latency distributions in Figure 3.4. I define the latency from the source of the measurement to the first



(a) (b)

Figure 3.4: Examples of latency variations between pairs of nodes: a) the latency between 66.189.0.29 and 200.31.70.18 exhibits variations due to load; because both 1-hop and 2-hop latencies have similar variations, we conclude that it is either network load from S to A or DNS load on A; b) the latency between 216.61.143.252 and 147.136.250.51 varies during the measurement; besides the occasional spikes given by load, there are long periods of time (from 1h to 8h) when the latency changes significantly (by 70ms)

DNS server (nsA in Figure 3.2) as the 1-hop latency, and the latency from the source to nsB through nsA as the 2-hop latency. The final latency is obtained by subtracting the two values. I show the distributions of 1-hop and 2-hop latencies in the top Figure 3.4(a) and (b). Every point on the plot is associated with one measurement. I make the following observations:

- the variation of latency in Figure 3.4(a) exhibits many short-duration oscillations for the first 350 minutes; this is most likely a load-based event. On the right side of the plot, the latency does not oscillate at all.

- the variation of latency in Figure 3.4(b) shows fewer oscillations and the

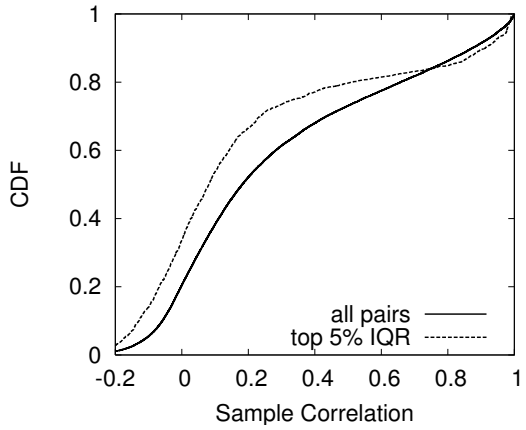


Figure 3.5: Cumulative distribution of sample correlation between 1-hop and 2-hop latencies for all pairs and only the top 5% of pairs when ordered by the interquartile range.

latency tends to stabilize around two values (30ms and 100ms) for periods ranging from 1 hour to 12 hours; this behavior suggests a routing-based event

- in Figure 3.4(a), the variations of the final, 1-hop and 2-hop latencies follow the same trends; in Figure 3.4(b), the 1-hop latencies remain constant over the first 200 intervals, while the 2-hop latencies change; this indicates the location of the event that causes the variation: a spike that appears on the 2-hop latency distribution but not on the 1-hop latency distribution must be caused by an event that occurred on the path between the two DNS servers

I compute the sample correlation for the 1-hop and 2-hop latencies for each pair in K200-allpairs-1h. I use this data set because it has the smallest time granularity of the ones that contain all-pair latencies and because it exhibits the greatest

variability. Figure 3.5 shows that there is less correlation among the pairs with the top 5% interquartile ranges—these are the more variable pairs. This indicates that the source of high variance typically lies between the two DNS servers probed by King. It also suggests that the source of the measurement has less impact on the variability of the data, as I confirm in Section 3.3.2.2.

I also compute the average absolute difference between consecutive measurements for each pair. This estimates how a prior measurement predicts a future one. A low average consecutive difference indicates that the data varies with low frequency (as in Figure 3.4(b)), while a high average consecutive difference indicates that the data varies with high frequency (as in Figure 3.4(a)). Figure 3.6 shows that for the pairs in the top 5% among interquartile ranges, the average consecutive difference is larger than for pairs in general. Indeed, one would expect that the more variable pairs have higher average consecutive differences. Less than 20% of those high-variance pairs have at most 30 ms of average consecutive differences; in those cases the cause of the variance is most likely due to path changes. For the remaining pairs, the variance changes rapidly; the source of the variance in those cases is most likely due to loaded DNS servers or high queuing delay at routers.

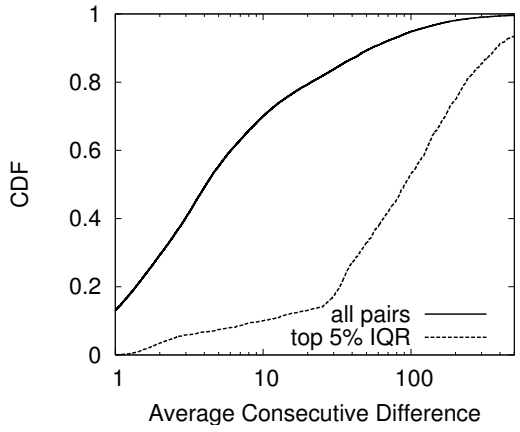


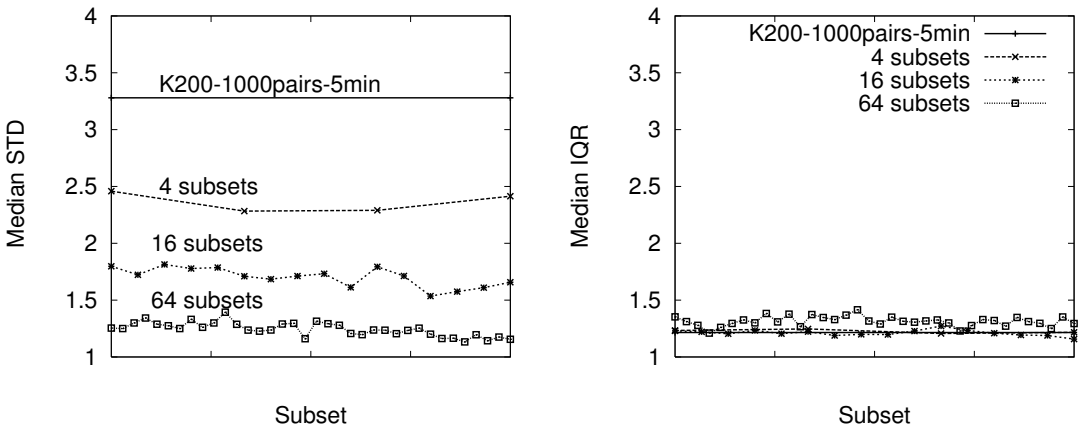
Figure 3.6: Cumulative distribution of the average difference between consecutive latency measurements.

3.3.2.2 Measurement-based causes of variation

Another source of variation may be the process of measurement itself. Next I verify whether the sampling interval, the time when each sample is measured, or the location of the measurer affect the variability of the data.

I split the K200-1000pairs-5min data set into k more coarsely grained subsets. In each subset, measurements for the same pair of nodes are collected at $k \times 5$ minute intervals. For example, when using $k = 4$ subsets, subset i contains all measurements taken at sample intervals i, $i + 4$, $i + 8$ and so on. Dividing the original data set in this way allows me to obtain k different measurement sets with $k \times 5$ minute sampling intervals. All subsets appear to start at five minute intervals over the course of $k \times 5$ minutes.

I compute the standard deviation (STD) and the interquartile range (IQR)

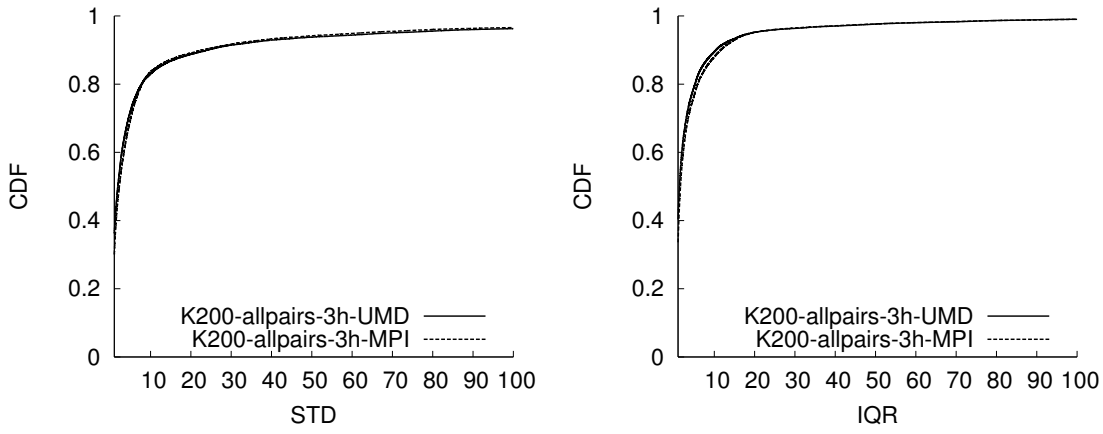


(a) (b)

Figure 3.7: Median a) standard deviation and b) interquartile range among the pairs in each subset in the K200-1000pairs-5min data set. Each point represents the median value for one of the subsets. As the sampling interval decreases, so does the median standard deviation.

for all pairs of nodes in each of the subsets, for $k = 1$ (the entire data set), $k = 4$, $k = 16$, and $k = 64$. Figure 3.7 shows the median STD and IQR. While the median STD decreases when using sparser samples, the median IQR remains approximately the same. We would expect that by sampling less often we are less likely to measure unusually high values. However, such values are always above the 75th percentile of the data, so they do not significantly affect the IQR. Also, the median STD and IQR do not change significantly between subsets, indicating that the time at which the measurement starts does not affect latency variance.

This analysis highlights a trade-off between sampling rate and the effect of rare high-latency measurements when we use a fixed list of the most recent latency



(a) (b)

Figure 3.8: Cumulative distributions for a) standard deviation, and b) interquartile range for two data sets collected simultaneously with experiments ran from University of Maryland, USA and Max Planck Institute for Software Systems, Germany.

measurements. A high-frequency sampling rate will observe more high-latency measurements, but those measurements will pass through the list more quickly. A low-frequency sampling rate does not observe high-latency measurements very often, but when it does, they remain in the list for a long period of time. This variation is typically mitigated through the use of median latency measurements, but since the IQR remains relatively stable it would also be safe to consider alternative aggregates (such as the mean) restricted to the middle 50% of the data.

Latencies measured with King may be subject to location bias. Since the source of the measurement is the same for all probes, it can introduce the same uncertainty in all measurements. I show next that such uncertainty is small.

I simultaneously start two King experiments and collect latencies among 200 IP addresses every three hours. Each experiment runs for 30 hours. The first experiment is run from a computer at University of Maryland and produces the data set K200-allpairs-3h-UMD, while the second is run from the Max Planck Institute for Software Systems, Germany and produces K200-allpairs-3h-MPI. For each pair in each data set, I compute its standard deviation and interquartile range and plot the cumulative distributions in Figure 3.8. The lines are almost identical showing that the location from where we run the measurements does not introduce additional uncertainty in the collected data. To confirm my findings, I again start simultaneously 10 King experiments measuring the same pair of nodes from 10 PlanetLab nodes distributed around the world. The variability of the data does not change with the location of the measurement.

3.4 Triangle inequality variations

In this section, I study the variation of triangle inequality violations and examine how well aggregate data sets that combine measurements taken over long periods of time capture the TIVs that were present during the measurements.

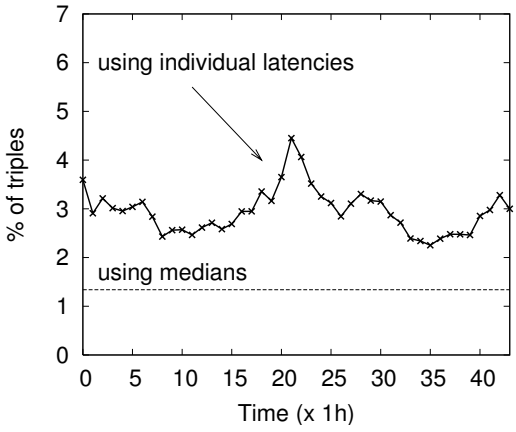


Figure 3.9: Percentage of TIVs of the total number of triangles for the K200-allpairs-1h data set.

3.4.1 TIVs vary over time

I count the number of triangle inequality violations after each sampling interval in the K200-allpairs-1h data set. I define a *good detour* as a detour that provides at least 10 ms and 10% latency reduction over the direct path. I consider only those violations that provide *good detours*. By considering only those violations that are significant, I protect my results from overstating the number of TIVs because of measurement error. Furthermore, applications that use triangle inequality violations to identify detour paths seek significant violations due to the overhead of relaying along the detour path.

Figure 3.9 shows the number of TIVs at every hour during the measurement. The vertical axis represents the percentage of bad triangles after each interval, out of all triples that have been measured during the interval. I define the median TIVs

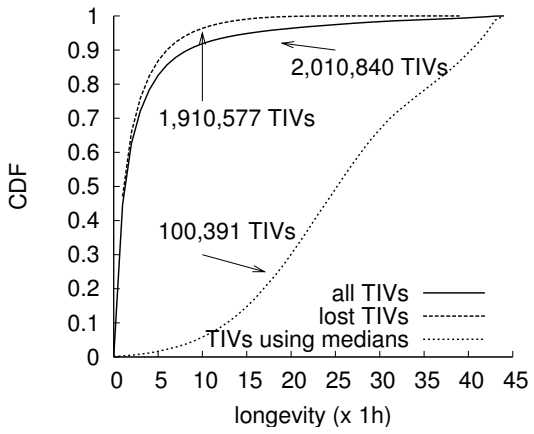


Figure 3.10: Cumulative distribution of the longevity of TIVs in the K200-allpairs-1h data set.

to be the TIVs computed using the median latency for each pair. The percentage of median TIVs is represented by the horizontal line at 1.34%. Figure 3.9 indicates that triangle inequality violations vary in time. However, at no point during the measurement process is the number of violations lower than what one would obtain using the medians. Thus, data sets that represent multiple measurements by their median values are conservative: they reveal fewer triangle inequalities than there were during the measurement process. Of course, if the lost TIVs are all short-lived, it may be beneficial not to reveal them; for instance, we only want to use long-lived TIVs for finding detour paths. I study next the longevity of TIVs.

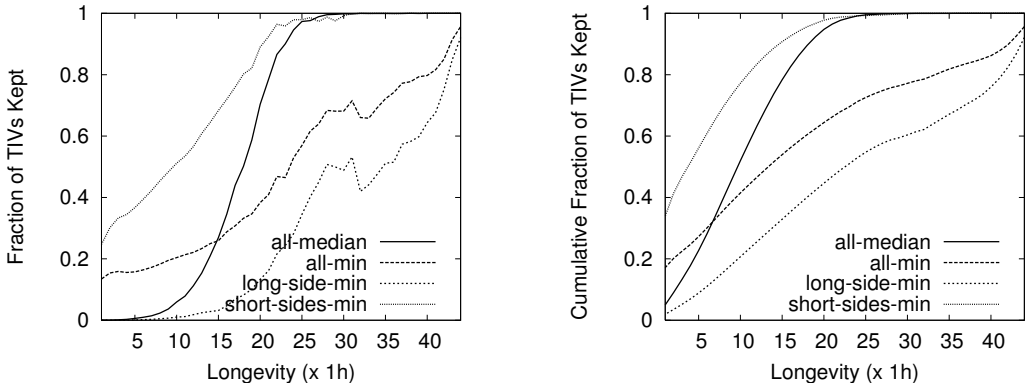


Figure 3.11: Probability and cumulative distributions of the fraction of TIVs that appear during the measurement and are preserved when computed on the aggregate data using one of the four methods: all-median, all-min, short-sides-min, long-side min.

3.4.2 Longevity

What happens to a TIV seen at one point during the measurement? Does it appear in the TIVs computed with medians? I expect that, due to extreme values in latency measurements, many triangles are short-lived—they are the effect of an unusually high latency.

