

Imperial College London
Department of Department of Electrical and Electronic Engineering

Improving End-to-End Internet Performance by Detouring

Sing Wang Ho

June 2012

Supervised by Moez Draief

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Department of Electrical and Electronic Engineering of
Imperial College London
and the Diploma of Imperial College London

Abstract

The Internet provides a best-effort service, which gives a robust fault-tolerant network. However, the performance of the paths found in regular Internet routing is suboptimal. As a result, applications rarely achieve all the benefits that the Internet can provide. The problem is made more difficult because the Internet is formed of competing ISPs which have little incentives to reveal information about the performance of Internet paths. As a result, the Internet is sometimes referred as a ‘black-box’. *Detouring* uses routing overlay networks to find alternative paths (or detour paths) that can improve reliability, latency and bandwidth. Previous work has shown detouring can improve the Internet. However, one important issue remains—how can these detour paths be found without conducting large-scale measurements?

In this thesis, we describe practical methods for discovering detour paths to improve specific performance metrics that are scalable to the Internet. Particularly we concentrate our efforts on two metrics, latency and bandwidth, which are arguably the two most important performance metrics for end-user’s applications. Taking advantage of the Internet topology, we show how nodes can learn about segments of Internet paths that can be exploited by detouring leading to reduced path latencies. Next, we investigate bandwidth detouring revealing constructive detour properties and effective mechanisms to detour paths in overlay networks. This leads to Ukairo, our bandwidth detouring platform that is scalable to the Internet and `tcpChiryo`, which predicts bandwidth in an overlay network through measuring a small portion of the network.

To Mom and Dad,

Acknowledgement

First, I would like to thank Moez Draief, my supervisor, for his guidance, support, and care throughout my PhD degree. His comments are helpful and I could not have finished this thesis without him. I am grateful for Peter Pietzuch and Thom both of which have given me many constructive advice (and beer) throughout my degree.

Thanks to the people in the Intelligent System Networks group: Julia, Niki, Arta, Harold, Miguel, Kyuhwa and Yan. Your encouragement and support really counts especially during stressful times. In particular, thank you to Harold and Miguel for proof-reading my thesis.

I also want to show my gratitude to my examiners, Kin Leung and Rio Miguel, for providing feedbacks to my dissertation.

Last but not least, my parents have always been there for me with their loving care and support.

Contents

1	Introduction	1
1.1	Main Contributions	3
1.2	Roadmap	5
2	Background and Related Work	6
2.1	Internet Routing	6
2.1.1	Inter-domain Routing	8
2.1.2	Intra-domain Routing	9
2.2	Measuring the Internet	9
2.2.1	Latency	10
2.2.2	Network Topology	11
2.2.3	Bandwidth	13
2.2.4	Testbeds	17
2.3	Detouring the Internet	19
2.3.1	Benefits from Detour Routing	19
2.3.2	Detouring Overlay Networks	21
2.3.3	Alternatives to Detouring	24
3	Latency Detour Discovery, An AS-level Approach	26
3.1	AS Path Deconstruction	27
3.2	Path Clustering	29
3.3	Decentralised Path Clustering	31
3.4	Evaluation	32
3.4.1	Centralised Detour Discovery	33
3.4.2	Decentralised Detour Discovery	37
3.4.3	Application of Path Clustering to other metrics	38
3.5	Analysis of detoured AS paths on PlanetLab	41
3.6	Summary	43

4	Reducing Bandwidth Measurement Overhead in Overlay Networks	44
4.1	Direct TCP Bandwidth Measurement	45
4.2	Relationship between TCP Bandwidth and Available Bandwidth	46
4.3	tcpChiryo: TCP bandwidth inference using loss tomography	52
4.3.1	Methodology	54
4.3.2	Evaluation	59
4.3.3	Analysis of Inferred Bottleneck Links	64
4.4	Summary	67
5	Feasibility of Bandwidth Detouring	68
5.1	Potential for Bandwidth detouring	69
5.1.1	Effects of Bandwidth Detouring on Latency	71
5.2	Bandwidth Detour Properties	73
5.2.1	Symmetry of Detours	73
5.2.2	Latency of Bandwidth Detours	73
5.2.3	Persistence in time	75
5.3	Bandwidth Detouring Mechanism	77
5.3.1	IP Detouring	77
5.3.2	TCP Detouring	80
5.4	Experimental Evaluation	81
5.4.1	IP and TCP detouring performance	81
5.4.2	TCP detouring performance on PlanetLab	82
5.5	Using Latency Detours for Bandwidth Detouring	83
5.6	Discussion	84
5.7	Summary	86
6	Ukairo - Practical Detour Discovery	87
6.1	Scalable Detour Discovery	89
6.1.1	Ranking Detours	90
6.1.2	Iterative Detour Discovery	95
6.1.3	Pairing Clients with Overlay Nodes	97
6.2	Three Models for a Detouring Architecture	100
6.3	Ukairo Architecture	102
6.3.1	Ukairo's Information Node	103
6.3.2	Path Selection Mechanism	104
6.3.3	Detour implementation	105

6.4	Evaluation	106
6.4.1	Internet detouring performance	107
6.4.2	Iterative detour node selection	114
6.4.3	Load balancing detour nodes	115
6.5	Summary	118
7	Conclusions and Future Work	119
7.1	Future Work	120
7.1.1	Towards an ISP-Friendly Detouring Overlay Network	120
7.1.2	Towards Passive Measurements	121

List of Figures

2.1	Example of the two-level hierarchy of the Internet	7
2.2	Example of IP and Router-level Topology	12
2.3	An example of detouring which improves latency	20
2.4	An Illustration of an Overlay Network	21
3.1	Example of AS-level detouring	28
3.2	Example of shared AS-links	29
3.3	Precision and recall of TIV alert thresholds	34
3.4	Latency improvement from Path Clustering	36
3.5	Latency improvement from Decentralised Path Clustering	37
3.6	Loss improvement from Path Clustering	39
3.7	Bandwidth Improvement from Path Clustering	40
3.8	AS-paths change due to detouring by Path Clustering	41
4.1	Measuring TCP bandwidth with different duration	46
4.2	Comparison of TCP bandwidth with ABets	49
4.3	Convergence rate using different path selection strategies	60
4.4	Factor errors for different inference algorithms	62
4.5	Bottleneck link loss rates for <code>tcpChiryo-max</code> and <code>-bern</code>	65
4.6	Distribution of bottleneck links to AS hop	66
5.1	Multi-hop bandwidth detouring	70
5.2	Potential for bandwidth, latency and loss improvement	71
5.3	Effects of bandwidth detouring on latency	71
5.4	Latency-conscious bandwidth detouring	72
5.5	Distribution latency of bandwidth detour paths	74
5.6	Longevity of bandwidth detours	76
5.7	Dynamics of IP detouring	79
5.8	TCP detouring versus IP detouring	82

5.9	TCP detouring on PlanetLab	83
5.10	Latency-based bandwidth detours	84
6.1	Improvement obtained by restricting the detour set	92
6.2	Bandwidth improvement of different ranking strategies	94
6.3	Improvement of client-overlay pairing using different metrics	98
6.4	Example of a peer-to-peer bandwidth detouring network	100
6.5	Example of an ingress-egress bandwidth detouring network	100
6.6	Example of an information node based detouring network	101
6.7	Overview of the Ukairo system architecture	102
6.8	Bandwidth improvement of Ukairo strategies on PlanetLab	108
6.9	Bandwidth improvement of Ukairo on public Internet destinations	110
6.10	Bandwidth improvement of Ukairo on residential broadband clients	112
6.11	Relative improvement of Ukairo on residential broadband clients	114
6.12	Bandwidth improvement using iterative detour selection	115
6.13	Concurrent downloads on Ukairo	116
6.14	Adding nodes to the same detour sites	117

List of Tables

2.1	Taxonomy of Available Bandwidth Tools	16
3.1	Classification accuracy of Path Clustering	35
3.2	AS-paths change due to detouring by Path Clustering	41
4.1	List of datasets for comparing available bandwidth tools	50
4.2	Training times of different inference algorithms	63
4.3	Types of bottleneck links found from different inference algorithms . . .	66

1 Introduction

The Internet is designed to provide a best-effort service. The philosophy behind the design of the Internet is to provide a robust service that can tolerate faults in parts of the network. This leads to our modern Internet routing which can provide basic point-to-point connectivity effectively but disregards the actual performances of these connections.

The Internet is made up of many self-administrated networks known as *Internet Service Providers* (ISPs). Internet routing is provided by these ISPs cooperating with each other, but at the same time compete and use contractual agreements to monetise Internet traffic sent through neighbouring networks. These agreements, in terms, use policies to govern how traffic flows. As a result, Internet routing creates longer or more congested routes than necessary.

There are many approaches to improving the performance of the Internet. Some have taken the inefficiency in Internet routing as a fact and find other workarounds. For example, ISPs upgrading their existing infrastructure, content providers creating replica to other locations using *content distribution networks* (CDNs), or clients downloading from multiple sources such as BitTorrent [4]. These solutions are either expensive or only work for popular contents leaving the niche but equally important long-tail content unoptimised. Others have considered redesigning Internet protocols to allow more flexibility for clients to choose their paths [133, 44, 103, 134, 139]. These solutions require major modifications to our current Internet architecture.

An alternative to the above solutions is to use *routing overlay networks*. An overlay

network is a collection of nodes deployed over another network to provide a particular service. Overlay networks still rely on the underlying routing protocol to carry traffic between the nodes. One particular class of overlay networks is routing overlay networks where the nodes are used to redirect Internet traffic. By deploying a routing overlay network, we can redirect traffic in a way that improves the performance of paths over the Internet. This concept of redirecting Internet traffic to improve path performance is known as *detour routing* and the nodes in the overlay which redirect traffic are known as *detours*.

Routing overlay networks have several key advantages. They work on top of the Internet meaning the underlying Internet routing protocols and Internet structure do not need to be changed. This means that overlay networks can be deployed readily and cheaply because they do not need specialised hardware for routing. Overlay networks can be scaled easily from a small deployment to a full fledged Internet service.

In this dissertation, we discuss how routing overlay networks can be used to improve various performance metrics of the Internet. In particular, we concentrate on latency and bandwidth which are arguably the two most important measures of performance of an Internet path. Latency is important for real-time applications such as stock quotes, live sports scores, and online gaming which require intimate player interactions. Bandwidth, on the other hand, is important for applications which transfer large amounts of data, such as file transfers, cloud storage systems, and video streaming applications.

A naive solution for discovering useful detours is to have clients measure the direct end-to-end path and all detour paths each time they connect to a new destination. As the number of clients grow, two problems arise. First, clients will flood the network with redundant traffic by sending many measurement probes. Secondly, detours in the overlay network will get overwhelmed reducing their effectiveness as a detour.

A particular concern is that measuring bandwidth generates a significant amount of data

over the Internet. Unlike latency which can be probed by sending as little as one packet per path, bandwidth consumes in the order of tens of thousands of packets in wide-area networks such as PlanetLab [52]. Therefore, it is important to reduce the average cost of bandwidth measurements.

Additionally, Internet measurements have large variances due to many factors, for example: i) congestion which is caused by queuing in routers, and ii) router maintenance such as routing table updates [98, 136]. Measurements can also expire due to permanent routing changes.

Our goal is to be able to deploy a live practical routing overlay network on the Internet. We avoid modifications to clients and arbitrary destinations to allow our overlay network to integrate with existing applications and to encourage clients usage.

To attract clients to use our detouring overlay, we design our system such that we never perform worse than the best-effort service that the Internet currently provides. This insures that we do not degrade the performance of a clients' path such that clients experience worse performance to one of their favourite destinations.

Another practical challenge is to design a user-friendly client. Ideally an effective client-side software should be run as a daemon in the background and only detour paths when necessary. Clients should have a better end-user experience of the Internet using our routing overlay network. In particular, our detouring mechanism should work seamlessly to clients such that they should not experience significant delays when discovering detours.

1.1 Main Contributions

The key contributions of this thesis are as follows:

- We introduce a novel technique for discovering latency detours in an overlay network.

Our technique finds latency detours by looking at the relationships between ISPs. Our

intuition is that certain sets of ISP policies would lead to inefficiencies in Internet routing. We design a hierarchical clustering algorithm that identifies these sets of policies and can also provide detour suggestions based on the sets identified. For latency, we are able to identify paths that have a detour with a 94.3% accuracy and suggest beneficial detours to these paths.

[55] *S. W. Ho, T. Haddow, J. Ledlie, M. Draief, and P. Pietzuch. Deconstructing Internet Paths: An Overlay for AS-Level Detour Route Discovery. In IPTPS, 2009.*

- Next, we investigate how we can reduce the amount of measurements when measuring the bandwidth of paths in a network. We present `tcpChiryo` which makes use of the fact that a path’s bandwidth can be decomposed into latency and loss rate. With a small amount of measurements and by intelligently selecting the paths to measure, we can infer the link’s loss rate from the path’s loss rate. From the inferred link’s loss rate, we can predict the bandwidth of a given path for which we have not measured.

[54] *S. W. Ho and M. Draief. tcpChiryo: TCP bandwidth Inference using Loss Tomography. pre-print, 2012.*

- We provide a detailed investigation into the potential of detour routing for bandwidth. We show that there are more opportunities for benefits in detouring for bandwidth than latency. We provide a longitudinal study of bandwidth detours and find that most bandwidth detours can last for long periods of time. We present and evaluate, both analytically and experimentally, two different mechanisms for deploying bandwidth detouring on the Internet.

[52] *T. Haddow, S. W. Ho, J. Ledlie, C. Lumezanu, M. Draief, and P. Pietzuch. On the Feasibility of Bandwidth Detouring. In PAM, 2011.*

- Finally, we implement and evaluate Ukairo a practical routing overlay network for bandwidth detouring. The routing overlay incorporates several methods for finding detours without incurring high measurement overheads. We rank the detours in terms of

their overall quality as a bandwidth detour and suggest detours for clients based on their ranking. We show that the construction of a detour rank list is scalable leading to a reduction in the number of measurements. We also present an iterative method which passively tracks the quality of each detour when the client uses them and improves upon its detour choices based on previous use.

We evaluate Ukairo in a broad variety of scenarios. We show that Ukairo can benefit when downloading files from PlanetLab, public mirrors and cloud storage systems. We evaluate scalability by running multiple clients concurrently. We also examine Ukairo on different residential broadband clients from around the world.

[53] *T. Haddow, S. W. Ho, C. Lumezanu, M. Draief, and P. Pietzuch. Ukairo: Internet-Scale Bandwidth Detouring. Technical Report DTR11-2, Imperial College London, 2011.*

1.2 Roadmap

The rest of the thesis is organised as follows.

Chapter 2 provides the necessary material to understand the rest of the thesis and compares our work with several related work.

Chapter 3 presents our latency detour discovery technique.

Chapter 4 investigates different methods of reducing bandwidth measurement overhead and presents our bandwidth inference technique, `tcpChiryo`.

Chapter 5 explores the potentials for detour routing in bandwidth.

Chapter 6 describes and evaluates Ukairo which is a routing overlay network for detouring bandwidth.

Chapter 7 summarises this dissertation and discusses further work.

2 Background and Related Work

The purpose of this chapter is to provide the relevant material that is required to understand routing overlay networks. We first present how the current Internet routes traffic and discuss how this leads to the existence of better performing Internet paths (in §2.1). Next, we investigate the different methods of obtaining the Internet topology, latency and bandwidth measurements in order to build a routing overlay network that can scale with respect to measurement overhead (in §2.2). Finally, we analyse what others have understood about detouring and how has other routing overlay networks been constructed (§2.3).

2.1 Internet Routing

How Internet routes traffic is complex and involves many protocols interacting with each other. Here, we only discuss the mechanisms in Internet routing that leads to the existence of better performing routes in the Internet.

The Internet operates in a competitive environment. Traffic that is sent between two hosts often traverses several competing *Internet Service Providers* (ISPs). These ISPs cooperate to provide data delivery service from one end to another, but at the same time compete by charging each other for relaying traffic.

There are thousands of ISPs in the Internet, each operating their individual network known as an *Autonomous System* (AS). Typically each ISP owns a single AS, but it is pos-

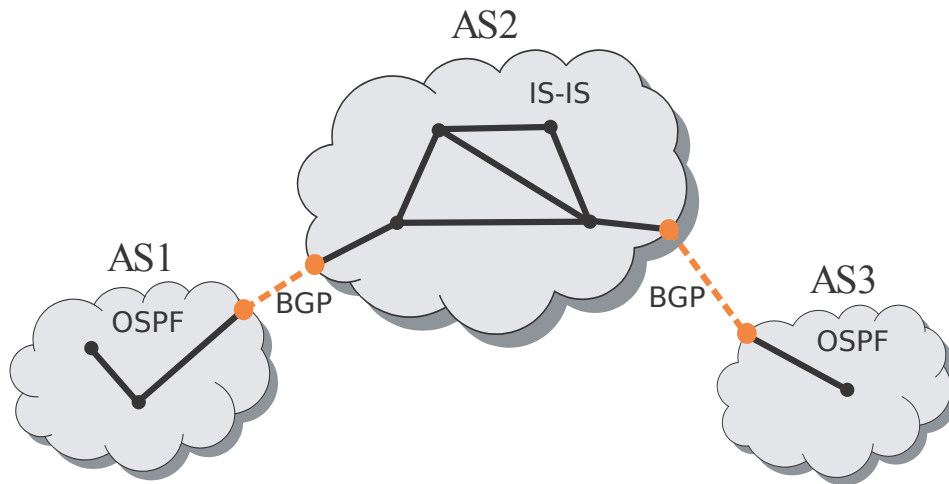


Figure 2.1: Example of the two-level hierarchy of the Internet

sible for an ISP to have multiple ASes. The relationships between ASes can be classified into two categories: *peering* and *provider-customer* relationships [42, 118]. While a peering relationship allows two ASes to mutually exchange traffic freely, a provider-customer relationship usually incurs a monetary cost (known as *transit cost*) when the customer AS sends traffic to its provider.

ASes govern their own network individually; each decides how routing should be operated within its network. They route traffic using devices called *routers* which use *routing tables* to direct traffic based on the destination that the data is destined to. A routing table contains a list of destinations that the router knows how to reach and is configured based on the routing protocol the AS uses.

We can see that the Internet is actually a two-level hierarchical structure. The top level (or the AS-level) routes traffic between ASes (inter-AS routing), and the bottom level (or the IP-level) routes traffic within an AS (intra-AS routing). While the Internet provides connectivity between ASes by using a single homogeneous routing protocol, known as *Border Gateway Protocol* (BGP), it gives flexibility to individual ASes to achieve their own goals by allowing them to freely choose their own routing protocols. Figure 2.1

illustrates an example of the hierarchical structure containing three ASes. We see that each AS maintains its own network (shown as a cloud) and use its own protocols such as Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), and RIP (Routing Information Protocol) to route within its network. The ASes are connected to each other using BGP.

We discuss how each level of the hierarchical structure can lead to sub-optimal Internet paths. We split our discussion into: i) inter-domain routing and ii) intra-domain routing.

2.1.1 Inter-domain Routing

The inter-AS routing protocol, or BGP, informs each AS of the destinations that are reachable and how a sequence of AS-hops (AS_PATH) can lead to these destinations. Each AS gathers from each neighbouring AS a list of reachable IP addresses (or IP prefixes) with the corresponding AS_PATHs to reach them. The AS aggregates these lists and adds the IP addresses from its own network on to the list then advertises the aggregated list back to its neighbouring ASes. There are often multiple paths that can reach the same destination, but BGP only chooses one of these paths to be added to the list before it is advertised.

One of the most important decisions in BGP routing is therefore selecting the AS_PATH to reach a destination when there are multiple paths available. This decision is set by local AS policies which are often driven by the individual AS's economic incentives. For example in a customer-provider relationship, the customer AS will remove paths to the provider AS if there is an alternative AS_PATH available in order to avoid paying transit costs. The AS_PATH suggested by BGP routing can therefore be circuitous even if a more direct path exists.

It is common in the Internet that there are multiple connection points between two ASes. Thus an AS also has to decide which connection point to use when a packet leaves the network. A common practice in peering agreements is to use *hot potato* (or early exit)

routing. This method chooses the connection point which has the lowest latency based on its current location but this does not guarantee it is the lowest latency for the entire path [138].

2.1.2 Intra-domain Routing

Intra-AS routing searches for the lowest-cost path within the network. The cost is typically computed based on various metrics in a link such as delay, throughput of a link, reliability and load. Not all Intra-AS routing protocols compute the cost in the same way. For example, Cisco's OSPF routers default to using throughput of a link to compute link cost whereas RIP searches for the path with the least hop count.

2.2 Measuring the Internet

Being able to capture the performance of individual paths on the Internet is important in order to optimising the performance. Measuring Internet paths is particularly challenging because competing ISPs reveal little information about the performance of their network due to the economic sensitivity of these information. Hence, from an end-user's perspective, the Internet is often viewed as a black-box.

Not surprisingly, there are thousands of papers that aim at capturing various information in the Internet. We here only focus on work that is relevant to this thesis. We study work which reveals the latency (or delay) of a path, the bandwidth (or data transfer rate) of a connection and the network topology of the Internet. We conclude this section by discussing available testbeds for collecting these metrics.

2.2.1 Latency

We define *latency* or *round trip time* (RTT) as the delay in sending a packet from a sender to a receiver and back.

Latency of a path is computed as the sum of:

- *propagation delay* which is the time it takes for the packet to travel,
- *transmission delay* which is the time needed to put all the bits of a packet onto the links,
- *queuing delay* which is the waiting time in a router before a packet is sent, and
- *processing delay* which is the time it takes for routers along the path to decide on the next-hop.

In wired networks like the Internet, propagation and transmission delays are fixed and deterministic for a given route. These delays are influenced by the access medium (such as fibre optics and copper wiring) and the technology used to transmit the data (e.g. ADSL, DOCSIS) respectively. In contrast, queuing and processing delays have larger variance and can be caused by a variety of factors such as congestion from competing traffic or background router maintenance processes [98].

Traditionally, latency can be measured using `ping` which sends a packet to the receiver to trigger the receiver to send a reply back to the sender. `ping` can trigger the receiver to reply in several ways: using *Internet Control Message Protocol* (ICMP), or by sending an ‘erroneous’ packet using UDP or TCP which will cause the receiver to report the error back to the sender. CAIDA [79] studied the various methods and found ICMP probes to be the most responsive in the Internet.

While the above method requires control at the source, King [49] is a tool that can measure the latency between two arbitrary hosts. King approximates the latency between a source and destination by their *authoritative name servers* and measures the latency between the two corresponding name servers instead. King first measures the latency to the source’s name server then requests the source to perform a recursive query to the

destination's name server. By taking the difference between the two measured times, King can estimate the latency between the source and destination. Recursive query in name servers is only an optional feature thus this method does not work all the time.

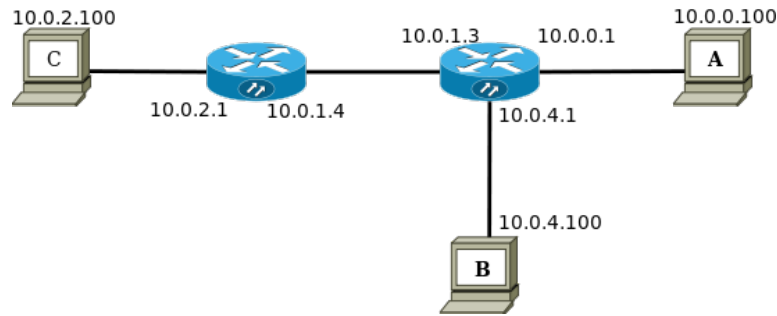
There has been substantial work in embedding Internet nodes into a *network coordinate system* based on latency measurements [31, 78, 86, 93, 99, 109, 121]. Using network coordinates, nodes can predict latency without having to perform any measurements and instead compute them based on the network coordinates. GNP [93], IDES [86], Lighthouse [99], ICS [78] and Virtual Landmarks [121], use fixed landmarks with predefined coordinates to triangulate the coordinates of other nodes. Vivaldi [31] and BBS [109] operate over successive iterations where each node updates its network coordinates each time to reduce the embedding error. Lee et al. [74] pointed out that network coordinates will always have errors due to routing inefficiencies in the Internet. Zhang et al. [128] and Lumezanu et al. [83] considered using network coordinates to find routing inefficiencies.

2.2.2 Network Topology

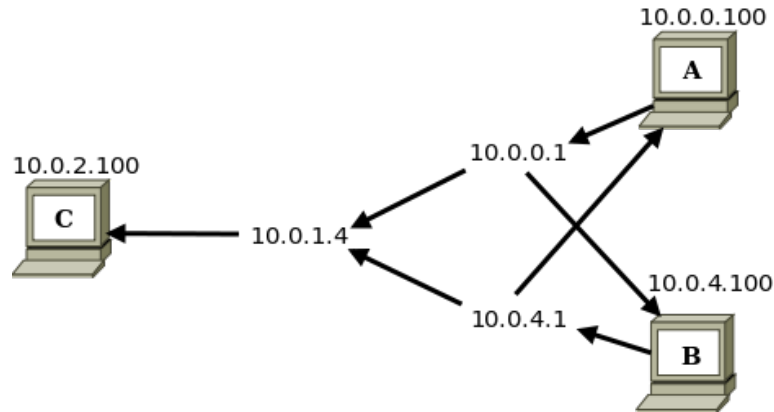
By combining the information from the network topology and end-to-end measurements, we can predict the performance of unmeasured paths. In routing overlay networks, this can be beneficial as a method of reducing measurement overhead, or network topology can be used to provide beneficial suggestion to detours. We examine two different network topologies in the Internet: the router-level topology and the AS-level topology.

Router-level topology. The IP addresses that is traversed in a path can be found using `traceroute` or the IP record-route option [63, 111, 112]. The IP record-route option records up to the first 9 IP-hops in a path.

While we can construct a simple graph using the IP addresses, this graph is not representative of the actual physical network. The reason is that each router has multiple IP addresses and the IP address that is returned often depends on the network interface at which



(a) Router-level Topology



(b) IP-level Topology

Figure 2.2: IP-level topology from `traceroute` or IP record-route can lead to more links in a network

the probe arrives from. We illustrate this with a simple example. Figure 2.2(a) shows the actual physical topology of the network between three hosts A, B, and C connected by two routers. Assume that the routers always return the IP address that is closest to the host (which is the standard practice for Cisco routers [111]). If we run `traceroute` for path A, B and A, C, the topology constructed would be that of shown in Figure 2.2(b). The IP-level topology has more links than the actual topology, leading to an inflated view of the actual Internet.

To construct a more useful topology, we have to group the IP addresses (known as *IP aliases*) from the same router together. There are several different techniques used to do this. Ally [116] notices that the IP-ID field is incremented for every error message a router sends out. Ally can verify IP addresses that are likely to be from the same router by

observing if the IP-ID is incremented. DisCarte [111] tries to infer the rule that routers use in their response to obtain a more accurate topology. Finally, APAR [50] and KAPAR [64] notice that subnets, which is the set of interfaces that are directly connected to each other (i.e. they are 1-hop away), use the same set of IP prefixes. Once the IP prefix of each subnets is found, the topology can be constructed by merging the subnets together.

AS-level topology. The AS-level topology can be found either directly from BGP routers or using the IP topology. IP addresses are mapped to their corresponding AS using various lookup services: *Regional Internet Registries* (RIRs), *Looking Glass Servers*, or through Team Cymru’s lookup service [122]. RIRs maintain a static records of the IP addresses each AS is allocated to. While Looking Glass Servers provide AS-link information from local BGP routers, Team Cymru aggregates these information from multiple BGP peers. Finally, the Route Views project [11] aggregates BGP data to provide a view of the AS topology.

Several projects have collected the Internet topology. DIMES [110] discovers the Internet topology using distributed agents. The data is made public for clients to download. Skitter [29] and Ark [3] collect measurements both actively and passively to map the Internet topology. Skitter is a predecessor of Ark. Ark collects its measurements measuring from more than 20 vantage points around the world. Finally, Ally is used in Rocket-Fuel [116] to map traceroute information into a geographical map of the world.

iPlane [84] collects traceroute and other measurements such as latency from several PlanetLab vantage points. Their aim is to build a topology of the Internet that can predict latency—and other performance metrics—given an arbitrary Internet path.

2.2.3 Bandwidth

In networks, the term *bandwidth* is a class of metrics that is related to the rate data that can be transferred across a series of links. Different bandwidth metrics can result in very

different bandwidth values. The three most common bandwidth metrics are: *capacity*, *available bandwidth* and *TCP bandwidth*. We first define these metrics then discuss how they are measured.

The capacity of a path is the maximum rate data that can be sent over the path in the absence of third-party traffic. The capacity of a path is limited by the *narrow link* which is the link that sends data at the lowest rate. Available bandwidth is the residual capacity of a path in the presence of cross-traffic in the Internet. The *tight link* is the link with the least available bandwidth. TCP bandwidth is the amount of data that can be sent using TCP per unit time.

While all three bandwidth metrics (capacity, available bandwidth and TCP bandwidth) are important in evaluating the performance of Internet paths, our work focuses on TCP bandwidth. It uses the most common protocol seen in the Internet and is also the carrier for HTTP traffic. The other two metrics can be measured with a significantly lower overhead than TCP bandwidth and can be useful in inferring TCP bandwidth.

Capacity. The capacity of a path can be measured by sending a pair of packets back-to-back [33, 34, 37, 62]. At the receiver, the two packets would arrive at two different times due to packet dispersion caused by the transmission delay. For example, when sending a 1000-byte packet over a path with capacity limit of 10 Mbps, the transmission delay would be 0.8 ms. The main difference between the various proposed approaches is in how noise is removed from the measurements due to cross-traffic and routers' processing delays.

Available bandwidth. In general, available bandwidth tools are attractive in bandwidth routing overlay networks. Available bandwidth can be measured without incurring a high measurement overhead and is less intrusive (meaning generates less packet loss to cross traffic) than measuring TCP bandwidth. There are many tools for measuring available bandwidth in the Internet. All the tools send multiple packets and observe the delays or

dispersion between the packets caused by the cross-traffic. The tools can be separated into three categories: *Probe Rate Model* (PRM), *Probe Gap Model* (PGM) and *Probe Delay Model* (PDM). Table 2.1 provides a taxonomy of the more commonly available bandwidth tools and the technique they use.

PRM tools send a train of packets at a pre-defined rate and observe the rate at which this train arrives at the receiver. If the send rate is below the available bandwidth, the rate at which the train is sent would be the same as the rate observed at the receiver. However if the send rate is above the available bandwidth, the packets will be dispersed due to the cross traffic and the rate observed at the receiver will be lower. Based on this observation, available bandwidth can then be probed by searching for the maximum send rate such that the send rate is the same as the receiving rate.

PGM tools require the capacity of the narrow link (i.e. the minimum capacity of the path). In PGM, two or more packets are sent back-to-back (i.e. immediately after one another). The receiver observes the ‘gap’, or time difference, between the two packets. This gap is caused by queuing in routers and the size of the gap can be used to compute the cross-traffic rate in the path. The available bandwidth can be computed as the difference between the capacity of the narrow link and the cross-traffic rate. PGM tools are known to under-estimate bandwidth when the tight link is not the same as the narrow link [72].

Similar to PGM tools, PDM tools also require knowing the capacity of the path. PDM tools observe the difference in latency from packet arrivals to infer whether there is queuing in the path or not. We can express *utilisation* ρ as the fraction of the number of probes queued over the number of probes sent. Given the capacity of the path, the available bandwidth can be computed as ρC where C is the capacity of the tight link.

