# Efficient Internet Topology Discovery Techniques

A thesis

submitted in partial fulfilment

of the requirements for the degree

of

Master of Science

at

The University of Waikato

by

**Alistair King**


THE UNIVERSITY OF
WAIKATO
*Te Whare Wānanga o Waikato*

February 3, 2010

# Abstract

Current macroscopic Internet topology discovery projects use large numbers of vantage points to conduct traceroute surveys of Internet paths. These projects send billions of unsolicited packets to millions of routers within the Internet. Due to the structure of the Internet, many of these packets are sent without gaining any new topology information. In this thesis, we implement and extensively test a large-scale doubletree system designed to increase the efficiency of topology mapping projects and reduce the load that they place on the Internet. Also, for all of the effort that current projects put into gathering data, the methods used do not discover, with confidence, the entire set of paths. We propose, implement and critique a novel algorithm, economical MDA traceroute, which is designed to discover a comprehensive topology in a manner which is more efficient than the current state of the art. We show that, compared to current methods, well over 90% link coverage can be obtained while reducing the number of probes used by over 60%. We also evaluate alternate methods for making large scale topology discovery projects more efficient and comprehensive; such as using BGP routing data to guide probing.

# Acknowledgements

The author would like to thank the following people for their contributions to this project:

Dr Matthew Luckie - for supervising and providing invaluable contributions and oversight to the project.

WAND Group - for providing all sorts of miscellaneous help such as feedback and advice on drafts and support with LaTeX.

CAIDA - for providing helpful advice and access to their Archipelago measurement infrastructure to conduct testing with (DHS S&T Cybersecurity Contract N66001-08-C-2029 and NSF grant CRI-0551542).

The Waikato University Masters Research Scholarship - for their funding contribution.

New Zealand Foundation for Research Science and Technology (FRST) contract UOWX0705 - for their funding contribution.

Thanks also to friends and family who have provided support in a multitude of ways, without which, this project would not have been possible.

# Contents

# List of Acronyms

**AS** Autonomous System

**ALSS** Adaptive Local Stop Set

**BGP** Border Gateway Protocol

**CAIDA** the Cooperative Association for Internet Data Analysis

**DNS** Domain Name Service

**ICMP** Internet Control Measurement Protocol

**IP** Internet Protocol

**IPID** Internet Protocol Identifier

**IPv4** Internet Protocol version 4

**ISP** Internet Service Provider

**MDA** Multipath Detection Algorithm

**TCP** Transmission Control Protocol

**TOS** Type Of Service

**TTL** Time To Live

**UDP** User Datagram Protocol

**WAND** Waikato Applied Network Dynamics

# Chapter 1

# Introduction

## 1.1 The Problem

Data collected by macroscopic Internet topology discovery projects is used in a number of ways by Internet researchers. For example, Faloutsos *et al.* use traceroute data to determine that the Internet structure exhibits power-law characteristics [12]. RadarGun uses traceroute to produce router-level Internet topologies [6]. Leskovec *et al.* use topology data to observe graph densification occurring in the Internet [21]. Due to the commercial nature of the majority of the networks that comprise the Internet, topology details are considered sensitive and thus are not readily available. The Internet's topology must therefore be inferred by probing. Probing, in the context of this thesis refers to the process of sending specially crafted packets, known as probes, into the Internet to elicit responses from the routers the forwarding path. Because Internet routing differs depending on geographic location, mapping projects use a large number of geographically distributed *vantage points* to generate comprehensive data. This combined with the sheer scale of the Internet Protocol (IP) address space means that inferring accurate, complete topologies is a difficult task. There has however, been significant criticism of the methods by which data is currently collected. Lakhina *et al.* show that the power-law observations made in [12] are perhaps an artifact of the measurement process, rather than the underlying topology [20]. Others have also investigated the effects of over-sampling the edges of the Internet [2, 13, 28, 34]. It is therefore vital for collection methods to be improved so the underlying topology is represented more accurately.

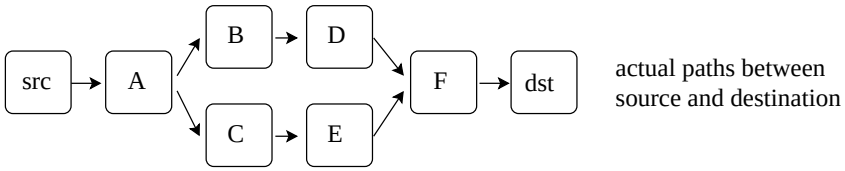



Figure 1.1: An example topology caused by load balancing. There are two paths that can be observed to the destination; A-B-D-F and A-C-E-F.