I define the longevity of a TIV as the number of intervals in which it appears. I do not require the intervals to be consecutive to avoid bias due to missing or extreme measurements. I compute the longevity for three categories of TIVs: all TIVs seen during the measurement, all median TIVs and all TIVs seen during the measurement but not when using medians (lost TIVs). Figure 3.10 shows the cumulative distributions of longevities for TIVs in the three categories.

More than 80% of all TIVs have a longevity of less than 5 hours, while almost

all (≥99%) TIVs computed with medians are seen for more than 5 hours and more than half of them for more than a day. Thus, using medians eliminates the short-term TIVs.

Of all TIVs, only 18% have a longevity of more than 5 hours. However, of these long-lived TIVs, 72% (not shown in the figure) are lost—they do not appear as median TIVs. Such violations are present long enough to be able to help an overlay routing application such as PeerWise—by exposing a shorter detour—but are not captured when the measurements are aggregated.

Scenarios where the medians create a TIV that does not exist, as in Figure 3.1 are extremely infrequent. For example, 128 triangles (0.1%) appear only when using medians and never during the measurement. Using medians only ignores 1.5% of the TIVs that appear more than half the time using individual latency measurements.

3.4.3 Alternative ways to compute TIVs

To better understand the effects of latency aggregation on the performance of a latency-reducing detour routing application, I investigate four different ways to compute the number of TIVs: *all-median*, *short-sides-min*, *long-side-min* and *all-min*. I described *all-median* in Section 3.4.1. In *short-sides-min*, when I verify whether a triple forms a TIV, I consider the minimum latencies for the potential

Method	**Intermediate TIVs preserved**	**Intermediate TIVs (long≤5) preserved**	**Intermediate TIVs (long>5) preserved**	**Final TIVs false**
all-median	4.9%	0.1%	28.1%	0.006%
all-min	23%	21.6%	30%	6%
short-sides-min	49.1%	46.6%	60.8%	15.3%
long-side-min	1.9%	0.01%	11%	< 0.001%

Table 3.2: Percentage of TIVs preserved or added by the various methods out of the total number of TIVs in the corresponding categories. For instance, out of all intermediate TIVs, we preserve 49.1% with the short-sides-min method. 15.3% of the TIVs computed with short-sides-min do not appear at all during the measurement.

short sides and the median latency for the long side. In *long-side-min*, I use medians for the short sides and minimum for the long side. In *all-min*, as in other previous studies [109, 106], I use the minimum latency values for every edge of the triangle.

All-median is conservative. While it eliminates many short-term TIVs, it also ignores 72% of the TIVs longer than five hours that appear during the measurement (§ 3.4.2). Intuitively, the *long-side-min* method decreases the number of TIVs that are preserved and provides a more conservative data set for evaluating latency-reducing overlay networks. On the other hand, the *short-side-min* approach preserves more TIVs but many of them may be short-lived and thus

potentially useless for detour routing.

Figure 3.11 shows the probability and cumulative distributions of the fraction of TIVs that are kept by each of the four methods, based on their longevity. Every point represents the fraction of TIVs with each longevity that are kept.Table 3.2 summarizes the results.

Short-sides-min loses less long-lived TIVs than *all-median* but also keeps more short-lived TIVs. Of all TIVs longer than 5 hours, *all-median* keeps 28% while *short-sides-min* keeps almost 60%. Using either of the two methods will better reflect the performance of latency-reducing detour routing applications. *Short-sides-min* keeps more TIVs but also 15% of the TIVs it computes are false: they never appear in individual measurements. *All-min* and *long-side-min* keep about as many short-lived TIVs as *short-sides-min* and *all-median*. However, neither *all-min* nor *long-side-min* keep as many of the very long-lived TIVs as the other two methods.

In conclusion, the *short-side-min* method of computing TIVs is suitable for applications that require an upper-bound on the number of TIVs. *all-min* understates heavily the number of TIVs (it keeps only 5%) and thus does not provide an accurate latency snapshot for evaluation. In my evaluations, I use *all-median*, because it provides a more conservative estimation, biased towards keeping long-lived TIVs—which are more useful for latency-reducing detour routing—and losing

short-lived ones.

3.5 Summary

In this chapter, I showed how to measure triangle inequality violations to better assess their utility in building a latency-reducing overlay network. Existing latency data sets are inadequate for evaluating TIVs because they aggregate multiple measurements taken at different times over long periods. Using new data sets, collected with the King measurement tool, I showed that TIVs are not illusions of the measurement process but real properties of Internet latencies and that the number of TIVs varies with time. Aggregating multiple measurements using medians provides a conservative estimation of the number of TIVs that existed at any point during the measurement, and is better suited for assessing the feasibility of detour routing than latency aggregation methods based on minimum.

Chapter 4

Using Triangle Inequality Violations

In this chapter, I study the effects, both positive and negative, of using triangle inequality violations for latency reduction.

In the first part, I study whether triangle inequality violations expose significant shorter detours that could be used by PeerWise. I use real-world latency data sets to answer several technical questions: how many TIVs are there in the Internet? how much latency reduction do they provide? how many nodes can take advantage of them? do the one-hop paths provided by TIVs offer sufficient latency reduction or are more complex, multi-hop paths necessary? I show that, although there are few TIVs, many nodes can obtain significant latency improvement by exploiting them.

In the second part, I examine the interaction between triangle inequality violations and BGP, the interdomain routing protocol. I use measured and predicted AS paths to verify whether the detour paths exposed by TIVs violate routing policies enforced with BGP. Understanding the interaction between TIVs and BGP offers new insights into how ISPs and users can work together to avoid less-than-optimal

paths while maintaining their tussle in equilibrium [16].

4.1 Data sets

In Chapter 3, I have shown that aggregating multiple measurements using medians provides a conservative estimation for the number of triangle inequality violations. In turn, such an estimation offers a worst-case scenario for the evaluation of a latency-reducing detour routing system such as PeerWise: the fewer the TIVs, the fewer the potential detours. Next, I apply the lessons learned in the previous chapter to collect three larger, more diverse data sets, that are better suited to assess the feasibility of a latency-reducing overlay network such as PeerWise. I describe the data sets below and summarize them in Table 4.1 (ignore the last three columns for now).

4.1.1 Latencies

PeerWise-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 [22] project. I use King [28] to measure all-to-all latencies between the 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 [22]. For each pair of nodes, I kept the median of all latencies measured at random intervals

Data set	Size	When	TIVs (triples)	TIVs (pairs)	Latency Reduction
PeerWise-King	1715 x 1715	2008	2%	51%	77ms
PeerWise-PL	213 x 213	2009	2%	50%	61ms
PeerWise-PL-Dest	325 x 448	2008	2%	21%	63ms
Vivaldi-King	1740 x 1740	2004	2%	91%	47ms
Vivaldi-PL	384 x 384	2004	3%	62%	77ms

Table 4.1: Latency data sets for evaluating TIVs.

for 20 days in February 2008. Of the 1,953 servers, I removed 238 that appeared to experience high load during the measurement, as described by Dabek *et al.* [22].

PeerWise-PL contains RTTs between 213 PlanetLab nodes, measured in January 2009. I selected one PlanetLab node per site and measured all-to-all latencies at random times over a week. The final data set contains the median values of the measurements.

PeerWise-PL-Dest contains RTTs from 389 PlanetLab nodes to 500 popular web servers, measured in January 2008. I selected the servers based on a ranking by the Alexa Internet Company [3] 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, I performed DNS lookups on each of the 500 names from the 389 PlanetLab nodes. I obtained 2932 distinct IP addresses in 796 /24 prefixes. I probed each prefix and each PlanetLab node from every PlanetLab node at random times over a week. I used the median RTT values to represent the link.

The latency collection process can produce incorrect data that may bias the results. I 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. I removed those nodes from the data set since they would artificially overstate the potential of PeerWise. The final latency matrix contains RTT values from 325 PlanetLab nodes to 718 prefixes corresponding to 448 websites.

I also use two data sets, Vivaldi-King and Vivaldi-PL, collected by Dabek *et al*. for the Vivaldi project, in 2004. Vivaldi-King was collected between 1740 IP addresses, in a similar manner as PeerWise-King. Vivaldi-PL captures all-to-all latencies between 384 PlanetLab nodes, but unlike PeerWise-PL, it contains more than one node per site.

The five data sets illustrate two scenarios in which PeerWise can be useful. La-

tency reduction on the symmetric data sets (PeerWise-King, PeerWise-PL, Vivaldi-King, Vivaldi-PL) shows the potential benefit to applications a set of peers may run, such as distributed multi-player network games or VoIP applications. On the non-symmetric PeerWise-PL-Dest, reduced latency shows benefit for users accessing popular servers that would not participate in PeerWise.

For each pair of nodes in the data sets, I find all one-hop detours. I consider only good detours, as in Section 3.4.1. This cutoff helps avoid impractical or dubious detours due to measurement error. In the PeerWise-PL-Dest data set, I find detours by server name: the detour path may end at a different IP address associated with the same name. I find that out of all triples of nodes in our data sets, at most 3% violate the triangle inequality: two of the nodes in the triple benefit from a *good detour* through the third node. As I will show next, although small, the number of TIVs accounts for many pairs of nodes that could benefit from a detour.

4.1.2 AS Paths

Understanding the AS paths beneath TIVs allows me to study their compliance with known routing policies and thus to assess the negative effects of exploiting TIVs for latency reduction. I select the PeerWise-King data set, because it is the largest, and augment it with AS paths between all pairs of nodes. To compute

as many AS paths as possible I use several sources: RouteViews, Looking Glass servers and iPlane [61]. To the best of my knowledge this is the first large latency data set between Internet hosts augmented with AS path information computed at the same time.

RouteViews [84] collects and archives BGP routing tables and updates from commercial ISPs. I gathered AS path information from 44 BGP core routers located in 38 ISPs in March 2008. In addition, I used paths obtained by Madhyastha *et al.* [61] by probing around 25,000 BGP prefixes from 180 public Looking Glass servers.

I augment RouteViews and Looking Glass measured paths with paths predicted by iPlane. iPlane measures paths from 300 PlanetLab sites to more than 140,000 BGP prefixes to predict end-to-end paths between any pair of hosts. The predicted path combines partial segments of known paths, exploiting the observation that routes from nearby sources tend to be similar [60].

I found AS paths for the pairs of nodes in the data set, 10.4% from RouteViews and 13.6% from Looking Glass. The reason for such low completeness is that most of the Looking Glass servers and RouteViews peers are close to the core of the Internet and are unlikely to capture paths between two edge ASes. iPlane predicts AS paths between 71.7% of the pairs. By combining RouteViews, Looking Glass, and iPlane, I find AS paths for almost 75% of the pairs of nodes in the data set.

4.2 TIVs and latency reduction

4.2.1 TIVs are important

Previous research on network coordinates [22] and network location [106] has generally discarded triangle inequality violations because their number is small compared to the total number of triples that do satisfy the triangle inequality. I confirm that the number of TIVs is indeed low but I also show that their effect on latency reduction can be significant.

For each data set, I count the number of triples that form bad triangles and the number of pairs of nodes that are long sides in bad triangles (*i.e.*, pairs that have an alternate shorter path). As in Chapter 3, to minimize the effect of the measurement process, I consider only TIVs that provide good detours (more than 10 ms and 10% latency reduction). I report the results in Table 4.1. Although the number of bad triangles is relatively low (less than 3%), they account for many paths not being shortest (at least 50% for the symmetric data sets). These results agree with those reported by Ledlie *et al.* [43] and underline the importance of detecting and exploiting TIVs. Although the percentage of triples is similar to the other data sets, there are fewer pairs that can use TIVs in the PeerWise-PL-Dest. This could be explained by the popularity of the web servers: they are over-provisioned and already have good paths leading to them.

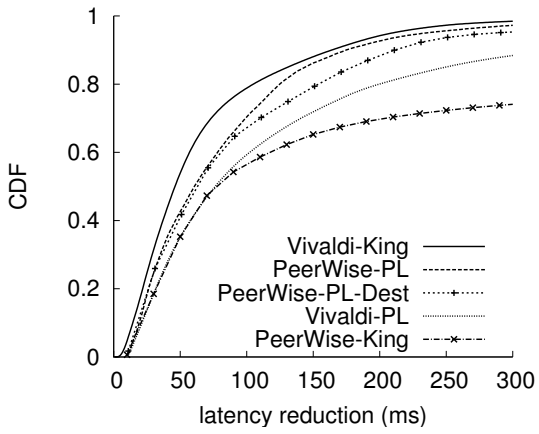


Figure 4.1: Cumulative distribution of potential latency reduction from TIVs, for the five data sets in Table 4.1

4.2.1.1 TIVs offer significant latency reduction

The latency reduction of a TIV is the difference between the length of the long side (the direct path) and the sum of the short sides (the detour path). If the latency reduction is low, even if many pairs of nodes benefit from TIVs, their benefit would be small. I show next that this is not the case.

I compute the latency reduction for every TIV in each of our data sets and plot the cumulative distribution in Figure 4.1. Recall that I have already limited the number of TIVs to those that offer at least 10ms or 10% reduction. More than 80% of the TIVs in each data set offer latency reduction of more than 20ms and the majority of TIVs improve the direct path with at least 50ms. I show the median latency reductions in Table 4.1.

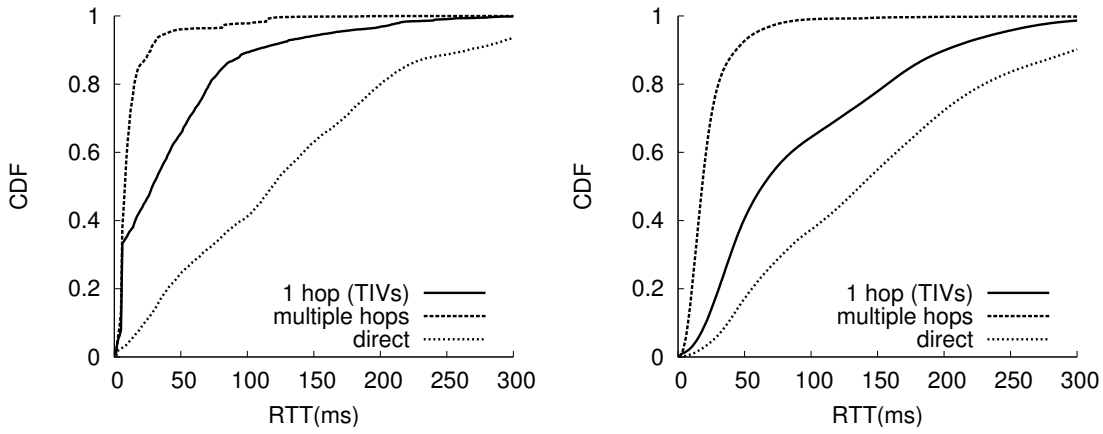
Data set	Improvement over direct path	
	one-hop (TIVs)	multiple hops
PeerWise-King	38%	70%
PeerWise-PL	60%	83%
PeerWise-PL-Dest	34%	73%
Vivaldi-King	61%	72%
Vivaldi-PL	60%	80%

Table 4.2: Latency improvement achieved with one-hop and multiple-hop paths.

4.2.1.2 One hop is enough

In discovering low-latency paths, it is tempting to allow paths of arbitrary length. However, the cost of optimal latencies is high; finding the paths would require an expensive routing protocol such as AODV [73], and ensuring cooperation across multiple hops is difficult [4,47,112]. Gummadi *et al.* observed that relaying through a single intermediate hop could escape many network failures [29]. I present a similar result for reducing latency: that limiting paths to a single hop—exposed by TIVs—is enough.