TCP Bandwidth. TCP bandwidth implements a congestion control algorithm to share the bandwidth of a path with other competing TCP flows. The algorithm can be broken down into two phases. The first phase, *slow start*, increases the growth rate of the TCP flow

Name	Author	Year	Technique	Publicly Available?
bprobe [22]	Carter et al.	1996	PRM	Yes
nettimer [69]	Lai et al.	2000	PRM	No
TOPP [89]	Melander et al.	2000	PRM	No
PTR [56]	Hu et al.	2003	PRM	Yes
pathload [58]	Jain et al.	2003	PRM	Yes
pathChirp [105]	Riberio et al.	2003	PRM	Yes
Abing [91]	Navratil et al.	2003	PRM	Yes
STAB [106]	Ribeiro et al.	2004	PRM	Yes
Envelope [61]	Kang et al.	2006	PRM	No
ImTCP [124]	Tsugawa et al.	2007	PRM	Yes
pathpair [70]	Lai et al.	2008	PRM	No
ASSOLO [45]	Goldoni et al.	2009	PRM	Yes
Delphi [104]	Riberio et al.	2000	PGM	No
IGI [56]	Hu et al.	2003	PGM	Yes
Spruce [117]	Strauss et al.	2003	PGM	Yes
Forcaster [92]	Neginhal et al.	2006	PDM	No

Table 2.1: Taxonomy of Available Bandwidth Tools

exponentially to the latency of the path. This quickly uses up the available bandwidth in the path which then leads to a packet loss event. This triggers the second phase where the growth rate is adjusted according to the *additive increase multiplicative decrease* (AIMD) algorithm.

There are different variants of TCP. The original variant is TCP-Reno which was later improved to respond better to packet losses in TCP-NewReno. However, TCP-NewReno is still known to be slow in high-speed networks that also have high latencies (known as LFN, *long fat networks*). There are many variants that attempt to address this issue such as Westwood [23], FAST TCP [129], HS-TCP [40], TCP-CUBIC [51] and Compound TCP [120]. TCP-CUBIC [51] is the standard in Linux since kernel-2.6.19 and makes modifications to the original AIMD algorithm. In the absence of loss, while the rate in the original AIMD algorithm increases linearly, the rate in TCP-CUBIC increases based on a cubic curve. Since the release of Windows Vista, Windows use a Compound TCP which uses jitter in latency to help adjust the TCP's rate.

TCP bandwidth can be measured directly by emulating a file transfer using the TCP protocol. This is most commonly done using bandwidth benchmarking tools such as `iperf`.

Alternatively, TCP bandwidth can be predicted from stochastic models. Mathis et al. has developed a simple equation to model the flow of TCP-NewReno [88]. The equation is as follows:

$$BW = \frac{MSS}{RTT} \frac{\Phi_{reno}}{\sqrt{p}} \quad (2.1)$$

The equation shows TCP bandwidth can be expressed as a function of MSS, the maximum segment size, RTT, the latency of the path, and p , the path's average loss rate. The constant Φ is dependent on the TCP acknowledgement strategies used in TCP Reno and is empirically found to be roughly 0.98 in the commonly used delayed-acknowledgement strategy. The model assumes that bandwidth is not limited by the sender's and receiver's buffer sizes.

More recently, the model has been refined by a number of authors to be more accurate [12, 17, 90, 96] by including slow-start and retransmission timeout mechanism in TCP. These refinements require additional parameters, such as TCP retransmission timeout, which cannot be retrieved prior to the start of a TCP flow. Other work [21, 114] focuses on short-lived TCP connections where the slow-start phase plays an important role in dictating the TCP bandwidth.

2.2.4 Testbeds

There are many publicly available testbeds that can collect bandwidth, latency and loss measurements. NS-2 [10] provides a simulated environment where users have full control over the network. Emulab [7] allows researchers to carry out repeatable and controlled experiments on real servers and networks. Measurement Lab [9] allows researchers to deploy their measurement tools online for home broadband users to test their connection.

Finally, PlanetLab provides a wide-area network where researchers can access to over 1000 nodes from 500 sites worldwide. These nodes are connected to each other via the Internet, providing realistic Internet paths with cross-traffic as well as access to other nodes in the Internet.

Though PlanetLab is the largest publicly available testbed available, there are several limitations using PlanetLab as a measurement platform: (i) it is biased towards academic sites [18]; (ii) the hosts are shared and other users on the same host may generate traffic affecting the experiment; (iii) and access and traffic restrictions limit low-level experimentation and large-scale measurement studies.

In this thesis, we use PlanetLab as our primary testbeds. Due to the relatively large number of academic sites on PlanetLab, it may not be representative in the Internet. First, academic sites are connected via research networks where commercial clients do not have access to [18]. These institutions often have bandwidth higher than residential clients. However, commercial ISPs disfavor measurements conducting over their network because of network security and business confidentiality concerns. In this thesis, we start by conducting experiments exclusively with PlanetLab nodes, then extending our results to include third-party hosts such as webserver and residential broadband clients in Chapter 6. Due to downtimes and bandwidth restrictions on PlanetLab, we use between 50 to 200 nodes from different sites at any one time.

We also use UkairoLab to circumvent problems that PlanetLab has. UkairoLab contains 10 geographically-dispersed nodes located in US, India, UK and France. The network connectivity between the UkairoLab nodes are provided by commercial providers. These nodes have been virtualised to provide kernel access.

2.3 Detouring the Internet

The Internet is designed to provide connectivity between end-hosts but place very little emphasis on the performance of its chosen routes. Internet only provides one path for a given pair of hosts. *Detour routing* (or *detouring*) improves the performance of Internet paths by providing alternative choices of paths to route to the destination. In this section, we first show that detouring can indeed improve the Internet path performance (§2.3.1). Next, we show how detouring has been applied in the past to provide these improvements (§2.3.2). Finally, we compare detour routing-based approach with other approaches that also aim to improve Internet path performances (§2.3.3).

2.3.1 Benefits from Detour Routing

From our discussion in §2.1, it may be obvious that Internet routing is driven by business relationship rather than end-user path performances and therefore there is scope for detour routing. In this section, we look at several past work [81, 82, 108, 138] that evaluates the benefits from detour routing and the likely causes of these benefits.

We first show an example of how detour routing can be beneficial and use it to define some basic terminology. Figure 2.3 shows how traffic is routed from Paris to Braga. The *regular path* taken by regular Internet routing would route the traffic to Geneva, Switzerland then to Madrid, Spain before entering Portugal and requires 71 ms to complete the round trip. By detouring through a node (or a *detour*) in Madrid, the *detour path* becomes less circuitous and traffic is no longer routed to Switzerland resulting in a significantly shorter path (46 ms).

Savage et al. [108] are the first to show that Internet routing does not provide the best path in terms of latency. They found by relaying traffic to an alternative node in the network, the latency of the detour path can be better than the regular path. In their datasets, they show that at least 20% of the paths can benefit from latency detouring. They also

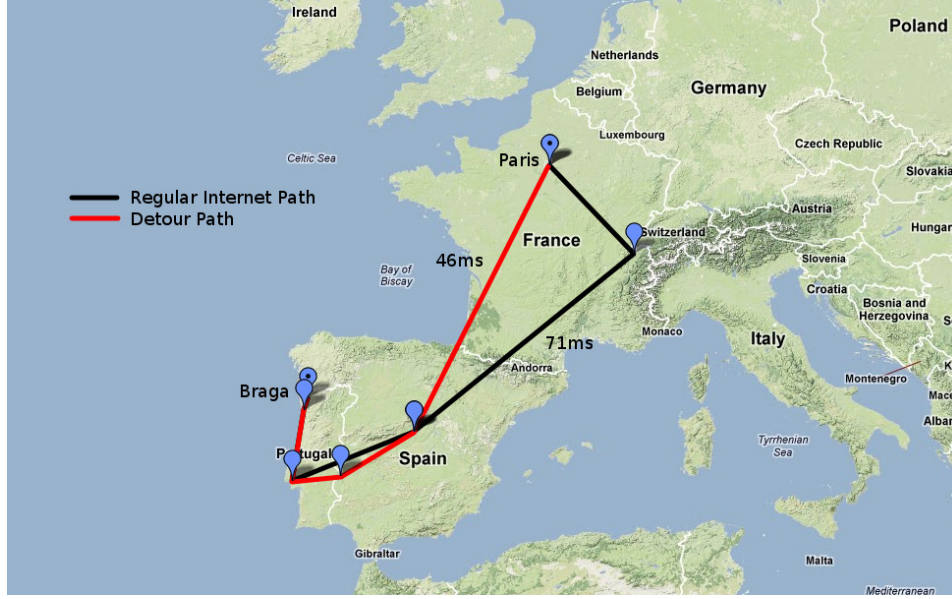


Figure 2.3: An example of detouring which improves latency

show that bandwidth detouring can be beneficial as well.

Zheng et al. [138] study the beneficial latency detour paths on PlanetLab. They observe that the existence of these beneficial detour paths stems from multiple factors. In addition to policies set between ASes, they show that beneficial detours can exist in: i) inter-domain routing due to hot-potato routing, and ii) intra-domain routing due to configurations of link weights which does not correspond to the latency of a link.

Lumezanu et al. [81] study the beneficial latency detour paths on an AS-level and classify these paths into: *possible* paths which are paths that can actually be found through BGP's decision process, and *impossible* paths which are paths that are *never* advertised on BGP. They observe that at least 25% of the detour paths are possible and 62% of these paths will have the same or lower transit cost than the regular path. In a different study [82], they performed a longitudinal study of latency detours and observe that almost all ($\geq 99\%$) of the detours persisted for 5 hours and more than half persisted for a day.

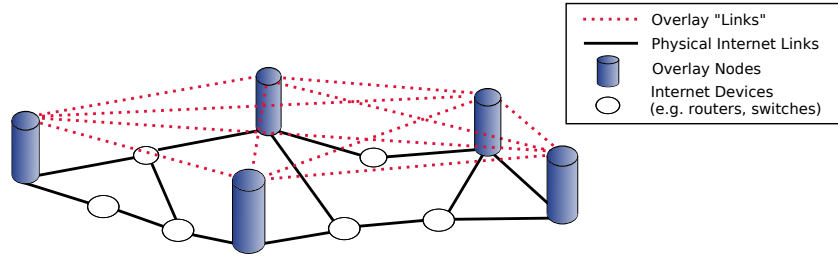


Figure 2.4: An Illustration of an Overlay Network

2.3.2 Detouring Overlay Networks

Overlay networks are groups of nodes placed in another network and rely on the other network to facilitate the connection between these nodes. Figure 2.4 illustrates an overlay network which consists of five overlay nodes (blue cylinders) built on top of a larger underlying network. The dotted line represents overlay connections which in reality are provided by a series of links (black lines) and routers (ovals).

Overlay networks provide additional services that the regular Internet routing protocol cannot provide. For example, they enable file sharing systems such as BitTorrent, voice communication services like Skype, secured network access as in VPN, or provide QoS services [119].

Detouring overlay networks apply the concept of detour routing on the Internet. The nodes in the overlay network acts as intermediaries to redirect traffic over the Internet. We split our discussion of the different overlay networks by the different performance metrics they aim to improve in the Internet.

Reliability. Resilient Overlay Network (RON) [13] is a project that aims at reducing the number of failures between hosts in the Internet. They are the first to show that an overlay network can be used to circumvent failures in the Internet to improve reliability with a live deployment. However, RON requires all-to-all measurements to find failures in its overlay network. Gummadi et al. [48] observe that it is sufficient to select a small number of detours at random to improve the reliability of paths. MONET [14] targets at

improving web availability for clients by making multiple simultaneous HTTP requests.

Latency. Detour routing has been most successfully applied to improving latency. Savage et al. [107] show that detour routing can reduce latency and outline an overlay architecture that uses *IP tunnels* to redirect Internet traffic. Peerwise [83] aims to encourage nodes to detour via each other by forming ‘peerings’ where both parties can benefit from detouring via each other. Peerwise discovers detours by first embedding each node into a network coordinate system. A large embedding error suggests opportunities for benefits in detouring. In comparison to our work (in §3), we discover detours by analysing AS paths in the Internet.

Bandwidth. Since bandwidth is an expensive metric to measure, all-to-all measurements do not scale as the size of the network grows. Bandwidth detouring overlay networks aim at reducing measurement overheads while providing useful detours for clients to use. Most of these designs have a detour discovery process which suggests a small set of candidate detours then clients perform measurements on this small set to find the best one to use.

Several projects take advantage of available bandwidth measurement tools to gather bandwidth measurements in order to reduce the measurement overhead incurred by the overlay (as discussed earlier in §2.2.3). Jain et al. [57] propose a video streaming overlay network that uses detouring to provide higher quality videos. They reduce measurement overhead by modifying (or shaping) the traffic flow pattern of a video stream to gather measurements implicitly using the available bandwidth measurement tools.

Bandwidth-Aware Routing Overlay Network (BARON) [75] uses both capacity and available bandwidth estimates to discover good bandwidth detours. BARON stores all-to-all capacity measurements in several distributed nodes. BARON uses these capacity measurements to suggest several detours for its clients. The clients then choose the best detour using available bandwidth measurement techniques.

SideStep [27] is an overlay network that uses *content distribution networks* (CDNs) to find detours without incurring a large overhead. Clients form *detour groups* where they detour via each other. The detour groups are formed from clients taking similar CDN redirections.

Zhu et al. [140] simulate a bandwidth detouring network where flows are allowed to switch paths to obtain a higher bandwidth. They compare two algorithms for selecting detours: a *proactive* approach which always switches the flow to the one with the most available bandwidth and a *reactive* approach which only switches a flow when more bandwidth is required.

In contrast to the above bandwidth overlay networks, our work, Ukairo, first suggests a set of detours that is based on the *quality* of each detour. Ukairo then picks the best detour from the set of suggested detours by simultaneously processing the user’s requests and measuring the bandwidth from the set. While most of the above systems are either in simulation [140] or experimental [57, 75] stage, Ukairo is designed to be ready-to-use where it can be deployed today and provides immediate benefits to end-users.

Unlike Jain et al.’s video streaming overlay [57], Ukairo does not shape the traffic thus providing the maximum download rate achievable. In contrast to BARON [75], Ukairo can process user’s requests on-demand avoiding the time-consuming process of measuring available bandwidth for the set of suggested detours. Our detour selection strategy is fundamentally different from SideStep [27]—ours select detours based on the quality of each detour node whereas SideStep uses third-party CDNs to select detours. Finally, Ukairo is different from Zhu et al. [139], we do not switch detours once the best detour has been chosen because this might affect the stability of the connection.

2.3.3 Alternatives to Detouring

Content distribution networks. Content distribution networks (CDNs) such as Akamai [95], Amazon CloudFront [1] and Coral [5], replicate content and place it at locations closer to clients. Clients are transparently redirected to a nearby CDN node using name servers. By downloading content from a closer server and not from the original source leads to better responsiveness and hence better end-user experience. CDNs analyse user trends to decide which content to replicate.

Not all contents can be replicated using CDNs. Since content requires storage and network resources to replicate, CDNs are only cost-effective for popular content. Content that cannot be stored in caches such as teleconferencing and personalised content where it is only accessed by a limited number of clients cannot be stored on CDNs. In comparison, detour routing can provide performance improvement for these other content and does not need the expensive infrastructure required in CDNs.

Multipath approaches. The performance can also be improved by simultaneously downloading from multiple sources. Peer-to-peer networks such as BitTorrent [4] separate the file into pieces so clients can simultaneously download different pieces of the same file from multiple peers. Bullet [66] disseminates the file systematically based on the bandwidth each peer has. Splitstream [24] constructs application-level multicast trees to disseminate its content. At the transport layer, multipath TCP [130] allows a single data stream to be split across multiple paths, with an aggregate bandwidth that is at least the best from one of its available single path.

These solutions often require clients to download from multiple sources to have benefits. Additionally, peer-to-peer solutions require modifications of clients and content servers by installing software to support these networks. Multipath TCP requires modifications to the TCP stack. Our Ukairo overlay (in §6) does not require any modifications to the content servers thus can be deployed immediately to take advantage of benefits from

detour routing.

Source Routing In source routing, users can choose the path each packet traverses through the network. Users select the intermediary nodes that the path takes. These intermediary nodes do not have to be connected directly by the same link. The primary goal in source routing is to expose users to multiple choices to route their traffic while complying to AS policies or allowing more complicated AS policies that BGP cannot express otherwise. Since paths are selected by users, ASes have less control over how traffic traverses through their network. It can also lead to security issue where an attacker can initiate a connection to the victim pretending to be someone trusted by the victim [38].

Pathlet [44] uses a path vector algorithm to disseminate segments of paths (known as pathlet) which represent a series of nodes used in source routing. Similar to BGP, the pathlets are disseminated using a path vector algorithm. By limiting the number of pathlets each node advertises, they show that routers only have to store a similar number of routing entries as in BGP. NIRA [134] suggests a new addressing scheme to incorporate AS-relationship. MIRO [133] uses BGP routes by default but allows clients to request an alternative path. Platypus [103] enables contracts to be made specifically for using source routing. By applying cryptography, the system delivers user's credentials to source routing enabled routers to test for policy compliance.

While the above source algorithm aims to provide path diversity, Feedback routing [139] aims to improve the performance of a path. Feedback routing performs AS-level source routing and switches AS-path if the quality of the current path is poor. The authors propose that routing should be done at designated routers at the edge of the network thus alleviating the requirement for individual clients to store the AS-level Internet topology.

3 Latency Detour Discovery, An AS-level Approach

Latency plays an important role in many Internet applications. In web-browsing, long latency extends webpage loading times which can cause users to give-up viewing a page. In voice and video conferencing, latency slows down the conversation and can cause the speaker to interrupt the conversation. Latency can also affect revenue streams over the Internet. For example, Amazon found every 100 ms would cost them 1% of their sales and Google found an extra 500 ms in search page generation time will drop traffic by 20% [65].

Regular Internet routing does not have the ability for user applications to switch to low-latency paths. Current content providers reduce latency by pushing their content towards the user geographically using content distribution network (CDN). However only popular content can be stored in CDNs, leaving the latency of personalised data and real-time applications unchanged.

Our approach uses *detour routing* which redirects the data to a *detour* where it is forwarded to the destination. Unlike Internet routing, detours are designated nodes in the Internet and perform packet forwarding above the transport layer—packets still rely on regular Internet routing to reach the detour node and destination. Detouring offers additional paths for end-user applications to satisfy network-specific needs.

Previous work has shown that detouring can improve latency [83, 107]. However an area that received less attention is how detours can be discovered scalably. Unfortunately, current detour discovery methods often involve large amount of measurements between sources, detours and destinations. A large amount of measurements may offset the benefits obtained through detouring.

Our method for discovering latency detours takes into consideration the AS-level Internet topology. Our intuition is that most improvements from detouring can be achieved by looking at the AS paths alone. By only considering the Internet topology on the AS-level, we drastically reduce the amount of data required to store path information. Additionally, AS paths are more stable than IP paths, thus reducing the frequency of re-measurement.

Our method first collects an Internet path measurement using `traceroute` then translates the IP addresses in the path to AS numbers. We deconstruct the AS path into pairs of ASes known as *AS-links*. Next, we construct *fingerprints* from clusters of regular paths. The fingerprints are used to identify groups of paths that benefit from the same detour.

Using the method we developed, we correctly classified 94.3% of the paths as having detours and 83.1% of the paths without detour on a 176-node PlanetLab dataset. 85.3% of the paths with detours benefited from the detour that our method suggests.

In this chapter, we first describe our clustering method and how it can be implemented in a decentralised manner. Next, we demonstrate the accuracy and quality of our detours from our method on PlanetLab. Finally, we provide a short analysis of the changes in AS path using our detouring methodology.

3.1 AS Path Deconstruction

An *regular path* (a, b) between two hosts a and b crosses multiple ASes $AS_1, \dots, AS_n$ given by regular Internet routing. Figure 3.1 illustrates the regular path (a, b) and a *detour path* (a, c, b) being a combination of two regular paths, (a, c) and (c, b) . The latency or

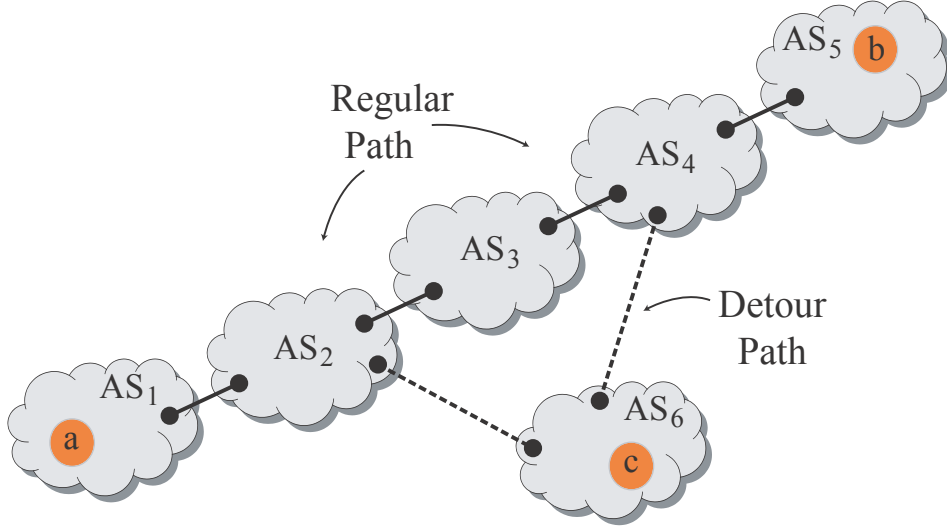


Figure 3.1: An example of AS-level detouring

round trip time of a detour path is the sum of the latencies of the two regular paths.

Detour routing exploits the fact that regular Internet routing does not map well to a metric space [31] and results in the existence *triangle inequality violation* (TIV) [135]. Equation (3.1) illustrates how the regular path (a, b) with round trip time (or latency) $rtt(a, b)$ can be detoured via c to have a shorter latency, hence forming a TIV.

$$\frac{rtt(a, c) + rtt(c, b)}{rtt(a, b)} < 1 \quad (3.1)$$

In previous work, it has been shown that most of the improvement can be obtained by relaying through a single detour node and relaying through multiple detour nodes suffers from diminishing return [80]. Therefore we focus on single-hop detours only.

In many cases, the existence of TIVs is the result of routing policies between ASes [138]. For example, a customer AS may avoid advertising routes which require the customer AS to pay its provider AS a *transit cost*. However, traffic flowing through a detour node that is in the customer AS will be treated as local traffic thus benefiting from these transit routes. Another example is a transit AS, such as Internet2 in the US, that only provide transits to some classes of traffic. A detour node in Internet2 enables third-party to transit through

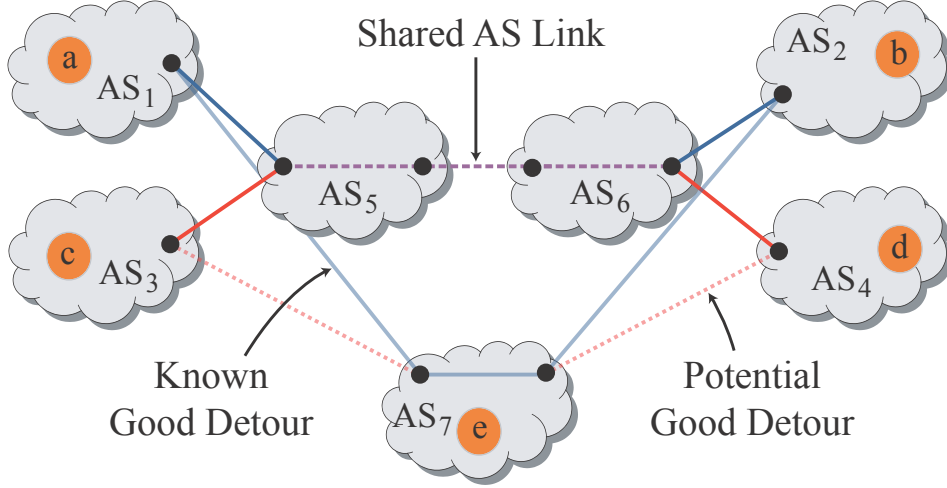


Figure 3.2: An example of shared AS-links

the AS thus leading to better latency.

Our method can be used to discover AS-level detours. We believe that most detours can be found at the AS-level. In our 176 PlanetLab dataset, we found 96.1% of all beneficial detour paths traverse at least one different AS than the regular path.

3.2 Path Clustering

The concept of our method is to first identify AS paths that are similar to each other, then choose an appropriate detour for such paths. In Figure 3.2 we show an example of a shared AS-link between two paths. Path (a, b) contains three AS links: AS_1 to AS_5 , AS_5 to AS_6 and AS_6 to AS_2 . Path (c, d) also contains the *shared AS link*, AS_5 to AS_6 , shown as a dashed line. Our hypothesis is that if (a, b) uses detour e then path (c, d) is likely to benefit from the same detour. We group paths that are similar together to form *clusters*. There are two types of clusters: *TIV clusters* which are paths that are similar and have TIV; and *no-TIV clusters* which are paths that do not have TIV.

We provide a formal definition of our algorithm. First, the similarity between two paths

is calculated using the *path similarity ratio*.

$$d((a, b), (c, d)) = \frac{1}{\max(m, l)} \sum_{i=1}^l \sum_{j=1}^m \mathbf{1}_{\{p_i^{a,b} = p_j^{c,d}\}} \quad (3.2)$$

Equation (3.2) shows how the path similarity ratio is computed between path (a, b) and (c, d) where $p_1^{a,b}, \dots, p_m^{a,b}$ and $p_1^{c,d}, \dots, p_l^{c,d}$ represent the set of AS-links for the two paths respectively. $\mathbf{1}$ is an indicator function which is the value one when the AS-links are identical and zero otherwise. For example, the path similarity ratio for the path (a, b) and (c, d) in Figure 3.2 is $1/3$.

Our approach merges paths that are similar together depending on their path similarity ratios. We add a constraint, the *similarity threshold* τ , which is the minimum ratio for two paths to be merged together (i.e. $d((a, b), (b, c)) > \tau$). For $\tau = 0.2$, the paths in Figure 3.2 would be merged together and the formed cluster would contain both paths $\{(a, b), (c, d)\}$.

Our method employs a hierarchical clustering to decide how clusters should be merged. Using the bottom-up approach, each path is initially treated as a cluster on its own (i.e. a singleton cluster). At each iteration, the pair of clusters, C_1 and C_2 , with the highest *cluster similarity ratio* $d_c(C_1, C_2)$ are merged. The cluster similarity ratio between two clusters, $C_1 = \{P_1, \dots, P_L\}$ and $C_2 = \{P'_1, \dots, P'_M\}$, where $P_i, i = 1, \dots, L$ and $P'_j, j = 1, \dots, M$ are AS paths, is calculated using the equation defined below:

$$d_c(C_1, C_2) = \frac{1}{LM} \sum_{i=1}^L \sum_{j=1}^M d(P_i, P'_j) \quad (3.3)$$

Note that the no-TIV clusters are never merged with TIV clusters and we need that $d_c(C_1, C_2) > \tau$ for two clusters to be merged together.

The similarity threshold $\tau \in [0, 1]$ can be used to tune the number of clusters resulting from our method. A higher threshold would have more clusters which would mean more

storage space is required. When $\tau = 1$, AS-paths must be identical for them to be merged together.

Each TIV cluster also tracks a list of detours each path has. The list is aggregated whenever the TIV cluster is merged. Our method currently suggests the most frequently used detour in a TIV cluster. Certainly load-balancing schemes can be used if detours become overwhelmed.

Our algorithm can take advantage of dynamic programming to speed up computation. This is most easily done by computing the *similarity matrix* which contains the path similarity ratios between all paths. Note that only half of the matrix needs to be computed because the path similarity ratio is symmetric (i.e. $d((a, b), (c, d)) = d((c, d), (a, b))$).

3.3 Decentralised Path Clustering

The decentralised path clustering aims to discover detours between any two nodes within a network. Using a decentralised approach reduces the measurement costs by measuring to a small subset of all paths compared to our centralised method. The paths are clustered using the path clustering strategy described above in §3.2. Nodes then exchange their clusters with each other through successive gossiping rounds. This disseminates information collected to other nodes in the network. Since only a small subset of paths is measured in the decentralised implementation, paths without detours cannot be assumed to be no-TIV paths thus no-TIV clusters cannot be reliably formed. We describe our decentralised approach in detail by breaking it into four steps.

1. Measurement collection. Each node randomly selects k other destination nodes where k is much smaller than the total number of nodes. Each node directly measures paths to the selected set of destinations and requests each destination node to provide the measurements to the remainder paths in the selected set. Therefore, each node has an all-to-all set of measurements for the selected k nodes. Next, each node identifies the TIV paths from

the dataset and records the AS paths for these paths along with their associated detour. Since k is small, the amount of measurements is significantly reduced.

2. TIV clustering. Once the measurements are gathered, each node performs path clustering described in the centralised version in §3.2 except that no-TIV clusters are never formed.

3. TIV cluster exchange. After the clusters are formed, they are disseminated to the other nodes. Each node randomly choose m other nodes and exchange their TIV clusters thus conducting m gossiping rounds. The TIV clusters which meets the similarity threshold τ are merged. In each gossiping round, each node only exchanges their original TIV clusters to prevent duplicated paths during cluster exchanges.

4. Detour discovery. To find detours for a given path, a node finds the most similar TIV cluster to the given path and chooses a detour from the set of ordered detours associated to the TIV cluster. If a path is most similar to a cluster that have no corresponding detours, the path is assumed to be a no-TIV path.

3.4 Evaluation

We investigate the accuracy of classifying TIV and no-TIV paths, and the quality of detours found using our centralised and decentralised version of the path clustering algorithm. We collected `traceroute` measurements from 176 PlanetLab nodes on December 10th, 2008. We run `traceroute` between all-pairs of nodes sending 100 `traceroute` probes per hop each with 1 second interval between the probes. The measurements are taken within a 24 hour period.

We construct the AS paths from `traceroute` measurements by following Mao et al.’s method [87]. Instead of using looking glass servers to map IP address to AS in Mao et al.’s method, we use Team Cymru’s service for quicker AS lookups. Team Cymru actively collects IP to AS mappings from BGP tables using over 50 different BGP peers. If

an IP address is not found in Team Cymru’s lookup service, we use the Regional Internet Registries (RIRs).

We found 9939 unique IP addresses from our `traceroute`, we resolve 99.58% using Team Cymru’s lookup service, 0.26% using RIRs, and the remainder 0.16% are reserved IP addresses that are unroutable on the public Internet. Our traceroutes successfully reached 20 614 destinations from each host which is 66.2% of all paths.

We found 69.2% of the paths can be improved by detouring and 47.5% of all paths can be improved by at least 100 ms. Due to the homogeneous nature of PlanetLab [18], we believe that more paths can benefit from latency detouring on other parts of the Internet.

3.4.1 Centralised Detour Discovery

In this section, we use our dataset to evaluate the accuracy and quality of detours in our centralised path clustering strategy.

Classification Accuracy

We compare the accuracy of our classification method with the TIV alert mechanism [128] which identifies TIV paths using network coordinate systems (where network coordinate systems are discussed in §2.2.1). While our method uses the AS-level topology of the Internet to find TIVs and discover detours, TIV alert mechanism can only classify TIV paths but cannot suggest valid detours. We first briefly describe how TIV alert mechanism operates before comparing it with our method.

TIV alert mechanism.