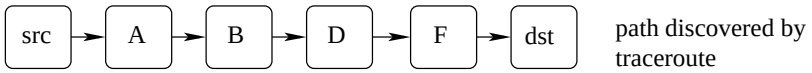



Figure 1.2: An example path discovered by traceroute when tracing the topology shown in Figure 1.1. Note that only one of the paths are discovered, the interfaces C and E are missed by this trace.

Macroscopic topology discovery projects such as CAIDA's Skitter [14] and Ark [7], DIMES [31] and iPlane [24] use a specialised version of the traceroute algorithm which allows them to discover the IP paths to large numbers of destination IP addresses from many vantage points. Traceroute is an algorithm that exploits a feature of the Internet Protocol. It works by sending packets which are designed to expire at a known distance into the network. When a packet expires in the Internet, RFC 792 [29] says the router should send an Internet Control Measurement Protocol (ICMP) time exceeded message to the sender. By causing these packets to expire at successive routers, a path can be incrementally discovered using the source address of the time exceeded messages.

There are two major flaws to this approach which this thesis addresses. When tracing a large number of destinations from a single vantage point, a tree, rooted at the vantage point, is observed. The effect of this is that the nodes toward the root of the tree are traversed repeatedly. For a set of destinations traced, there are a substantial number of probes that are sent without gaining any new information. The doubletree [9, 10, 11] algorithm proposed by Donnet *et al.* addresses this problem by remembering encountered topology to reduce redundant probing. This thesis investigates several issues surrounding this algorithm in macroscopic Internet topology discovery projects.

The other major problem that macroscopic topology discovery projects face is due to conventional traceroute algorithms not discovering a comprehensive topol-

ogy. This is because there is rarely only one path between two points in the Internet [5]. To provide redundancy in case of network failures, network operators will often provision alternate links. In order to distribute traffic over these alternate links, network operators use a technique known as load balancing. Load balancing routers forward packets out over two or more outgoing links. The implications of this is that because regular traceroute only discovers one path between the source and destination, it will miss any alternate paths. For example, Figure 1.1 shows a hypothetical set of paths between a source and destination. When traceroute is used to conduct a trace to the destination, it only discovers the single path shown in Figure 1.2, missing the alternate path.

These two problems efficiency and comprehensiveness, form the major thrust of this thesis. The goal is to create algorithms which allow a comprehensive topology to be reliably discovered while minimising the number of probes sent. There is an inherent tension between these two goals however: in making the topology discovery more comprehensive, we must send more probes to ensure that all the potential paths have been seen. However, in making the topology discovery more efficient, we make assumptions about how the networks are structured in order to avoid re-probing sections of the path which have been previously discovered. This creates the potential for topology to be missed. There is no perfect solution to this problem; however the algorithms we present make improvements to the current state of the art. We also investigate alternate sources of information with which we can guide probing, such as Border Gateway Protocol (BGP) routing data to reduce probes sent.

## 1.2 Thesis Overview

This chapter outlines the problems that the thesis addresses and how the thesis has made contributions to the field of macroscopic Internet topology discovery. The next chapter provides some background information which allows the thesis to be read in the right context.

In Chapter 3 we present the economical Multipath Detection Algorithm (MDA)

traceroute algorithm that we have developed. This algorithm allows high-confidence topology discovery like MDA traceroute [4] but remembers the interfaces previously observed by other vantage points so that it can minimise the number of probes that it sends. We discuss the motivations behind the design and the details of the algorithm. Chapter 4 presents an in-depth comparison of various large scale probing methods. We have carried out several experiments using variations of traceroute, doubletree, MDA traceroute and economical MDA traceroute. The results from these experiments are presented, and the various methods compared and discussed. Chapter 5 addresses the issues that arise when conducting large scale measurements using a cooperative probing scheme as with doubletree. We present a method and implementation which is able to cope with vantage points which probe at different speeds to one another, thus improving the overall efficiency of the system. Chapter 6 is a discussion about the local stop set in doubletree. We have conducted extensive experimentation and simulation regarding the optimal lifetime of the local stop set. In Chapter 7 we discuss using BGP routing data to influence probing. We investigate using BGP routing information to infer path lengths such that tracing can specifically target sections of paths which have not yet been probed thoroughly. Chapter 8 concludes the thesis and outlines areas which are open to further research.