I compare, for each pair of nodes, the direct path latency, the latency on the best one-hop path (the detour path latency of a TIV in which the pair of nodes is long side) and the shortest path latency (allowing multiple hops, computed



(a) (b)

Figure 4.2: Allowing one-hop detours achieves good latency in **a)** PeerWise-PL and **b)** PeerWise-King.

using Dijkstra's algorithm). Figure 4.2 presents the latency distributions for the PeerWise-King and PeerWise-PL data sets. I show relative latency improvements over the direct path in Table 4.1. Using one-hop detours (TIVs) achieves almost half of the latency reduction possible over the shortest path.

4.3 TIVs and BGP

In the previous section, I have shown that triangle inequality violations expose shorter paths that could benefit many nodes in the Internet. In this section, I will study the negative effects of exploiting TIVs for latency reduction. Sending traffic along the detour paths, exposed by TIVs, instead of default paths, chosen by BGP, has the potential to disrupt traffic engineering and policy routing in the

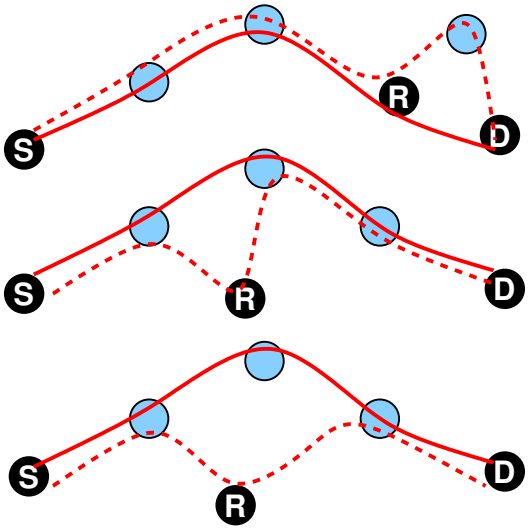
Total Detours		793,693	
Impossible AS Paths		460,830	(58%)
Cause	Customer transit	343,381	(75%)
	Peer transit	117,449	(25%)
Type	Truly disjoint	302,207	(66%)
	Borderline	153,057	(33%)
	Undercover	5,503	(1%)
Possible AS Paths		197,453	(25%)
Traffic Eng.	Relay AS not on direct path	56,813	(29%)
	Direct, detour paths differ	103,215	(52%)
	Direct, detour paths same	37,425	(19%)
Path length	Shorter than direct	17,770	(9%)
	Equal to direct	75,032	(38%)
	Longer than direct	104,651	(53%)
Transit cost	Smaller than direct	35,541	(18%)
	Equal to direct	96,751	(49%)
	Greater than direct	65,161	(33%)
Unknown		135,410	(17%)

Table 4.3: Detour paths are *possible* (may be available to the BGP decision process) or *impossible* (not advertised by BGP). Percentages inside the tables are relative to the total possible or impossible paths. Categories separated by horizontal lines overlap.

Internet [81]. For example, in a TIV ABC, the detour path (A, B, C) may violate the transit agreements between the ISPs of A, B and C (maybe because B's ISP is a customer of both A's and C's ISPs). Do all shorter detour paths violate policies? Or are they simply not selected by BGP because of its lack of mechanisms to minimize delay? Do detour paths traverse a different set of ASes that makes them more attractive from the users' perspective, but less attractive from the ISPs' perspective? Answering such questions is important for understanding the effects of exploiting TIVs for end-to-end latency reduction.

It is not surprising that the BGP path selection process may prefer longer, policy-compliant paths to shorter, policy-violating detour paths. I ask the following question: to what extent are detour AS paths available to BGP?

I separate all AS detour paths in the data sets into two categories: *impossible* and *possible*. A path is *impossible* when it could not have been advertised by a neighbor, possibly because it could not have been advertised by a neighbor's neighbor and so on. Common inter-domain routing rules [25] state that customers should not advertise routes learned from a provider to peers or other providers. This prevents the customer from being used as transit between two of its providers (customer transit). Similarly, routes learned from peers are advertised to customers and not to providers or other peers, preventing peer transit. Otherwise, a path appears *possible*, though traffic engineering or other rules may have led to the



undercover

depth = 0, width = 1

---

borderline

depth = 1, width = 1

---

truly disjoint

depth = 1, width = 2

Figure 4.3: Examples of impossible detour AS paths.

selection of an alternate.

To assess whether detour paths traverse possible or impossible paths, I use the AS relationships inferred by CAIDA [12]. Directed AS edges belong to one of four categories: customer-to-provider, provider-to-customer, peer-to-peer and sibling-to-sibling. A policy compliant AS path should have zero or more customer-to-provider edges followed by zero or one peer-to-peer edges, followed by zero or more provider-to-customer edges. Sibling-to-sibling edges may appear anywhere on the path.

Table 4.3 classifies the detour paths. The row labeled "Unknown" corresponds to the AS paths for which we cannot give an indisputable classification using the AS relationship data set. 58% of the detour paths in the data set are non-compliant (*i.e.*, include customer or peer transit). This is not surprising, since detour paths go through end hosts, which are generally customers and may be in stub ASes. I describe the cells of this table in the following discussion, first for impossible, and then for possible paths.

4.3.1 How Impossible Are the Impossible Paths?

I ask the following question: How severe are the policy violations of the impossible paths? For each detour path I define its prefix and its suffix. The prefix is the longest common subpath to appear at the beginning of both the detour path and a policy compliant path between the same pair of nodes, while the suffix is the longest common subpath to appear at their end. Based on the prefix and the suffix, I define two measures to capture the severity of policy violation of a detour path: *width* and *depth*. The width is the number of valid AS edges that would be required to connect the suffix and the prefix to obtain a policy compliant path. The depth is the minimum number of AS edges that have to be traversed from the relay to the end of the prefix or the beginning of the suffix. Based on the values of width and depth I classify the impossible detour paths into *undercover*, *borderline*,

and *truly disjoint*. I present an example of each type in Figure 4.3 and describe them below:

undercover (depth = 0) (1% of impossible detour AS paths)

Because the depth is 0, the relay of the detour lies on a compliant path. Although both direct and detour traffic enter the AS of the relay, they use different peering points to exit.

borderline (depth = 1, width $\leq$ 1) (33%)

Borderline compliant detours diverge from the compliant path only to traverse the relay before returning quickly. These paths might be discovered by BGP, given enough information about the relay location.

truly disjoint (all other cases) (66%)

These paths are disjoint enough from any compliant path that we do not believe BGP could find them.

To summarize, approximately a third of the "impossible" paths may in fact be available to BGP.

4.3.2 Possible Paths

25% of the detours in the data set follow compliant paths. Therefore, they can be learned by BGP. Only traffic engineering decisions or a lack of configuration

can stop these paths from being advertised and learned. BGP routers select paths based on cost, performance, length, and even which path is advertised first. Since I do not know precisely why any path was chosen, I consider here a few possible explanations.

4.3.2.1 Traffic Engineering

Each AS must pay some cost to carry traffic in its internal network. ISPs engineer their networks and routing to minimize this cost, while improving performance, choosing early-exit routes that deliver packets at the nearest exit, or divert traffic to balance load. Although I do not have explicit information about these choices, I infer when such traffic engineering occurs. For example, for 41% of the possible detour paths, the AS of the relay node lies *on the direct path*, yet the detour and the direct paths are different. This may occur because traffic, when redirected through the relay, will traverse a different peering point than the default traffic.

These results suggest that detours may take advantage of shorter paths by overriding common traffic engineering practice. The number of detours due to minimizing internal cost may be higher than we have observed; we can only identify such detours when the relay is on the direct path.

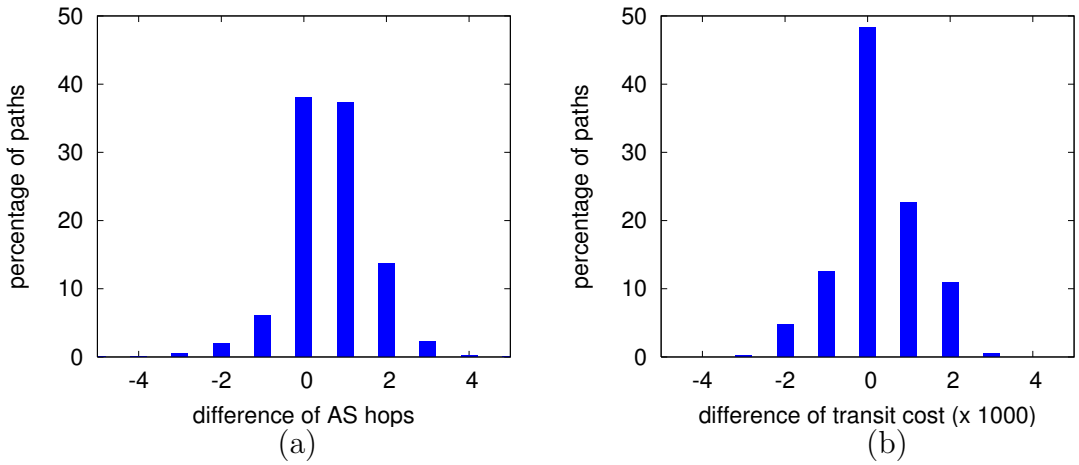


Figure 4.4: Possible detour AS paths have bigger (a) path length, and (b) transit cost.

4.3.2.2 Path Length

When choosing among otherwise equal paths, BGP selects the one with the fewest ASes. Because a detour path traverses an additional relay point, we expect it to use more ASes than the corresponding direct path. For each pair of nodes, I compute the difference in number of AS hops between the detour path and the corresponding direct path. Over 80% of the *possible* detour paths traverse at least as many ASes as the corresponding direct paths. This suggests that latency is not reduced by eliminating ASes traversed.

4.3.2.3 Transit Cost

Although not visible in BGP data, the price an ISP pays to its provider may make a path more or less preferred. Traversing larger networks implies greater

expense. I define the transit cost of a path as the maximum degree—number of AS-to-AS peerings—of all ASes on the path. Table 4.3 contains the results of the comparison between the transit cost of detour paths and corresponding direct paths. The transit cost of the detour paths is significantly higher than that of direct paths.

4.4 Summary

In this chapter, I used real world latency and AS path data sets to examine the positive and negative effects of exploiting triangle inequality violations for latency reduction. I have shown that TIVs are an important resource for improving end-to-end latency. Although the number of TIVs in my data sets is relatively small ($< 5\%$ of all triples), more than half of the nodes can exploit them to find shorter detours.

I also described how the shorter paths provided by TIVs compare with the default paths offered by BGP. As one might expect, many of these shorter paths appear impossible to BGP: providers or peers provide transit. Almost one-third of these paths, however, are compliant: internal traffic engineering decisions and not BGP may lead to longer paths being chosen as default. This shows that finding detours routes that abide by routing policies is possible. Further, of the non-compliant paths, one third could be made compliant: the detour path uses

a direct customer to transit between two peering ASes that could, if properly configured, provide the shorter, router-level path.

Chapter 5

Discovering detours with network coordinates

In this chapter, I show how to scalably detect TIVs and predict detours. I have shown in the previous chapter that TIVs can provide detours for many nodes. One way to discover these detours is to measure latencies on all possible paths between a source and a destination. However, this approach would limit the scalability of a latency-reducing overlay. I use network coordinates to find detours while significantly decreasing the number of measurements a node must perform.

Network coordinate systems [69,22,65,51,20,89,76,99,110,104,1] are a popular approach for estimating the latency between two nodes with few measurements. When run on a set of nodes, they assign each node a position in a finite metric space (also called *the embedding space*) and estimate the latency between two nodes as the distance between their coordinates in the space. The measurement cost is much reduced because nodes have to collect latencies to only a small set of other nodes. Further, a node can make instant latency estimations, without waiting for measurements to finish.

Network coordinates are not perfectly accurate. Triangle inequality violations

are one reason for the inaccuracy. Because TIVs are not allowed in metric spaces by definition, the latency estimations computed by network coordinates do not always match the real latencies between nodes. I show that one can use these discrepancies to predict the existence of triangle inequality violations without performing extensive measurements.

5.1 Network coordinates

Network coordinate systems [69, 22, 65, 51, 20, 89, 76, 99] provide a scalable method to estimate latency between nodes in the Internet without measurement. Their key insight is to associate nodes with coordinates in a geometric space that characterize their location in the network, and estimate the latency between two nodes as the distance between their coordinates. In this section, I review well-known coordinate systems and discuss their properties and design decisions.

The research on network coordinates is abundant. I enumerate the most established systems: GNP [69], Vivaldi [22], IDES [65], ICS [51], PIC [20], BBS [89], Lighthouses [76], Virtual Landmarks [99], and Pyxida [80]. They all converge in their desired properties, to scalably and accurately estimate Internet latency, but diverge in the design decisions that lead to these properties.

The most important properties of any coordinate system are accuracy and scalability. *Accuracy* is the property of a system to produce coordinates for its

nodes such that the difference between the real distance and the estimated distance between two nodes is minimized. *Scalability* ensures that the performance of the system does not degrade when the number of participants increases.

All coordinate systems have three important components to their designs: *space selection*, *probing* and *positioning*. Space selection involves how to calculate the distances between points, whether the embedding space is Euclidean, how many dimensions it has, etc. Probing is the process of measuring latency to a few peers—other nodes participating in the system—, or landmarks. Positioning is the optimization process of using probe results to assign a coordinate to every node in space.

5.1.1 Internet modelling and space selection

The first stage in the design of a network coordinate system is choosing the geometric space of the embedding. Ideally, one would like to use a space that best fits the Internet, such that the metric of the space approximates Internet latencies. GNP, Lighthouses, Virtual Landmarks, ICS, PIC and Vivaldi use an n-dimensional Euclidean coordinate space, motivated by the fact that latencies in the Internet are dominated by geographic distance.

Many systems use adjusted Euclidean spaces or non-Euclidean spaces to model the Internet. Dabek *et al*. [22] introduce spherical coordinates, motivated by the

fact that the modeled distances are computed on the spherical surface of the Earth. However, since paths in the Internet do not wrap around the Earth, the authors have abandoned the spherical model for a simpler quasi-Euclidean space: the height model augments n-dimensional Euclidean spaces with a height that captures the time needed to traverse the access links from a node to the core of the Internet. Lee *et al.* [46] add a localized adjustment term to Euclidean coordinates to account for the non-Euclidean effect of triangle inequality violations. Shavitt and Tenkel [90] propose Hyperbolic spaces, motivated by the jellyfish structure—a core in the middle with many tendrils—of the Internet [100].

5.1.2 Probing and data collection

To compute its network coordinate, a node must first measure latencies to a set of other nodes, which we call landmarks. The landmarks may or may not be participants to the system. The selection of landmarks and measurement data collected is very important in allowing nodes to correctly position themselves in the embedding space.

GNP, Lighthouses, Virtual Landmarks, ICS and IDES use a fixed infrastructure of landmarks to determine coordinates. GNP uses all known landmarks to infer coordinates. To mitigate the effect of landmark failures, Lighthouses, ICS and IDES, allow nodes to probe any subset of landmarks. In particular, IDES offers

nodes the possibility to measure distances not only to a subset of the predefined landmarks but also to other nodes that have already computed their coordinates.

PIC, Vivaldi and BBS do not need specialized infrastructure nodes. In PIC, a new participating node can pick any node, whose coordinates have already been computed, as a landmark. The authors propose three different strategies: pick nodes at random, pick the closest nodes, and pick some nodes at random and others as the closest nodes. Dabek *et al*. find that, in Vivaldi, the best results are obtained when choosing preferentially landmarks that are closer but also communicating with some distant nodes. Vivaldi does not require the selected landmark to have computed its coordinates, but it requires each node to know the accuracy of its own coordinate.