The TIV alert mechanism identifies TIV paths by first embedding the latency dataset into a Euclidean network coordinate system. This gives each node a coordinate where the Euclidean distance between any two nodes represents the latency of the path. Of course triangle inequality violations cannot be embedded into an Euclidean metric space. TIV

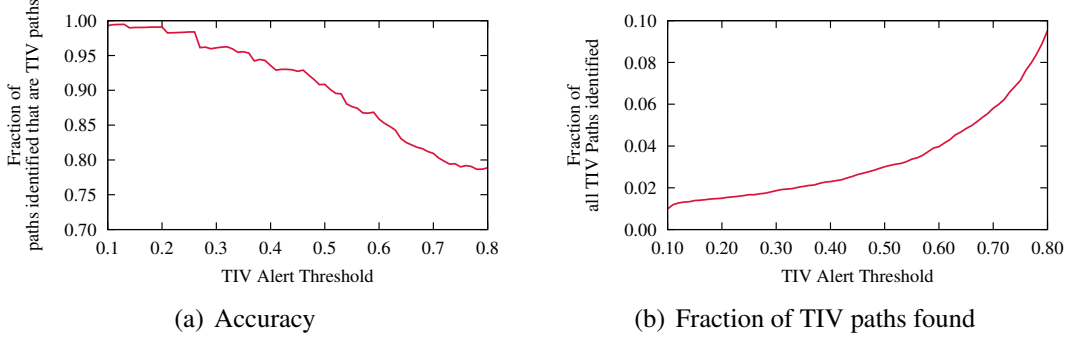


Figure 3.3: Accuracy and fraction of TIV paths found for different TIV alert thresholds

alert mechanism exploits this fact and assumes that for a given TIV path, the measured latency would be longer than the Euclidean distance between the two nodes. A TIV path is identified when the ratio between the measured latency and the distance between the two corresponding nodes is smaller than a predefined TIV alert threshold τ_{alert} .

We evaluate the TIV alert mechanism on our dataset. In previous work the TIV alert mechanism to only identify TIV paths which give large improvement between the detour and regular path [83, 128], here we aim to identify all TIV paths in a network. We follow their work and use a 5D metric space with Vivaldi [31] to construct our network coordinates.

In our experiment we vary the threshold τ_{alert} and observe the accuracy of TIV alert mechanism at identifying TIV paths. Figure 3.3(a) shows the accuracy of using TIV alert mechanism in identifying TIV paths for τ_{alert} between 0.1 and 0.8 at 0.01 increments and Figure 3.3(b) shows the proportion of all TIV paths identified for τ_{alert} between the same range. Though with smaller TIV alert threshold τ_{alert} finds TIV paths more accurately, it also finds a smaller portion of TIV paths. For example, if we set τ_{alert} at 0.5, 90.9% of the paths are correctly classified as TIV paths but this is only 3.0% of all TIV paths.

Using our path clustering algorithm described in §3.2, we can make use of the AS-level topology to classify TIV paths more accurately. The best detour for each path is associated with each TIV path. We increment the similarity threshold from 0 to 1 at 0.2 interval and

τ	0	0.2	0.4	0.6	0.8	1
Clusters Formed	4913	5028	5357	9417	13104	20327
TIV Accuracy	93.9	94.3	94.3	94.2	93.8	94.6
No-TIV Accuracy	82.3	82.0	83.1	83.9	86.5	88.8
Overall Accuracy	90.1	90.2	90.6	90.9	91.6	92.8

Table 3.1: Path clustering: Percentage of paths with and without detours that were classified correctly and incorrectly.

plot the classification accuracy and the number of clusters formed in Table 3.1. We notice that at large similarity threshold, more clusters are formed but have a slightly higher overall classification accuracy.

If we set the similarity threshold to 0.4, it gives a good balance between classification accuracy and the number of clusters formed. With this value of τ , we formed 5357 clusters of which 47.6% are no-TIV clusters. Even though no-TIV paths only contribute to 33.8% of all paths there is a higher proportion of no-TIV clusters because no-TIV paths are more heterogeneous and therefore less likely to be clustered. Using all measurements, we are able to classify 94.3% of all TIV paths correctly and 83.1% of no-TIV paths respectively. The higher mis-classification in no-TIV paths is also due to no-TIV clusters being more discrete.

Quality of Detours

Figure 3.4 shows the fraction of improvements from the regular path using path clustering. The figure only shows paths that can be improved (i.e. TIV paths). We compare our path clustering algorithm path clustering with brute force which exhaustively searches for the best detour for each TIV path and a random strategy, random-10 which randomly picks the best detour from 10 randomly chosen nodes. In our algorithm, we choose from the two most frequently featured detour node and pick from the better one.

We also compare our algorithm with detours obtained through a network coordinate

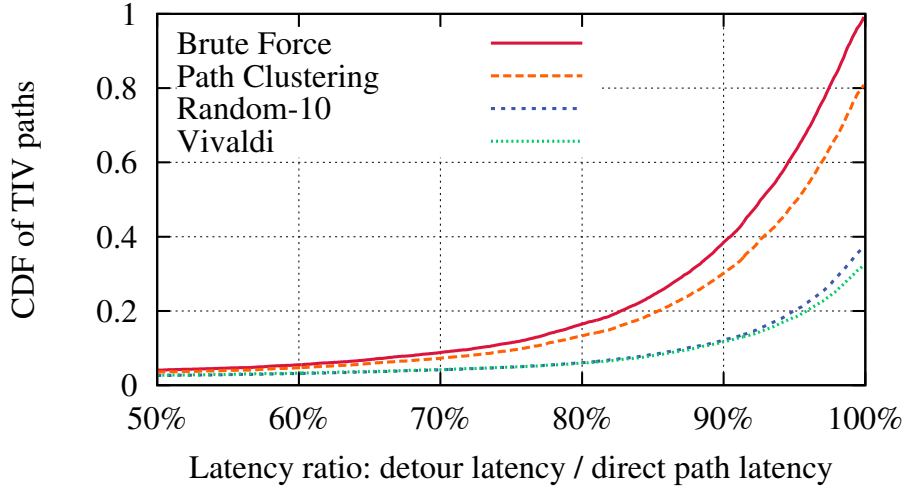


Figure 3.4: Path Clustering can obtain most of the benefits from detouring

system. Network coordinate systems can suggest potentially useful detours by picking a path with the shortest distance between two nodes which is not the regular path. More formally, for a given regular path (a, b) , we find a detour c where $\|\mathbf{x}_a - \mathbf{x}_c\| + \|\mathbf{x}_c - \mathbf{x}_b\|$ is the smallest. We implemented a network coordinate system using Vivaldi. We assume every path is a TIV path; that is every path has a detour. For fairness in comparison with our algorithm, vivaldi suggests two detours with the shortest distance for a given path, and pick the one that offers the most improvement.

Our clustering algorithm is able to provide benefits to a large proportion of paths. Our algorithm can improve 80.9% of all TIV paths. Our dataset also shows that 38.5% of the paths can benefit from 10% reduction in latency and our algorithm can find 78.4% of such paths. Unlike brute force which stores a detour per path (20 614 paths), our algorithm stores two detours per cluster (5357 clusters). By forming clusters, the detours discovered can be useful to other paths that have the same AS fingerprints.

random-10 and vivaldi improve 37.4% and 32.3% of all TIV paths respectively. Our result suggests that TIV paths cannot be accurately extracted from network coordinate systems. Due to the abundance of TIV paths, it causes significant distortion to the network coordinate system. Furthermore, a network coordinate system assumes the forward and

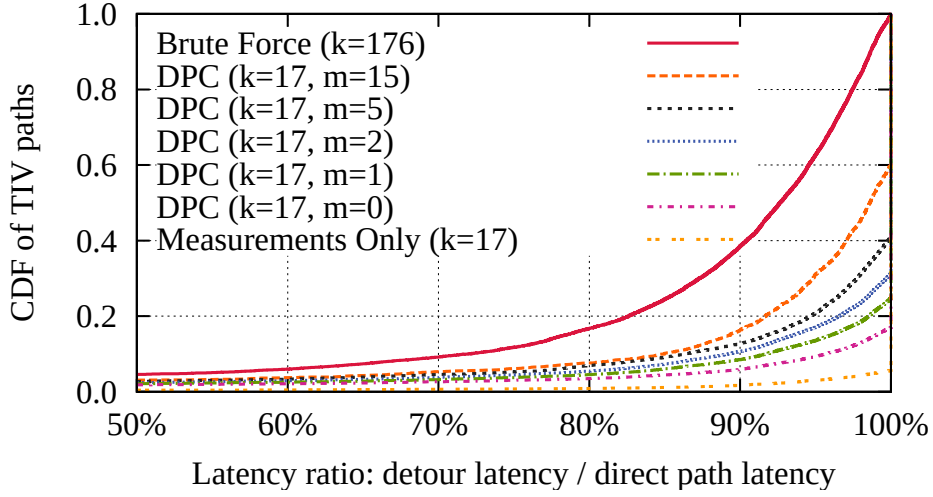


Figure 3.5: The quality of detours obtained from decentralised path clustering for different values of m

reverse path have the same latency which may not be true in the actual Internet.

3.4.2 Decentralised Detour Discovery

We investigate the quality of detours obtained from decentralising our path clustering algorithm described in §3.3. In this experiment, each node measures the latency to k random destinations and performs m gossip rounds to construct TIV clusters. We choose a small $k = 17$ which covers roughly 10% of all nodes to reduce the amount of measurements performed. Using this value of k each node, on average, finds 59 TIV paths and construct 32 TIV clusters prior to cluster exchanges.

Figure 3.5 compares the decentralised path clustering (DPC) algorithm when $k = 17$, brute force where every node considers all destinations (i.e. $k = 175$), and measurement only where we do not cluster paths at all. The graph shows that without clustering, around 5.7% of all possible detours can be found. After clustering but without exchanging clusters, we find 17.0% of all TIV paths. As DPC performs more cluster exchanges from none ($m = 0$) to 15, more TIV paths with better quality detours can be found but with diminishing returns.

3.4.3 Application of Path Clustering to other metrics

We also investigate the centralised path clustering algorithm for finding detours in two other metrics namely, loss and bandwidth.

Loss

First we define the loss equivalent variant of a TIV path. A path is considered to be a TIV path if the loss of the regular path $loss(a, b)$ is more than the detour path $loss(a, c, b)$, where $loss(a, c, b) = 1 - (1 - loss(a, c))(1 - loss(c, b))$, and can be expressed by the Inequality 3.4.

$$\frac{loss(a, c, b)}{loss(a, b)} < 1 \quad (3.4)$$

Apart from the definition of TIV path, path clustering is performed in the same way for loss as for latency. We construct TIV and no-TIV clusters using path clustering described in §3.2. We set the similarity threshold τ to 0.4.

In this experiment, we use the same 176 node dataset to evaluate path clustering on loss described earlier in §3.4. A path's loss is measured by the percentage of packet losses in the 1200 probes sent to the destination host at a rate of 11.8 kbps. Our dataset shows that most paths have no observable losses (72.5%). Most paths that have losses can benefit from detouring—5629 (or 27.3% of all paths) are TIV paths. The algorithm forms 2993 TIV clusters and 3415 no-TIV clusters which is significantly more than our latency dataset. This suggests loss is more heterogeneous in a network than latency.

Figure 3.6 shows the quality of detours obtained using our path clustering, brute force and random-10. It suggests that path clustering less effective in selecting a detour for loss than latency. Since most paths do not have losses, there are ample paths for detouring thus random-10 performs well compared to path clustering. random-10 can reduce losses in 93.9% of all paths compared to 83.3% in path clustering. This is significantly more than the number of clusters formed in latency. Unlike random-10 which probes 10 detour

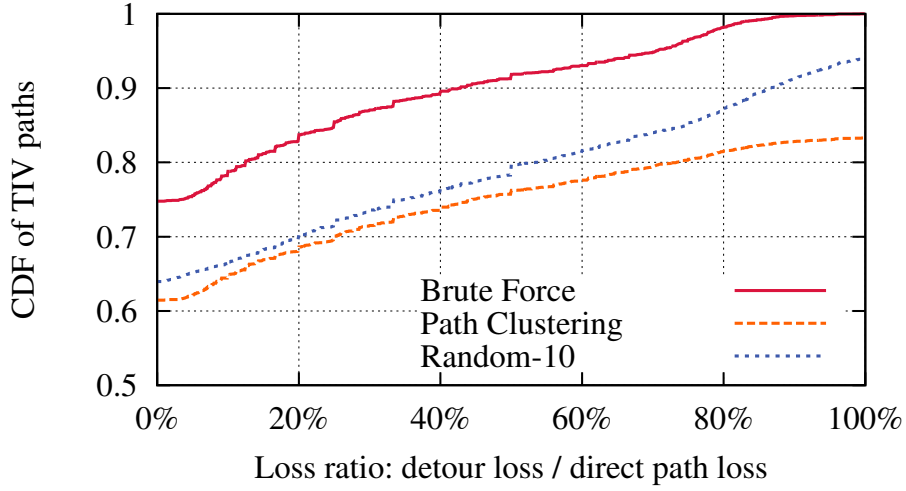


Figure 3.6: Comparison between path clustering with random-10 and brute force for loss

paths plus the regular path, path clustering only probes 3 paths including the regular path.

Bandwidth

To evaluate path clustering on bandwidth, we collected a separate dataset. On 30 August, 2010, we chose 85 PlanetLab nodes which have bandwidth caps of at least 100 Mbps. Between these nodes, we collected the AS paths and bandwidth measurements. Similar to the previous dataset, we collect AS paths by sending traceroute probes between all node pairs then translating the IP addresses using Team Cymru’s AS lookup service. The all-pairs bandwidth measurements are collected by running `iperf` for 5 s. The short run time for `iperf` prevents nodes from exceeding PlanetLab’s daily bandwidth limit. We insure that each node measures to at most one other node at any one time.

After collecting the dataset, we classify each path into TIV and no-TIV paths. We define a TIV path if the regular path’s bandwidth $bw(a, b)$ is smaller than both its detour path bandwidth $bw(a, c, b)$. The detour path’s bandwidth $bw(a, c, b)$ is assumed to be the minimum bandwidth between the source-to-detour $bw(a, c)$ and the detour-to-destination $bw(c, b)$. The Inequality (3.5) shows a TIV path $bw(a, b)$ which can benefit from detour c .

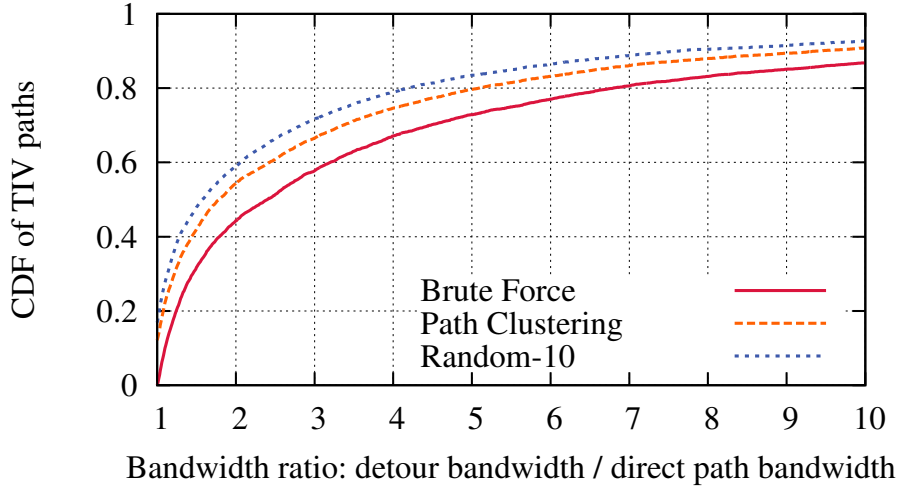


Figure 3.7: Comparison between path clustering with random-10 and brute force for bandwidth

$$\frac{bw(a, c, b)}{bw(a, b)} > 1 \quad (3.5)$$

After classifying TIV and no-TIV paths, we perform path clustering to obtain TIV and no-TIV clusters. Between the 85 PlanetLab nodes, we collected 6987 paths which have both bandwidth and AS paths measurements. In bandwidth, most of the paths contains beneficial detour—96.1% of the paths can benefit from detouring. There are 690 TIV clusters and 216 no-TIV clusters formed when $\tau = 0.4$.

We evaluate our path clustering method on bandwidth in the same way as loss and latency. In Figure 3.7, we show bandwidth can be increased by many folds by detouring—using brute force, we found that 55.7% of all paths with detours can double in bandwidth. Our path clustering approach can double the bandwidth for 45.8% of all TIV paths. Later, we discuss a different strategy for bandwidth detouring which can double the bandwidth for 43.2% of the paths without requiring all nodes to perform measurements.

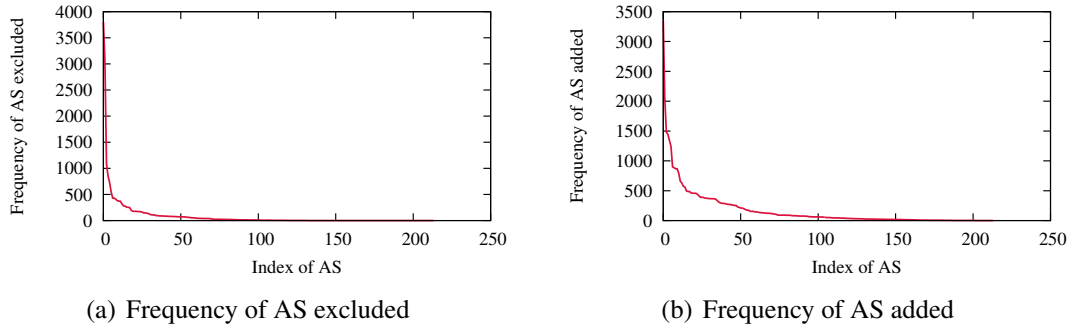


Figure 3.8: Frequency of each AS excluded or added due to detouring are skewed towards a small number of ASes

Excluded ASes		Added ASes	
AS Name	Frequency	AS Name	Frequency
Internet2	3806	Level3	3349
GEANT	3085	TeliaSonera	2063
Transpac	1085	Open Access	1482
PacificWave	860	AT&T	1445
LambdaRail	729	Cogent	1339

Table 3.2: Frequency of Top 5 ASes that are excluded or added due to detouring

3.5 Analysis of detoured AS paths on PlanetLab

In this section, we compare the regular AS paths and the detoured AS paths chosen by path clustering on PlanetLab. We study the effects of detouring on an AS path. First we present results on changes in AS path lengths due to detouring then we investigate the ASes that are excluded due to detouring and new ASes that are included during detouring.

Our findings show that detouring on PlanetLab typically increases the number of hops in an AS path. Using our centralised path clustering method and latency dataset described in § 3.4.1, we observe that on average a regular AS path length is 5.26 long and a detour AS path is 6.65 long. 78.1% of the detour AS paths are longer than their corresponding regular path, 8.0% of the detour paths are shorter and 13.9% are the same length. This is expected because our detour nodes are not situated in the core of the Internet thus, detouring will generally increase the length of an AS path.

Figure 3.8(a) shows the frequency of each AS that is excluded due to detouring decisions made by path clustering. The graph shows that ASes that are excluded from detouring are heavily skewed to a small number of ASes. The Top 22 ASes (10% of all 214 unique ASes) represent 75.9% of the AS that have been excluded.

Figure 3.8(b) shows the frequency of each AS that has been added due to detouring. The distribution of the added ASes are also skewed towards a small number of AS. The Top 22 ASes represent the frequency of 55.7% of newly added ASes due to detouring.

Previous work has shown that ASes are more likely to form agreements with an AS that is better connected to the rest of the Internet [26]. This preferential attachment explains why path clustering adds or omits the same few ASes which leads to a skewed distribution.

Table 3.2 shows that the Top 5 ASes used in detouring are different from those that are excluded. We observe that the Top 5 ASes that are excluded are different from research and educational network and the added ASes are more commercially orientated. Since most nodes on PlanetLab are from research networks, it is not surprising that the AS-path provided by regular Internet routing uses research networks. Regular Internet routing can only suggest one path, our path clustering algorithm can find alternative paths with better latencies that Internet routing cannot show.

Our implementation does not consider AS policies. Our clustering algorithm, which stores AS fingerprints, can incorporate AS-policies to become more ISP-friendly. For example, we can avoid suggesting detours for fingerprints that contain AS-links which are not advertised. Similarly, we can choose to only suggest detours which have an equal or lower transit costs than the regular AS-path where the difference in transit costs between the detour path and regular path can be found using Lumezanu et al.’s method [81]. We also discuss the implications of detouring have on ISPs in more detail at the end of the next chapter (in §5.6).

3.6 Summary

We have introduced a novel method for discovering latency detours that reduces the latency between two hosts. Unlike previous work, our method incorporates the AS-level topology of the Internet to search for detours. Our clustering algorithm groups paths into TIV and no-TIV clusters. The cluster of TIV paths can benefit from the same set of detours because they have common AS-links.

In our experiments, we showed that there is a substantial number of paths that can benefit from latency improvement through detouring—between 176 PlanetLab nodes, 69.2% of the regular paths can benefit from detouring. Using our path clustering algorithm, we were able to identify 94.3% of the TIV paths correctly and provide improvement to 85.3% of these paths.

We compared our method with those obtained through network coordinate system. By considering AS level information, our method can identify and provide more useful detours than metric-based systems. In contrary to network coordinate systems, our method can also be adapted to bandwidth and loss for discovering detours.

4 Reducing Bandwidth Measurement Overhead in Overlay Networks

TCP is the most dominant protocol in the Internet and is responsible for carrying over 80% of the Internet traffic [41, 60]. It is often used in large file transfers such as FTP and BitTorrent because of its automatic retransmission algorithm. Given a sufficiently large file, TCP achieves a long-term average transfer rate which is referred as the *steady-state* TCP bandwidth.

Measuring the steady-state TCP bandwidth of a path generates large overheads—often proportional to the bandwidth of the connection—by flooding the connection with data to the point of congestion. Our goal is to reduce the amount of traffic TCP generates when measuring a given network.

First, we investigate how long it takes for TCP to reach steady-state by conducting TCP bandwidth measurements for different durations (in §4.1). A shorter time reduces measurement overheads but may also cause TCP to fail to reach steady-state. In our study, we show that 5 s is sufficient for TCP to reach steady-state on most PlanetLab paths.

Next, we study the relations between *available bandwidth* and TCP bandwidth (in §4.2). While available bandwidth is the unused bandwidth in a path and is the maximum amount of bandwidth that can be sent without interfering with cross-traffic or exceeding the capacity of the path, TCP bandwidth is designed to occupy as much of the available

bandwidth as possible. Measuring available bandwidth instead of TCP bandwidth has the advantage that it generates substantially less traffic on the Internet.

Finally, we consider using bandwidth measurements to infer the properties of individual links in a network (in §4.3). The inferred link properties can then be used to estimate bandwidth of other unmeasured paths. In this approach, we achieve reduction in measurement overhead by measuring only a subset of the network.

4.1 Direct TCP Bandwidth Measurement

TCP Bandwidth can be measured directly using TCP benchmarking tools such as `iperf`. `iperf` establishes an end-to-end TCP connection then floods the connection with arbitrary data. TCP bandwidth is measured by the rate of data received at the receiver's end. An important parameter to set in `iperf` is the *duration* which denotes the length of time that `iperf` floods the connection for. In this section, we investigate the effects of this parameter on TCP bandwidth.

If the duration is too short, it would prevent TCP bandwidth from reaching steady-state and hence lead to an under-estimate of the actual steady-state bandwidth. However, long duration would consume excessive bandwidth resulting in large measurement overheads. We investigate how different periods have on the accuracy of `iperf` bandwidth measurements.

On 11 Oct, 2009, we ran `iperf` between 50 randomly chosen PlanetLab nodes with a duration of 3, 5 and 10 s. Figure 4.1 shows that a longer duration leads to a higher bandwidth estimate. As we increase the duration, our measurements converge towards the steady-state bandwidth but the convergence is slower as the duration becomes larger. For example, when we increase the duration by 2 s from 3 s to 5 s, the difference in the portion of paths that are over 10 Mbps is 4.5%, however increasing the duration by another 5 s only yields an extra 2.1% of the paths that are over 10 Mbps.

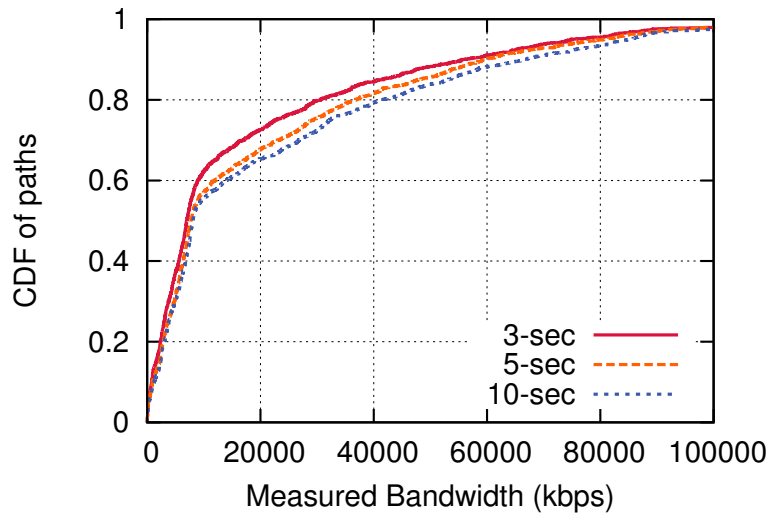


Figure 4.1: Increasing `iperf`'s duration leads to better bandwidth estimates but also increases measurement overhead

Though longer duration leads to more accurate estimation of the steady-state bandwidth, these also generate significantly larger amount of traffic. In our experiment, we found TCP bandwidth consumes on average 12.2 MB when the duration is 5 s but is increased by a factor of 2.17 when the duration is doubled. We found setting `iperf` to 5 s obtains a good balance between the amount of bandwidth consumed and the accuracy in measuring the steady-state bandwidth.

4.2 Relationship between TCP Bandwidth and Available Bandwidth

Available bandwidth is the unused bandwidth there is in a connection. TCP's *additive increase/multiplicative decrease* (AIMD) algorithm is closely related to available bandwidth. TCP first increases its rate linearly until a packet loss occurs which happens when all the available bandwidth has been used then multiplicatively reduces its rate of transfer in response to the loss. This process is repeated until the transmission terminates. Thus,

available bandwidth should be a good approximation for TCP bandwidth [58].

Available bandwidth estimation tools (ABets) measure the available bandwidth and we can use this as an estimate for the actual TCP bandwidth. Available bandwidth tools have two key advantages over direct TCP methods such as `iperf`: i) they are less intrusive (i.e. they are less likely to generate packet loss in competing traffic), and ii) require fewer probes to measure. We provide a description of how these tools work in the background chapter in §2.2.3.

There are a number of studies in the past which compare the publicly available ABets [30, 46, 47, 67, 72, 113, 125]. Shriram et al. [113] compare ABets on a high-speed 1 Gbps network with 4-hops whereas Lao et al. [72] perform their comparison in a low-speed 10 Mbps network. Goldoni et al. [46] evaluate a large number of ABets using a laboratory testbed. Labit et al. [67] compare ABets on real Internet paths in a French research network. Urvoy-Keller et al. [125] monitor three Internet paths across a European research network (GEANT) using ABets over 11 days. Guerrero and Labrador [47] study ABets based on several factors such as accuracy, reliability, measurement time, etc.. Lai et al. [70] compare available bandwidth tools on two selected paths on PlanetLab. Finally, Croce et al. [30] show that when using ABets to measure multiple paths concurrently, the accuracy of these tools decreases significantly and thus is not suitable in large networks.

While most of the above studies have been conducted in laboratory settings or measure available bandwidth through artificially generated cross-traffic, we focus on more realistic Internet paths across the globe by evaluating ABets on PlanetLab. Unlike Lai et al. [70]’s study, we use PlanetLab to test a substantially large number of paths, more than any comparison studies mentioned above. In our study, we select PlanetLab nodes from different sites which have bandwidth limits above 10 Mbps.

We compare TCP bandwidth measured using `iperf` with four publicly-available ABets. Based on the techniques these ABets used, we categorise the tools into either PRM or

PGM methods (described in §2.2.3). Two of our tools are PRM-based (`pathload` and `pathChirp`); one uses PGM (`spruce`); and the final one uses both (`igiptr`) which also produces two different available bandwidth estimates.

We evaluate ABets as follows. We generate four separate datasets corresponding to each tool. In each dataset, we first run `iperf` for 5 s to capture the TCP bandwidth then rest for 5 s before measuring the available bandwidth. To capture the variance of TCP bandwidth itself, we also compare the result with two consecutive `iperf` measurements. Since `spruce` requires the end-to-end capacity estimate, we use `pathrate` [33] to measure the capacity of each path before running `spruce`.

Note that `pathrate` in normal mode takes a very long time to measure per path (on average 12 mins). Instead, we use `pathrate` in quick mode and limit the measurement time to 30 mins per path when gathering capacity measurement. Despite these changes `pathrate` may still not return results thus the number of paths we can measure with `spruce` is significantly less.

We evaluate the ABets under three criteria: i) *relation with TCP bandwidth*, where we compare the available bandwidth estimates of each tool with TCP bandwidth measurements obtained through `iperf`, ii) *measurement speed*, where we evaluate the time it takes to obtain a measurement, ii) and *intrusiveness* where we compare the rate at which the ABets send their probes.

Relation with TCP bandwidth. Figure 4.2 shows the distribution of TCP bandwidth against the available bandwidth measured by each of the four ABets. We observe that there is a significant difference between TCP bandwidth and the measured available bandwidth. Our `iperf` measurements are substantially lower than available bandwidth measurements gathered by any of the ABets. For example, `iperf` measurements show that roughly 65% of the paths are below 40 Mbps but the ABets suggest only 20% of the paths are below 40 Mbps.

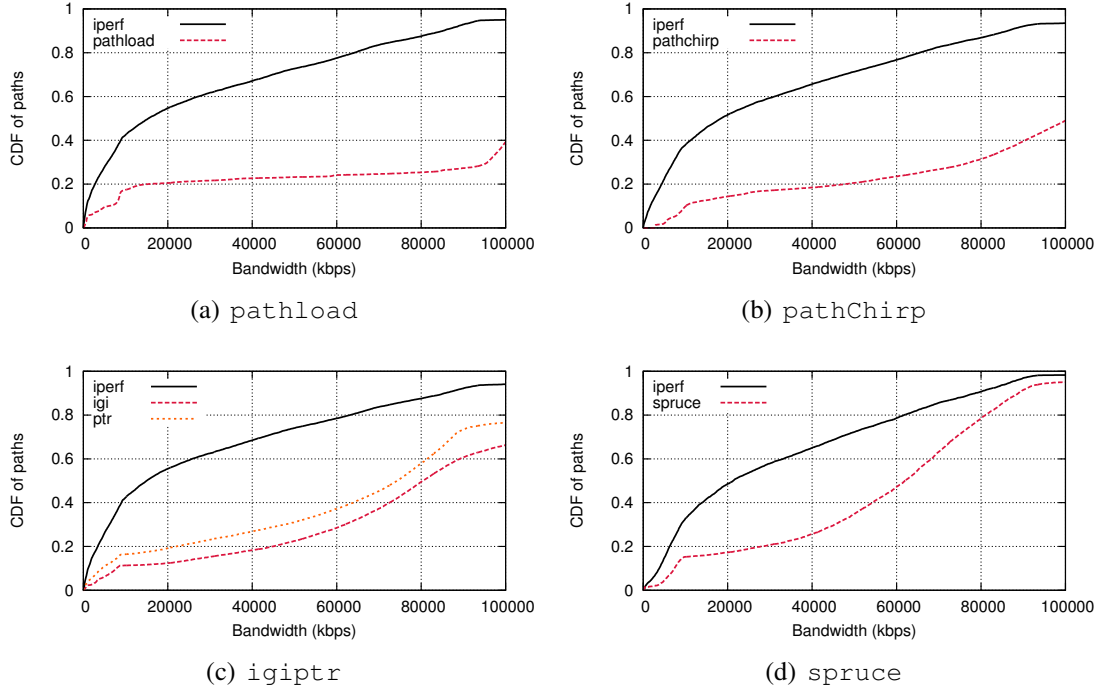


Figure 4.2: Comparison of TCP bandwidth and available bandwidth for different ABets

Excluding the measurements from `spruce` which are not representative of PlanetLab’s bandwidth distribution, the ABets suggest that between 12% to 20% of the PlanetLab paths are bottlenecked at 10 Mbps, but our `iperf` measurements suggest that approximately 40% of the paths are bottlenecked at 10 Mbps. We believe that this is caused by *traffic shaping* in the Internet. Traffic shaping artificially limits a connection to a *shaping-rate* which is below the capacity of the connection (in this case, limited to 10 Mbps). Traffic shaping is typically implemented using a token-bucket algorithm which allows small bursts of packets to momentarily exceed the shaping-rate. While `iperf` is limited by the shaping-rate as it continuously sends large amount of data, ABets transmit in bursts small enough such that it can exceed the shaping-rate and therefore detects the available bandwidth greater than the shaping-rate. A recent measurement tool, *ShaperProbe* [32], exploits this fact to detect traffic shaping on the Internet. *ShaperProbe* floods the connection at the rate of the path’s capacity long enough such that the traffic is

Date	Tool name	Comparable paths	Number of nodes	Average send rate	Average run-time (secs)
11 Feb, 2011	pathChirp	13607	125	750 kbps	33.77
9 Feb, 2011	igiptr	20025	160	338 kbps	11.01
3 Feb, 2012	pathload	20803	158	2.60 Mbps	55.03
3 Feb, 2012	iperf	20803	158	22.1 Mbps	7.97
6 Feb, 2012	pathrate	13134	118	54 kbps	72.13
7 Feb, 2012	spruce	6399	118	156 kbps	14.9

Table 4.1: List of datasets for comparing available bandwidth tools

shaped. The receiver will observe a permanent decrease in the receiving rate if the traffic is shaped.