Measuring the latency to a node by probing it may be expensive for the node since others may probe it at the same time. Thus, when possible the probing is piggybacked on application traffic. This is especially effective in the systems that do not rely on infrastructure nodes to position themselves.

5.1.3 Positioning

After measuring latencies to the landmarks, each node uses the measurements to compute its own network coordinate. In GNP, PIC, Vivaldi and BBS, each node assigns initial values to its coordinate and starts a numerical optimization process

whose goal is to minimize a predefined error objective function. The error captures the difference between real latencies and embedded latencies from the node to its landmarks. In Lighthouses, ICS, Virtual Landmarks and IDES, participants use the measured latencies to the landmarks as their initial coordinate vector (Lipschitz embedding). They apply different dimensionality reduction methods to eliminate the dimensions which have the least impact on the final position.

GNP and PIC use the Simplex Downhill method to minimize the sum of squared relative errors. Unfortunately, the Simplex Downhill is an algorithm that converges very slowly and its results vary based on the initial coordinates assigned to the nodes. Furthermore, the method does not guarantee convergence, risking of getting stuck in local minimum regions. Other approaches have obtained better convergence by relating to optimization problems in physics. Vivaldi chooses the best coordinates by simulating a network of springs, with a spring placed between every pair of nodes in the network. The rest length of the spring is the real distance between the nodes and the current length is the embedding distance. The algorithm minimizes the squared-error of the embedding by simulating the movement of each spring towards the low energy position. I provide more details on Vivaldi in Section 5.3.1. BBS [89, 90] simulates a set of particles in a force field, each particle corresponding to a node. Each pair of particles is affected by a force depending on the embedding error of the distance between the associated nodes. The goal of

the optimization is to find the position of each particle such that its potential is minimized.

Lighthouses allows any node to compute its coordinates by probing any subset of the landmark nodes and determining the local base through Gram-Schmidt orthogonalization. ICS and Virtual Landmarks use Principal Component Analysis (PCA) to reduce dimensionality. PCA is based on matrix factorization and transforms a data set that consists of a large number of correlated variables into a new set of uncorrelated variables, which characterize the network topology. IDES associates two coordinate vectors to each node, an incoming vector and an outgoing vector. The two vectors are derived after the factorization of the matrix comprising the distances to landmarks. The estimated distance from a node A to a node B is the dot product between the outgoing vector of A and the incoming vector of B.

5.2 Network coordinates and TIVs

In this section, I show why network coordinate systems cannot estimate Internet latencies perfectly and how I use this inacurracy to scalably discover detours. In Section 5.3, I present evaluation results that support my claims.

Most network coordinate systems embed Internet nodes into metric spaces. A metric space is an ordered pair (M, d), where M is a set of points and d a distance

function on M, such that, $\forall x, y, z \in M$, the following properties hold:

$$d(x,y) \geq 0 \text{ (non-negativity)} \tag{5.1}$$

$$d(x,y) = 0 \text{ if and only if } x = y \text{ (identity of indiscernibles)} \tag{5.2}$$

$$d(x,y) = d(y,x) \text{ (symmetry)} \tag{5.3}$$

$$d(x,z) \leq d(x,y) + d(z,y) \text{ (triangle inequality)} \tag{5.4}$$

The space formed by all nodes in the Internet, with the latency (or the round-trip time) as the distance between two nodes, is not a metric space. While it respects the non-negativity and identity of indiscernibles properties, the Internet latency space fails to satisfy the symmetry and the triangle inequality properties. Asymmetric routing [71,42], which occurs when the path from node A to node B is different from the path from node B to node A, leads to the asymmetry of latency. In Chapters 2 and 3, I described the causes of triangle inequality violations. Both asymmetric routing and triangle inequality violations are inconvenient for network coordinate systems. Because my goal is to detect detours, I focus here only on how TIVs impact the accuracy of coordinates.

Any three Internet nodes that violate the triangle inequality cannot be embedded accurately into a space that prohibits TIVs, such as any metric space. Inherently, the more triangle inequality violations there are, the more imprecise the embedding. Although detrimental to the distance estimation, the inaccuracy in coordinates introduced by embedding a bad triangle can be helpful in determin-

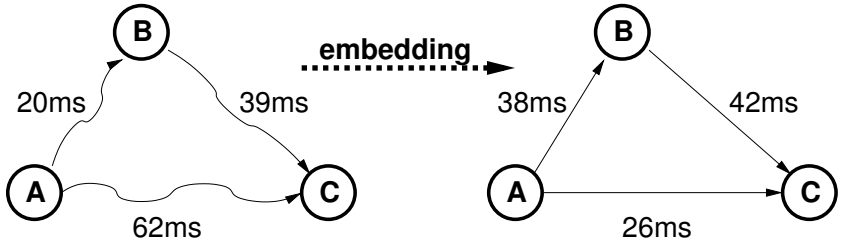


Figure 5.1: Embedding three points that form a TIV into a metric space introduces inaccuracies.

ing which nodes and links belong to bad triangles. With this information, nodes could proactively search for shorter detours or advertise their position as relays in shorter detours for others.

How can triangle inequality violations impact the embedding? I define the embedding error (or simply, the error) ε between a pair of nodes A and B as the difference between the embedding distance and the real distance between A and B.

$$\varepsilon(A, B) = d(A, B) - rtt(A, B) \tag{5.5}$$

Embedding a bad triangle ABC into a metric space may cause high errors on the edges of the triangle (see Figure 5.1). The errors must be significant enough to reverse the sign of the triangle inequality. Thus,

$$rtt(A, C) > rtt(A, B) + rtt(B, C) \tag{5.6}$$

becomes

$$d(A, C) \leq d(A, B) + d(B, C) \tag{5.7}$$

This, in turn, implies one of following three possibilities:

- the error of the long side is negative ($d(A,C) - rtt(A,C) < 0$)
- the sum of errors of the short sides is positive ($(d(A,B) - rtt(A,B)) + (d(B,C) - rtt(B,C)) > 0$)
- both

Consequently, *I expect that the more negative the error of an edge, the higher the probability that the edge is a long side in a triangle inequality violation. Conversely, the more positive the error, the better the chances for the edge to be a short side in a TIV.* I show this to be true in the next section.

5.3 Evaluation

In this section, I use simulations of the Vivaldi network coordinate system to evaluate two hypotheses:

- triangle inequality violations impact the accuracy of network coordinates
- the embedding error between two Internet nodes indicates whether the nodes are more likely to form the long side of a TIV (need a detour) or the short side of a TIV (provide a detour)

I consider two of the three main data sets described in Chapter 4: PeerWise-King and PeerWise-PL. I do not experiment with PeerWise-PL-Dest because it is

intended to assess the performance of a system where not all nodes participate in PeerWise and therefore run a network coordinate system.

5.3.1 Vivaldi

I chose the Vivaldi network coordinate system because it is distributed and adaptive, running without global state and accommodating the dynamics of the network. Vivaldi simulates a system of springs where each spring corresponds to a pair of nodes. The rest length of a spring emulates the real distance between two nodes while the actual length is the distance computed by the embedding. The energy of each spring is proportional to its displacement (the difference between the rest length and the current length). The algorithm runs iteratively at each node and simulates the progress of the springs toward a state with minimum energy. At every step, each node will be pushed to a new position that minimizes the displacement of the springs it is connected to. Two factors affect the position of a node after each step: the magnitude (M) and the direction (D) of movement. M is proportional to the displacement of the associated springs and D is the opposite of the gradient of the energy function with respect to the position of the node. After computing the magnitude and the direction of movement the following rule updates the coordinates of a node:

$$x = x + \delta \times M \times D$$

where δ is the timestep between two consecutive updates.

I consider an n-dimensional coordinate space. The energy of a spring between nodes x and y is:

$$E_{xy} = \frac{1}{2}k(rtt(x,y) - d(x,y))^2$$

where $rtt(x,y) - d(x,y)$ is the displacement of the spring and k is the elasticity constant.

Dabek *et al.* [22] use coordinates in Euclidean space augmented with a height value h. The distance between two nodes x and y, with $x = (x_1, x_2, \ldots, x_n, h_x)$ and $y = (y_1, y_2, \ldots, y_n, h_y)$, is:

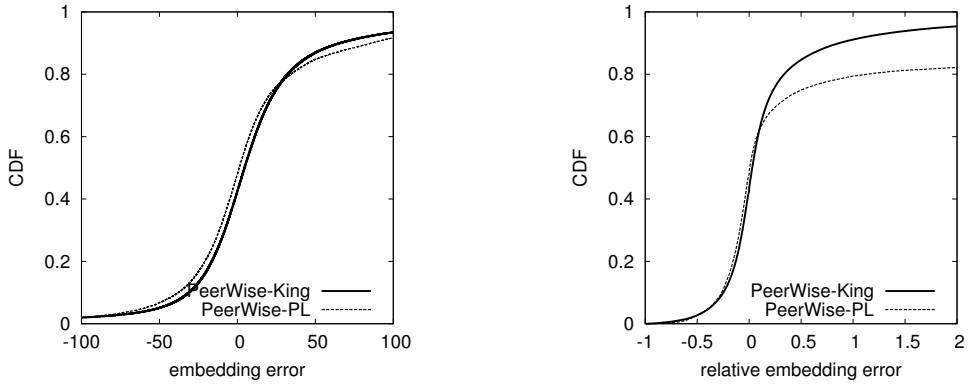
$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} + h_x + h_y$$

After computing the gradient of the energy function, they obtain the following expressions for M and D:

$$M = \frac{rtt(x,y) - d(x,y)}{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}} \qquad D = x - y$$

5.3.2 Methodology

I use the Harvard Vivaldi [43] network simulator. Each node has 32 neighbors: half selected as the closest nodes in network latency and the rest chosen at random. Each node starts at the origin of the space, and moves using Vivaldi's adaptive



(a) (b)

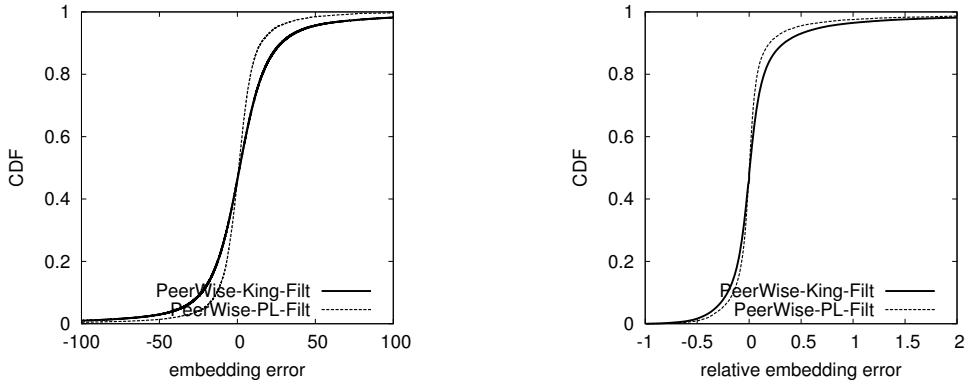
Figure 5.2: Cumulative distributions of **a)** absolute and **b)** relative errors for the PeerWise-King and PeerWise-PL data sets.

timestep to converge quickly. I choose a two-dimensional Euclidean space augmented with heights due to its simplicity and because it was shown to produce good embeddings [22], better than for other types of spaces such as Hyperbolic [58, 90]. In particular, Euclidean spaces have been motivated by the fact that latencies in the Internet are dominated by geographic distance and that paths generally do not "wrap around" the Earth [22].

5.3.3 TIVs impact the accuracy

Next, I study the accuracy of the embeddings using error distributions and show how it depends on the number of TIVs.

I evaluate the accuracy of Euclidean Vivaldi by absolute and relative errors computed over all pairs of nodes in PeerWise-King and PeerWise-PL. Absolute



(a) (b)

Figure 5.3: Cumulative distributions of **a)** absolute and **b)** relative errors for the PeerWise-King-Filt and PeerWise-PL-Filt data sets.

error is the difference between the embedded distance and the real distance (Equation 5.5); relative error is the absolute error divided by the real distance. Other accuracy metrics, like relative rank loss [52] better capture the usefulness of the embedding for applications, but by relying only on the relative distance to nodes they tend to overstate the importance of small errors.

Figure 5.2(a) presents the distribution of the absolute embedding error for the data sets. Each point corresponds to one pair of nodes. Vivaldi exhibits similar distributions for both data sets, with more than 80% of the pairs having an error within the range [-50,50].

I also plot the distribution of relative errors in Figure 5.2(b). A relative error of 1 between a pair of nodes means that the embedded distance is twice the real

	% of all predicted distances		
Data set	within 25% of real	within 50% of real	within 100% of real
PeerWise-King	66%	82%	91%
PeerWise-King-Filt	80%	93%	96%
PeerWise-PL	61%	74%	81%
PeerWise-PL-Filt	86%	95%	98%

Table 5.1: Summary of prediction errors. We show, for each data set, the percentage of distances estimated to be within 25%, 50% and 100% of the real distances. Higher values are better.

distance; of -0.5 that the embedded distance is half the real distance. Of the two data sets, Vivaldi performs worst on PeerWise-PL. For example, almost 20% of the distances are predicted to be at least four times as large as the real distances in PeerWise-PL, compared to fewer than 5% in PeerWise-King. I summarize the error prediction results in Table 5.1.

I now verify whether the accuracy improves when I eliminate triangle inequality violations. Since most of the nodes participate in at least one TIV, I cannot eliminate all nodes that create a TIV. Instead, for each node, I count the number of TIVs that it participates in and eliminate from each data set the 10% of nodes that participate most TIVs. I obtain two new data sets, PeerWise-King-Filt and PeerWise-PL-Filt, of 1544 and 192 nodes. By doing the filtering (and removing

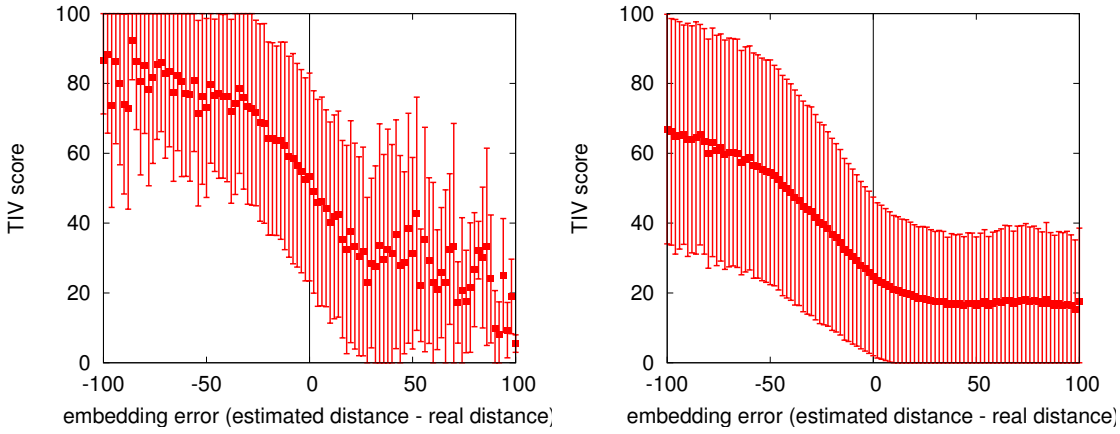
10% of the nodes), I am removing 83% of the TIVs in PeerWise-PL and 72% of the TIVs in PeerWise-King. Figure 5.3 shows the error distributions for new data sets. Table 5.1 presents the resulting accuracy statistics.

As expected, Vivaldi performs much better on the data sets with fewer triangle inequality violations. After removing the 10% of the nodes that participate in most TIVs, more than 80% of the distances are predicted to be within 25% of the real values. Triangle inequality violations can disrupt the embedding considerably. My results are similar to those obtained by Wang *et al.*, who showed that Vivaldi has improved accuracy when the edges that are part of triangle inequality violations are removed from the probing process [102].

5.3.4 Embedding errors indicate TIVs

I conjectured that as the error between two nodes decreases towards -∞, the probability that the two nodes form a long side in a TIV (rather than a short side) increases. Similarly, as the error increases towards ∞, the two nodes will rather be the end points of a short side than of a long side in a TIV.

To capture the presence of each pair of nodes in triangle inequality violations, I define the TIV score. A TIV score is given to each pair of nodes and represents the percentage of the number of times the nodes form a long side in a TIV out of the total number of times the two nodes are present in a TIV. A TIV score of 0 means



(a) (b)

Figure 5.4: Average number of TIVs versus estimation error for **a)** PeerWise-PL and **b)** PeerWise-King: as the estimation error decreases, it is more likely that the pair is a long side in a bad triangle.

that the pair appears only as short side, while a TIV score of 100 indicates that the two nodes form only long sides in TIVs. Unlike the TIV severity metric [102], which is computed only for the bad triangles in which an edge is long side, the TIV score accounts for all bad triangles in which an edge is present.

I compute the embedding error for each pair and plot it against the average TIV score in Figure 5.4, for the PeerWise-King and PeerWise-PL data sets. To create the plot, I took all pairs of nodes that had the same estimation error and averaged their TIV scores. The error bars correspond to one standard deviation in each direction. The figure shows that as the estimation error of a pair of nodes becomes more negative, the nodes form more and more long sides. When the estimation

error becomes larger, the number of short sides that a pair forms increases. Thus, a pair of nodes with a negative estimation error has a higher chance of needing a shorter path; when the nodes have a large estimation error between them, they are more likely to be part of a shorter path for another node.

5.4 Summary

In this chapter, I have shown that triangle inequality violations can be scalably detected with 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. These errors in Internet coordinates indicate the presence of TIVs and the type of error shows whether a path is a likely detour to exploit or a pathologically long link for which a detour can be found. Such a result is of the greatest importance in designing and building PeerWise. To find the best peers automatically, a PeerWise node would simply have to compute its network coordinate and use the error in the embedding (how over- or under-estimated any link latency is) to find Internet paths to be avoided or preferred.

Chapter 6

Mutual advantage

Triangle inequality violations provide an excellent opportunity for latency-reducing overlay routing. Participants benefit from detour paths significantly shorter than the corresponding direct paths and discover these detours with few measurements using network coordinates.

Such a simple setting is not fair for the intermediate nodes that provide the detours. The cost of using a detour is not paid by the node who benefits from it; it accumulates instead at the relay node, who must use its own resources to forward the detour traffic. The heterogeneity of connectivity of Internet users means that well-connected nodes are more likely to offer detours, while poorly connected nodes are more likely to need detours. This asymmetry may discourage the well-connected nodes from joining—they have little to gain and would pay highly. The design of a latency-reducing routing overlay should include an incentive mechanism, a means by which nodes compel each other to provide at least as much service as they receive.

I propose to introduce *mutual advantage* as a design principle in PeerWise:

overlay edges should exist only between nodes that provide detours for each other: each is an intermediate node in a detour for the other. Nodes negotiate connections based strictly on mutual advantage, and overlay paths follow only these connections.

In this chapter, I study the feasibility of introducing mutual advantage into the design of PeerWise. I answer several technical questions: Is there mutual advantage in the Internet latency space? Will all nodes find mutually advantageous latency agreements, or are some nodes universally disadvantaged? I show that mutual advantage exists in the Internet: perhaps contrary to expectations, that there are not only "haves" and "have nots" of low-latency connectivity. The mutual advantageous requirement reduces the number of destinations available through detours to about half, yet even popular websites, using content distribution services such as Akamai are reachable through detours.

6.1 Motivation

Several distributed protocols and applications use *mutual advantage* as part of their design [18, 50, 91, 21]. BitTorrent [18] peers that download the same file trade blocks the other is missing. In backup systems [21], 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 [25].

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 [77, 87]. 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. Third, mutual advantage avoids the tragedy of the commons [30] 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 [85]. Finally, pairwise, mutually advantageous peerings provide a powerful, dynamic, fine-grained admission control mechanism. Connections are not made based on the membership to a group, but are negotiated individually by each participant with all other participants.

The requirements imposed by the mutual advantage principle on who can connect to whom are reminiscent of the bilateral connection game (BCG) [19], a special case of network formation game. In BCG, a link between two nodes is established only with the consent of both nodes. However, nodes construct links that minimize the cost of reaching other participating nodes, whereas my goal is to design an overlay network where nodes create peerings that offer detours to destinations that do not necessarily participate.

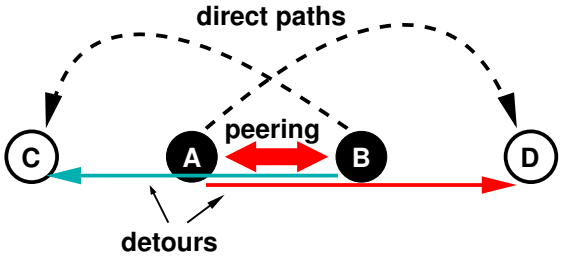


Figure 6.1: Finding mutually-advantageous detours: A discovers a detour to D through B; B also finds that it can reach C faster if it traverses A; A and B create a mutually advantageous peering which they both use to get more quickly to their destinations.

The key idea behind mutual advantage is that two nodes can cooperate to obtain faster end-to-end paths without either being compelled to offer more service than they receive. Nodes negotiate and establish pairwise connections to each other based strictly on mutual advantage. Figure 6.1 shows an example. 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. Therefore they can establish a pairwise peering.

6.2 Limitations of Mutual Advantage

I assess the potential performance loss when finding detours with mutual advantage. Because I restrict detour paths to mutually advantageous peerings, I would not expect PeerWise to find the shortest detours or find detours to all destinations. Using simulations on the PeerWise-King and PeerWise-PL-Dest data sets,

I show that nodes can find shorter paths to the majority of destinations for which a shorter detour exists, despite the requirement of mutual advantage. I find that mutually advantageous detours exist even for popular destinations hosted on many prefixes.

6.2.1 Methodology

I 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, I use Vivaldi [22]. I 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 [57]. I evaluate alternative relay selection methods in Section 7.2.2.

For each pair of nodes in the data sets, I find all one-hop detours. As before, I consider only good detours (detours that provide at least 10 ms and 10% latency reduction over the direct path). In the PeerWise-PL-Dest data set, I may find detours by server name: The detour path may end at a different IP address associated with the same name.

6.2.2 Is There Mutual Advantage in the Internet?

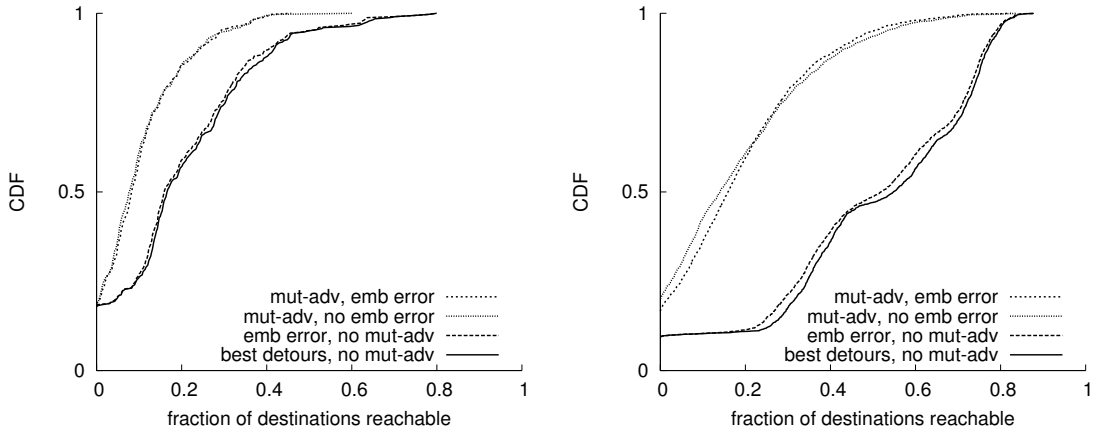
How much mutual advantage exists in the data sets? I define a *potential peering* to exist between two nodes that can provide a detour to each other, as between A

Data set	**Median**	**25th perc**	**75th perc**
PeerWise-PL-Dest	47%	22%	57%
PeerWise-King	75%	50%	87%

Table 6.1: Percentage of potential peerings for each node. Half of the nodes in PeerWise-PL-Dest have potential peerings with 47% of the other nodes (in PeerWise-King with 75%).

and B in Figure 6.1. The number of potential peerings for a node represents the number of neighbors with which the node can construct mutually advantageous peerings. In Table 6.1, I show statistics about potential peerings. 50% of the nodes in either data set have have potential peerings with 47% of the rest of the nodes in PeerWise-PL-Dest, and 75% in PeerWise-King. The table also shows that there is more mutual advantage in the PeerWise-King data set than in PeerWise-PL-Dest.

Next, I show that mutual advantage sacrifices few detours. I study the fraction of destinations that each node can reach more quickly via mutually advantageous peerings in Figure 6.2. 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



(a) (b)

Figure 6.2: Distribution of the fraction of destinations reachable through mutually advantageous peerings for PeerWise-PL-Dest data set (a), and PeerWise-King data set (b). 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 PeerWise-King, all nodes have many detours available, and mutual advantage is less costly. In both, embedding error finds nearly all detours.

among either set (constrained to mutual 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 6.2).

6.2.3 Detours to Nearby Destinations

The destinations in PeerWise-PL-Dest include both regionally and globally popular websites. I 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

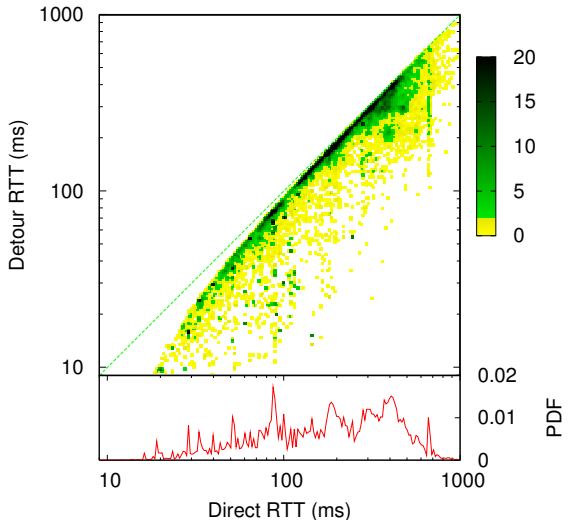


Figure 6.3: PeerWise-PL-Dest: When a detour exists, density plot of detour path RTT versus the direct path RTT (top), and PDF of direct path RTTs (bottom).

"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 6.3, I show that latency reduction is not limited to distant destinations. Because my rule to define a "good" detour requires at least 10 ms of reduction, few very short paths are featured. However, detours are found for direct paths too short to cross the Atlantic or Pacific oceans (< 100 ms).

6.2.4 Multiple-IP Websites

PeerWise has the potential to be effective even for websites using content distribution. For faster content delivery, around 20% of the popular websites in the PeerWise-PL-Dest data set are served from geographically distributed locations.

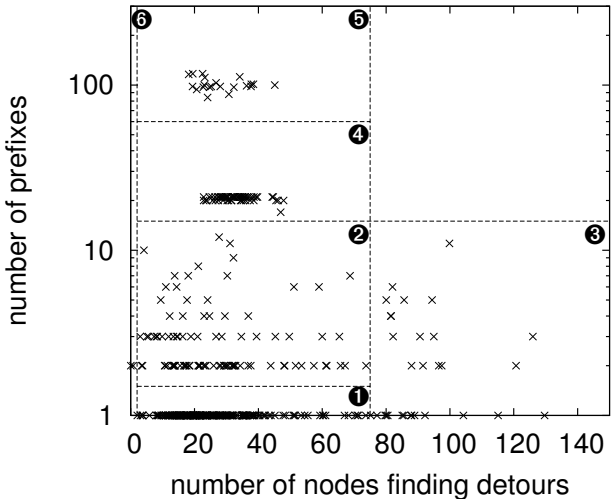


Figure 6.4: Detours to mirrored websites: The figure presents number of nodes that find detours versus number of prefixes for each website. Table 6.4 describes the six regions in the figure.

User requests are transparently directed to the geographically (or administratively) nearest IP address.

Using the PeerWise-PL-Dest data set, I 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 6.4 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. I divide the plot into six regions and describe each in the accompanying table.

Figure 6.4 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.

	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, mutually-advantageous detours found.
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`.

Table 6.2: Detours to mirrored websites: the number of nodes that find detours (c) versus number of prefixes for each website (p). The table describes the six regions in Figure 6.4.

6.2.5 Simulation Limitations

I discuss here the potential limitations of our simulation.

First, the 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. My evaluation might favor VoIP applications where the endpoints are well distributed and no endpoint is orders of magnitude more popular than the others.

Second, as described in Chapter 3, 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. I believe that this sharing prevents research networks from providing an unfair advantage in latency reduction. I have even observed detours between PlanetLab nodes—routing within the

academic network is not so latency-optimal as to prevent detours.

Finally, I 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.

6.3 Summary

I this chapter, I introduced mutual advantage as a design principle for routing overlay networks in general and PeerWise in particular. Mutual advantage restricts the virtual edges in PeerWise to those pairs of nodes that can provide detours to each other: each is an intermediate node in a detour for the other. Using real-world latencies, I show that introducing mutual advantage into detour discovery does not limit considerably the number of detours that are found. Even popular websites, using content distribution services are available through mutually-advantageous detours.

With triangle inequality violations exposing detours, network coordinates scalably predicting TIVs and mutual advantage offering fairness, all the pieces are in place for the design of PeerWise. In the next chapter, I present the components that make up PeerWise and the policies of each PeerWise participant for finding detours.

Chapter 7

Designing a Latency-Reducing Routing Overlay Network

In this chapter, I describe the design of PeerWise in two main parts: mechanisms and policies.

The key components of PeerWise are network coordinates, neighbor tracking and pairwise negotiation. Besides using simple network coordinates, as described in Chapter 5, I 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 and maintains connections promising mutual benefit.

The second part of the chapter focuses on the policy decisions that each PeerWise node makes. I evaluate neighbor selection and relay selection algorithms. I show that coordinates can be used to choose among detours. The environment is quite different from previous work on latency prediction using coordinates [22,69].

Instead on focusing on source-to-destination, PeerWise 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.

7.1 Mechanisms

In this section, I focus on the key mechanisms of PeerWise: detour detection using network coordinates for scalability, neighbor tracking for improving efficiency, and pairwise negotiations for fairness.