Though our goal is to investigate whether there is an ABets that is suitable for estimating TCP bandwidth, we observe that the available bandwidth measurements for the ABets often disagree with each other. Figure 4.2 also shows that the distributions between the ABets are different. Even tools that use the same techniques such as the PRM tools (pathload, pathChirp and ptr) have very different available bandwidth distributions. Previous work has suggested two potential problems with ABets: i) measuring available bandwidth requires precise packet timings that is difficult to achieve on the publicly shared PlanetLab machines (for example measuring a 100 Mbps with 1500 byte packet requires timestamps of at least $120 \mu s$ precision) [68]; ii) network cards may coalesce multiple packets that it receives together to reduce load which leads to erroneous timestamps [102]. Jain et al. [59] also pointed out ten difficulties in designing ABets. We believe a more careful and in-depth evaluation is necessary to obtain a tool that maybe suitable for the Internet.

Intrusiveness. We assess how intrusive a tool is by the average bit rate the tool sends its traffic at. The available bandwidth tools are designed to avoid triggering losses thus should have a low bit rate compared to `iperf`. In Table 4.1, we see this is indeed the case and tools probe available bandwidth at a much lower rate than `iperf`. `spruce` is

the least intrusive in terms of average rate at which packets are sent, because `spruce` limits its probe rate to the minimum of 240 kbps and 5% of the measured capacity.

Measurement speed. Measurement time is particularly important when measuring a large number of paths. Quicker measurements can imply that measurements can be made more frequently. Table 4.1 shows that the slowest tool is `pathload` which on average takes 55 s to complete a measurement. In contrast, `igiptr` takes the shortest time to measure available bandwidth (11 s). However, the runtime for this tool is still 38% longer than `iperf`. Like `iperf`, measuring multiple paths by running the ABets concurrently will cause interference which will lead to an estimate lower than the actual available bandwidth [30].

To some extent, the runtime for most of these tools can be controlled. Both `pathload` and `igiptr` terminate when a pre-defined error tolerance has been reached. Thus, accepting a larger error tolerance would make the tools run more quickly. In `spruce` and `pathChirp`, the runtime can be controlled more directly—both of these tools stop after a pre-set time or a number of packet trains have been sent.

In summary, we observe that the available bandwidth measured by the ABets we used and TCP bandwidth measured by `iperf` can be very different on PlanetLab paths. We show that the available bandwidth estimates from different ABets often disagree with each other. These tools are designed to reduce intrusiveness and measurement times when measuring available bandwidth but are often tested in laboratory settings or with artificial traffic generators which are not realistic to the Internet. We have to evaluate ABets in more diverse Internet environments with knowledge of *ground truth* (i.e. the actual available bandwidth), before we can fully understand the relationship between available bandwidth and TCP bandwidth.

4.3 tcpChiryo: TCP bandwidth inference using loss tomography

In the above sections, we discussed two approaches for measuring end-to-end TCP bandwidth: direct TCP measurement and estimation from available bandwidth. These methods have their shortcomings—direct TCP measurement consumes a significant amount of bandwidth and thus is impractical at scale, and the relationship between available bandwidth estimation tools and TCP bandwidth is unclear. An alternative approach is to apply *network tomography* which aims to infer link properties from end-to-end measurements. The link properties can then be used to estimate other unmeasured paths.

We cannot apply network tomography directly on TCP bandwidth because it is an end-to-end path property and cannot be split to individual links. Instead, we decompose TCP bandwidth into two components: loss and latency [88, 51]. Since a path’s loss rate can be separated into its link’s loss rate, we can use network tomography to infer the link’s loss rates from the path’s. We measure latency directly because the measurement overhead is small compared to bandwidth.

We introduce `tcpChiryo`¹ which uses network tomography to infer TCP bandwidth of unmeasured paths. `tcpChiryo` first collects latency and the topology of the network and also measures the TCP bandwidth of a subset of the network. Using the TCP bandwidth and latency measurements, we can obtain the path’s loss rates of a subset of paths where we use network tomography to obtain the path’s loss rates of the unmeasured paths. Finally, using the inferred loss rates and measured latency, `tcpChiryo` obtains estimates of TCP bandwidth for unmeasured paths in a network. Our method never measures loss rates directly because loss rates are too small to be measured on high-speed paths.

`tcpChiryo` reduces the overhead by avoiding measuring all-pairs TCP bandwidth which generates significant amount of traffic (as shown in §4.1). `tcpChiryo` can also

¹`tcpChiryo` means TCP estimation in japanese romanji

extract link’s loss from Internet paths.

`tcpChiryo` relies on network tomographical techniques to infer the loss rates of individual links. Network tomography is first coined by Vardi [127] and refers to the use of end-to-end observations to infer properties of a network such as link’s latency, link’s loss, or the topology of the network. Castro et al. [25] provide an excellent summary of work in network tomography prior to 2004.

Loss tomography is a branch in network tomography which aims to infer the loss rate of links. Loss tomography was initially investigated on tree networks, i.e. from a single source or receiver’s perspective. Bu et al. [20] show that the loss of link’s from a single source can be inferred using expectation maximisation (EM) methods. Duffield et al. [35] extend the algorithm to use multicast probes which reduces measurement overhead. Duffield et al. [36] also provide an alternative solution which uses back-to-back packets for networks that do not support multicast. Xi et al. [131] provide a more detailed analysis of the EM approach in tree topologies.

As an alternative to EM methods, link’s loss can be inferred using *linear programming* (LP) [43, 97, 137]. Padmanabhan et al. [97] analyse link losses from a server perspective with an interest to identify congested links (i.e. loss rates greater than 1%). Zhang et al. [137] compare the inference accuracy of several LP models in the Internet core. Ghita et al. [43] successfully apply LP to infer link’s loss from multiple sources to multiple receivers.

Our work uses loss tomography to infer TCP bandwidth rather than loss and do not measure loss directly and measure TCP bandwidth and latency to find the loss rate of a connection. As far as we know, there has not been a known method of using network tomography to infer TCP bandwidth. We first describe how `tcpChiryo` is implemented (in §4.3.1). We evaluate our implementation by comparing `tcpChiryo` with a recent loss tomography tool, `netscope` (in §4.3.2). We also provide an analysis of the link’s

loss that is inferred using `tcpChiryo` (in §4.3.3).

4.3.1 Methodology

`tcpChiryo` first gathers topology and latency measurements for all nodes in a network. Both of these information can be obtained quickly and cheaply using `traceroute`. `tcpChiryo` then clusters IP addresses together to reduce the size of the topology and increase the number of shared links in the network. Next, `tcpChiryo` measures the TCP bandwidth of a small subset of paths in the network then decomposes the measurements into path's loss. The path's loss and topology is fed into an EM-based algorithm to generate a link-loss mapping. Using this mapping, we infer the bandwidth of the other paths. In the following section, we explain each step in detail.

Clustering IPs. In `traceroute` measurements, packets can arrive at the same router via different physical network ports. This causes `traceroute` to report multiple IP addresses for the same router because each port has a different IP address [15]. These addresses would have the same network properties thus can be clustered together. Clustering IPs reduces the size of the network topology which offers two benefits: i) it makes the network more scalable, and ii) it increases the number of shared links in the network which provides links that are more likely to be useful to other paths.

The IP address of similar devices can be clustered by their origin AS and their geographical location. The AS can be found from public whois servers, BGP looking glass servers, or regional Internet registry (RIR). `tcpChiryo` uses Team Cymru's whois lookup service which collects their IP to AS mapping from over 50 BGP peers. Occasionally, some IP addresses do not reach any Team Cymru's BGP peers and therefore do not appear in Team Cymru's listing. In that case, we use the RIRs database.

The geographical location can be found using `undns` from the Rocketfuel's project [116]. Rocketfuel resolves bandwidth measurements from the hostname of the IPs. It takes ad-

vantage of clues left in hostnames to reveal geographical locations. For example, Cogent encodes the location of their major routers into hostnames using the three letter airport code (e.g. ‘`lax`’ represents a router near Los Angeles). We added our own lookup rules to improve the number of geographical locations found using this method. In `tcpChiryo`, IP addresses that cannot be resolved to their geographical locations are not clustered.

Path Selection. By making conscious selection of paths to measure, we can estimate paths more accurately with less measurements. Our path selection strategy selects paths whose link has been measured the least. In each path selection iteration, we first compute the number of times each link has been measured by end-to-end path measurements. We find the set of links that have been measured least and randomly select a path which involves one of the links in this set.

Decomposing bandwidth measurements. We can make better inference of link properties if we remove path-dependent properties such as end-to-end latency from our bandwidth measurements. Mathis et al. [88] describe how TCP bandwidth can be broken down into latency and loss rates. They observe that TCP bandwidth in steady-state is inversely correlated to the product of latency RTT and the square-root of path’s loss. The formula is based on the TCP Reno variant of the TCP stack and is described as follows,

$$BW = \frac{MSS\Phi_{reno}}{RTT\sqrt{p}}, \quad (4.1)$$

where Φ_{reno} is a constant between 0.93 and 1.22 depending on how the acknowledgements in TCP are handled. We do not measure loss rate directly because it depends on the rate at which probes are sent.

Equation (4.2) describes a different variant of TCP stack—TCP CUBIC. This variant is implemented as the default variant since Linux 2.6.18 and is the TCP stack that `tcpChiryo` uses. Based on the expected congestion window size described by Ha et al. [51],

we derive this equation which describes the steady-state bandwidth for TCP CUBIC.

$$BW = \frac{MSS}{\sqrt[4]{RTT}} \frac{\Phi_{cubic}}{\sqrt[4]{p^3}} \quad (4.2)$$

In the above equation, Ha et al. [51] has computed that Φ_{cubic} is 1.17. We assume MSS, which is the amount of data carried in one packet, is 1460 bytes which is a typical value for most Internet paths.

Both equations assume two conditions to be true: i) the maximum congestion window size is never reached, ii) and TCP retransmission timeout events rarely occur. The former is generally true in modern operating systems which automatically adjusts their buffer size. The later happens when there is a large variation in latency which is rare in wired connections.

Comparing the two equations, we see that the bandwidth of a TCP CUBIC variant is less sensitive to changes in latency because of the square of square root relationship. This meant TCP CUBIC will have a larger throughput in paths with long latencies than TCP RENO does.

Creating link-to-loss mapping. We infer our end-to-end path loss p to link's loss l using EM methods. EM is an iterative method for estimating parameters based on a set of observations. It consists of two steps: 1) *expectation step* which computes the estimates based on the current parameters, 2) *maximisation step* which updates the parameters by comparing the estimates with the set of observations.

We consider two different EM algorithms in our implementation which differ from the loss model used to compute the path's loss p from link's loss l . The first model, which we refer as the *Bernoulli model*, assumes that loss is a Bernoulli process thus each link's loss is independent of each other. The path's loss p_i on path i and its set of links $\mathcal{L}_i$ can

Algorithm 1: Iterative update rule for $\hat{l}$ using the Bernoulli model

```

1 foreach Path  $i$  do
2    $\hat{p}_i \leftarrow \left[ 1 - \prod_{j \in \mathcal{L}_i} (1 - \hat{l}_j) \right]$ 
3    $\epsilon \leftarrow \hat{p}_i / p_i$ 
4   foreach Link  $j$  in  $\mathcal{L}_i$  do
5     Store  $\epsilon$  in  $\mathcal{E}_j$ 
6 foreach Link  $j$  do
7    $\hat{l}_j \leftarrow \text{Average}(\mathcal{E}_j \hat{l}_j)$ 

```

be computed as,

$$p_i = 1 - \prod_{j \in \mathcal{L}_i} (1 - l_j) \quad (4.3)$$

However, loss rates are not necessarily independent of each other—packet losses earlier in the path will reduce the rate of arrival for later links and thus packets are less likely to be dropped at later hops in a path. Hence, we propose an alternative model, *maximum model*, which assumes the link with the largest loss rate can sufficiently approximate the path’s loss. We often refer the link with the largest loss rate as the *tight link* or the *bottleneck link*. Equation (4.4) describes the relationship between path’s loss p_i and its set of links $\mathcal{L}_i$ using the maximum model.

$$p_i = \max \{j \in \mathcal{L}_i : l_j\} \quad (4.4)$$

Based on these two models, we derive our EM algorithms to infer link’s losses. We denote `tcpChiryo` that is based on the Bernoulli model and maximum model to be `tcpChiryo-bern` and `tcpChiryo-max` respectively.

For a given link j in the set of links $\mathcal{L}_i$, we denote the inferred link’s loss as $\hat{l}_j$. We initialise the link $\hat{l}_j$ to the maximum path’s loss that uses link j . The purpose of initialising the values is to reduce the number of iterations for the algorithms and provide sensible starting points.

Algorithm 2: Iterative update rule for $\hat{l}$ using the maximum model

```

1 foreach Path  $i$  do
2    $b \leftarrow \arg \max \left\{ j \mid \forall \mathcal{L}_i : \hat{l}_j \right\}$ 
3   Store  $p_i$  in  $\mathcal{E}_b$ 
4 foreach Link  $j$  do
5    $\hat{l}_j \leftarrow \text{Average}(\mathcal{E}_b)$ 

```

Algorithm 1 represents the EM algorithm used by `tcpChiryo-bern`. Given measured path i , Algorithm 1 first computes the *inferred path's loss rate* $\hat{p}_i$ using Equation (4.3); that is the expectation step (line 2). In our maximisation step, we compute the errors between the inferred and measured path's loss rates then we update our link's loss rate $\hat{l}_j$ based on the errors between these measurements. In more detail, we compute the *relative error* ϵ defined as the ratio between the inferred path's loss rate $\hat{p}_i$ with the actual path's loss rate p_i (line 2–3). Next, we store this error temporarily in a list $\mathcal{E}_j$ corresponding to each link j that path i used (line 4–5). We repeat this process for all measured paths, storing the error ϵ into $\mathcal{E}_j$ for each measured path. Finally, we update our inferred link loss $\hat{l}_j$ by taking product between the previous link loss value and the average relative error in $\mathcal{E}_j$ (line 6–7).

Algorithm 2 shows how `tcpChiryo-max` operates. Given measured path i , the bottleneck link b is identified. Next, `tcpChiryo-max` stores p_i into a list $\mathcal{E}_b$ associated with the bottleneck link. Note that if there are ties when searching for the bottleneck link, `tcpChiryo-max` stores p_i in $\mathcal{E}_b$ associated to both links. Once this process has been repeated for all the measured paths, the link's loss is updated by the average of the list $\mathcal{E}_b$.

We found both algorithm to stabilise quickly and can be terminated after 10 iterations.

4.3.2 Evaluation

In this section, we compare `tcpChiryo-bern` and `tcpChiryo-max` with `netscope` [43] which is a recent loss tomography algorithm, and `min-bw` which uses a standard EM algorithm to infer bandwidth directly. We first evaluate the effectiveness of our path selection strategy (described in §4.3.1) by comparing our strategy with two different random-based path selection strategies. Next, we show that `tcpChiryo` can infer bandwidth more accurately than the other two algorithms. Finally, we compare the time it takes for each algorithm to create the link-to-loss mapping for a given set of measured paths.

`netscope` is a loss tomography tool that uses a set of measured path's loss p_i and traceroute measurements to infer the link's loss. `netscope` uses Gauss-Jordan elimination to remove links and approximate them with 0 loss rate. To select the appropriate links to remove, `netscope` has to re-order the paths in descending order of loss rates. After the elimination process, `netscope` uses LP to find the link-to-loss mapping. LP is implemented using L1-norm minimisation with an additional constraint which limits the link's loss rates to be positive [115]. `netscope` is shown to be more accurate than just applying L1-norm minimisation with positive constraints alone.

Our implementation of the `min-bw` is the same as our Algorithm 2 except in line (2) where the measured path's loss p_i is replaced by the measured bandwidth of the path and we replace $\arg \min$ with $\arg \max$.

We use one dataset to evaluate our algorithms. We collect an all-to-all set of bandwidth measurements and `traceroute` between 85 Planet-Lab nodes on 30 August, 2011. We use PlanetLab nodes that are from different sites and have no bandwidth limits. For our end-to-end latency measurements, we use the median of three probes. We use `iperf` to collect TCP bandwidth measurements for all-pairs of paths. Our final dataset is 91.7% complete and contains 6546 paths.

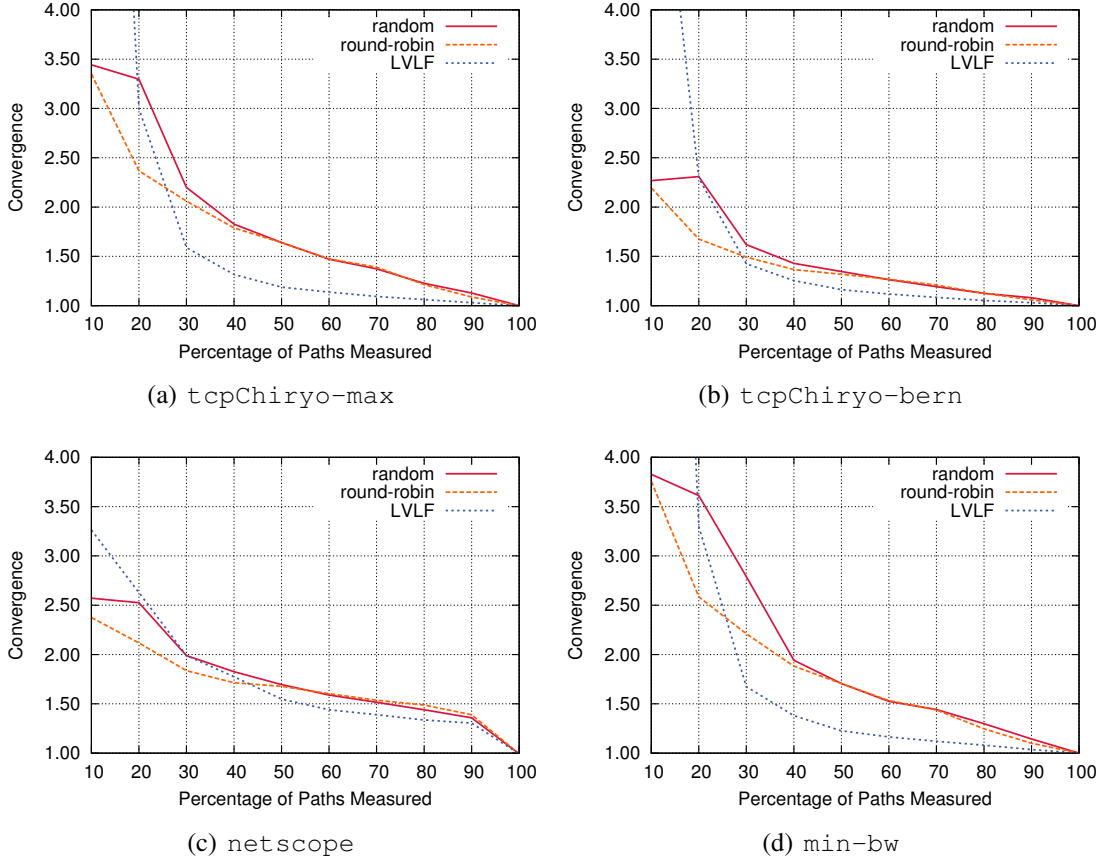


Figure 4.3: Our LVLF path selection strategy is better than random-based strategies after measuring roughly 30% of the paths

Path Selection. We compare our path selection strategy, denoted as least visited link first (LVLF) and described in §4.3.1, with two random-based path selection strategy: i) random which uniformly at random selects a path to measure, and ii) round-robin which picks each source in a round-robin fashion and selects a random destination to measure. We evaluate the three strategies by comparing how fast each path selection strategy converges towards the link-to-loss mapping when all paths are measured. We define *convergence* as the average ratio between the inferred link’s loss rates at the different stages during path selection and the inferred link’s loss rates when all paths are measured.

Figure 4.3 shows the convergence of each algorithm using each path selection strategy between 10% to 100% of all paths at 10% intervals. The experiment is repeated 10 times

for consistency. Figure 4.3(a), 4.3(b), and 4.3(d) show that the three EM algorithms and LVLf are a poor estimate when very few paths are measured, but become better than both random and round-robin when more than 30% of the paths are measured. We also found that LVLf would have measured each link at least once in its path selection after selecting approximately 30% of all paths, so measuring additional paths is used to refine the link’s estimate.

On the contrary, using random and round-robin strategies do not effectively cover all the links at early stages in its path selection process which leads to slower rate of convergence. This is particularly important for algorithms such as `tcpChiryo-max` and `min-bw`, which are sensitive to a single bottleneck link in the path (i.e. the link with the largest loss rate or lowest bandwidth in a path respectively).

In `netscope`, the path selection strategies have insignificant impact on the convergence rate. This is due to the `netscope`’s link elimination process which strips most benefits from our path selection process.

We compare the inference accuracy of paths by the *average bandwidth factor error*. We define the factor error as the ratio of a path’s estimated bandwidth and the path’s actual bandwidth. The factor error is a logarithmic error that is strictly positive—and similar to decibels, thus which leads to our average factor error defined as the following:

$$10^{|\log_{10}(BW_{est}/BW_{actual})|} \quad (4.5)$$

Accuracy.

In Figure 4.4, we use LVLf to select from 10% to 90% of all paths at 10% increments of the paths to measure and show the average bandwidth factor errors when estimating the unmeasured paths using each algorithm. We observe both versions of `tcpChiryo` perform better than the other two algorithms (`min-bw` and `netscope`)—when measuring 30% of all paths, both versions of `tcpChiryo` are roughly 9% and 8% better

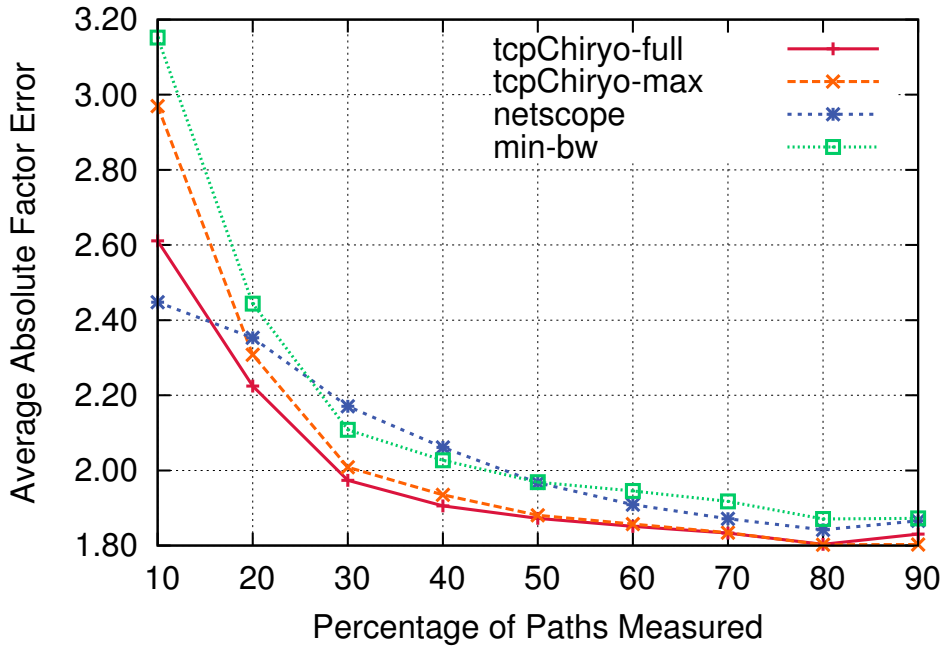


Figure 4.4: Both tcpChiryo algorithms are more accurate than `netscope`

than `netscope` and `min-bw` respectively. Unlike `min-bw`, our two `tcpChiryo` algorithms remove the latency component from bandwidth before applying EM methods which make them more accurate. `tcpChiryo-bern` performs slightly better than `tcpChiryo-max` because of a more accurate model of loss. As expected, beyond 30% of the paths measured, our three EM algorithms (both `tcpChiryo` and `min-bw`) suffer from diminishing returns because LVLf has measured to each link at least once (shown earlier in Figure 4.3).

Training time. We denote training times as the time it takes to compute the link-to-loss mapping from a given set of measured paths. We assess the scalability of `tcpChiryo`, `netscope` and `min-bw` with respect to training time. We also compare our EM-based algorithms with standard LP algorithm, by dividing `netscope`'s training time into two phase: i) the time to compute Gauss-Jordan elimination, ii) and the time to solve the linear problem by LP. The Gauss-Jordan elimination is computed using partial pivoting written in `perl` and LP is performed using the publicly available `lp_solve-5.5` library [8].

Pct of Paths	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
min-bw	0.55	0.76	0.99	1.19	1.41	1.63	1.84	2.06	2.28	2.48
tcpChiryo-max	0.72	1.04	1.36	1.68	1.98	2.26	2.57	2.90	3.21	3.47
tcpChiryo-bern	1.86	3.35	4.86	6.33	7.80	9.50	11.09	12.42	14.26	15.43
netscope (LP)	1.28	6.49	15.24	27.57	44.83	63.93	89.60	118.00	151.87	193.20
netscope (GJ)	110.92	424.91	903.72	1505.94	2256.63	3100.95	4044.01	5179.62	6278.45	7416.63

Table 4.2: The EM-based approach has shorter training time than `netscope`

All algorithms are tested on a quad-core 2.4 GHz machine with 4 GB RAM.

In Table 4.2, we show the training times with 10% to 100% of the paths selected using random path selection strategy from our 85-node dataset. The three EM-based algorithms have very low training times. When finding the link-to-loss mapping with 50% of all path, `netscope` requires 38 mins to complete, and the most expensive EM algorithm, `tcpChiryo-bern` requires only 7.8 s. Another advantage to EM approaches is that training time grows at a linear rate relative to the number of paths measured thus we can predict the time it will take to train a given set of paths. We compute that `min-bw`, `tcpChiryo-max` and `tcpChiryo-bern` require 0.38 ms, 0.53 ms, and 2.36 ms to train each path respectively.

Computing the Gauss-Jordan elimination for our 6546 by 3509 matrix requires over 2 hrs. It may be possible that the process can be speeded up by taking advantage of the fact that the matrix is sparse.

We also compare the time it takes to solve the LP problem with EM-based algorithms. The LP problem given to `netscope` has less links because of the Gauss-Jordan elimination. This is quicker to solve than a LP problem without the elimination process because the extra links add more constraints to the problem. Yet, we find the EM-based algorithms to perform faster than LP-based algorithm—with 50% of all paths, while `netscope` takes 44.83 s to create the link-to-loss mapping, `tcpChiryo-bern` only requires 7.8 s.

4.3.3 Analysis of Inferred Bottleneck Links

A *bottleneck link* is the link with the highest loss rate in a path and is the link that would influence the bandwidth of the path the most. Studying the bottleneck links can help identify where congestion is most common. Network operators can then devise mechanisms to reduce congestion in those part of the network (e.g. by re-routing or deploying web-cache).

We find bottleneck links by creating a link-to-loss mapping using the entire dataset. Not all links in the mapping are bottleneck links, we include links that have the highest loss rate for at least one path. In this section, we first show that bottleneck link loss rates obtained by `tcpChiryo-bern` and `tcpChiryo-max` have a strong correlation. Next, we describe the distribution of bottleneck links on the Internet.

Previous work on loss tomography [36, 97, 94] is restricted to measuring link’s loss rates that are above 0.01 due to the amount of time it takes to probe smaller loss rates. By decomposing our bandwidth measurements, we can infer loss rates which are much smaller—for example, 91.9% and 86.8% of the bottleneck links have loss rates of less than 0.01 in `tcpChiryo-bern` and `tcpChiryo-max` respectively.

We compare the inferred loss rates of the bottleneck links from `tcpChiryo-bern` and `tcpChiryo-max`. We find that the bottleneck links chosen by the two algorithms are similar. In `tcpChiryo-bern` and `tcpChiryo-max`, there are 1036 and 1189 bottleneck links out of 3509 links. Though only 712 bottleneck links are found by both algorithms, the ones that are not found have very small loss rates—for example, in `tcpChiryo-bern`, the median bottleneck link loss rate is 1.49×10^{-4} but the bottleneck links that appeared in `tcpChiryo-bern` and not `tcpChiryo-max` have a median loss rate of 3.46×10^{-5} .

Figure 4.5 depicts a scatter plot of the loss rates for the 712 bottleneck links inferred by the two algorithms. The green dashed line shows when the loss rates are equal. We

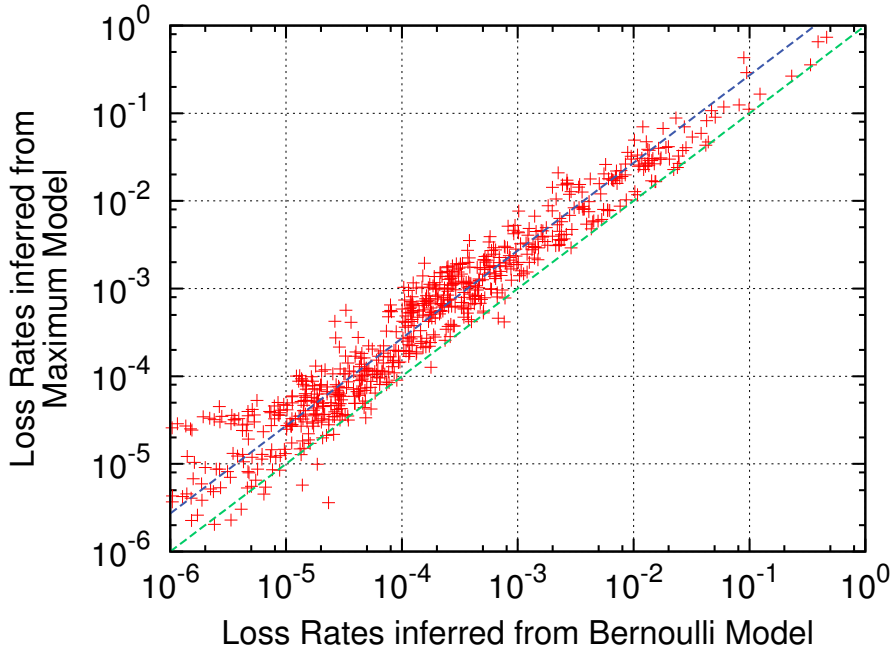


Figure 4.5: Comparison between the bottleneck link loss rates inferred by `tcpChiryo-max` and `tcpChiryo-bern`

see there is a strong correlation between the loss rates between the two algorithms. We compute the Pearson's correlation coefficient and the Spearman's rank correlation which are 0.95 is 0.96 respectively. These correlation values imply that bottleneck links preserve the same order in terms of their loss rates derived from either algorithms.

Figure 4.5 also shows that the loss rates from the Bernoulli model are less than the maximum model, but moreover, the loss rates are smaller by a constant factor. The blue dashed line shows the best fit between the bottleneck link loss rates of the two algorithms, we find that the bottleneck link loss rates obtained from `tcpChiryo-bern` is roughly 37% of `tcpChiryo-max`. Since the maximum model bases the entire path's loss on a single link whereas the Bernoulli model is a stochastic model involving all links, this suggests that a bottleneck link accounts for 37% of the losses in a path.

Next, we investigate the location of the bottleneck link within a given path using `tcpChiryo-bern`. Figure 4.6 shows the frequency distribution of the number of AS

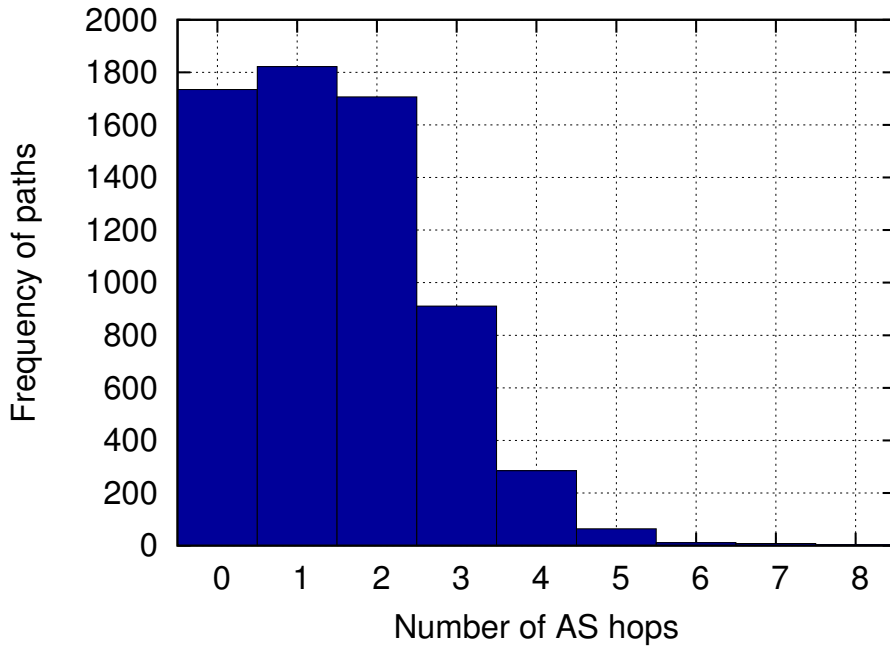


Figure 4.6: Distribution of bottleneck links are skewed towards the first few AS hops

Algorithm	Inter-AS	Intra-AS	Unknown
tcpChiryo-max	46.5% (553)	48.6% (578)	4.9% (58)
tcpChiryo-bern	45.5% (472)	50.4% (522)	4.1% (42)
min-bw	46.9% (568)	48.5% (587)	4.6% (56)
netscope	39.2% (203)	58.1% (301)	2.7% (14)
all links	37.8% (1328)	58.8% (2062)	3.4% (119)

Table 4.3: Comparison of the types of bottleneck links found using each algorithm with all links

hops taken before reaching the bottleneck link for all paths. We observe that bottleneck links are often close to the source. In particular, most bottleneck links are located at the first two AS hops of the path—30.5% of the path’s bottleneck link is within the first AS hop and 54.7% is within the first-two AS-hops.

We investigate whether bottleneck links are likely to be within an AS (intra-AS) or between the boundaries of two ASes (inter-AS). Distinguishing these bottlenecks is important because solutions to reducing congestion in the two types of links are different. For example, while intra-AS congestion can be solved by re-routing within the network,

inter-AS congestion may require renegotiations in AS agreements. Table 4.3 shows the type of links found in our dataset (all links) and the type of bottleneck links found by each algorithm. Unknowns are due to errors in resolving IP addresses to their corresponding AS. While we note that there are more intra-AS links than inter-AS links for all links or any algorithm, the difference between the two groups is significantly smaller for our EM algorithm—in all links there are 37.8% intra-AS links but in our EM algorithms we found approximately 46% are bottleneck links. This shows that inter-AS links are *more likely* to be bottleneck links than intra-AS links, but there are more intra-AS links in the Internet.

4.4 Summary

In this chapter, we seek to reduce the measurement overhead when measuring steady-state TCP bandwidth. We first discuss end-to-end measurement approaches where we found direct measurements generate a significant amount of traffic (on average 12.2 MB per path on PlanetLab) and available bandwidth estimation tools do not accurately represent TCP bandwidth. Next, we demonstrate that `tcpChiryo` can estimate TCP bandwidth by measuring selected paths in the network. We show that measuring 30% of the network provide the best tradeoff between accuracy and number of paths measured. By inferring link properties within the network, `tcpChiryo` also reveals the loss rates of individual links in the network.

5 Feasibility of Bandwidth Detouring

Detour routing provides alternative paths for user applications to choose that regular Internet routing cannot provide. Choosing a beneficial detour route can improve latency, availability, bandwidth, or loss of end-to-end connection.

Previous work has focused on improving detouring on latency [107] and reliability [13]. One key potential benefit of detouring is in detouring for bandwidth. Bandwidth is important in many Internet applications. For example, HD video-on-demand, video conferencing, or remote medical assistance, all require a substantial amount of bandwidth to work. Furthermore, with the emergence of cloud data centres, more users and businesses are storing content on the cloud using applications such as Dropbox.

Current solutions to increase bandwidth are: i) increase the capacity of the current network, ii) or limit bandwidth-intensive applications that have low commercial value like peer-to-peer file sharing. The first approach requires expensive upgrades to the network infrastructure while the latter limits the capabilities of the actual network itself. In contrast, our solution diverts Internet traffic to take advantage of less congested paths in the Internet in order to improve bandwidth.

Detouring for bandwidth is more difficult than latency or availability. Bandwidth is more expensive to measure—bandwidth measurements require many probes to be sent over long periods of time [58, 117, 123]. Bandwidth also varies significantly over time due to cross-traffic and requires frequent re-measurements to maintain accuracy [136].

In this chapter, we start by exploring the potential benefits from bandwidth detouring.

Next, we discuss various properties of bandwidth detours such as the longevity of bandwidth detours. We analyse two different bandwidth detouring mechanism: IP-level and TCP-level detouring; and evaluate them experimentally on PlanetLab.

5.1 Potential for Bandwidth detouring

In this section, we demonstrate the existence of useful bandwidth detours and potential benefits of these detours for bandwidth. We carry out our investigations on two datasets generated from PlanetLab and UkairoLab (described in §2.2.4).

Using PlanetLab nodes, we generated a list of 256 PlanetLab nodes which had bandwidth cap higher than 10 Mbps on May 3rd, 2010. Our experiments use less nodes due to downtimes of individual nodes.

We use UkairoLab to provide full kernel access which is necessary for evaluating IP-level detouring. UkairoLab contains 10 geographically dispersed nodes supplied by commercial providers. The bandwidth for these paths are lower than PlanetLab—the median bandwidth for UkairoLab and PlanetLab is 4.87 Mbps and 6.54 Mbps respectively.

Our goal is to understand whether bandwidth detours exist and the potential benefits one can gain from bandwidth detouring. We measure TCP bandwidth using `iperf` between 152 PlanetLab nodes and it is run for 5 secs. We assume the detour bandwidth is the minimum bandwidth of two paths: between the source-to-detour, and between the detour-to-destination.

In our first experiment, we investigate the potential improvement in bandwidth by detouring via one or more detours. In Figure 5.1, we find that 96.6% of the paths can benefit from detouring. The median improvement in bandwidth from taking one detour is 18.6 Mbps and by a median factor of 2.24 increase. The figure also shows that roughly 40% of the paths cannot improve their bandwidth by more than 10 Mbps. This is likely due to the paths having a capacity limit of 10 Mbps. Our results show and confirm that

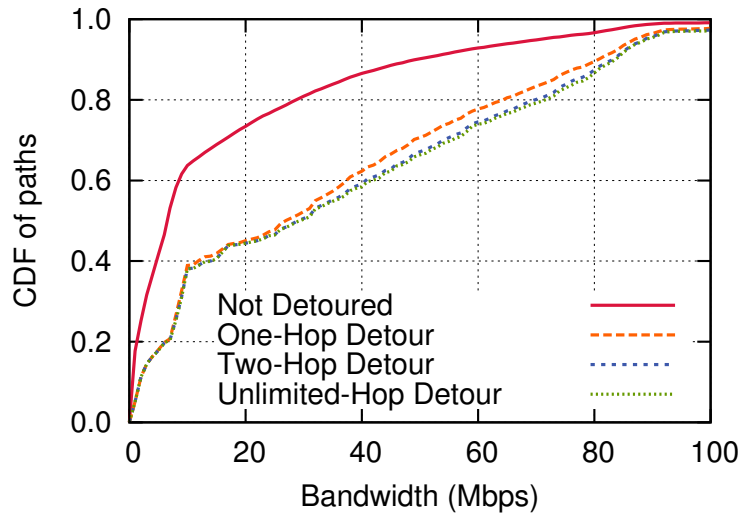


Figure 5.1: Detouring via more than one detour hop yields little additional benefits

detouring via more than one detour provides very little additional benefit [75].

Our next experiment compares the potential benefits of detouring for different metrics namely: bandwidth, loss and latency. We measure loss rate by sending 1200 UDP probes with 1472 byte payload each at 100 ms intervals which is the same rate as a typical VoIP connection [84]. We take the median latencies of these probes as the latency of a path. The latency of a detour path is computed as the sum of the latency between the source-to-detour and between the detour-to-destination; whereas the loss of a detour path is computed as $1 - (1 - \text{loss}(a, c))(1 - \text{loss}(c, b))$ where $\text{loss}(a, c)$ and $\text{loss}(c, b)$ are the loss of source-to-detour and detour-to-destination respectively.

In Figure 5.2, we show that bandwidth detours can offer a more substantial gain than latency detours—more than half of the paths can double in bandwidth while only 13.5% of that paths can reduce their latency by a half. Only 27.2% of the paths can benefit from loss detouring because most paths do not have losses at this low rate.

There is a larger scope for bandwidth improvement than latency because bandwidth can have several orders of magnitude of difference between paths. For example, detouring traffic from a path limited by a 10 Mbps-link to another which is limited by a

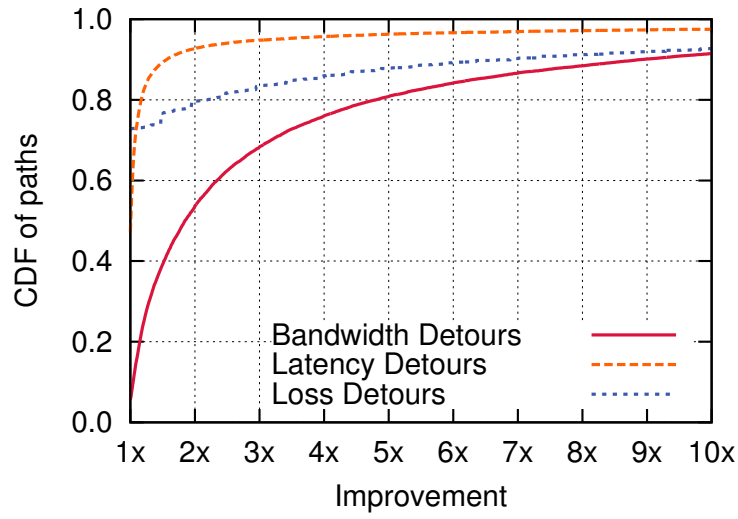


Figure 5.2: Bandwidth can be improved significantly more than latency or loss using detouring.

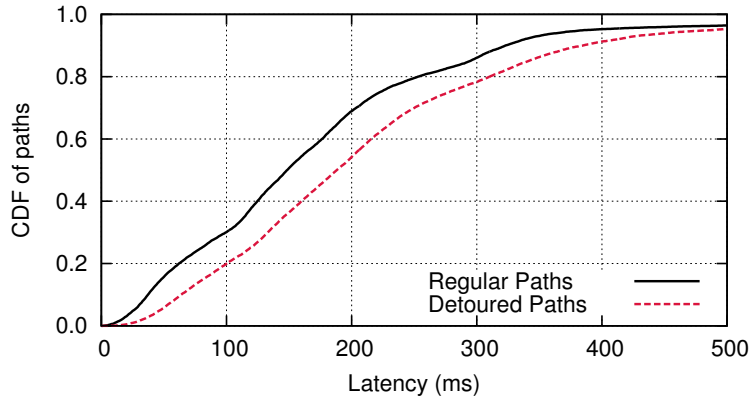


Figure 5.3: Choosing the best bandwidth detour increases the latency of a path.

100 Mbps-link would likely see large factors of improvement. Latency, on the other hand, the detoured path latency is bounded by geographical distances between the source and destination.

5.1.1 Effects of Bandwidth Detouring on Latency

An important question is what happens to the latency of a connection when we detour for bandwidth. Figure 5.3 shows the regular path's latency and detour path's latency when we pick the best bandwidth detour; and there is an increase in latency in most paths due

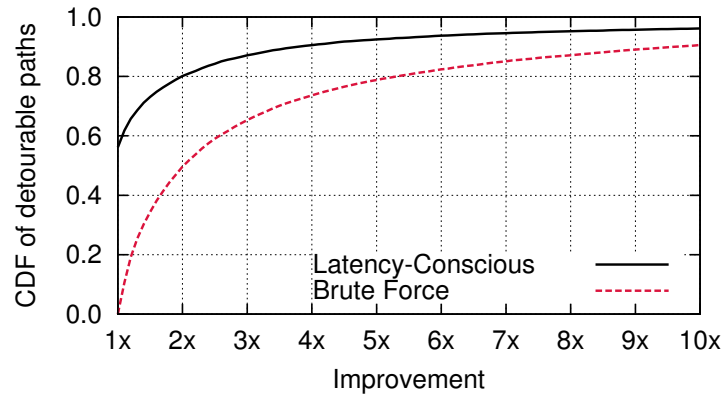


Figure 5.4: Less paths can be detoured when we choose bandwidth detours that does not increase end-to-end latency.

to bandwidth detouring. The median increase in latency between the regular path and the detour path is 23 ms. We found that 85.4% of the paths increase in latency (not shown in graph). Depending on the application, the difference can have varying degree of effects on end-user experiences. Applications should distinguish between data-intensive or time-sensitive traffic and detour only when necessary.

The next question is whether we increase bandwidth without increasing latency and quality of such detours? Figure 5.4 compares brute-force, the bandwidth improvement from choosing the best bandwidth detour, with latency-conscious, the best bandwidth detour that does not increase the regular path's latency. The figure shows that 43.9% of the paths that benefit from bandwidth detouring have a detour which does not increase the path's latency. This is expected because as shown earlier in §5.2, there are less paths that can be improved by means of latency detours; thus the choice of detours becomes limited. We believe that most applications have preference for either bandwidth or latency improvements rather than both. For example, HD streaming applications give precedence to their bandwidth requirements over the actual latency of the path.

5.2 Bandwidth Detour Properties

5.2.1 Symmetry of Detours

A detour is defined as *symmetric* if the detour benefits both the forward and reverse direction of the regular path. Congestion in the forward path rarely causes congestion in the reverse path so we would expect bandwidth to be different in each direction [19]. Yet our results show that 89% of 18 036 paths for which we have measurements in both directions, have at least one symmetric detour. We believe that this happens because the quality of a detour is dependent on the connection properties of the detour such as the upload and download speed rather than the congestion of a path. We found symmetric detours to be better than asymmetric detours—the median improvement of a symmetric detour is 39% as oppose to 16% for asymmetric detours.

5.2.2 Latency of Bandwidth Detours

We study the latency of regular paths in bandwidth detours. First, we investigate the latency of regular paths which benefit from bandwidth detouring. Figure 5.5 shows the latency distribution of all regular paths and paths which have a bandwidth detour that improves by a factor of at least 1, 2, 5, and 10 respectively. We notice that there are three distinct clusters of latencies for all paths. Previous work has shown that these clusters are formed due to the geography of the world [73, 135]. The three clusters roughly represent paths that are: intra-continental, and crossing one and two oceans.

Most paths benefiting from bandwidth detouring have long latencies. For example, while 32.3% of all paths have less than 100 ms, only 21.7% are below this threshold for paths with at least double the bandwidth. Intuitively, we would expect paths with low latency to make less AS-hops and thus be more likely to be well-connected and less likely to benefit from bandwidth detouring. Figure 5.5 also shows that most paths that benefit

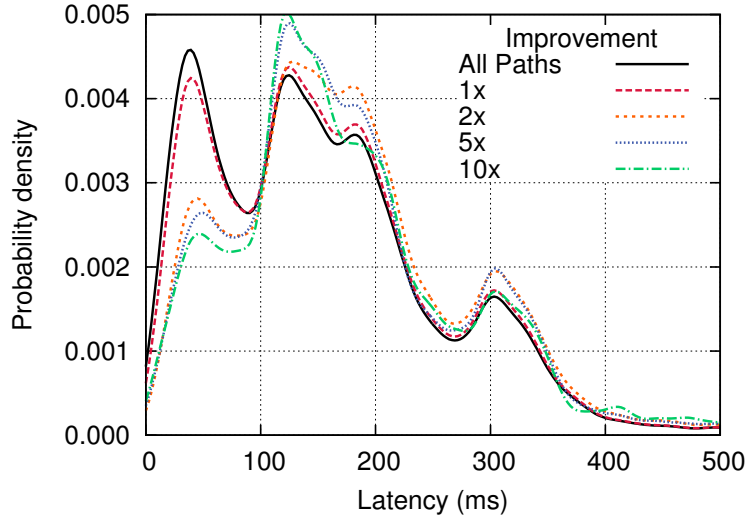


Figure 5.5: Bandwidth detouring benefits paths with longer latencies.

from detouring are in the second cluster (between 100 ms and 200 ms).

Skewness of Detours

Though beneficial detour paths have a higher latency than the regular path (shown earlier in §5.1.1), we find that beneficial detours usually have a *low* latency to the source or destination. Given an regular path (a, b) with latency $rtt(a, b)$ the *skewness* of a detour path $skewness(a, c, b)$ via detour c is defined as:

$$skewness(a, c, b) = \frac{|rtt(a, c) - rtt(c, b)|}{\max\{rtt(a, c), rtt(c, b)\}} \quad (5.1)$$

As skewness tends towards 0 the detour node becomes closer to the source or destination. Lower skewness values are more likely to offer beneficial detours—the median skewness value for detours that provide improvement is 0.43 whereas the median skewness value of all detours is 0.58. The reverse is also true; as the skewness approaches 1, the detours are less likely to be beneficial. Our dataset shows that detours that offer no benefit at all have a median skewness of 0.58. Our findings suggest detours that are close to the source or destination are more likely to be connected by high-capacity links, and hence more likely

to be good detours.

5.2.3 Persistence in time

The duration for which a detour can last is important to determine that bandwidth detouring is feasible. Detour discovery requires significant amounts of measurements and time to complete. Long-lasting detours increase the period for which the detour can be used and reduce the frequency of re-measurements.

In the next experiment, we conduct a longitudinal study on bandwidth detours. We collect an all-pair measurement between 100 randomly chosen PlanetLab nodes at 4am (GMT) everyday from Dec 9, 2010 to Jan 5, 2011 (except Dec 18 due to a failure). We exhaustively search for all beneficial detours on the Dec 9 dataset and observe whether the detours are still valid on later dataset.

We select detours on Dec 9, 2010 which improve the regular path by at least a factor of 1.0 to 2.0 at 0.2 increments then we test whether these detours are still beneficial at a later date. Figure 5.6 shows the fraction of valid detours after Dec 9, 2010. The graph shows a gradual decrease in the validity of the detours in the first 19 days. For example, 85.4% of the detours that can improve the regular path by a factor of two persisted after the first day, and this percentage drops to 78.5% by the 19th day. This bodes well for bandwidth detouring because a large number of detours persisted for long periods of time meaning re-measurements can be conducted less frequently.

We notice an abrupt change in detours after the 20th day on Dec 29 and this is permanent, lasting for the remaining 8 days. This suggests that there is a change in AS routes that permanently alters the bandwidth of multiple paths.

The figure also shows detours with a greater improvement factor are more likely to persist for longer periods of time. For example, choosing detours that provide a factor of two improvement is on average, 14.7% more likely to persist than detours that improves

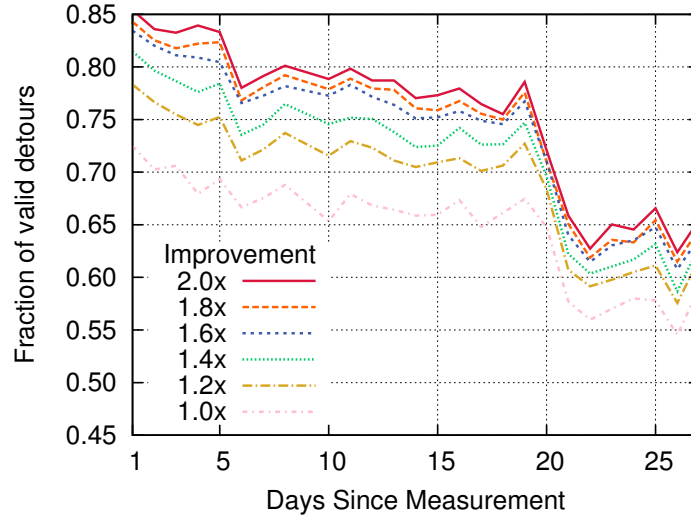


Figure 5.6: Longevity of bandwidth detours

by a factor of one. The trade-off is that selecting detours with higher improvements means less paths can be improved—while 94.3% of the paths can benefit from bandwidth detouring, 53.0% of the paths can be improved by a factor of two in the Dec 19 dataset.

From our experiment, we conclude that detours can be reused over long periods of time. We recommend re-measuring every two weeks to avoid substantially stale data. The long expiry times for bandwidth detours is particularly important compared to detours in other metrics such as latency and loss because bandwidth measurements are time consuming and consume more traffic to measure. In real deployments, the relatively high measurement overhead can be leveraged because detours remain effective over long periods of time.

We can also consider alternatives instead of re-measuring the entire network. For example, we can alleviate the need for frequent re-measurements by observing for major path changes using periodic traceroute probes which generates considerably less traffic; or we can update the bandwidth or validity of our detours by collecting feedback from clients while they use the detours. We leave this investigation to future work.

5.3 Bandwidth Detouring Mechanism

We study two different mechanism for detouring: a network-level detouring mechanism, *IP detouring*; and a transport-level detouring mechanism, *TCP detouring*. In this section, we perform a simple stochastic analysis of the two mechanisms and evaluate the performance of the two mechanisms on UkairoLab.

We denote the source-to-detour and detour-to-destination paths as first and second leg of a detour respectively.

5.3.1 IP Detouring

IP detouring can be thought of as a simplified implementation of *source routing*. Each packet is encapsulated with the destination IP at the source then the packet is sent to the detour where the destination IP is extracted and the packet is forwarded. IP detouring supports the two most important protocols in Internet applications, namely TCP and UDP. This mechanism only requires modifications at the source and detour, thus is transparent to the destination host. IP detouring can be applied to other metrics, such as latency.

We provide a simple analysis of steady state behaviour of TCP bandwidth using a stochastic model of TCP [88]. Equation (5.2) is a square-root formula that describes the TCP bandwidth, BW, in relation to the packet loss p and the average round trip time RTT. The parameter Φ , is dependent on the TCP acknowledgement strategy and the size of each packet (or better known as the maximum segment size MSS). $\Phi = \text{MSS} \times 0.93$ when delayed acknowledgement is used and $\Phi = \text{MSS} \times 1.22$ when an acknowledgement is sent on every packet.

$$\text{BW} = \frac{\Phi}{\text{RTT}\sqrt{p}} \quad (5.2)$$

We make two assumptions: the latency of the detour path is the sum of the latency of the two constituent paths and the loss rates of the two paths are independent. With these

assumptions, we derive the bandwidth of a path using IP detouring BW_{IP} . We denote p_1 , RTT_1 , BW_1 , and p_2 , RTT_2 , BW_2 as the average loss, latency and bandwidth of the first and second leg of the detour respectively. Equation (5.3) estimates the bandwidth of the detour path.

$$BW_{IP} = \frac{\Phi}{RTT_1 + RTT_2} \sqrt{\frac{1}{1 - (1 - p_1)(1 - p_2)}} \quad (5.3)$$

Removing $p_1 p_2$ and substituting Equation (5.2) to replace p_1 and p_2 yields:

$$BW_{IP} \approx \frac{RTT_1 RTT_2 BW_1 BW_2}{RTT_1 + RTT_2} \sqrt{\frac{1}{(RTT_1 BW_1)^2 + (RTT_2 BW_2)^2}} \quad (5.4)$$

Equation (5.4) can predict the bandwidth of a path that uses IP detouring given that we know the latency and bandwidth for both legs of the path.

Comparison with the minimum model

Bandwidth prediction of a detour path is typically based on a *minimum model* where the detour bandwidth estimate is the minimum of the two constituent paths. The intuition in this model stems from the fluid model where the volume of fluid that can be pushed through two pipes (or the two detour legs) depends on the individual rate of the two, but how accurate is this model? We compare our derived Equation (5.4) with that of the minimum model. First, we modify the equation to represent bandwidth and latency as ratios of the two detour legs. We denote α as the ratio of the latency between the first and second leg (i.e. $RTT_1 = \alpha RTT_2$) and similarly, β as the ratio of bandwidth between the first and second leg. Substituting the two ratios produce the following equation:

$$BW_{IP} \approx \frac{\alpha}{1 + \alpha} \sqrt{\frac{1}{1 + \alpha^2 \beta^2}} BW_1 \quad (5.5)$$

Equation (5.5) shows that the absolute values of the latency does not alter the detour

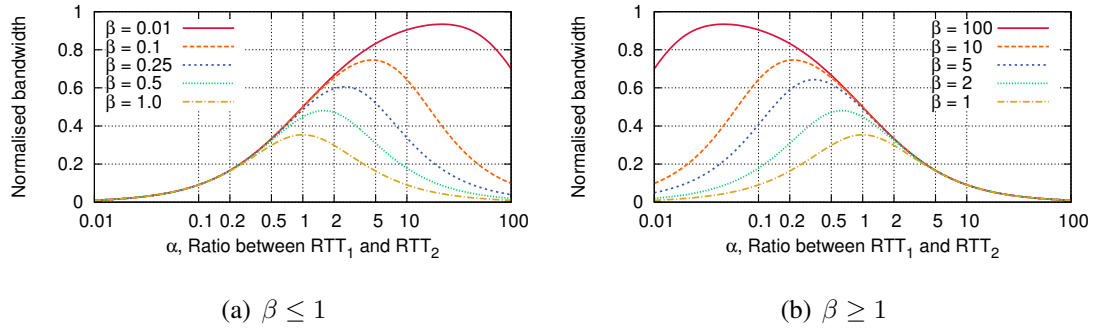


Figure 5.7: The bandwidth and latency of both legs affect the detour bandwidth in IP detouring.

bandwidth but it is the ratio between the two latency values that matters. Since $\alpha > 0$ and $\beta > 0$, we can see that the bandwidth obtained from IP detouring BW_{IP} is strictly less than the minimum model. Next, we differentiate this equation with respect to α to obtain the maximum bandwidth for different values of β . Equation (5.6) shows an inverse relationship between α and β which maximises the detour bandwidth.

$$\alpha^3 = \frac{1}{\beta^2} \quad (5.6)$$

To compare the detour bandwidth using IP detouring with the minimum model, we normalise the detour bandwidth by the minimum of the two constituent paths. Figure 5.7 shows the normalised bandwidth for different values of α and β . The figure shows three key points: i) using IP detouring, the bandwidth of the detour path is asymptotically below the minimum bandwidth of both legs as supported by Equation (5.5); ii) the peak bandwidth for a given β is described by Equation (5.6); iii) in the worst case scenario, where $\alpha = \beta = 1$, the detour bandwidth is reduced to a factor of 0.354 from the minimum of the two paths.

5.3.2 TCP Detouring

In TCP detouring, two TCP connections are established before data is sent: one at each leg of the detour path respectively. TCP detouring is analogous to *split-TCP*. This can be achieved by using SOCKS proxies at the detour. SOCKS proxies are transparent to the destination thus no modification is necessary at the destination to use it. For simplicity in our analysis, we assume that a client is downloading from a server where data flows from the second leg to the first leg.

Consider when the first leg of the path has a lower bandwidth than the second leg, then the buffer at the detour is never empty and the bandwidth would be determined by the first leg. Now consider the opposite case, where the first leg has a higher bandwidth than the second leg, then any packets that arrive at the detour have sufficient bandwidth to be sent to the first leg immediately thus the bandwidth would be governed by the second leg. In other words, the TCP bandwidth achieved using TCP detouring is the minimum of the two legs (i.e. the minimum model). Baccelli et al. [16] provide a stochastic analysis of the growth of TCP windows in the two detour legs.

An important question is how big do the buffers need to be at the detour? Unlike, IP detouring where buffers only need to be large enough such that the packets can be transmitted onto the link; in TCP detouring, buffers at the detour have to be large enough such that if there is a packet loss the packet can be retransmitted. To insure this property holds, the buffer at the detour should be large enough to hold the data that is in transmission. This is the same size as the *bandwidth delay product* (BDP) which is computed as the product between latency and bandwidth. Similarly, the buffer at the detour for uploading a file should be at least the size of BDP of the second leg.

5.4 Experimental Evaluation

We evaluate IP and TCP detouring performance on UkairoLab and PlanetLab. Using UkairoLab, we validate our predicted IP and TCP detouring performance. Next, we share our experience of deploying a TCP detouring mechanism on PlanetLab on a larger set of nodes. We were unable to evaluate IP detouring on PlanetLab because there is significant load which causes severe delays between time-slices.

5.4.1 IP and TCP detouring performance

We compare IP and TCP detouring on UkairoLab. We first collect an all-pairs bandwidth measurements. We predict the throughput of the detoured connection for IP detouring using Equation (5.4) in §5.3.1. Next we use IP and TCP detouring to measure the bandwidth of all-pairs via all detours.

Figure 5.8 shows the performance of IP and TCP detouring on each of the path. We first note that the improvement from IP detouring is less than TCP detouring—for example, 23% as opposed to 40% of the paths improve by at least 5 Mbps using IP and TCP detouring respectively. This confirms our earlier analysis in §5.3.1 that the loss rate of each leg in IP detouring is aggregated which deteriorates the detour bandwidth.

Though our TCP detouring mechanism takes advantage of the effect from split-TCP, most improvement comes from choosing a good detour with high bandwidths. Classical split-TCP assumes latency is largely unchanged and benefits come purely from responding to losses more quickly in a path. Our result shows that 77% of all detours provide at least 10% and 1 Mbps bandwidth improvement; of the detours where the intermediate leg latencies are lower than the regular path latency (which stand to benefit most from a split TCP connection), only 28% provide similar improvements.