7.1.1 Network Coordinates

Every PeerWise node must compute its own network coordinate before searching for detours. I use Vivaldi [22] for network coordinates because it is distributed and scalable. 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 7.1(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

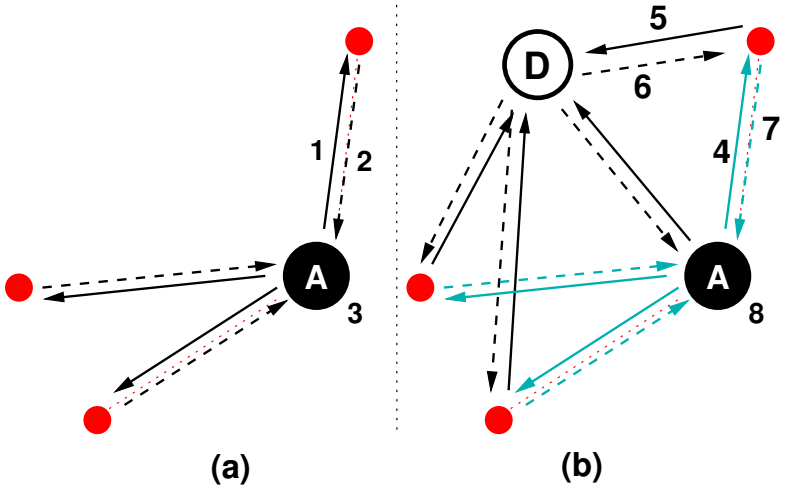


Figure 7.1: (a) *Computing network coordinates for a PeerWise node*: A measures RTT to its neighbors and asks for their coordinates (1); after it receives the replies (2) it computes the coordinate that minimizes the squared estimation error (3); (b) *Computing network coordinates for a non-PeerWise 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).

overlay, it will not provide its own network coordinate. I therefore extend Vivaldi to allow a node to compute a virtual network coordinate for any non-participating node. I will often refer to any node that does not participate to PeerWise as a (non-participating) destination, since it can only be a destination for detours.

To generate virtual network coordinates for non-participating node in Vivaldi, a participating node chooses to become temporarily responsible for that destination. The node runs Vivaldi on behalf of the destination with one minor adjustment. Since the destination is not participating in the system, it cannot manage its own neighbor set or actively gather the round trip times needed to compute the coor-

dinate. Instead, the participating node uses its own neighbor set as the neighbor set for the destination, and requests that those neighbors measure the latency to the destination. Figure 7.1(b) depicts this method. The extensions are similar to those described by Ledlie *et al.* [45].

Requiring all nodes to compute virtual coordinates for all non-participating destinations would limit the scalability of PeerWise. I include a gossip mechanism to disseminate the calculated coordinates throughout the system. At fixed intervals (10 s in my 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 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, I 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 destination is popular. If the desti-

nation is not popular, a node trying to discover a detour to it will need to compute its virtual coordinate. Since this requires that the node's neighbors measure the round trip time to the destination, 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 destination already (because the destination 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. I evaluate these predictions in Section 7.2.3.

7.1.2 Neighbor Tracking

The success of PeerWise 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. I 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 7.2, 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, I limit the number of neighbors of each node. Neighbors with higher potential to offer the best detours replace less-efficient neighbors. I consider and evaluate different methods for ranking potential neighbors in Section 7.2.1. Because PeerWise allows a node to exchange information about neighbors with neighbors, I expect each node to have ample choices.

7.1.3 Pairwise Negotiation and Maintenance

PeerWise nodes negotiate with their neighbors to request or advertise alternate routes. As discussed in Chapter 5, 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 7.3(c)). Nodes are not limited to this simple strategy. In Section 7.2.2, I evaluate different policies for choosing relays and deciding whether

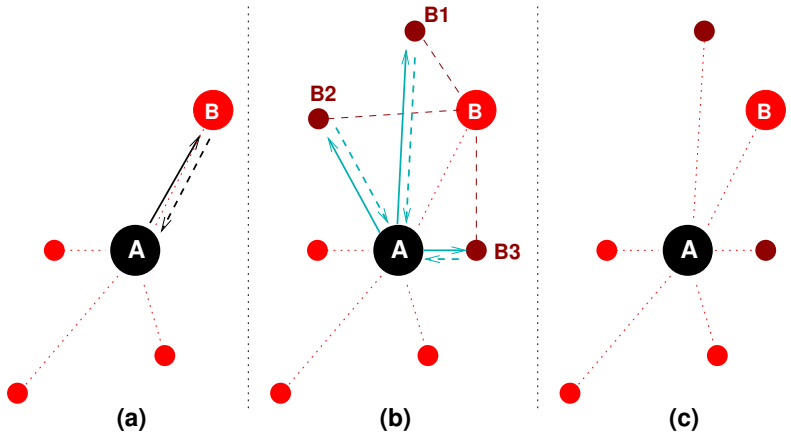


Figure 7.2: Neighbor Tracking. (a) A chooses the neighbor to which it has the highest embedding error and requests its neighbor set; (b) A measures RTTs to each of the nodes received from B; (c) A adds to its neighbor set those nodes to which it has a positive embedding error.

to request detours for a destination.

Actively requesting detours may be inefficient, especially if the connection to the destination is short-lived. 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 7.3(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

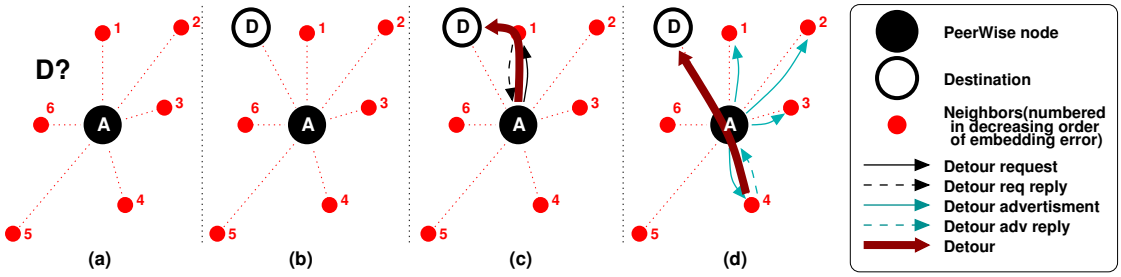


Figure 7.3: Detour Requests and Advertisements. (a) A wants to connect to destination D; (b) A discovers the network coordinate of D using Vivaldi or Virtual Vivaldi; (c) A requests a detour to D from the neighbor to which it has the highest embedding error; (d) A advertises its path to D to all neighbors that have positive embedding error to A.

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 Chapter 5. Requests for detours are accepted only when both the sender and the receiver find mutual advantage in forwarding each other's traffic.

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.

A service level agreement (SLA) is a formal contract between two peers that establishes all aspects of the service that each provides to the other. Nodes in routing overlays are inherently selfish and SLAs are an efficient method to curtail the effects of the selfishness. SLAs ensure that each node receives the expected level of quality-of-service even when the traffic demand in the network varies and the mutual advantage offered by a peering is time dependent.

I identify four performance metrics to be used as the basis for an SLA between two PeerWise users:

- **detours**: the number of detours that a node offers to the other
- **latency reduction**: the minimum latency reduction that a node offers to the other to a specific destination
- **bandwidth**: the average bandwidth at which one node forwards packets for the other to a specific destination
- **burstiness**: the maximum bandwidth at which one node forwards packets for the other to a specific destination

Each SLA is associated with a re-evaluation timeout which triggers a verification of the SLA against the traffic exchange since the previous timeout. I assume

that each PeerWise user has the means to monitor the traffic over all its peerings. Because IP is best effort and cannot provide measurable service, I do not require the bounds on the performance metrics defined in the SLA to be strictly enforced; it is enough if most of the packets sent over the peering satisfy the bounds specified by the SLA. Furthermore, I enforce frequent re-evaluations (on the order of minutes or hours rather than days or weeks). In doing so, I seek to protect the users against long periods when although the SLA is not violated, the peering is unusable.

I present an example of a simple SLA between two PeerWise users. Assume two nodes, A and B, discover mutual advantage and decide to form a peering. The SLA of the peering may state that A and B must offer each other latency reductions of at least 10ms for 95% of the packets that each send to their destinations C and D. The re-evaluation timeout is set to 60 seconds. Furthermore, the bandwidth that each peer uses to forward each other's packets must not exceed 20kB/s averaged over the 60 seconds, with its maximum value always below 100kB/s.

PeerWise nodes frequently renegotiate existing peerings to account for latency changes and to find the best detours available. They do not, however, monitor the byte-level usage of a peering. In my implementation, as described in Chapter 8, I establish SLAs based on the number of detours that each node is offering to the other. Applications built on top of PeerWise could use and combine all other SLA

metrics.

My SLA design draws from the agreements between autonomous systems in the Internet. Similarly to AS SLAs, PeerWise SLAs are intended to be maintained over long periods of time. In this way, long term reputation can motivate cooperation. However, AS SLAs are defined over much larger time scales. I require SLAs to be verified more often because traffic fluctuations may have bigger effects on a single link that connects two users than on a collection of links that interconnects two autonomous systems.

7.2 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 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, I 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, I collected latency measurements and coordinates for 262 PlanetLab nodes and the 448 popular web servers. I considered only the PlanetLab nodes responsive at the time of the measurement. To gather this PeerWise-Pyxida data set, I used Pyxida [80], an implementation of the Vivaldi coordinate system. To compute coordinates for the web servers, with colleagues, I extended Pyxida with the virtual coordinate algorithm. Every 30 seconds, for 18 hours on January 14, 2008, I took a snapshot containing RTT measurements and coordinates (virtual and non-virtual). I 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).

7.2.1 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

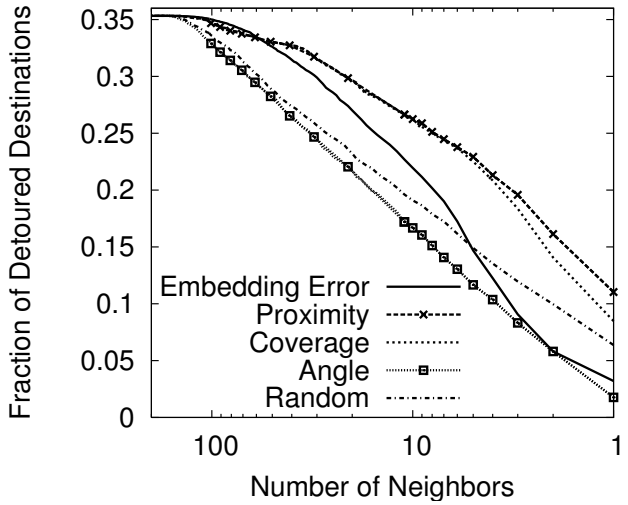


Figure 7.4: *Neighbor selection algorithms.* As the number of legitimate neighbors is restricted, coverage, proximity and embedding error (for 32 or more neighbors) algorithms preserve the most detours.

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, I 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, I do not expect access patterns to be nearly so predictable. Instead, I wish to determine, when a new neighbor arrives, whether it is likely to provide a shortcut to a useful region in coordinate space.

I consider a few traffic-independent neighbor selection policies, expecting that a combination of schemes would perform best. I 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 7.4, I compare these neighbor selection algorithms. I vary how many neighbors a node can have from 1 to 200. At each step, I 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. I choose proximity as the primary neighbor selection metric because it performs similarly to coverage

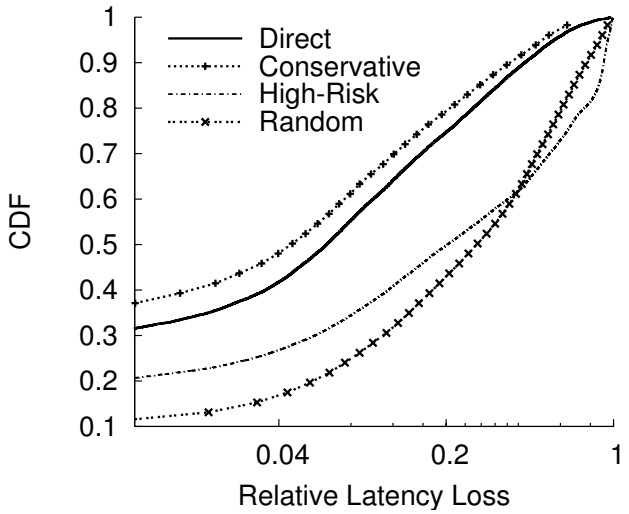


Figure 7.5: Prediction quality among relays. Best detours are found through relays selected using the direct and conservative algorithms.

and is easier to use.

7.2.2 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, I hope to reduce the number of relays that one needs to simultaneously contact to find a good detour when contacting a

destination for the first time.

I 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, I 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.

I select 32 neighbors for each node using the proximity-based algorithm and evaluate the four relay-selection algorithms. In Figure 7.5, I 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.

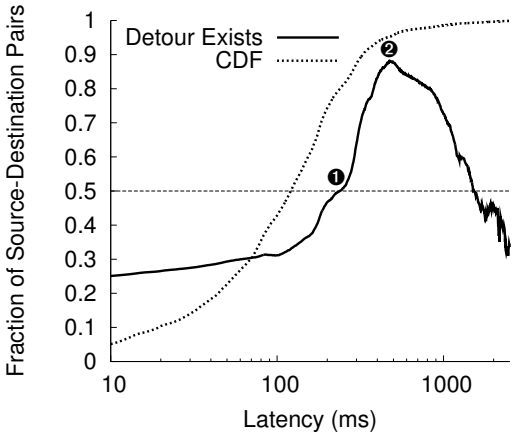
7.2.3 Deciding Whether to Relay

Deciding whether to use a detour depends on a prediction of whether it will improve application performance. This has two components: whether the traffic is sensitive to latency and whether a known neighbor is likely to provide a detour path. I 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, I 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.

7.2.3.1 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.



❶	
CDF	0.80
Frac	0.50
ms	236

❷	
CDF	0.95
Frac	0.88
ms	474

Figure 7.6: As the latency to a destination increases, so does the probability that there is a detour.

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 7.6, I 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. I expect that actual node behavior, in terms of when to seek out a detour, will be application dependent. For instance, a node may *always* try to find a detour for frequently contacted destinations.

7.2.3.2 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.

I 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. I 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

	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 7.1: 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.

not exist or finding a bad relay) are considered incorrect. I summarize all possible situations in Table 7.1. I 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. I 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.

7.3 Summary

In this chapter, I presented the design of PeerWise. First, I discussed its key components: network coordinates (including virtual coordinates for non-participating destinations) for detour detection, neighbor tracking for improved efficiency and scalability, and pairwise negotiation for fairness. Then, I presented the decision space of each PeerWise node for choosing the best neighbors (that provide good detours to many destinations) and relays (that provide good detours to specific destinations). In the next section, I implement PeerWise and present evaluation results from a wide-area deployment.

Chapter 8

Implementation and Evaluation

In this chapter, I describe results from the deployment of PeerWise under real network conditions on the PlanetLab testbed. First, I describe the implementation and then show that PeerWise can *quickly* find mutually advantageous detours that offer *significant* and *continuous* latency reduction. Then, I confirm that PeerWise detours can speed short web transfers in practice.

8.1 Finding Detours

In this section, I study the quality of detours found by PeerWise. I conduct experiments that support the following hypotheses:

- PeerWise finds mutually-advantageous detours
- The detours discovered by PeerWise offer significant and continuous latency reductions

8.1.1 Implementation

I 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 7.1. I use Pyxida [80] for computing coordinates, since it is the only network coordinate system implementation I am aware of that is tested extensively under realistic network conditions [43]. Pyxida is written in Java and uses the Vivaldi algorithm [22] to compute coordinates for nodes. Each Pyxida node maintains a variable number of neighbors, updated constantly, and probes them at regular intervals. I augmented Pyxida to compute virtual coordinates for hosts that do not participate in PeerWise, as described in Chapter 7.