We have evaluated both TCP and IP detouring mechanisms both analytically and experimentally. While IP detouring supports both TCP and UDP traffic, TCP detouring should

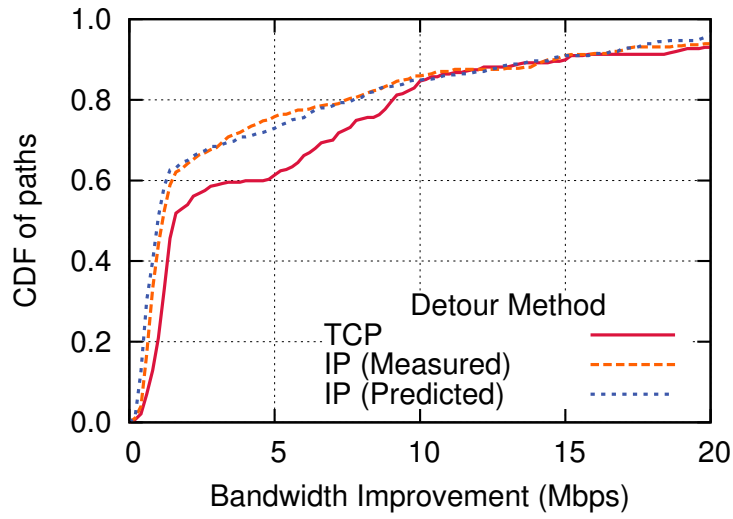


Figure 5.8: TCP detouring improves actual bandwidth between node pairs significantly more than IP detouring on UkairoLab.

be used when possible because it yields a higher gain from detouring.

5.4.2 TCP detouring performance on PlanetLab

We evaluate the TCP detouring mechanism with a larger set of nodes using PlanetLab. We divide the experiment into two phases: prediction and validation. In the first phase, we randomly select 50 PlanetLab nodes and perform all-pairs measurements between the nodes. The measurement is halted after 90 mins and we find 1845 of 2019 paths are detourable. In the validation phase, we select the best detour for each path based on measurements from the prediction phase, then we measure the bandwidth of both detour legs as well as the detour path using the TCP detouring mechanism. Since we avoid concurrent measurements, the second phase takes longer. After 11 hours, we are able to obtain 689 detourable paths.

Figure 5.9 plots the distributions of (i) measured regular path bandwidth; (ii) estimated detour bandwidth at the prediction phase and (iii) at the validation phase; and (iv) achieved detour throughput. First we found there is a significant increase in bandwidth from TCP

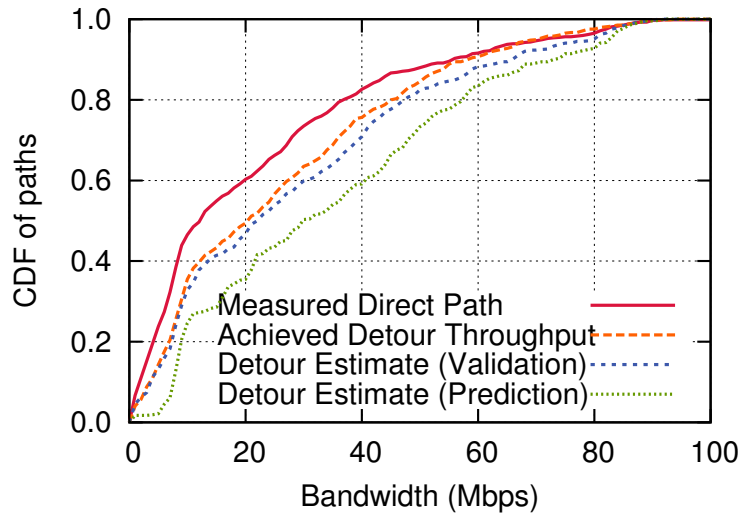


Figure 5.9: Validation of estimated and actual bandwidth using TCP detouring on PlanetLab

detouring; the median bandwidth increased from 12 Mbps to 21 Mbps. We also note that our TCP detouring mechanism performance is limited below 50 Mbps suggesting a bottleneck due to node performance. There is substantial difference between the detour estimate at the time of prediction and validation phase. Due to variations in bandwidth over time [136], the detour chosen during the prediction phase may no longer be the ‘best’ detour. However, these detours still provide a sizeable improvement over the regular path.

5.5 Using Latency Detours for Bandwidth Detouring

Earlier in §5.2.2, we have raised the possibility that lower latency paths are more likely to be well-connected. We investigate whether good latency detours can be effective bandwidth detours. We collect both latency and bandwidth measurements on 10 265 paths between 136 PlanetLab nodes. We choose the best latency detour for each given path and compute the bandwidth by taking the minimum of the two detour legs. Figure 5.10 plots the distribution of estimated bandwidth for the best bandwidth and latency detours found using brute-force search, and the estimated bandwidth by choosing a detour randomly.

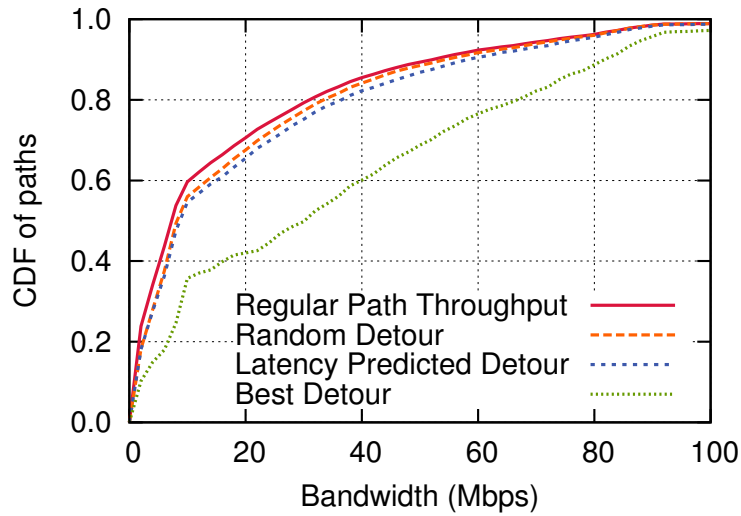


Figure 5.10: Latency performs no better than random selection for discovering bandwidth detour paths.

As mentioned earlier, though the best bandwidth detour can improve over the regular path significantly, in reality this is difficult to achieve due to variations in bandwidth. Our graph shows that selecting the best latency detour for bandwidth detouring is no better than randomly choosing a detour. This implies that it is ineffective to choose bandwidth detours based on latency detour discovery methods.

5.6 Discussion

In this chapter, we have shown that significant improvements can be obtained through bandwidth detouring. What are the implications of our results? We discuss this from two perspectives: i) the validity of our results and ii) the effects detouring would pose on the public Internet.

Although PlanetLab is the only available wide-area test-bed, it raises concerns about the validity of the results for the wider Internet. PlanetLab represents a subset of the actual Internet that is based on educational and research networks. We expect routing to be more transparent between ASes than the rest of the Internet because there are less conflicts of

interests, yet we find bandwidth not to be optimal on PlanetLab—we were able to discover bandwidth detours among PlanetLab nodes. On the other hand, we would expect the rest of the Internet to be more commercial-oriented which would have more complex AS agreements leading to more opportunities for detouring.

Collecting measurements on the commercial Internet would be difficult due to the amount of bytes transferred in bandwidth measurements. A possible alternative to PlanetLab is Measurement Lab [9] which can collect measurements from public users. However, they can only measure bandwidth to a limited number of designated servers.

Another problem is that academic networks from PlanetLab are connected better to the core of the Internet than a typical residential Internet user. Residential user’s bandwidths are often limited by the final stretch of a connection from a webserver (i.e. the “last mile”) where this stretch is connected using low bandwidth access mediums such as twisted pair. Since the bottleneck bandwidth is at the “last mile” there is little opportunity for detouring to change the bandwidth. We study this further in §6.4.1.

What will happen when bandwidth detouring becomes widely adopted? Bandwidth detour routing redirects traffic away from congested areas of a network to provide better end-user experiences and improves the utilisation of the Internet. Certainly, it can violate network-level routing policies such as moving traffic towards an AS that requires transit cost. While this may seem undesirable at first, there are already plenty of applications which shift traffic to obtain better performances such as BitTorrent [4] and CDNs [95].

Network operators should be made aware of these applications and take appropriate actions to handle these types of traffic. Recent proposals [71, 132] to make overlay networks to find paths that are ISP-friendly is complementary to our work.

5.7 Summary

We have assessed the possibility of exploiting detours to improve the bandwidth of end-to-end connection in the Internet. We found bandwidth detouring have great potential for improvement—we found more than half of the paths can double in bandwidth. We explore the various properties in bandwidth detours and found most detours with significant improvement have long latency (over 100 ms) and most bandwidth detours last for periods of over 2 weeks. Next, we discuss an IP-level and a TCP-level detouring mechanism. With analytical and experimental evidence, we show that TCP detouring performs significantly better than IP detouring. Yet, IP detouring can support UDP and be used as a mechanism for other metrics such as latency.

Our study discusses the feasibility of bandwidth detouring and offers insights to how detours can be discovered. However, how bandwidth can be implemented in a practical and scalable architecture remains an open question. In our next chapter, we introduce a system which provides bandwidth detouring that supports multiple clients.

6 Ukairo - Practical Detour Discovery

Bandwidth is an important performance metric for the Internet. In the previous chapter, we have shown that we can achieve substantial gains when detouring for bandwidth—with 50 detours, 43.9% of the paths can at least double in bandwidth on PlanetLab. To translate this benefit to Internet clients, we explore the challenges in deploying a practical bandwidth detouring platform.

One of the key issues in bandwidth detouring is the large amount of bandwidth measurements that has to be conducted to discover detours. The number of measurements often grows with the number of clients in the system as well as the number of nodes in the detouring platform. For a practical bandwidth detouring overlay network, the cost in performing measurement must be amortised with the benefits clients can gain. Thus it is necessary to limit the amount of measurements while maintaining the benefits from a bandwidth detouring overlay.

We explore three different methods in discovering detours for arbitrary paths: i) *detour ranking* ranks a list of detours to identify the most beneficial detours from the rest, ii) *iterative detour discovery* allows clients to find a set of useful detour through successive downloads, and iii) *client-to-overlay pairing* associates clients to one of the overlay nodes where detours are chosen based on the recommendation of the overlay node. All these methods aim at reducing measurements while providing useful detours.

Based on our offline analysis, we construct Ukairo¹, a practical bandwidth detouring

¹Japanese romaji for “detour”

platform for Internet clients. Ukairo provides bandwidth detours that reduce HTTP download times for arbitrary Internet paths. Like a regular HTTP download, Ukairo’s download process is on-demand and seamless to the end-user.

Ukairo operates in three steps. First, it measures the bandwidth within a set of overlay nodes and ranks these nodes based on their potential as a detour. Next, it uses the list of ranked detours to apply different detour selection strategies which finds a smaller set of candidate detours. The various strategies allow Ukairo to adjust between the bandwidth gains of individual clients and balance the load on the detours. Finally, from the small set of candidate detours, clients identify the best detour route by running parallel downloads for a short period of time. Over successive usage, clients refine their set of candidate detours and replace the poorly performing detours with better ones.

In this chapter, we first demonstrate that most of the bandwidth improvement can be achieved from a subset of all detours. Our PlanetLab experiment shows that with 50 randomly chosen detours, we can obtain 80% of the detour’s benefit that an overlay with 150 nodes can achieve. This can be enhanced by measuring 10% of the paths in the 150-node overlay to obtain a detour ranking where we achieve 92.4% of all improvements (§6.1.1).

We describe several detouring strategies that are based on the detour’s ranking. We show that these strategies are effective in improving bandwidth for clients (§6.4.1) and the strategies can be used to make our detouring platform more scalable (§6.4.3).

We evaluate our Ukairo deployment on PlanetLab. We show that clients can substantially increase their bandwidth when downloading from public web servers and cloud storage systems. We notice that 80% of the paths can increase in bandwidth to these destinations and 30% of the paths can quadruple in bandwidth to Amazon-S3’s cloud storage system (§6.4.1).

6.1 Scalable Detour Discovery

For our bandwidth detouring overlay to operate to Internet-scale, it is impractical to measure all paths via all detours for arbitrary end-hosts. Instead we select a small set of candidate detours $\mathcal{S}$ that are potentially beneficial to the path where we limit the size of the set $|\mathcal{S}|$ to five or less. Thus the number of measurements to discover detour for an arbitrary path becomes $|\mathcal{S}|$ plus the regular path. In our Ukairo implementation, we show that it is possible to find the best detour from this set without incurring any measurement overhead (in §6.3.2).

We discuss three different methods of discovering detours scalably which have unique properties for finding detours, load-balancing and managing overhead. They are complementary and can be used in-conjunction with each other in a detouring platform.

Dataset. We study various methods for discovering bandwidth detours on PlanetLab. We collected a dataset containing bandwidth, traceroute and latency measurements on 6th August, 2010. We choose at most one PlanetLab node from each site which has a bandwidth limit of at least 10 Mbps. The TCP bandwidth measurements are collected by running `iperf` for 5 s. The latency measurements are gathered by taking the median of 1200 probes each with 1500 bytes sent at 100 ms interval. We also convert the IP addresses in our traceroute measurements to their associated AS using Team Cymru’s AS lookup².

Between 199 PlanetLab nodes, we measured the bandwidth, latency and traceroute for 38 102, 34 895 and 26 256 paths respectively. On average, `iperf` consumes 11.4 MB to measure a path.

We first define the bandwidth of a detour path and the metric we use to measure the improvement of a detour. For a given path (i, j) detouring via a detour l is predicted to have a detoured bandwidth of b_{ilj} which is computed as the minimum bandwidth between

²Team Cymru is a lookup service that provides IP-to-AS mapping collected from more than 50 BGP peers

the two legs of the detour b_{il} and b_{lj} , i.e. ,

$$b_{ilj} = \min(b_{il}, b_{lj}) . \quad (6.1)$$

In the previous chapter (in §5.4.2), we evaluated the validity of this assumption.

We quantify the *quality of a detour* $d_l(i, j)$ for a given path (i, j) by the relative bandwidth increase that the detoured bandwidth b_{ilj} has over the original bandwidth b_{ij} as described in the following equation,

$$d_l(i, j) = \frac{\max(b_{ilj}, b_{ij})}{b_{ij}} . \quad (6.2)$$

6.1.1 Ranking Detours

Not all bandwidth detours are useful, some bandwidth detours are more beneficial than others. We find that ranking detours can provide two advantages: i) it can restrict the size of the overlay network while providing similar benefits when all detours in the bandwidth overlay are used, and ii) it can find a small set of detours that are potentially better than the rest.

We build a detour ranking based on the quality of each detour d_l , which is defined as average relative bandwidth improvement of the set of paths that have been measured as given by,

$$d_l = \frac{\sum_{\text{paths}(i,j)} d_l(i, j)}{|\text{paths}(i, j)|} , \quad (6.3)$$

where $\text{paths}(i, j)$ represents the set of measured paths.

Restricting Overlay Size

By limiting the number of detours in a system we can drastically reduce the number of bandwidth measurements taken. We explore the effectiveness of a detouring overlay when

we restrict our overlay size to a set of detours chosen by rank.

We divide our n PlanetLab nodes into two disjoint sets: a detour set $\mathcal{D}$ with k nodes, and a client set $\mathcal{C}$ with $n - k$ nodes. We further divide the nodes in the detour set $\mathcal{D}$ into two groups: i) *effective detours*, $\mathcal{L}$, are detours from the detour set $\mathcal{D}$ that are selected to perform detouring ($\mathcal{L} \subset \mathcal{D}$), ii) while *inactive detours* are detours in the detour set $\mathcal{D}$ which do not (i.e. $d_l(i, j) = 1$ for all inactive detours).

Though constructing a detour rank requires taking measurements, we can select paths to measure in order to obtain a useful detour ranking without having to measure all paths. Before any measurements are taken, there is little information available to suggest the paths to measure, but we can select paths that form triangles (i, l, j) that allow us to compute $d_l(i, j)$. We use a simple algorithm to select the order of paths to measure. The algorithm first selects each detour l in a round-robin fashion, then randomly picks a path (i, j) for which $d_l(i, j)$ has not been computed. It measures the triplet of paths (i, j) , (i, l) and (l, j) . If some of the paths in the triplet have been measured previously, the algorithm uses the value from the previous measurement. This process is repeated until a predefined percentage of the overlay has been measured.

We compare the different selections of effective detours $\mathcal{L}$ by computing the *aggregate bandwidth improvement* of an overlay. For each path (i, j) where $i, j \in \mathcal{C}$, we find the best effective detour l_{ij} from the detour set $\mathcal{D}$. The aggregate bandwidth improvement is the sum of the relative bandwidth increase for $(i, j) \in \mathcal{C}$ using the best detour, i.e.

$$\sum_{\text{paths}(i,j)} d_{l_{ij}}(i, j). \quad (6.4)$$

Figure 6.1 shows the bandwidth improvement of $\mathcal{D}$ normalised to the bandwidth improvement when the size of detour set is $k = 150$. We vary the number of effective detours from 1 to 150 nodes. We select our effective detours from the detour set $\mathcal{D}$ in two ways: i) at random, ii) or based on the rank of a detour d_l computed when 10%, 40%, 60%, and

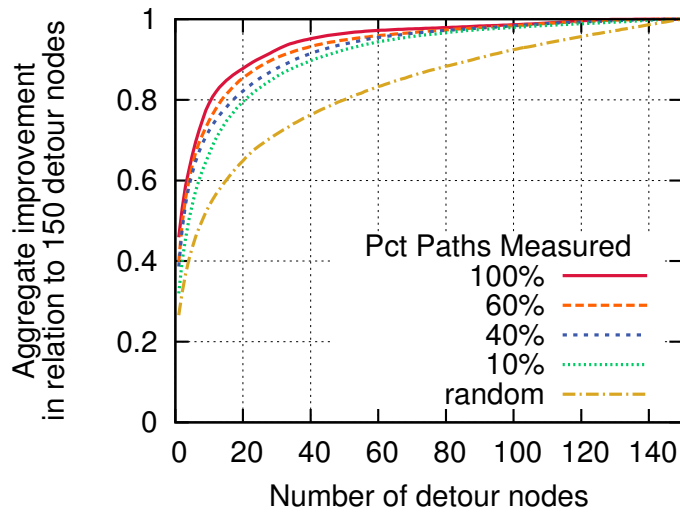


Figure 6.1: Improvement obtained by restricting the detour set to fewer nodes

100% of the overlay has been measured. We repeat this process 1000 times where we randomly select a fresh detour set $\mathcal{D}$ to test the different detour topologies.

We make three observations from Figure 6.1. First, as the number of effective detours increases the improvement provided by an additional detour suffers from diminishing returns. Second, if the effective detours are selected based on a detour's rank (at 100% of paths measured), we can drastically reduce the number of active detours without reducing the benefit of the overlay. For example, to obtain at least 80% of the bandwidth improvement from our 150 node overlay, we only need to, on average, include 11 detours if the active detours are selected by rank, but require 50 detours when they are selected randomly.

We also note that measuring a very small portion of paths is sufficient to obtain a detour ranking that is useful. For example, with 50 active detours, measuring all-to-all detour paths to build a ranking can obtain 96.5% of all benefits from the overlay, measuring only 10% of all paths can already provide 92.4% but we significantly reduce the number of measurements. Measuring a small portion of paths removes detours that have relatively low bandwidth to the core of the Internet and therefore is sufficient to eliminate low

quality detours.

Detour Selection

A detour ranked list can also be used to find a small *candidate detour set* $\mathcal{S}$ for providing bandwidth improvements to an arbitrary path. In the previous section, we showed that the detour set $\mathcal{D}$ can be restricted to a smaller set $\mathcal{L}$ of effective detours while maintaining similar performance to $\mathcal{D}$. Here, we compare various detour selection strategies that select the set of candidate detours $\mathcal{S}$ from our effective set $\mathcal{L}$.

Using the effective detours $\mathcal{L}$, clients still need to select the best detour to use for an arbitrary destination. Due to the size of $\mathcal{L}$, it is impractical for clients to search for the best detour in $\mathcal{L}$. Using several detour strategies, we filter $\mathcal{L}$ to a smaller candidate detour set $\mathcal{S}$ that is feasible for clients to discover detours in.

The candidate detour set $\mathcal{S}$ should ideally be tailored to a specific client. On an Internet-scale deployment, searching for $\mathcal{S}$ specific to a client will incur a substantial amount of measurements and is computationally expensive due to the large, $\binom{|\mathcal{L}|}{|\mathcal{S}|}$, search-space. We compare two classes of detour selection strategies: *client-agnostic* strategies where the detour selection process is the same regardless of the client, and a *client-specific* strategy where the candidate detour set is customised to the client.

We describe four different strategies, the first three are client-agnostic strategies and the last is client-specific.

Random-k. This strategy chooses k nodes uniformly at random. The strategy performs poorly for choosing the detour selection set $\mathcal{S}$ but we include this for comparison.

Best-ranked-l. All clients select the l best-ranked detours from $\mathcal{L}$. The size of l can be chosen depending on the load of the l detours which can be collected when the client uses the bandwidth detouring overlay.

Load-balanced-l-m. The previous strategy may overload the l best-ranked detours. The

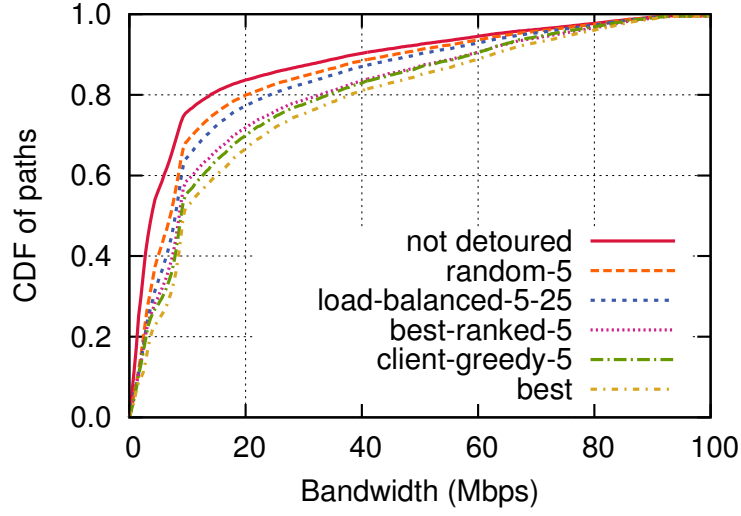


Figure 6.2: Comparison of the effectiveness of different ranking strategies to reduce the set of detour nodes to be considered by clients

load-balanced strategy first selects m best-ranked detour nodes such that $m > l$ from $\mathcal{L}$, then it randomly selects l detours from the list of m detours. While this strategy degrades the quality of detours, it allows the bandwidth detouring overlay to support more clients by spreading the load across the m detours.

Client-greedy- l . In this strategy, we first construct a *client-specific ranking* that is specific to a client i then select the l -best ranked detour from the ranking. To compute the client-specific ranking, we apply the same detour ranking equation as in Equation (6.3) except we only include paths that are from client i (i.e. the set of paths (i, j) where $j \in \mathcal{C}$). Though this is an approximation to the *optimal* client-specific set, this is computationally feasible because it does not require calculating $\binom{|\mathcal{L}|}{|\mathcal{S}|}$ combinations. However, clients still perform a significant amount of measurements to each destination to obtain the ranking. Later, we describe how we can alleviate this cost by collecting measurements passively in §6.1.2.

We evaluate our detour selection strategies by splitting the PlanetLab dataset into two sets. Of the 199 PlanetLab nodes, we randomly select 50 PlanetLab nodes to represent

as detours $\mathcal{D}$ and assign the rest interchangeably as clients or destinations. We regenerate the detour set 1000 times to test different sets of detours.

Figure 6.2 shows the bandwidth distribution of paths using the different strategies. We compare the distribution of regular paths (not detoured) to: i) the best of 5 randomly chosen detours (random-5), ii) the 5 best ranked detours selected from $\mathcal{L}$ (best-ranked-5), iii) 5 randomly selected detours from 25 best ranked detours (load-balanced-5), and iv) a client-specific strategy which also selects 5 nodes using the client-greedy-5 strategy. We also include the bandwidth distribution of the best possible detour by exhaustively searching through $\mathcal{D}$ as a comparison (best).

We see that best-ranked-5 provides significantly higher bandwidth than regular and random-5. best-ranked-5 also performs close to the best bandwidth achievable and client-greedy-5 but without requiring the client to perform measurements which is necessary to obtain the client-specific ranking. While best-ranked-5 is more vulnerable to detours becoming overloaded, load-balanced-5-25 offers modest improvement by including lower quality nodes to balance load. Depending on the load of top ranking detours, the detouring overlay can adjust its detour selection strategy to provide improvement to a larger set of clients.

6.1.2 Iterative Detour Discovery

When clients use the bandwidth detouring overlay, information regarding the effectiveness of each detour used can be fed-back to the clients. This information can be used to modify the clients' copy of the detour list $\mathcal{L}$ to obtain a client-specific detour ranking. Using this ranking, clients can create their client-specific detour selection set $\mathcal{S}$ that converges towards their optimal set.

Clients refine their detour selection set $\mathcal{S}$ as measurements are gathered when using the bandwidth detouring overlay. Each client i is given a list of detours $\mathcal{L}$. The list $\mathcal{L}$ can

Algorithm 3: Iterative refinement of detour node set $\mathcal{S}^i$

```
1  $\mathcal{L}^i :=$  ranked detour list with scores
2  $\mathcal{S}^i :=$  candidate detour set
3  $l^i :=$  test detour

4 download using path selection with  $(\mathcal{S}^i, l^i)$  as detours
5  $\mathcal{L}^i \leftarrow$  update scores using path selection results
6  $\mathcal{S}^i \leftarrow$  select  $s$  top-scoring detours from  $\mathcal{L}^i$ 

7 if  $b_{il_{ij}} < b_{ij}$  then
8   // test detour did not beat regular path on last test
9    $l^i \leftarrow$  select node from  $\mathcal{L}^i$  with probability =  $score$ 
```

optionally be ranked using the *best-ranked* detouring strategies discussed in §6.1.1 to offer a good set of detours to choose from before the client has gathered any measurements. We refer to this list specific to the client i as $\mathcal{L}^i$.

Algorithm 3 shows how clients iteratively refine their detour selection set $\mathcal{S}^i$ through successive downloads. After each download, clients update their list $\mathcal{L}^i$ based on the performance of each detour which in turn updates $\mathcal{S}^i$. The algorithm can be described as follows.

Given an instance of $\mathcal{L}^i$, client i picks the highest-scoring $|\mathcal{S}^i|$ detours from $\mathcal{L}^i$ to form the candidate detour set $\mathcal{S}^i$. In addition to $\mathcal{S}^i$, we also select one additional *test detour* l^i , with probability weighted by its current score, from $\mathcal{L}^i$ that is not in $\mathcal{S}^i$. The purpose of the test detour is to speed up convergence of our $\mathcal{S}^i$ towards the client specific set.

As the client i downloads from a destination j , bandwidth measurements can be collected as part of the path selection process. The client i learns the bandwidth to destination j (i.e. $bw(i, j)$) and the detour bandwidth via the candidate detours $\mathcal{S}^i$ and the test detour l^i . The list $\mathcal{L}^i$ is then updated to incorporate the new measurements. The new score of each detour is computed using the average quality of detour d_l described earlier in Equation (6.3). The list of detour nodes $\mathcal{L}^i$ is then re-ranked in descending order and the set $\mathcal{S}^i$ is reselected based on the new ranking (lines 5–6). We re-pick the test detour only if it did not show improvement compared to the regular path (lines 7–9).

While the detour selection set $\mathcal{S}^i$ provides consistency in improvement from the regular path, the test detour l^i ensures quick convergence towards the optimal set of client-specific as more downloads are performed (cf. §6.4.2).

6.1.3 Pairing Clients with Overlay Nodes

An alternative method to discovering detours scalably is to associate clients or destinations to a node in the bandwidth detouring overlay such that the overlay node shares the same detouring properties as the corresponding clients or destinations. For example, let a client i and a destination j be associated to nodes a and b in the overlay respectively. Through measurements within the overlay, we find that l_{ab} to be a beneficial detour for path (a, b) , then l_{ab} would also be beneficial to path (i, j) . This method alleviates clients from having to do any measurements by selecting a node in the overlay to suggest detours on their behalf.

In comparison to the ranking strategy (in §6.1.1), each overlay node would contain a unique set of detours where clients select from. Therefore, this method uses a more diverse set of detours than best-ranked-5 in our ranking strategy which would provide better load-balancing.

Unlike overlays [107, 140, 57] which redirects all the client's traffic to an ingress node in the overlay network (see 6.2), our method only uses overlay nodes to provide detour suggestions. Our method is guaranteed to perform as good as the regular path suggested by regular Internet routing.

We denote the set of clients and destinations as $\mathcal{C}$ and the nodes in the bandwidth detouring overlay as $\mathcal{D}$. The associations become a bipartite graph between the two sets. Our objective is to match the client and destination $i, j \in \mathcal{C}$ to two nodes $a, b \in \mathcal{D}$ such that their detouring properties are similar that is $d_l(i, j) \approx d_l(a, b)$. We assume that the client-destination pair has no detours if they are matched to the same node (i.e.

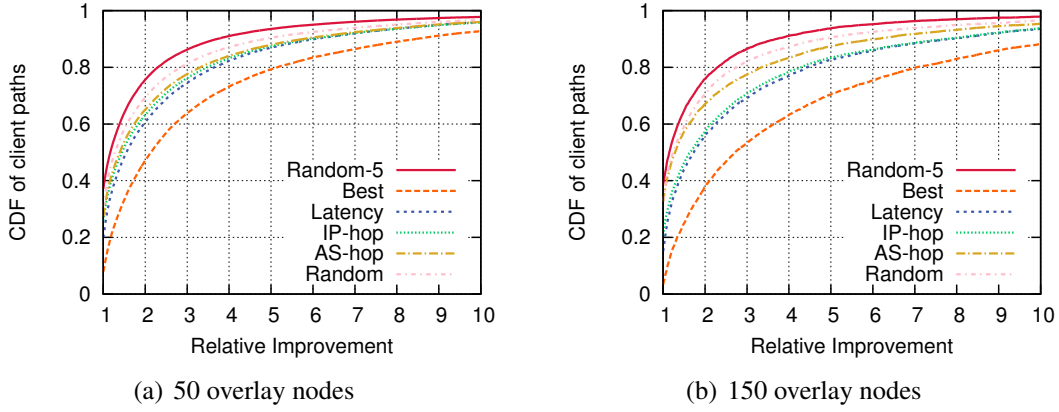


Figure 6.3: Improvement of client-overlay pairing using different metrics

$d_l(a, a) = 1$ for any detours l).

For an Internet-scale bandwidth detouring overlay, the clients and destinations set $\mathcal{C}$ can be arbitrarily large. Therefore, the associations between $\mathcal{C}$ and $\mathcal{D}$ have to be made without performing substantial amount of measurements. We propose three metrics that would be suitable for determining the matching: lowest latency, fewest IP-hop count, and fewest AS-hop count. These metrics are chosen because they have a low measurement cost and reflect the proximity in terms of geographical locations or network distances between the nodes in the two set, $\mathcal{C}$ and $\mathcal{D}$.

Like previous experiments, we divide our dataset into two disjoint sets, $\mathcal{C}$ and $\mathcal{D}$, where there are k PlanetLab nodes in $\mathcal{D}$. We use the nodes in $\mathcal{C}$ as clients and destinations interchangeably. For each given *client path* $(i, j) \in \mathcal{C}$, we map the two nodes, i, j , to $a, b \in \mathcal{D}$ using one of the three metrics respectively. We find the best 5 detours for path (a, b) to form a candidate detour set $\mathcal{S}$ by exhaustively searching through all-pairs of nodes within $\mathcal{D}$. We use this candidate detour set $\mathcal{S}$ as detour suggestions for the regular path (i, j) and pick the path with the highest bandwidth from the set or regular path (i, j) . We repeat this experiment 1000 times randomly selecting $\mathcal{D}$ every time.

Figure 6.3(a) shows the distribution of the relative improvement defined in Equation (6.2) when the overlay size is $k = 50$. We compare the relative improvement of our method

based on the three metrics (latency, IP-hop, and AS-hop). We also add: i) random-5 which randomly selects 5 detours from $\mathcal{D}$, ii) random which associates the clients and destinations with nodes in $\mathcal{D}$ at random, and iii) best which always picks the best detour from the overlay.

We observe all three metrics perform similarly in terms of relative improvement. They all show better improvements than both random strategies (random-5 and random). We speculate that the size of the overlay is too small for any of the three metrics to make strong associations between the two sets, $\mathcal{C}$ and $\mathcal{D}$. For example, the average latency, IP-hop and AS-hop for the association between the two sets are: 25.8 ms, 8.1 IP-hops, and 2.1 AS-hops respectively.

In Figure 6.3(b), we repeat the same experiment except that the size of the overlay $\mathcal{D}$ is increased to $k = 150$ nodes. The average latency, IP-hop and AS-hop for the associations between the two sets become: 11.61 ms, 6.8 IP-hops, and 1.7 AS-hops respectively. Due to the stronger associations, there is an increase in the relative improvement—for example, in latency, 41% of the client paths can at least double in bandwidth when the overlay size is $k = 150$ but this is only true for 37% of the paths if the overlay size contains 50 nodes. There is less benefit from the larger overlay when using AS-hop as a metric because AS-paths are too short to benefit from the increase.

Though our pairing method shows improvement over the regular path, compared to our earlier detour ranking method (in §6.1.1) the improvement gained is less. We believe that for this method to be effective, there must be sufficiently large number of nodes in the overlay such that they are ‘close’ to the client. Of course, with a larger overlay, the measurement costs within the overlay would increase. Therefore it is important to place detours close to the client to maximise the benefit of this strategy.

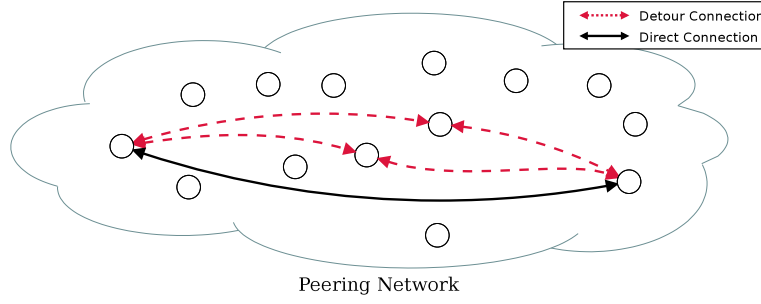


Figure 6.4: Example of a peer-to-peer bandwidth detouring network

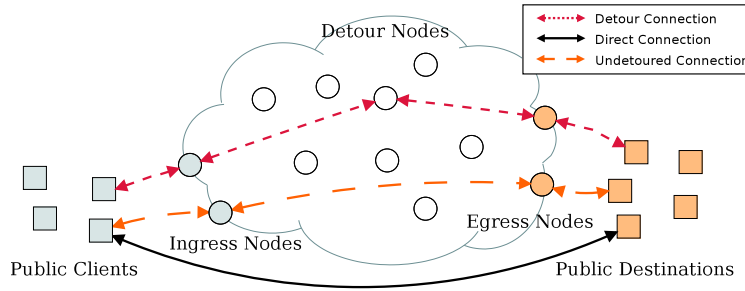


Figure 6.5: Example of an ingress-egress bandwidth detouring network

6.2 Three Models for a Detouring Architecture

In general, bandwidth detouring overlays can be classified into three generic models based on their architecture. The main difference between the three models is who selects the detour of a given path. The three models are: *peer-to-peer network* where the peers (or clients) select the detour, *ingress-egress detouring network* where the detours are chosen by the nodes in the overlay, and *information node based detouring network* where detours are selected by an auxiliary node (which we will refer as an *information node*).

Figure 6.4 illustrates a peer-to-peer networks such as BitTorrent [4], SplitStream [24] and Bullet [66]. In a peer-to-peer network, every node is a client as well as a detour. Clients exploit path diversity by downloading from multiple peers (or nodes). This architecture requires the node with the original content to be a peer in the network. This means that the source of the content must support the protocol used in the peer-to-peer network; thus it cannot be used to connect to arbitrary webservers.

Figure 6.5 shows an example of an ingress-egress detouring platform. An ingress-

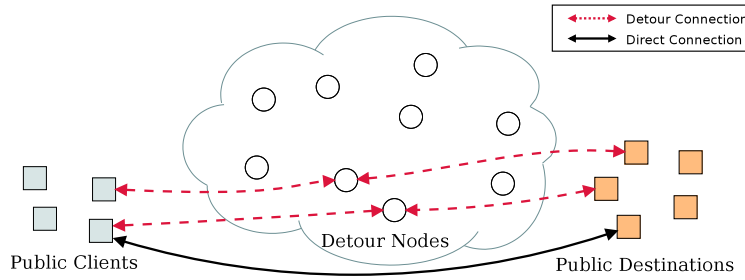


Figure 6.6: Example of an information node based detouring network

egress network is composed of ingress nodes, egress nodes, and detours. Ingress nodes provide an entry point for a client to connect to; detours provide path diversity; and egress nodes allow traffic to leave the network to the desired Internet destination. These three types of nodes are often interchangeable. Through measurements within the network, ingress nodes can select the best path to any other nodes within the network.

Clients benefit from detouring relaying its traffic to the overlay network and rely on the network to choose the path that is most beneficial. Clients first connect to a pre-determined ingress node. The ingress node determines the egress point that is—by some metric—closest to the destination. The ingress node then determines the path with the highest bandwidth to the egress node then forwards the traffic. Note that a path at ingress that does not have a detour (long-dashed orange line) is not the same path as the regular path (a solid black line).

There are several key issues in using this architecture. First, it is unknown how a client-destination pair can be associated with an ingress-egress pair. Since clients and destinations can be arbitrarily large, this association must be made with low measurement cost. Second, the ingress and egress nodes can become congested introducing a potential bottleneck in the system. Finally, this model makes the assumption that traffic relayed over the network is better than a regular connection between the client and destination.

The third architecture is an information node based detouring network (shown in Figure 6.6). Information nodes (not shown in figure) are responsible for coordinating mea-

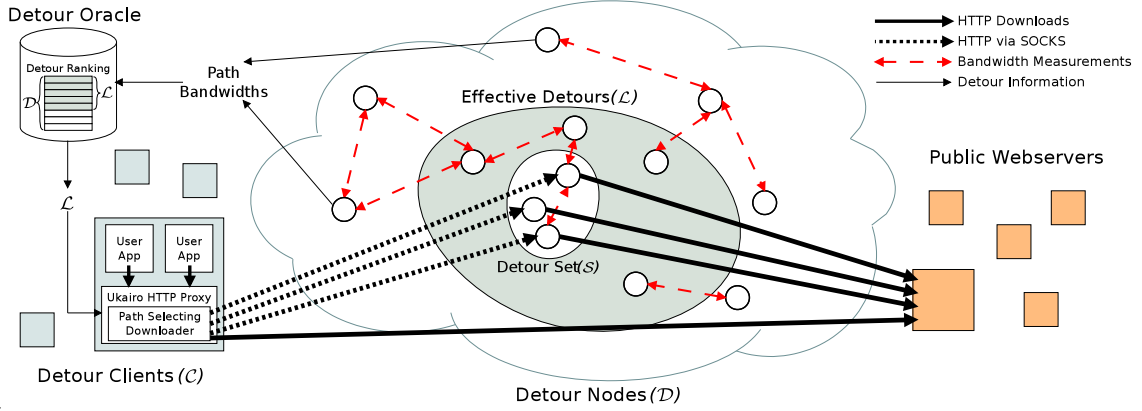


Figure 6.7: Overview of the Ukairo system architecture

measurements within the detour network and disseminating information about the detours to the clients. Clients then use this information to decide how to detour their traffic using the detour node. Examples of this type of architecture are Baron [75] and SideStep [27].

Our goal is to provide bandwidth improvement for Internet-scale clients to arbitrary destinations. A peer-to-peer architecture is not suitable because it cannot connect to arbitrary destinations. We raised several concerns regarding the use of egress-ingress model. An information node based network is more preferable for its simplicity. In the next section, we describe Ukairo and we describe in detail how to implement a bandwidth detouring network using an information node based network.

6.3 Ukairo Architecture

Up to this point in the chapter, we have discussed three methods for discovering bandwidth detours scalably (in §6.1.1), and different models for implementing a detouring architecture (in §6.2). We now combine the two concepts together to construct Ukairo, an Internet-scale bandwidth detouring overlay for HTTP traffic.

Figure 6.7 gives an overview of how Ukairo facilitates bandwidth detouring. Ukairo adopts an information node based structure described in §6.2. It has three main compo-

nents: i) an information node, which coordinates measurements between the set of detours $\mathcal{D}$ and disseminates this information to the clients; ii) a client's path selection mechanism, which picks the best path from a list of suggested detours $\mathcal{S}$ or the regular path if none are beneficial; iii) and the detour itself which provides higher bandwidth paths to clients.

From a client's point of view, a client is first given a list of detours $\mathcal{L}$ with their detour ranking score from the information node. When an application in the client invokes the Ukairo detouring service, the client first filters the list and selects a smaller subset of detours $\mathcal{S}$ to perform path selection. The client starts downloading useful data from the destination as soon as path selection starts. Once the path selection selects the best path, the client disconnects the connection from the other contending paths and completes the download using the best path.

6.3.1 Ukairo's Information Node

The main tasks of an information node is to schedule measurements amongst the set of detours $\mathcal{D}$ and provides clients with a set of detours that can be used to detour for *any* destinations. The information node decides how measurements should be conducted within the overlay and collects these measurements. The measurements are analysed and given to the clients upon their request.

The information node orchestrates measurements by requesting detour nodes to measure bandwidth by running `iperf` for 5s. There are two stages in Ukairo's measurements. The objective of the first stage is to restrict the size of the detouring overlay and pick an effective detour set $\mathcal{L}$ from the rest of the nodes. For example in our PlanetLab deployment, we measure 10% of the overlay for 199 PlanetLab nodes using our path selection process discussed in §6.1.1 to reduce the number of effective detours $\mathcal{L}$ to 50 PlanetLab nodes.

The second stage involves measurements amongst the effective detours $\mathcal{L}$, where we

aim to find the rank of each detour node within $\mathcal{L}$. We perform all-pairs measurements between these nodes to identify the ranking of each detour, however, since this is already a restricted set, we have already significantly reduced the measurement costs.

The two measurement stages operate on two different periods. The first stage is completed more sparsely because there is already a set of effective detours $\mathcal{L}$; whereas in the second stage can react more closely to the rank of the selected detours. Since detours are long-lived (shown in §5.2.3), we recommend the first stage to be performed monthly and the second stage on a bi-weekly schedule.

The second task of the information node is to disseminate the set of detours to clients. Depending on the congestion between the detours, the information node can enforce the clients to use different detouring strategies (described in 6.1.1). For example, instead of disseminating the full list of effective detours $\mathcal{L}$ (as shown in Figure 6.7), it can disseminate a smaller list of candidate detours such as the best five ranked detours to encourage clients to use the better detours when the usage is low. Since the ranked list is small in size, our information node can be mirrored to different locations to provide redundancy and accessibility.

6.3.2 Path Selection Mechanism

Ukairo’s path selection mechanism aims to find the specific detour from a small set of detours that is most useful for an arbitrary path. The mechanism is designed to be transparent to applications (i.e. applications are not aware that path selection is being performed) so applications only need to make minimal changes to implement Ukairo’s path selection mechanism.

Ukairo’s path selection mechanism is split into two phases: the *sampling phase* where the best path is discovered by establishing parallel HTTP connections and simultaneously downloading the file from several paths; and the *completion phase* where the download

of the remainder of the file is completed.

Sampling phase. A client is given a list of detours $\mathcal{S}$ chosen by the iterative detour discovery algorithm (in §6.1.2). The path selection algorithm also finds the size of the file by establishing a regular direct connection to the webserver. The file is divided into $N/(s+1)$ segments and downloaded simultaneously using the HTTP/1.1 “Range” field [39].

Ukairo downloads in parallel for 5 s or until it has finished downloading its share. Our experience shows that 5 s is sufficient for TCP to distinguish the best path from the $(s+1)$ paths.

Completion phase. After sampling phase is completed, the client downloads the parts of the file that has not been completed using the best detour path. If the regular path has a higher bandwidth than the detour paths, then download is completed using the regular path. Since the client does not download any duplicate data, the overhead generated by Ukairo’s path selection mechanism is minimal.

It is important that the sampling phase is kept as short as possible. Since downloading from multiple TCP flows in parallel will affect the fairness of contending TCP traffic in the Internet. In Ukairo’s path selection mechanism, the duration of the sampling phase is kept short and the slow start in a TCP connection limits the negative effect of multiple simultaneous flows. As future work, Ukairo can implement multipath-TCP such as [130] to maintain fairness while measuring from multiple TCP paths.

6.3.3 Detour implementation

Detours relay TCP connections by implementing a SOCKSv5 [76] proxy server. SOCKSv5 is transparent to the destinations and thus can connect to arbitrary TCP destinations. For our implementation, we choose `3proxy` because it is lightweight.

A detour traffic flows from a client’s application to an arbitrary HTTP server as follows:

- i) the application is configured to relay their traffic through the Ukairo HTTP proxy lo-

cated at the client, ii) when the Ukairo client receives HTTP requests from the application, it establishes a TCP connection with a remote detour proxy using SOCKSv5 protocol, iii) the remote proxy completes the connection with the webserver.

Clearly, detouring results in two TCP connections: one between the client and the detour, and another between the detour and destination. Our implementation benefits from the effect of *split-tcp* whose bandwidth of the detoured path is close to the minimum of the TCP throughput of the two TCP connection. As shown in the previous chapter (in §5.3), this provides a significantly higher bandwidth than establishing one long TCP connection through the client, detour and destination.

Our use of the SOCKSv5 protocol incurs a small delay while the connection is being established. Let the round-trip time between the client and the detour be RTT_1 and similarly, the detour and the destination be RTT_2 . The client first establishes a TCP connection with the detour and exchanges the SOCKS-related information which will cost $2RTT_1$. Next, establishing a TCP connection between the detour and the destination and replying to the client will cost $RTT_1 + RTT_2$. Thus setting up a SOCKSv5 TCP connection will take $3RTT_1 + RTT_2$. This delay is likely to be larger than the latency of a non-detoured connection (demonstrated in §5.2.2). We also note that this delay implies that it is favourable to connect to a detour that is closer to the client. For large downloads, this overhead becomes insignificant compared to the benefits of bandwidth detouring.

6.4 Evaluation

We evaluate Ukairo on a PlanetLab deployment. Our aim is to demonstrate the benefits Ukairo can provide in improving bulk HTTP downloads from different Internet clients and destinations.

6.4.1 Internet detouring performance

We evaluate the detouring performance of Ukairo by using PlanetLab nodes as a set of detours $\mathcal{D}$. We start with 201 geographically dispersed set of PlanetLab nodes chosen from different sites as detour nodes. We randomly choose 50 nodes to represent our effective detour set³. We deploy our information node in an university machine. We evaluate the detouring platform with clients and destinations from other PlanetLab nodes, public Internet web servers and servers support cloud services (Amazon S3 and Dropbox), as well as residential broadband clients.

Detouring to PlanetLab destinations

First, we consider detouring between PlanetLab nodes. We choose 30 PlanetLab nodes as clients and also another 30 PlanetLab nodes as destinations such that none of these nodes are not detours.

For each of the 30 destinations, we deploy a lightweight webserver, `thttpd`, to imitate a public webserver. Each client downloads a 40 MB randomly generated binary file from the webserver which provides statistically significant result for paths with high bandwidth. We limit any serial single HTTP transfer of a file to 120 s to reduce the time in measuring low bandwidth paths. We observe that we cannot reliably achieve over 40 Mbps on PlanetLab webserver (likely due to limited memory and IO contentions) thus we limit our results to regular paths below this rate. This means that we cannot provide improvement to 15% of the regular paths as shown in Figure 6.2.

We compare the bandwidth improvement for different detour ranking strategies: `best-ranked-5` and `load-balanced-5-25` with `best-ranked-1` and `load-balanced-1-25` to demonstrate the importance of having multiple candidate detours.

³Normally, the set of effective detours $\mathcal{L}$ would be selected using the method described in §6.3.1 earlier. However, due to down times in PlanetLab nodes, we find it more reliable to maintain a random set of 50 detours.

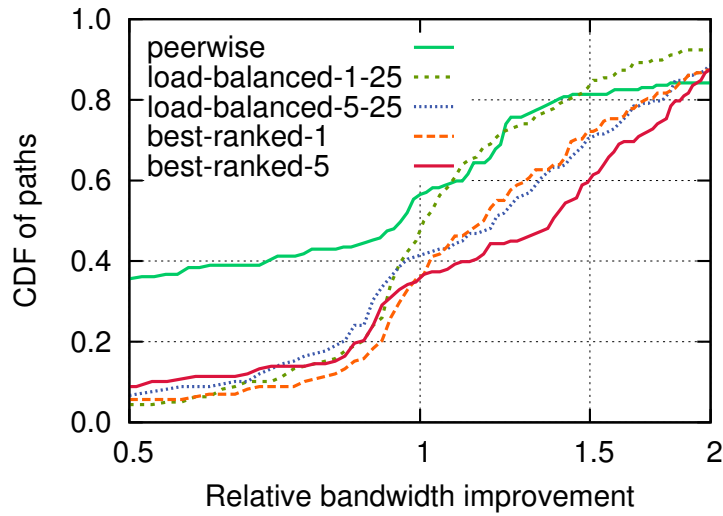


Figure 6.8: Detouring improves bandwidth on the majority of lower-bandwidth Internet paths

We also compare our approach with *peerwise*, where bandwidth detours are selected based on the latency of the detour path. *PeerWise* [80] finds the detour with the largest latency reduction between a client and destination. *Peerwise* chooses detours using a network coordinate system which is scalable (i.e. without making all-to-all measurements). In our implementation, we use the detour with the largest latency reduction and not use our path selection mechanism.

Figure 6.8 shows the relative improvement as the ratio between the detoured bandwidth and the regular path. When the ratio is greater than one, it shows that the regular path has benefited from detouring and when the ratio is less than one the contrary is true. Paths decrease in bandwidth can be due to measurement noise, detouring overhead or a change in bandwidth in the regular or detour path.

We measure between 209 paths in which 75.6% is under 40 Mbps. During the sampling phase in path selection, we observe that this is on average 3% faster than the download speed in the completion phase. This suggests that the parallel download process during the sampling phase does not gain a significantly unfair share of the network resource.

Figure 6.8 demonstrates that *best-ranked-5* performs the best—showing improvement

in 64.2% of the paths, and with 40.0% of the paths increased in bandwidth by 50%. As expected, load-balanced-5-25 shows a smaller improvement (58.6% of the paths show improvement) because it chooses from lower quality detours, however, having the advantage that it reduces the load of the best 5 detours.

Our best-ranked-1 strategy performs significantly better than load-balanced-1-25—27.8% and 16.2% of the paths can see an improvement of 50% respectively. This shows that a single carefully selected detour in the Internet can offer significant benefits.

best-ranked-1 and load-balanced-1-25 both perform relatively well because there are less paths to be compared by the path selection mechanism, therefore less likely to select the wrong detour path. During the sampling phase of the path selection, a larger set of candidate detours $\mathcal{S}$ will increase the contention for bandwidth between the parallel TCP flows thus the measurement result from each flow also become more noisy.

Finally, we note from the figure that using latency-based detour selection such as peer-wise does not yield effective detours. We observe 64.4% of the paths lose half the bandwidth using such scheme. This confirms our earlier results (in §5.5) that latency-based detour selection does not provide significant gain.

Detouring to public destinations

We also consider public Internet destinations to provide a more realistic scenario in which Ukairo would operate. We demonstrate that Ukairo can reduce the download times from webserver such as Linux kernel mirrors and from two different cloud services, namely *Amazon-S3* [2] and *Dropbox* [6].

For our public webserver, we use a list of 400 different server mirrors hosting Linux kernel sources. Most of these mirrors are hosted on commercial sites. We choose a file that is roughly 40 MB in size from these mirrors to download. To minimise the negative impact of downloading multiple large files from these webserver, each client only downloads

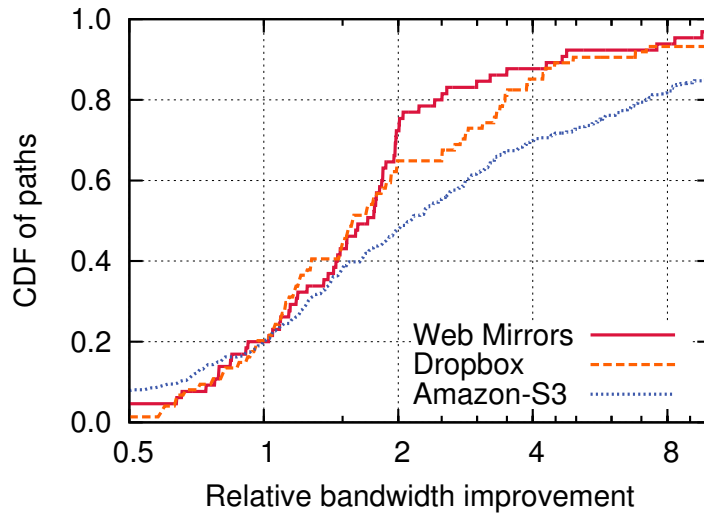


Figure 6.9: Detouring improves bandwidth to arbitrary public Internet destinations

from each mirroring site at most once.

We also experiment with downloading from two cloud services, Amazon-S3 and Dropbox. At the time when we run our experiment, the Amazon-S3 cloud service provides cloud file storage at five different datacenters around the world (Tokyo, Singapore, UK-Dublin, US-California, and US-Virginia). Dropbox is also a cloud storage service which uses one of the Amazon’s datacenters, namely US-Virginia. We evaluate the download performance by uploading a randomly generated 40 MB file to each Amazon datacenter as well as Dropbox.

We use 50 detour nodes on PlanetLab and also 10 clients and 78 clients for the public web server experiment and cloud downloading experiment respectively. We evaluate 86 different paths in our public web server experiment, 432 paths in our Amazon-S3 cloud service, and 78 paths in Dropbox.

Figure 6.9 shows the distribution of the relative bandwidth improvement of these public destinations when we use our best-ranked-5 strategy. First, we note that the improvement from public destinations are significantly higher than our PlanetLab counterpart. For example, in Amazon-S3, we observe that 61.8% of the paths can increase in bandwidth

by 50% which is only true for 40.0% of the paths if the destinations are on PlanetLab (as shown in Figure 6.8). We believe the bandwidth performance of our previous PlanetLab experiment is constrained due to overloaded PlanetLab nodes.

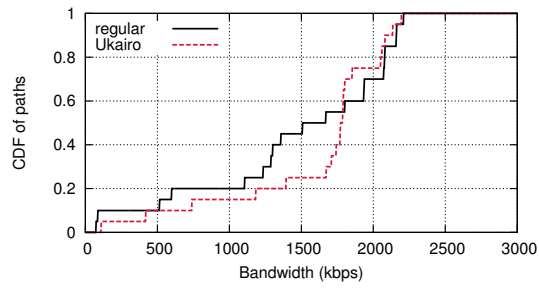
The public web servers, Amazon-S3 and Dropbox all show an improvement in bandwidth in roughly 80% of the paths. We also see that 50.1% and 35.1% of the paths increase in bandwidth by at least two-fold to Amazon-S3 and Dropbox respectively; whereas 26.7% obtain a similar benefit to public web servers. We presume the difference between the webserver and Amazon-S3 is that Amazon's datacenters are strategically located to provide high bandwidths therefore have a higher potential for large improvements. The performance in Dropbox is slightly worse than Amazon's, we believe this is caused by additional provisioning processes that Dropbox has to perform in addition to Amazon-S3's.

Detouring for residential broadband

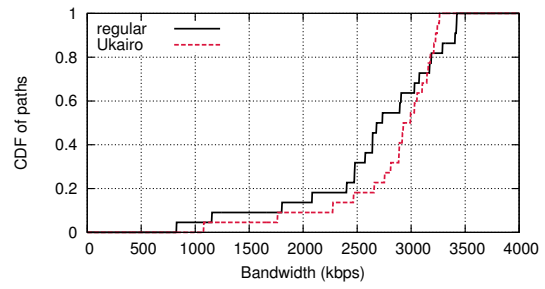
In the past, the major bottleneck in bandwidth has been at the "last mile" of a path. This bottleneck has been gradually shifting as faster bandwidth technologies become available to residential homes such as ADSL2+, DOCSIS and fibre-to-the-home networks [77]. This shift meant the congestion of a path is closer to the core of the Internet, which allows residential clients to detour around the congested segment of the path therefore improving the bandwidth of their connection.

We investigate whether Ukairo can be beneficial to residential broadband clients. We explore eight different residential broadband where six are located in London-UK, one in Hong Kong, and one in Taiwan. Six of our clients use ADSL and two use DOCSIS. The maximum broadband speeds of our clients range from 2.2 Mbps to 43 Mbps. Each client downloads a 40 MB file from at least 20 destinations chosen at random from our list of Linux kernel mirrors. We use best-ranked-5 as our detouring strategy.

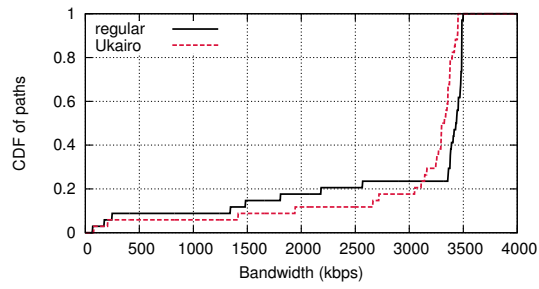
Figure 6.10 shows the distribution of bandwidth for a regular download and the down-



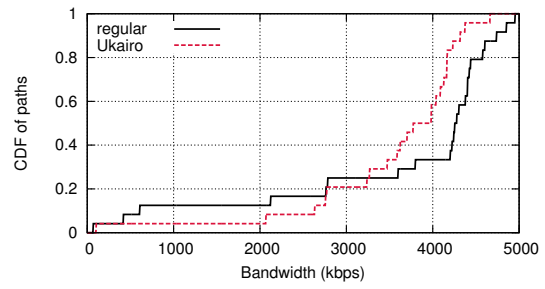
(a) DSL-London-1



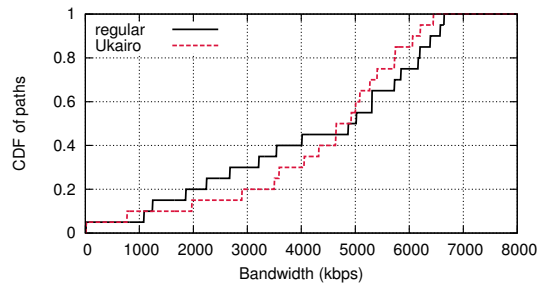
(b) DSL-London-2



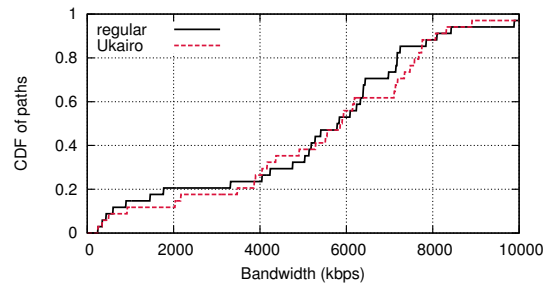
(c) DSL-London-3



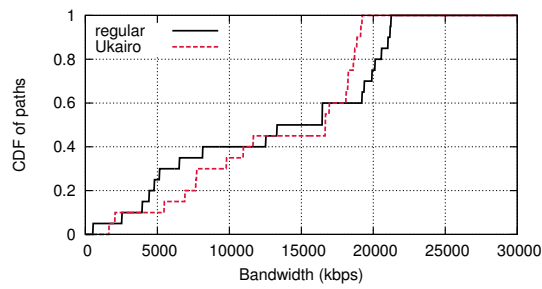
(d) DSL-HK



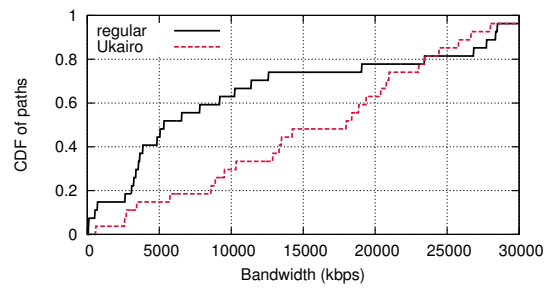
(e) DSL-London-4



(f) DSL-Taiwan



(g) DOCSIS-London-1



(h) DOCSIS-London-2

Figure 6.10: Bandwidth of different residential broadband clients using Ukairo

load achieved using Ukairo for each individual client. We sort the graphs in order of the client’s maximum bandwidth observed. We notice that in all the clients, when the download speed is close to the client’s maximum bandwidth (i.e. the client’s bottleneck bandwidth), Ukairo performs worse than the regular path. This is most clearly illustrated in the low-speed ADSL broadband clients such as DSL-London-3 shown in Figure 6.10(c). The difference is due to the additional overhead incurred by Ukairo during the TCP setup of the proxy connection mentioned in §6.3.2. This overhead is larger when clients are further from the detour. We do not quantify this bandwidth reduction any further because it depends on several factors: the delay between the client and the detour, the client’s connection speed, and the actual size of the file that the client downloads.

We compute the *net improvement* for each client defined as the sum of the difference between the detour’s path bandwidth and regular path’s bandwidth. With the exception of our DSL-HK client (shown in Figure 6.10(d)), all our broadband clients show a gain in net improvement where the faster DOCSIS clients gain more benefits than DSL clients. The DSL-HK client does not benefit from detouring due to the larger TCP setup overhead because it is located further away from the set of candidate detours $\mathcal{S}$ which are located in US and UK.

We plot the distribution of relative improvement of 20 randomly sampled paths from each residential clients in Figure 6.11 (all). Additionally, we show the subset of paths when path selection selects one of the five detours as the preferred path (detoured). Though individually not all clients benefit from detouring, we see Ukairo can provide a substantial improvement to some paths—more than 12.8% of the paths at least doubled in bandwidth. This improvement is more noticeable when we consider only paths that are actually detoured where we see 19.8% of the paths improve by the same factor. We also observe only 43.3% of the paths show any improvement in bandwidth, however, the bandwidth of the paths do not decrease significantly when using Ukairo—only 0.6% of

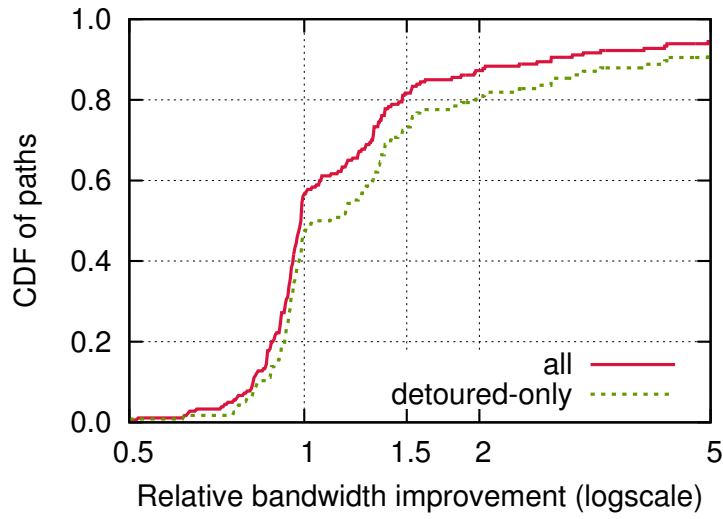


Figure 6.11: Relative Improvement of all broadband residential clients

the paths decreased in bandwidth by more than half. The key insight is when clients are bottlenecked by their broadband speed, due to Ukairo’s overhead, the speed would be reduced slightly; however, when the clients are not bottlenecked locally, Ukairo has the potential to provide a significant improvement.