I 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, I minimize the communication overhead. On the average, every node consumes less than 1KB/s (including Pyxida traffic).

8.1.2 Deployment

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

- **All-dest:** Each node tries to find detours to all 500 popular websites (described in Section 6.2) 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 [11, 105].

The discussion focuses on the **All-dest** experiment, but I summarize the results from **Rand-dest** and **Zipf-dest** in Tables 8.1 and 8.2. Recall that the destinations are already very popular servers, many of which use content distribution. Therefore, **All-dest** is not a best case scenario.

I 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. I limit the number of neighbors of each node to 32

for scalability and use the *proximity* policy for selecting neighbors. I 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, nodes 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. I 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.

I 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. I show that most nodes find mutually advantageous detours and that these detours lead to significant and stable latency reduction.

8.1.3 PeerWise finds detours

For each node, I count the destinations that can be reached using a mutually advantageous detour for the duration of the experiment. Figure 8.1 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. The results are consistent with those of the evaluation in Chapter 6.2 (see Figure 6.2). 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**.

8.1.4 PeerWise finds detours quickly

How quickly are the detours discovered? I compute the fraction of destinations to which a detour is discovered by PeerWise within the first 10 minutes, 1 hour and 5 hours. Figure 8.1 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.

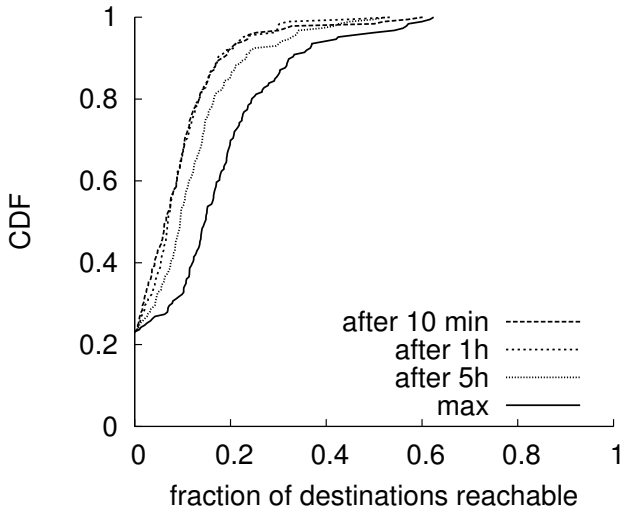


Figure 8.1: Fraction of the popular destinations reachable through mutually advantageous detours from PlanetLab.

8.1.5 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. I show that this is not the case. Recall that I have set a threshold: I consider only those detours that offer reduction of more than 10 ms and 10% of the direct-path latency. Here I focus on the latency reductions negotiated by PeerWise. In Section 8.2, I show how these reductions hold when user traffic traverses the detour path.

I 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. I show the median, 10th and 90th percentiles in Table 8.1. The median latency reduction is 29 ms or 26% of the latency of the direct path.

	Latency reduction (§ 8.1.5) relative (absolute)		
	median	**10 percentile**	**90 percentile**
All-dest	26% (29ms)	12% (12ms)	63% (131ms)
Rand-dest	25% (33ms)	12% (13ms)	60% (115ms)
Zipf-dest	24% (27ms)	12% (13ms)	59% (76ms)

Table 8.1: Characteristics of PeerWise detours: latency reduction.

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.

8.1.6 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.

I consider two metrics: *longevity* and *variability*. Longevity captures how PeerWise nodes maintain latency reduction once a detour is discovered. I define the

	Longevity (§ 8.1.6) % of (src,dest) pairs			**Variability (§ 8.1.6)** % of (src,dst) pairs		
	≥0.9	**0.5-0.9**	**<0.5**	**1**	**2-10**	**>10**
All-dest	54%	18%	28%	67%	2%	31%
Rand-dest	36%	19%	45%	51%	23%	26%
Zipf-dest	31%	31%	38%	48%	23%	29%

Table 8.2: Characteristics of PeerWise detours: longevity and variability.

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 8.2 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.

8.2 Using Detours

In this section, I 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, I download the front page of each of the 500 popular websites to which a mutually-advantageous detour exists. I 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, I install the *tinyproxy* HTTP proxy on every PlanetLab node that can be used as a relay. I run each transfer 100 times, alternating whether detour or direct comes first, and record the individual completion times.

I verify whether the detours promised by PeerWise are seen by the web transfers. For each (source, destination) pair with a detour in PeerWise, I 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 8.2 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.

I explain the dissonance between the PeerWise view and the application view next. PeerWise detours are determined by network-level pings. On the other hand,

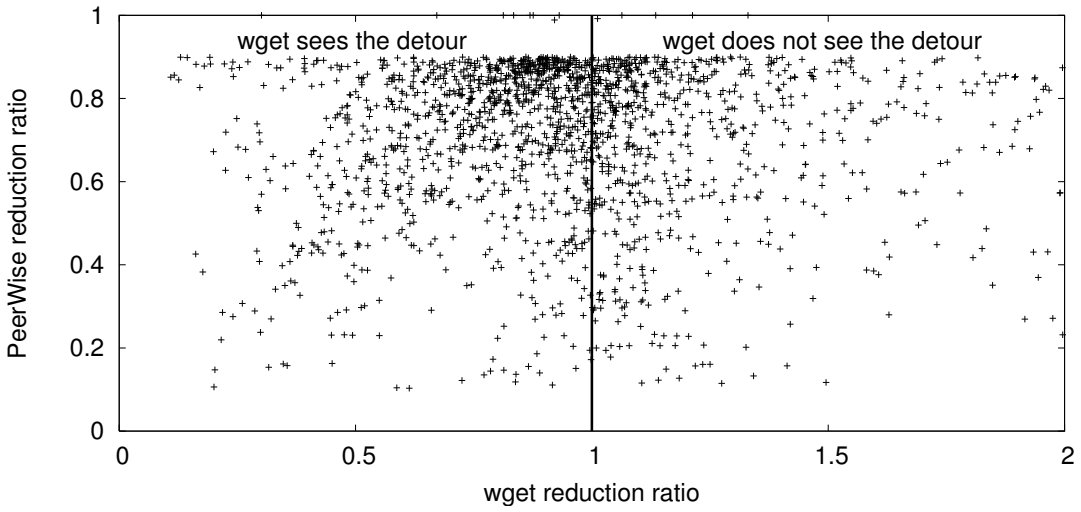


Figure 8.2: *Wget* latency reduction versus PeerWise latency reduction: 58% of all PeerWise detours achieve latency reduction in real life.

the *wget* end-to-end latency includes server and proxy wait 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, I instrument the experiment as follows. During the web transfers, I run *tcpdump* on every relay node and log all proxy traffic. Using the packet timestamps, I 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 8.3 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

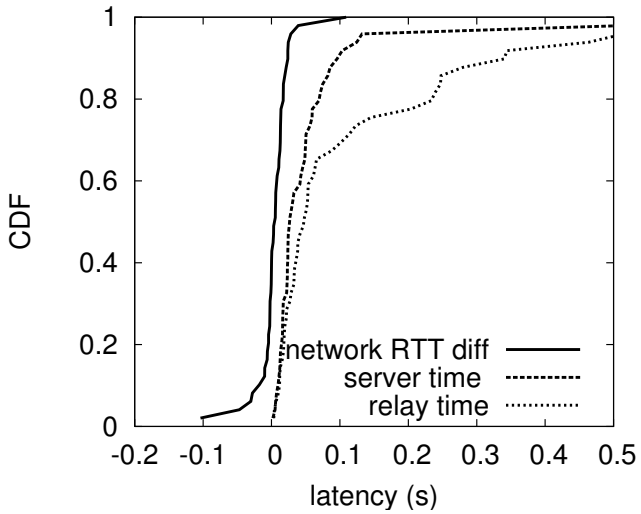


Figure 8.3: Distributions of average server wait times, relay times, and difference between *wget* and PeerWise RTTs for all detour transfers. Relay times inflate application latencies the most.

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. I believe that the slowness of PlanetLab is the main factor that contributes to the unusually high relay time for our transfers. To confirm, I 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 I 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 the web transfers would see the detours promised by PeerWise. I conclude that PeerWise has the potential to improve application performance.

8.3 Summary

In this chapter, I described the implementation of PeerWise and presented results from its evaluation on the PlanetLab testbed. I showed that PeerWise nodes find detours to popular destinations, that the 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. I then showed how PeerWise detours translate into real life and how user applications, such as the *wget* download application, can benefit.

Chapter 9

Conclusions and Future Work

9.1 Thesis and contributions

In this dissertation, I support the following thesis: *It is possible to make end-to-end communication between users in the Internet simultaneously faster, scalable and fair, by relaying solely on inherent properties of the Internet latency space.*

The leitmotif of my work is the triangle inequality violation (TIV). The failure of the Internet latency space to satisfy the triangle inequality provides an opportunity to improve end-to-end latencies by redirecting traffic between two nodes through a third node with which the first two form a TIV. That network coordinates do not work well with TIVs is an opportunity to scalably identify TIVs. Finally, because there are many pairs of nodes in the Internet latency space that are relays in each other's detours, I can provide fairness with a simple, locally-enforced mutual advantage principle: only those nodes that offer detours to each other are allowed to communicate.

I bring the following contributions. First, I perform a large-scale, extensive

study on triangle inequality violations in the Internet and show that they provide an excellent opportunity for detour routing. Using collected real-world latencies between nodes in the Internet, I demonstrate that TIVs are real and not illusions of the latency measurement process. Although the number of TIVs varies with time, aggregating multiple measurements using medians offers conservative estimates on the number of TIVs at any point in time. I analyze TIVs quantitatively and qualitatively and show that, albeit their low number, TIVs offer significant latency reduction to many.

Second, I propose and evaluate a **scalable technique to discover TIVs** with few measurements. My technique exploits the impossibility of perfectly embedding a triangle inequality violation in a metric space. I use network coordinates to place nodes that form TIVs in an Euclidean space and rely on the error in distance estimation using coordinates to indicate whether a pair of nodes is part of or needs a detour.

Third, I introduce **mutual advantage as a novel principle in the design of routing overlay networks**. Only pairs of nodes that are in a symbiotic relationship—each is an intermediate relay on a detour of the other—can peer. I show that mutual advantage does not limit significantly the number or the quality of detours that can be found, even to very popular destinations.

Finally, inspired by the properties of triangle inequality violations in the In-

ternet, I **design and build PeerWise, the first scalable and fair latency-reducing routing overlay network**. PeerWise is a step towards practical, flexible overlay routing, where participants are guaranteed that their costs to participate do not outweigh their benefits, and where they can negotiate paths based on their own performance.

9.2 Future work

I separate future work into two parts: extensions to PeerWise and new directions originated from the components of PeerWise.

9.2.1 Extensions to PeerWise

PeerWise is a latency-reducing overlay network. Providing detours that maximize a metric other than latency seems as an obvious next step. Next, I discuss how I intend to proceed in offering detours that route around failures, have lower-loss, or higher bandwidth.

Detours around failures

It seems straightforward to apply the latency-optimized overlay toward routing around failures. Although Gummadi *et al.* [29] show that a random selection can often skirt those failures that can be avoided, the peering links chosen by PeerWise

might be too correlated to maximize resilience. For example, they may all traverse a usually high-performance but failed link. PeerWise may have an advantage in that the mutual forwarding relationship cemented over time could be more quickly applied after failures.

Establishing pairwise agreements for failure recovery is different than for low latency. Peerings are intended to be used only when failures on the direct paths occur and thus a node does not know *a priori* if or how much it will use a peering. I identify several peering strategies to be used when fast failure recovery is desired:

- **random-k peering**

  Similarly to SOSR [29], each node chooses k other nodes as intermediaries to carry the traffic in case there are failures on the default paths to destinations. Gummadi *et al.* showed that a good value for k is 4: when four other intermediate nodes are used, their system is able to route around 92% of the failed paths. Oftentimes, the detour paths are not the most efficient: paths that offer more benefit in terms of end-to-end latency than the default recovery paths do exist. By selecting the k intermediate nodes among the ones with the highest embedding error, it may be possible to use network coordinates to find more efficient routes around failures. It remains to be seen how much selecting k nodes with high errors instead of k random nodes affects the recovery rate.

- **history-based peering**

  There are certain types of failures that tend to occur frequently in the same places: router misconfigurations, operating system failures, routing loops. If information about these failures is available, nodes can better devise their peering strategy to peer with nodes that are more likely to help find alternate paths. For example, if there is evidence that packets traveling through a certain AS tend to enter routing loops frequently, it is best to route them through a node that does not belong to the faulty AS.

- **locality peering**

  In locality peering, pairwise agreements are established between nodes that are located geographically close but that are part of different IP networks. The intuition behind this strategy stems from the work of Wang *et al.* [103]. Large Internet providers cover the same geographical regions and it is not uncommon that end-users that are geographically close to each other are clients of different ISPs. When physical-layer errors—such as cable cuts—occur, they only affect paths through one domain, leaving the paths through the other domains as a viable alternatives for recovery. Network coordinates could detect potential peers that are geographically close, but are part of different domains.

Lower-loss detours

Can PeerWise provide low-loss paths? First, for fairness to exist, pairwise peerings must be established, so there has to be a level of symmetry in the space formed by the loss rates between nodes in the Internet. Pairs of nodes should be able to help each other in finding low-loss paths to destinations. Second, for **scalability**, there must be a way of selecting the good nodes based only on partial information. I envision a solution that, similarly to latency detours, uses embedding errors in network coordinates. If an edge is underestimated, it means that its real latency is higher than the estimated distance. This may happen because the edge is the long side in a bad triangle or because other measurements done by one of the endpoints of the edge constrain the coordinates and move the other endpoint closer than necessary. Intuitively, if the relative error of the edge is also high, then we can assume that the edge is unusually long because it experiences congestion. A measurement study of the correlation between loss rates and triangle inequality violations is necessary and would shed more light on the feasibility of extending PeerWise to finding low-loss detours.

Higher-bandwidth detours

Application to bandwidth [31] may be more difficult, not because it would be a challenge to find pairs of nodes connected by high bandwidth, but because finding

third nodes that have high bandwidth connections to one but not both of the overlay participants seems difficult. A challenging measurement study would be required to show the potential of such overlay.

9.2.2 Social map of the Internet

Understanding the structure of the Internet is paramount. With precise network maps, the effectiveness of many applications and protocols would be easier to evaluate and the consequences of failures would be easier to predict. Most efforts in mapping the Internet focus exclusively on physical connectivity: identifying all nodes (routers) and connections between them. However, physical links are sometimes insufficient for predicting performance; how nodes interact with each other is a better predictor than whether they share a wire or not.

A different vision is to study the Internet as a *social* medium of communication. I will attempt a new structural decomposition of the network and identify relationships between nodes that transcend physical connectivity. PeerWise offers one such type of relationship: symbiosis, the mutual ability of two nodes to benefit from each other's position or resources. Such a relationship cannot be captured with simple topology maps and is of utmost importance for evaluating the feasibility and performance of a fair overlay network such as PeerWise. I believe that many more such relationships exist, each leading to a different map. For example,

dependency between nodes occurs when one node's performance relies exclusively on the other's, such as the source node in a detour relying on the relay for transit.

A social map of the network is exciting because it would change the way researchers view the Internet. It would create more complete models that, coupled with topology maps, would better predict the performance of new protocols. Social network maps could help overlay network construction, routing protocol design or measurement techniques by revealing new patterns of dependency or redundancy between nodes. They would also benefit the day-to-day operation and running of the network (*e.g.*, by identifying which links or nodes to overprovision, where to establish peering agreements). Studying the social structure of the Internet at the network level requires leveraging existing measurement techniques and studies, while devising new ones that capture social interplays between nodes.