Ukairo selects detours that are geographically similar to the regular path. The two most used detours of the five detours are located in Germany and Washington, DC. They are responsible for detouring 73.4% of all detoured traffic. The German detour is used when the destination is located in Europe and the one in Washington, DC is used when the destination is in North and South America, Asia, and Australia. While the two detours may not be close to the client or destination, using the `traceroute` tool reveals that they usually lie close to the regular network path.

6.4.2 Iterative detour node selection

We evaluate the iterative probing strategy described in §6.1.2 where clients refine their detour selection set $\mathcal{S}$ as they download files. We conduct our experiment on PlanetLab where we select 10 clients to download a 40 MB file from 35 other PlanetLab nodes at

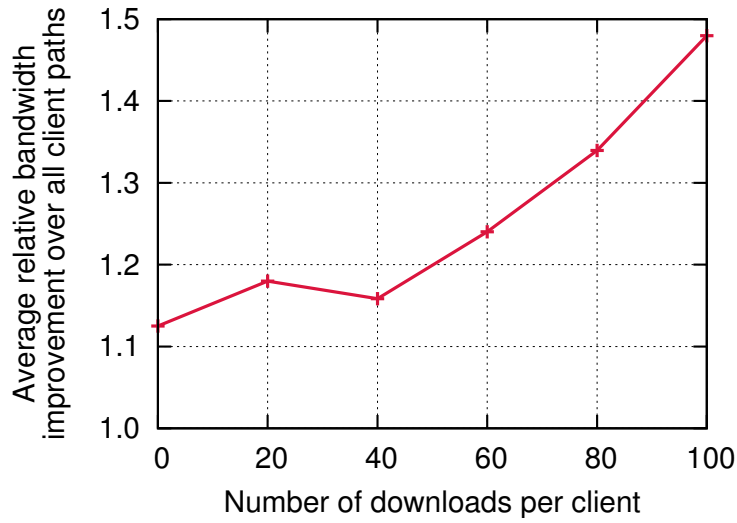


Figure 6.12: Iterative detour selection improves the quality of the clients' detour sets over time

random. Clients are given a list of $\mathcal{L}$ detours with the detour's rank. This list is used to refine the detour selection set $\mathcal{S}$ upon successive downloads using Algorithm 3.

Figure 6.12 shows the average relative bandwidth improvement after each iteration. As the number of downloads increase, clients tailor its detour set to provide greater bandwidth gains from detouring. Prior to any downloads, the average relative improvement is a modest 12% increase. After approximately 100 downloads, the relative bandwidth improvement is increased to a factor of 1.48. We also find that our iterative client eventually converges towards a detour selection set $\mathcal{S}$ that is specific to that client.

6.4.3 Load balancing detour nodes

Our previous experiments evaluated Ukairo when it is used by a single client. However, in a realistic setting, Ukairo has to be able to support many concurrent clients. In this section, we evaluate how Ukairo reacts to multiple clients using the detouring platform at the same time.

To evaluate the number of simultaneous clients Ukairo can support. We increment the number of clients in our experiment from 10 to 120 concurrent clients. Each client

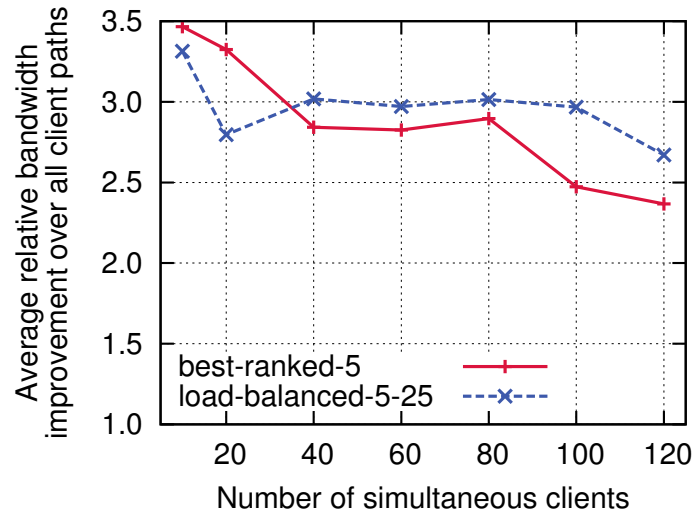


Figure 6.13: A load-balancing detour strategy can be effective for many concurrent clients

downloads from one of our 20 PlanetLab webservers. We find it difficult to saturate a PlanetLab detour while complying with PlanetLab’s daily bandwidth limit, thus we artificially limit the bandwidth of a detour to 50 Mbps. We run this experiment twice, once using the best-ranked-5 strategy then again using our load-balanced-5-25 strategy.

Figure 6.13 depicts the average relative improvement of the client paths as we increase the number of concurrent clients from 10 to 120. At fewer than 40 clients, our best-ranked-5 strategy performs better than our load-balanced-5-25 strategy. As we increment the number of clients in our best-ranked-5 strategy, the five best ranked detours become congested leading to a lower relative improvement. On the contrary, load-balanced-5-25 spreads the load over a larger set of detours therefore the decrease in relative improvement is less significant.

We also test Ukairo when we are not restricting the bandwidth of a detour. We host our own detour at an university. We investigate whether adding more machines to the detour site would increase the number of concurrent clients.

Figure 6.14 shows the percentage of paths detoured by path selection algorithm as we increase the number of concurrent clients from 5 to 100. We also consider adding up to

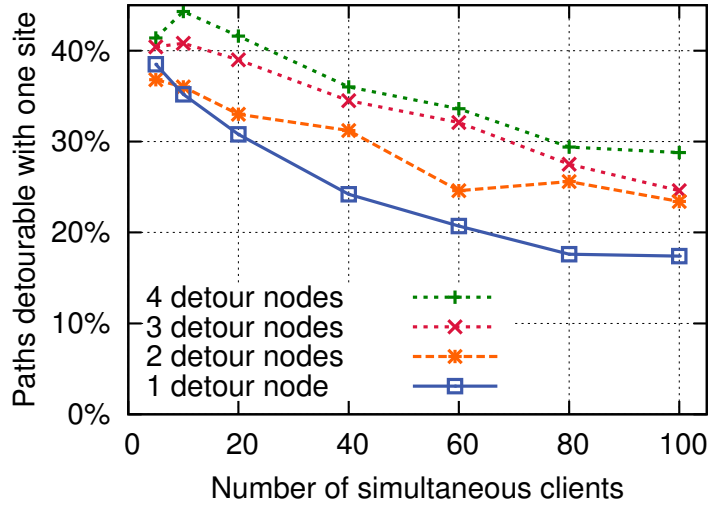


Figure 6.14: Adding nodes to a detour site increases the number of supported clients

four other detours onto the same site. We observe that the average relative improvement is $2.5 \times$, similar to our previous experiment.

The graph shows that as we increase the number of clients the fraction of paths detoured decreases. For example, in the case where we use one detour at the site, 35% of the paths can be detoured when there are 5 concurrent clients and this percentage decreases to 18% when there are 100 concurrent clients.

We observe that when we add more detours to the site, more paths can benefit from detouring. At 100 concurrent clients, 4 detours at the same site would provide 29% of the improvement as opposed to 18% when there is only a single machine. While with one detour the peak bandwidth used is 280 Mbps, four detours at the same site consume up to 600 Mbps. This suggests that instead of the network, the performance of the detours can play an important role in determining the number of clients it can support as well. As demonstrated, it is possible to add additional detours on to the same site to utilise the network better.

6.5 Summary

In this chapter, we demonstrate that bandwidth detouring can improve the bandwidth for many Internet clients. We show that a few well-chosen detours can provide a significant benefit in bandwidth detouring. Taking advantage of this fact, clients can find bandwidth detours without having to measure the bandwidth of a large set of detour nodes. We also discovered that only a small number of detours are needed to provide improvements to many clients concurrently.

Using Ukairo, clients can benefit from a reduction in HTTP download times. We show that for public web servers and cloud storage systems, Ukairo can double the bandwidth for 50% of the paths from PlanetLab clients. While residential broadband clients are still limited by their broadband speed, for Internet paths that are well below the client's broadband speed Ukairo can offer bandwidth improvement.

7 Conclusions and Future Work

In this thesis, we have shown that end-to-end bandwidth and latency can be improved using detour routing. We have shown that detours can be found effectively without incurring high measurement costs.

We began by presenting a novel technique for finding latency detours. Our hierarchical clustering algorithm first finds *fingerprints* (which are tuples of AS-links) then associates these fingerprints with an appropriate detour. Our algorithm is accurate in identifying TIV paths which are paths that benefit from detouring. It also shows significant benefits in detouring for other metrics such as bandwidth.

Next, we present `tcpChiryo` which employs concepts from network tomography to predict TCP bandwidth in an overlay from a small subset of measured paths. Using a practical methodology and by intelligently selecting paths to measure in the overlay, we can infer the bandwidth of other unmeasured paths. As far as we know, no one has applied and evaluated network tomography for TCP bandwidth. We show that this approach can also reveal congestion in a network which may be useful for diagnosing the Internet.

Finally, we study the benefits and scope of bandwidth detouring which then leads to our implementation and evaluation of Ukairo—a bandwidth detouring overlay network. We found that TCP detouring mechanism is superior to IP detouring. By ranking bandwidth detours in terms of their quality, we can scalably discover detours. Our Ukairo path selection mechanism, which selects the best detour from a small candidate set of detours, allows clients to seamlessly download a file on-demand.

7.1 Future Work

We propose two future directions to our overlay network which are complementary to our current work. In one direction, we suggest taking AS policies into consideration to build an ISP-friendly detouring overlay network. On the other hand, we propose a passive approach to capture measurements as clients use Ukairo.

7.1.1 Towards an ISP-Friendly Detouring Overlay Network

There may be mutual benefits for both end-users and ISPs to cooperate in choosing beneficial detours. This provides end-users with higher performance Internet paths while ISPs can potentially benefit from a lower transit cost.

There have been significant efforts towards more ISP-friendly overlays such as [71, 132], especially in peer-to-peer networks [28, 100, 101]. Some work [71, 132] suggests ISPs to reveal some information about the underlying policies for detouring. In peer-to-peer overlay networks, researchers suggest finding peers that are within the same AS domain leading to a higher bandwidth path without incurring transit costs on ISPs [28]. While this solution suffices when there is a replica of the content within the local AS domain, it does not work for content which is far away from the local AS domain.

The goal of our approach is to provide incentives for ISPs to deploy detours. We first uncover AS relationships through BGP tables or from Route View servers such that we can identify provider-customer and peering relationships. Recent work has also shown that it is possible to infer the cost in a provider-customer relationship [81, 126]. We can find a detour path whose total transit cost is strictly equal to or potentially *lower* than the regular path.

We can leverage on our current Ukairo infrastructure to suggest candidate detours. Clients can then filter the candidate detours based on the transit costs taken for each detour. Thus, clients need to know the AS-topology of the Internet and the corresponding

AS relationships in order to find ISP-friendly detours. Fortunately, this information can be retrieved without generating a large overhead—in iPlane-Nano [85], they show that the Internet’s AS topology can be stored in a file that is less than 7 MB.

7.1.2 Towards Passive Measurements

Our current approach in Ukairo actively measures among detours to find TCP bandwidth information. By capturing the on-going TCP streams that use the detours, we can passively obtain measurements. This having the obvious advantage of not requiring to send any measurement probes.

We attempt to extract bandwidth measurements from both legs of the detour: client-to-detour and detour-to-destination. For the IP detouring mechanism (described in Chapter 5), the TCP bandwidth for either legs can be estimated by observing where packet losses occur on each leg of the path. This can be computed based on the amount of data the client sent (include packets that have been lost) which can be found at the client-side proxy, the amount of data the detour sent and the total amount of data transferred. Using the equation we derived for IP detouring in §5.3.1, we can estimate the TCP bandwidths of individual detour legs.

For the faster TCP detouring mechanism, we can only reveal the bandwidth of the leg that is congested. This can be done by observing the size of the buffer at the detour as suggested by Baccelli et al. [16] in his ns-2 experiments. Assuming a download using TCP, a large buffer implies that the client-detour leg is congested and a small buffer implies the detour-destination leg is congested. The reverse would be true for an upload stream.

Bibliography

- [1] Amazon CloudFront. <http://aws.amazon.com/cloudfront/>.
- [2] Amazon S3. <http://aws.amazon.com/s3/>.
- [3] Archipelago Measurement Infrastructure. <http://http://www.caida.org/projects/ark/>.
- [4] Bittorrent. <http://www.bittorrent.com>.
- [5] Coral CDN. <http://http://www.coralcdn.org>.
- [6] Dropbox. <http://www.dropbox.com>.
- [7] Emulab. <http://www.emulab.net>.
- [8] lp_solve 5.5.2.0. <http://lpsolve.sourceforge.net/5.5/>.
- [9] Measurement Lab. <http://www.measurementlab.net>.
- [10] Network Simulator (ns2). <http://www.isi.edu/nsnam/ns/>.
- [11] University of Oregon Route Views Project. <http://www.routeviews.org>.
- [12] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. A Stochastic Model of TCP/IP with Stationary Random Losses. *TON*, 13:356–369, 2005.
- [13] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *SOSP*, 2001.
- [14] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Rohit N. Rao. Improving Web Availability for Clients with MONET. In *NSDI*, 2005.
- [15] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding Traceroute Anomalies with Paris Traceroute. In *IMC*, 2006.
- [16] François Baccelli, Giovanna Carofiglio, and Serguei Foss. Proxy Caching in Split TCP: Dynamics, Stability and Tail Asymptotics. In *INFOCOM*, 2008.
- [17] François Baccelli and David R. McDonald. A stochastic model for the throughput of non-persistent TCP flows. *Perform. Eval.*, 65(6–7):512–530, 2008.
- [18] Suman Banerjee, Timothy G. Griffin, and Marcelo Pias. The Interdomain Connectivity of PlanetLab Nodes. In *PAM*, 2004.
- [19] Michael Borella, Debbie Swider, Suleyman Uludag, and Gregory B. Brewster. Internet Packet Loss: Measurement and Implications for End-to-End QoS. In *ICPPW*, 1998.
- [20] Tian Bu, Nick G. Duffield, Francesco Lo Presti, and Don Towsley. Network Tomography on General Topologies. In *SIGMETRICS*, 2002.
- [21] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling TCP latency. In *INFOCOM*, 2000.
- [22] Robert L. Carter and Mark E. Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. *Perform. Eval.*, 27-28:297–318, 1996.

- [23] Claudio Casetti, Mario Gerla, Saverio Mascolo, M. Y. Sanadidi, and Ren Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wirel. Netw.*, 8(5):467–479, 2002.
- [24] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *SOSP*, 2003.
- [25] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network Tomography: Recent Developments. *Statistical Science*, 19:499–517, 2004.
- [26] Qian Chen, Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Willinger. The Origin of Power Laws in Internet Topologies Revisited. In *INFOCOM*, 2002.
- [27] David Choffnes and Fabián E. Bustamante. On the Effectiveness of Measurement Reuse for Performance-Based Detouring. In *INFOCOM*, 2009.
- [28] David R. Choffnes and Fabián E. Bustamante. Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems. In *SIGCOMM*, 2008.
- [29] kc claffy, Tracie E. Monk, and Daniel McRobb. Internet tomography. In *Nature*, 1999.
- [30] Daniele Croce, Marco Mellia, and Emilio Leonardi. The Quest for Bandwidth Estimation Techniques for Large-Scale Distributed Systems. *SIGMETRICS*, 37:20–25, 2010.
- [31] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM*, 2004.
- [32] Constantine Dovrolis and Partha Kanuparth. ShaperProbe: End-to-end Detection of ISP Traffic Shaping using Active Methods. In *IMC*, 2011.
- [33] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet dispersion techniques and a capacity estimation methodology. *TON*, 12(6):963–977, 2004.
- [34] Allen B. Downey. Using Pathchar to Estimate Internet Link Characteristics. In *Measurement and Modeling of Computer Systems*, 1999.
- [35] Nick G. Duffield, Joesph Horowitz, Francesco Lo Presti, and Don Towsley. Explicit Loss Inference in Multicast Tomography. *Information Theory, IEEE Trans.*, 52(8):3852–3855, 2006.
- [36] Nick G. Duffield, Francesco Lo Presti, Vern Paxson, and Don Towsley. Network loss tomography using striped unicast probes. *TON*, 14(4):697–710, 2006.
- [37] Taoufik En-Najjary and Guillaume Urvoy-Keller. PPrate: A Passive Capacity Estimation Tool. In *E2EMON*, 2006.
- [38] Rik Farrow. Source address spoofing. <http://technet.microsoft.com/library/cc723706.aspx>.
- [39] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616), 1999.
- [40] Sally Floyd. HighSpeed TCP for Large Congestion Windows (RFC3649), 2003.
- [41] Marina Fomenkov, Ken Keys, David Moore, and kc claffy. Longitudinal study of Internet traffic in 1998-2003. In *WISICT*, 2004.
- [42] Lixin Gao. On Inferring Autonomous System Relationships in the Internet. *TON*, 9(6):733–745, 2001.
- [43] Denisa Ghita, Hung Nguyen, Maciej Kurant, Katerina Argyraki, and Patrick Thiran. Netscope: Practical Network Loss Tomography. In *INFOCOM*, 2010.
- [44] Philip Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet Routing. In *SIGCOMM*, 2009.

- [45] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. Assolo, a New Method for Available Bandwidth Estimation. In *ICIMP*, 2009.
- [46] Emanuele Goldoni and Marco Schivi. End-to-End Available Bandwidth Estimation Tools, an Experimental Comparison. In *TMA*, 2010.
- [47] Cesar D. Guerrero and Miguel A. Labrador. On the applicability of available bandwidth estimation techniques and tools. *Comput. Commun.*, 33(1):11–22, 2010.
- [48] Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *OSDI*, 2004.
- [49] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *IMW*, 2002.
- [50] Mehmet H. Gunes and Kamil Sarac. Inferring Subnets in Router-level Topology Collection Studies. In *IMC*, 2007.
- [51] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS*, 42:64–74, 2008.
- [52] Thom Haddow, Sing Wang Ho, Jonathan Ledlie, Cristian Lumezanu, Moez Draief, and Peter Pietzuch. On the Feasibility of Bandwidth Detouring. In *PAM*, 2011.
- [53] Thom Haddow, Sing Wang Ho, Cristian Lumezanu, Moez Draief, and Peter Pietzuch. UKAIRO: Internet-Scale Bandwidth Detouring. Technical Report DTR11-2, Imperial College London, 2011.
- [54] Sing Wang Ho and Moez Draief. tcpChiryo: TCP bandwidth Inference using Loss Tomography. *pre-print*, 2012.
- [55] Sing Wang Ho, Thomas Haddow, Jonathan Ledlie, Moez Draief, and Peter Pietzuch. Deconstructing Internet Paths: An Overlay for AS-Level Detour Route Discovery. In *IPTPS*, 2009.
- [56] Ningning Hu and Peter Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *Selected Areas in Communications, IEEE Journal on*, 21(6):879–894, 2003.
- [57] Manish Jain and Constantine Dovrolis. Path selection using available bandwidth estimation in overlay-based video streaming. *Comput. Netw.*, 52:2411–2418, 2008.
- [58] Manish Jain and Constantinos Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *TON*, 11(4):537–549, 2003.
- [59] Manish Jain and Constantinos Dovrolis. Ten Fallacies and Pitfalls on End-to-End available Bandwidth Estimation. In *IMC*, 2004.
- [60] Wolfgang John and Sven Tafvelin. Analysis of Internet Backbone Traffic and Header Anomalies observed. In *IMC*, 2007.
- [61] Seong-Ryong Kang, Xiliang Liu, Amit Bhati, and Dmitri Loguinov. On Estimating Tight-Link Bandwidth Characteristics over Multi-Hop Paths. In *ICDCS*, 2006.
- [62] Sachin Katti, Dina Katabi, Charles Blake, Eddie Kohler, and Jacob Strauss. MultiQ: Automated Detection of Multiple Bottleneck Capacities Along a Path. In *IMC*, 2004.
- [63] Ethan Katz-Bassett, Harsha V. Madhyastha, Vijay Kumar Adhikari, Colin Scott, Justine Sherry, Peter van Wesep, Thomas E. Anderson, and Arvind Krishnamurthy. Reverse traceroute. In *NSDI*, 2010.
- [64] Ken Keys. Internet-Scale IP Alias Resolution Techniques. *SIGCOMM CCR*, 40(1):50–55, 2010.
- [65] Ron Kohavi and Roger Longbotham. Online Experiments: Lessons Learned. *Computer*, 40(9):103–105, 2007.
- [66] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *SOSP*, 2003.

- [67] Yann Labit, Philippe Owezarski, and Nicolas Larrieu. Evaluation of Active Measurement Tools for Bandwidth Estimation in Real Environment. In *E2EMON*, 2005.
- [68] Kevin Lai and Mary Baker. Measuring Link Bandwidths Using a Deterministic Model of Packet Delay. In *SIGCOMM*, 2000.
- [69] Kevin Lai and Mary Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *USITS*, 2001.
- [70] Zhaohong Lai, Chris Todd, and Miguel Rio. Pathpair: a fast available bandwidth estimation tool with asymptotic one way delay comparison model. *IET Commun.*, 3(6):967–978, 2009.
- [71] Raul Landa, Eleni Mykoniati, Richard G. Clegg, and Miguel Rio. Modelling the Tradeoffs in Overlay-ISP Cooperation. In *IFIP Networking*, 2012.
- [72] Li Lao, Constantine Dovrolis, and M. Y. Sanadidi. The Probe Gap Model can Underestimate the Available Bandwidth of Multihop Paths. *SIGCOMM CCR*, 36:29–34, 2006.
- [73] Jonathan Ledlie, Margo Seltzer, and Peter Pietzuch. Proxy Network Coordinates. Technical Report Technical Report 2008/4, Imperial College London, 2008.
- [74] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On Suitability of Euclidean Embedding of Internet Hosts. In *SIGMETRICS*, 2006.
- [75] Sung-Ju Lee, Sujata Banerjee, Puneet Sharma, Praveen Yalagandula, and Sujoy Basu. Bandwidth-Aware Routing in Overlay Networks. In *INFOCOM*, 2008.
- [76] Marcus Leech, Matt Ganis, Ying-Da Lee, Ron Kuris, David Koblas, and LaMont Jones. SOCKS Protocol Version 5 (RFC 1928).
- [77] Tom Leighton. Improving Performance on the Internet. *Commun. ACM*, 52:44–51, 2009.
- [78] Hyuk Lim, Jennifer C. Hou, and Chong-Ho Choi. Constructing Internet Coordinate System Based on Delay Measurement. *TON*, 13(3):513–525, 2005.
- [79] Matthew Luckie, Young Hyun, and Bradley Huffaker. Traceroute Probe Method and Forward IP Path Inference. In *IMC*, 2008.
- [80] Cristian Lumezanu, Randy Baden, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Symbiotic Relationships in Internet Routing Overlays. In *NSDI*, 2009.
- [81] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle Inequality and Routing Policy Violations in the Internet. In *PAM*, 2009.
- [82] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle Inequality Variations in the Internet. In *IMC*, 2009.
- [83] Cristian Lumezanu, Dave Levin, and Neil Spring. PeerWise Discovery and Negotiation of Faster Paths. In *HotNets*, 2007.
- [84] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An Information Plane for Distributed Services. In *OSDI*, 2006.
- [85] Harsha V. Madhyastha, Ethan Katz-Bassett, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane Nano: Path Prediction for Peer-to-Peer Applications. In *NSDI*, 2009.
- [86] Yun Mao and Lawrence K. Saul. Modeling Distances in Large-scale Networks by Matrix Factorization. In *IMC*, 2004.
- [87] Zhuoqing Morley Mao, Jennifer Rexford, Jia Wang, and Randy H. Katz. Towards an Accurate AS-Level Traceroute Tool. In *SIGCOMM*, 2003.

- [88] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM CCR*, 27(3):67–82, 1997.
- [89] Bob Melander, Mats Björkman, and Per Gunningberg. A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In *GLOBECOM*, 2000.
- [90] Vishal Misra, Wei-Bo Gong, and Don Towsley. Stochastic Differential Equation Modeling and Analysis of TCP-Window Size Behavior. Technical Report ECE-TR-CCS-99-10-01, University of Massachusetts, 1999.
- [91] Jiri Navratil and R. Les. Cottrell. ABwE: A Practical Approach to Available Bandwidth Estimation. In *PAM*, 2003.
- [92] Mradula Neginhal, Khaled Harfoush, and Harry Perros. Measuring Bandwidth Signatures of Network Paths. In *IFIP Networking*, 2007.
- [93] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, 2002.
- [94] Hung X. Nguyen and Patrick Thiran. Network Loss Inference with Second Order Statistics of End-to-End Flows. In *IMC*, 2007.
- [95] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The Akamai Network: A Platform for High-Performance Internet Applications. *SIGOPS*, 44:2–19, 2010.
- [96] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *TON*, 8(2):133–145, 2000.
- [97] Venkata N. Padmanabhan, Lili Qiu, and Helen J. Wang. Server-based Inference of Internet Link Lossiness. In *INFOCOM*, 2003.
- [98] Konstantina Papagiannaki, Sue Moon, Chuck Fraleigh, Patrick Thiran, Fouad Tobagi, and Christophe Diot. Analysis of Measured Single-Hop Delay from an Operational Backbone Network. In *INFOCOM*, 2002.
- [99] Marcelo Pias, Jon Crowcroft, Steve Wilbur, Tim Harris, and Saleem Bhatti. Lighthouses for Scalable Distributed Location. In *IPTPS*, 2003.
- [100] Michael Piatek, Harsha V. Madhyastha, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Pitfalls for ISP-friendly P2P design. In *HotNets*, 2009.
- [101] Fabio Picconi and Laurent Massoulie. ISP Friend or Foe? Making P2P Live Streaming ISP-Aware. In *ICDCS*, 2009.
- [102] Ravi Prasad, Manish Jain, and Constantinos Dovrolis. Effects of Interrupt Coalescence on Network Measurements. In *PAM*, 2004.
- [103] Barath Raghavan and Alex C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *SIGCOMM*, 2004.
- [104] Vinay J. Ribeiro, Mark Coates, Rudolf Riedi, Shriram Sarvotham, Brent Hendricks, and Richard Baraniuk. Multifractal cross-traffic estimation. In *ITC Conference on IP Traffic, Modeling and Management*, 2000.
- [105] Vinay J. Ribeiro, Rudolf Riedi, Richard Baraniuk, Jiri Navratil, and R. Les. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM*, 2003.
- [106] Vinay J. Ribeiro, Rudolf H. Riedi, and Richard G. Baraniuk. Locating Available Bandwidth Bottlenecks. *IEEE Internet Computing*, 8(5):34–41, 2004.
- [107] 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.

- [108] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The End-to-End Effects of Internet Path Selection. In *SIGCOMM*, 1999.
- [109] Yubal Shavitt and Tomer Tankel. Big-bang simulation for embedding network distances in Euclidean space. In *INFOCOM*, 2003.
- [110] Yuval Shavitt and Eran Shir. DIMES: Let the Internet Measure Itself. *SIGCOMM CCR*, 35(5):71–74, 2005.
- [111] Rob Sherwood, Adam Bender, and Neil Spring. Discarte: A Disjunctive Internet Cartographer. In *SIGCOMM*, 2008.
- [112] Rob Sherwood and Neil Spring. Touring the Internet in a TCP Sidecar. In *IMC*, 2006.
- [113] Alok Shriram, Margaret Murray, Young Hyun, Nevil Brownlee, Andre Broido, Marina Fomenkov, and kc claffy. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *PAM*, 2005.
- [114] Biplab Sikdar, Shivkumar Kalyanaraman, and Kenneth S. Vastola. Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno, and SACK. *TON*, 11(6):959–971, 2003.
- [115] Han Hee Song, Lili Qiu, and Yin Zhang. NetQuest: A Flexible Framework for Large-Scale Network Measurement. In *SIGMETRICS*, 2006.
- [116] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP Topologies with Rocketfuel. *TON*, 12:2–16, 2004.
- [117] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *IMC*, 2003.
- [118] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. In *INFOCOM*, 2002.
- [119] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. OverQos: An Overlay based Architecture for Enhancing Internet Qos. In *NSDI*, 2004.
- [120] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. Technical Report MSR-TR-2005-86, Microsoft Research, 2005.
- [121] Liying Tang and Mark Crovella. Virtual Landmarks for the Internet. In *IMC*, 2003.
- [122] Team Cymru. <http://www.team-cymru.org>.
- [123] Ajay Tirumala, Les Cottrell, and Tom Dunigan. Measuring end-to-end bandwidth with Iperf using Web100. In *PAM*, 2003.
- [124] Tomoaki Tsugawa, Cao Le Thanh Man, Go Hasegawa, and Masayuki Murata. Inline bandwidth measurements: Implementation difficulties and their solutions. In *E2EMON*, 2007.
- [125] Gillaume Urvoy-Keller, Taoufik En-Najjary, and Alessandro Sorniotti. Operational Comparison of Available Bandwidth Estimation Tools. *SIGCOMM CCR*, 38:39–42, 2008.
- [126] Vytas Valancius, Cristian Lumezanu, Nick Feamster, Ramesh Johari, and Vijay V. Vazirani. How Many Tiers? Pricing in the Internet Transit Market. In *SIGCOMM*, 2011.
- [127] Yehuda Vardi. Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data. *Journal of the American Statistical Association*, 91(433):365–377, 1996.
- [128] Guohui Wang, Bo Zhang, and T. S. Eugene Ng. Towards Network Triangle Inequality Violation Aware Distributed Systems. In *IMC*, 2007.
- [129] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *TON*, 14(6):1246–1259, 2006.

- [130] Damon Wischik, Costin Raiciu, Adam Greenhalgh, , and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *NSDI*, 2011.
- [131] Bowei Xi, George Michailidis, and Vijayan N. Nair. Estimating Network Loss Rates Using Active Tomography. *Journal of the American Statistical Association*, 101:1430–1438, 2006.
- [132] Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4P: Provider Portal for Applications. In *SIGCOMM*, 2008.
- [133] Wen Xu and Jennifer Rexford. MIRO: Multi-path Interdomain Routing. In *SIGCOMM*, 2006.
- [134] Xiaowei Yang, David Clark, and Arthur W. Berger. NIRA: A New Inter-Domain Routing Architecture. *TON*, 15(4):775–788, 2007.
- [135] 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.
- [136] Yin Zhang, Nick Duffield, Vern Paxson, and Scott Shenker. On the Constancy of Internet Path Properties. In *IMC*, 2001.
- [137] Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network Anomography. In *IMC*, 2005.
- [138] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy G. Griffin. Internet Routing Policies and Round-Trip-Times. In *PAM*, 2005.
- [139] Dapeng Zhu, Mark Gritter, and David R. Cheriton. Feedback Based Routing. *SIGCOMM CCR*, 33(1):71–76, 2003.
- [140] Yong Zhu, Constantinos Dovrolis, and Mostafa Ammar. Dynamic overlay routing based on available bandwidth estimation: A simulation study. *Comput. Netw.*, 50:742–762, 2006.