9.2.3 Node location framework

In PeerWise, I use network coordinates to scalably find nodes under latency constraints, specifically those nodes that form TIVs. Many distributed applications in the Internet require finding nodes that satisfy various location constraints. Content distribution [24] is more efficient when performed through peers that minimize delay to receivers. In online games, distributed matchmaking algorithms attempt to select the game server that minimizes the average latency to all players [10, 1]. Un-

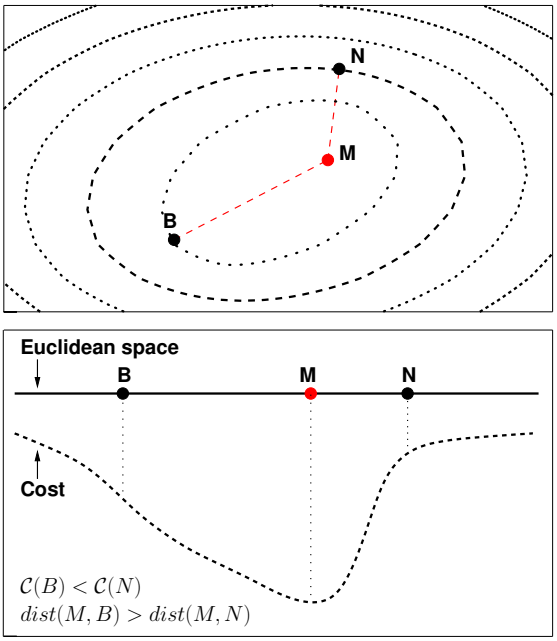


Figure 9.1: Nearest neighbor is not enough. M is the theoretical lowest cost point, N and B are nodes in the geometric space. Although N is closer to M than B, the cost of B is lower, making it more desirable than N.

like PeerWise, where I use the embedding error of network coordinates to identify nodes in TIVs, general node location queries cannot take advantage of the disagreement between network coordinates and TIVs. They find nodes under latency constraints by first computing the best coordinate that satisfies the constraints and then returning its nearest neighbor(s) [78, 44].

Finding the nearest neighbor using network coordinates is not always sufficient. The node closest in coordinate space to a theoretically optimal point may not be the best node. Consider the problem of computing the centroid of a set of nodes. Figure 9.1(top) shows the contour plot of a possible cost function. B and

N represent nodes in the Internet and M is the theoretically optimal centroid for a set of nodes (not shown on the plot). Although N is closer to M than B, B is in fact a better choice for the centroid, since it lies on a lower cost contour.

I consider generalizations of node location using network coordinates. Finding nodes under latency constraints can be naturally expressed as a cost optimization problem, where the cost of each node is determined by an arbitrary continuous function over the node coordinates. I sketch the design of Sherpa, an overlay network that finds the lowest cost node for distributed applications. Sherpa uses a modified compass routing algorithm [38] on the network coordinate space to discover the nearest neighbor to a theoretical optimal point and a gradient descent algorithm based on Voronoi region information to find the lowest cost node. Initial simulations on two real world latency data sets show that Sherpa finds nodes with cost that is significantly lower than the nearest neighbor and close to the optimal, even considering the embedding error of network coordinates.

My general formulation for node location captures a diversity of problems and extends the applicability of network coordinates beyond nearest neighbor applications. I intend to show how to effectively find low cost game servers for online game participants (where both the absolute latency and the relative disparities are important), relays that leverage split-TCP protocols [34] (to increase throughput between two nodes), and overlay paths that verifiably avoid nodes or regions in

the Internet.

Bibliography

[1] Sharad Agarwal and Jacob R. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *SIGCOMM*, 2009.

[2] Akamai. `http://www.akamai.com`.

[3] Alexa. `http://www.alexa.com/`.

[4] Luzi Anderegg and Stephan Eidenbenz. Ad hoc-VCG: A truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In *MobiCom*, 2003.

[5] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *SOSP*, 2001.

[6] Katerina Argyraki and David R. Cheriton. Loose source routing as a mechanism for traffic policies. In *Workshop on Future directions in network architecture*, 2004.

[7] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *SIGCOMM*, 2002.

[8] Suman Banerjee, Seungjoon Lee, Bobby Bhattacharjee, and Aravind Srinivasan. Resilient multicast using overlays. In *Sigmetrics*, 2003.

[9] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *NetGames*, 2004.

[10] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *SIGCOMM*, 2008.

[11] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, 1999.

[12] CAIDA AS relationships dataset. `http://www.caida.org/data/active/as-relationships/`.

[13] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast cooperative environments. In *SOSP*, 2003.

[14] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Anthony I.T. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal of Selected Areas in Communication*, 2002.

[15] David Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM*, 1988.

[16] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussles in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, 2005.

[17] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[18] Bram Cohen. Incentives build robustness in BitTorrent. In *P2PEcon*, 2003.

[19] Jacomo Corbo and David Parkes. The price of selfish behavior in bilateral network formation. In *PODC*, 2005.

[20] Manuel Costa, Miguel Castro, Antony Rowstron, and Peter Key. PIC: Practical Internet coordinates for distance estimation. In *ICDCS*, 2004.

[21] Landon Cox and Brian Noble. Samsara: Honor among thieves in peer-to-peer storage. In *SOSP*, 2003.

[22] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, 2004.

[23] Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. Computing the types of the relationships between autonomous systems. In *INFOCOM*, 2003.

[24] Michael J. Freedman, Eric Freudenthal, and David Maziéres. Democratizing content publication with Coral. In *NSDI*, 2004.

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

[26] Gnutella. `http://www.gnutella.com`.

[27] Albert Greenberg and Bruce Hajek. Deflection routing in hypercube networks. *IEE Transactions on Communications*, 40(6):1070–1081, 1992.

[28] Krishna Gummadi, Stefan Saroiu, and Steven Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Internet Measurement Workshop*, 2002.

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

[30] Garrett Hardin. The tragedy of the commons. *Science*, 162(3859):1243–1248, 1968.

[31] Sing Wang Ho, Thom Haddow, Jonathan Ledlie, Moez Draief, and Peter Pietzuch. Deconstructing Internet paths: An approach for AS-level detour route discovery. In *IPTPS*, 2009.

[32] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal of Selected Areas in Communication*, 20(8), 2002.

[33] Internet Growth Statistics. `http://www.internetworldstats.com/emarketing.htm`.

[34] Rahul Jan and Teunis J. Ott. Design and implementation of split TCP in the Linux kernel. In *Globecom*, 2006.

[35] H. Tahilramani Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi. BANANAS: An evolutionary framework for explicit and multi-path routing in the Internet. In *Workshop on Future directions in network architecture (FDNA)*, 2003.

[36] Kazaa. `http://www.kazaa.com`.

[37] Dejan Kostic; Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *SOSP*, 2003.

[38] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *CCCG*, 1999.

[39] James F. Kurose and Keith W. Ross. *Computer Networking: A Top Down Approach*. Addison-Wesley, 2009.

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

[41] Craig Labovitz, Abha Ahuja, Abhijit Abose, and Farnam Jahanian. Delayed Internet routing convergence. In *SIGCOMM*, 2000.

[42] Karthik Lakshminarayanan and Venkata N. Padmanabhan. Some findings on the network performance of broadband hosts. In *IMC*, 2003.

[43] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network coordinates in the wild. In *NSDI*, 2007.

[44] Jonathan Ledlie, Peter Pietzuch, Michael Mitzenmacher, and Margo Seltzer. Wired geometric routing. In *IPTPS*, 2007.

[45] Jonathan Ledlie, Margo Seltzer, and Peter Pietzuch. Proxy network coordinates. Technical report, Imperial College London, 2008.

[46] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of euclidean embedding of Internet hosts. In *Sigmetrics*, 2006.

[47] Dave Levin. Punishment in selfish wireless networks: A game theoretic analysis. In *NetEcon*, 2006.

[48] Dave Levin, Randy Baden, Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. Motivating participation in internet routing overlays. In *NetEcon*, 2008.

[49] Dave Levin, Adam Bender, Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. Boycotting and extorting nodes in an internetwork. In *NetEcon+IBC*, 2007.

[50] Dave Levin, Rob Sherwood, and Bobby Bhattacharjee. Fair file swarming with FOX. In *IPTPS*, 2006.

[51] Hyuk Lim, Jennifer C. Hou, and Chong-Ho Choi. Constructing internet coordinate system based on delay measurement. In *IMC*, 2003.

[52] Eng Keong Lua, Timothy Griffin, Marcelo Pias, Han Zheng, and Jon Crowcroft. On the accuracy of the embeddings for Internet coordinate systems. In *IMC*, 2005.

[53] Cristian Lumezanu, Randy Baden, Dave Levin, Bobby Bhattacharjee, and Neil Spring. Symbiotic relationships in Internet routing overlays. In *NSDI*, 2009.

[54] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality and routing policy violations in the internet. In *PAM*, 2009.

[55] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality variations in the Internet. In *IMC*, 2009.

[56] Cristian Lumezanu, Sumeer Bhola, and Mark Astley. Online optimization for latency assignment in distributed real-time systems. In *ICDCS*, 2008.

[57] Cristian Lumezanu, Dave Levin, and Neil Spring. PeerWise discovery and negotiation of faster paths. In *HotNets*, 2007.

[58] Cristian Lumezanu and Neil Spring. Measurement manipulation and space selection in network coordinates. In *ICDCS*, 2008.

[59] Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. Decentralized message ordering for publish/subscribe systems. In *Middleware*, 2006.

[60] Harsha V. Madhyastha, Thomas Anderson, Arvind Krishnamurthy, Neil Spring, and Arun Venkataramani. A structural approach to latency prediction. In *IMC*, 2006.

[61] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *OSDI*, 2006.

[62] Ratul Mahajan, David Wetherall, and Thomas Anderson. Negotiation-based routing between neighboring ISPs. In *NSDI*, 2005.

[63] Ratul Mahajan, David Wetherall, and Thomas Anderson. Mutually controlled routing with independent ISPs. In *NSDI*, 2007.

[64] G. Malkin. RIP Version 2. RFC 2453, IETF, 1998.

[65] Yun Mao and Lawrence K. Saul. Modeling distances in large-scale networks by matrix factorization. In *IMC*, 2004.

[66] J. Moy. OSPF Version 2. RFC 2328, IETF, 1998.

[67] Aki Nakao and Larry Peterson. Scalable routing overlay networks. In *ACM SIGOPS Operating Systems Review*, 2006.

[68] Akihiro Nakao, Larry Peterson, and Andy Bavier. A routing underlay for overlay networks. In *SIGCOMM*, 2003.

[69] T. S. Eugene Ng and Hui Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, 2002.

[70] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV*, 2002.

[71] Vern Paxson. End-to-end routing behavior in the Internet. In *SIGCOMM*, 1996.

[72] Vern Paxson. End-to-end Internet packet dynamics. In *SIGCOMM*, 1997.

[73] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, IETF, 2003.

[74] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM CCR*, 33(1):59–64, 2003.

[75] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2007.

[76] Marcelo Pias, Jon Crowcroft, Steve Wilbur, Tim Harris, and Saleem Bhatti. Lighthouses for scalable distributed location. In *IPTPS*, 2003.

[77] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In *NSDI*, 2007.

[78] Peter Pietzuch, Jonathan Ledlie, Michael Mitzenmacher, and Margo Seltzer. Network-aware overlays with network coordinates. In *IWDDS*, 2006.

[79] Internet Protocol. Internet protocol. RFC 791, IETF, 1981.

[80] Pyxida. `http://pyxida.sourceforge.net/`.

[81] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in Internet-like environments. In *SIGCOMM*, 2003.

[82] Barath Raghavan and Alex C. Snoeren. A system for authenticated policy-compliant routing. In *SIGCOMM*, 2004.

[83] Yakov Rekhter and Kirk Lougheed. A Border Gateway Protocol 4 (BGP-4). RFC 1771, IETF, 1994.

[84] RouteViews. `http://www.routeviews.org`.

[85] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of Internet content delivery systems. In *OSDI*, 2002.

[86] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: A case for informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, 1999.

[87] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM CCR*, 29(5):71–78, 1999.

[88] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of Internet path selection. In *SIGCOMM*, 1999.

[89] Yuval Shavitt and Tomer Tankel. Big-bang simulation for embedding network distances in euclidean space. In *INFOCOM*, 2003.

[90] Yuval Shavitt and Tomer Tankel. On the curvature of the Internet and its usage for overlay construction and distance estimation. In *INFOCOM*, 2004.

[91] Michael Sirivianos, Jong Han Park, Xiaowei Yang, and Stanislaw Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *USENIX*, 2007.

[92] Skype. `http://www.skype.com`.

[93] Neil Spring, Ratul Mahajan, and Tom Anderson. Quantifying the causes of path inflation. In *SIGCOMM*, 2002.

[94] Jeremy Stribling. Planetlab all pairs ping. `http://www.pdos.lcs.mit.edu/~strib/pl_app/`.

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

[96] Lakshminarayanan Subramanian, Matthew Caesar, Cheng Tien Ee, Mark Handley, Morley Mao, Scott Shenker, and Ion Stoica. HLP: A next generation inter-domain routing protocol. In *SIGCOMM*, 2005.

[97] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *NSDI*, 2004.

[98] Paul Syverson, David Goldschlag, and Michael Reed. Anonymous connections and onion routing. In *Security and Privacy*, 1997.

[99] Liying Tang and Mark Crovella. Virtual landmarks for the Internet. In *IMC*, 2003.

[100] Sudhir Tauro, Christopher Palmer, Georgos Siganos, and Michalis Faloutsos. A simple conceptual model for the internet topology. In *Global Internet*, 2001.

[101] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *USENIX Security*, 2000.

[102] Guohui Wang, Bo Zhang, and T. S. Eugene Ng. Towards network triangle inequality violation aware distributed systems. In *IMC*, 2007.

[103] Hao Wang, Yang Richard Yang, Paul H. Liu, Jia Wang, Alex Gerber, and Albert Greenberg. Reliability as an interdomain service. In *ACM Sigcomm*, 2007.

[104] Li wei Lehman and Steven Lerman. PCoord: Network position estimation using peer-to-peer measurements. In *International Symposium on Network Computing and Applications*, 2004.

[105] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *SOSP*, 1999.

[106] Bernard Wong, Aleksandrs Slivkins, and Emin Gn Sirer. Meridian: A lightweight network location service without virtual coordinates. In *SIGCOMM*, 2005.

[107] Wen Xu and Jennifer Rexford. Miro: Multi-path interdomain routing. In *SIGCOMM*, 2006.

[108] Xiaowei Yang. NIRA: A new Internet routing architecture. In *Workshop on Future Directions in Network Architecture (FDNA)*, 2003.

[109] Bo Zhang, T.S. Eugene Ng, Animesh Nandi, Rudolf RIedi, Peter Druschel, and Guohui Wang. Measurement-based analysis, modeling, and synthesis of the Internet delay space. In *IMC*, 2006.

[110] Rongmei Zhang, Y. Charlie Hu, Ziaojun Lin, and Sonia Fahmy. A hierarchical approach to Internet distance prediction. In *ICDCS*, 2006.

[111] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy G. Griffin. Internet routing policies and round-trip times. In *PAM*, 2005.

[112] Sheng Zhong, Jiang Chen, and Yang Richard Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *INFOCOM*, 2003.

[113] Dapeng Zhu, Mark Gritter, and David R. Cheriton. Feedback based routing. In *HotNets*, 2002.