## 1.3 Contributions

This thesis provides a number of contributions to the field of macroscopic Internet topology discovery.

The most notable of these is the conception, development and implementation of an algorithm which provides confidence that a complete topology has been discovered whilst reducing the number of probes that must be sent when compared to the algorithm proposed in [5] by Augustin *et al.*. This algorithm collects information regarding the destinations to which a node has been seen in the path to. This allows MDA traceroute to remain in an economical tracing mode while it can determine that a section of path has been probed enough to give confidence that all of the topology has been discovered. Testing of this algorithm on a production topol-

ogy discovery platform (Archipelago) found that it was able to maintain around 90% link coverage while reducing the number of probes sent by 64% compared with MDA traceroute. Thus significantly reducing both the time used and amount of unsolicited traffic generated. We also investigate the causes of the undiscovered topology. We discover that a previously unidentified type of load balancing, per-source/destination is responsible for a large portion of this topology loss.

In addition to this, the existing topology discovery algorithm, doubletree [11, 9, 10] has been implemented within scamper [22], the active measurement software used by the Cooperative Association for Internet Data Analysis (CAIDA) on their Archipelago platform. Doubletree improves the efficiency of the regular traceroute algorithm by collecting information about previously discovered topology which allows it to stop probing when a known path is encountered. Although doubletree has been previously tested in simulation, it has never been implemented in a way such that it can be used in a real-world topology discovery project. This thesis describes the challenges involved in implementing a distributed system which allows large-scale collection of data using the doubletree method. Substantial analysis has been conducted on this system and on the data that it produces, such as how often the information held should be cleared and how to deal with under-performing vantage points.

Another area of research that the thesis describes is the use of BGP routing data to influence and guide active measurement algorithms. BGP data provides information about the Autonomous System (AS) path to a given destination. We investigate how to best use the AS path information to build an estimated width of each AS. We propose and simulate an method which is able to learn these widths and use them to infer at the IP level, the sections of the path which have not been previously traced.

# Chapter 2

# Background

## 2.1 Introduction

This chapter introduces three active probing techniques which we leverage to improve the efficiency and coverage of macroscopic Internet topology discovery projects. The first of these - traceroute - is the method currently used in macroscopic topology discovery projects, however it is inefficient and does not gather a comprehensive topology. The other two algorithms - doubletree and MDA - aim to improve the efficiency and coverage respectively, of traceroute.

This chapter also discusses the topic of macroscopic Internet topology discovery and defines two coordination methods - team probing and cooperative probing - which are used to coordinate large sets of vantage points. Team probing divides the work amongst participating vantage points such that each vantage point only traces a fraction of the overall destination list. However, with cooperative probing, all vantage points trace all destinations, but share information about topology they have discovered to help reduce the work that the other vantage points must do.

Finally, we introduce techniques for alias resolution. Alias resolution involves reducing the interface-level data obtained by traceroute, where each node represents an interface on a router, to a router-level graph such that each node represents a router.

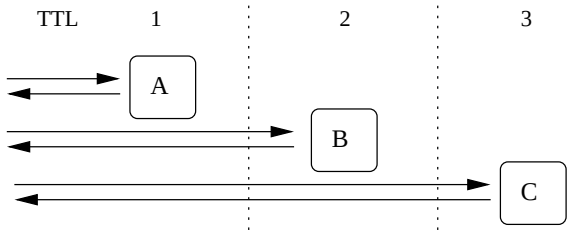



Figure 2.1: an example of how the ICMP time exceeded messages can be used to infer a path. A packet is sent with a TTL of one so that it expires at A. A replies with an ICMP time exceeded message, the source address (A) of the message is used to identify the router. This process is continued for TTLs two and three, discovering routers B and C. The links are inferred between the discovered interfaces, to give a path of A-B-C.

## 2.2 Traceroute

Traceroute [18] is perhaps the most ubiquitous per-hop Internet measurement technique. Traceroute works by exploiting an IP feature which allows a response to be elicited from a router at a known distance into the network. This is achieved by manipulating the Time To Live (TTL) field in the IP header of a packet which is then sent toward the destination of a path to be inferred. The TTL field is supposed to be used to prevent a packet from never exiting a routing loop. Each router that a packet visits on a path decrements the field by one. When the value reaches one, the packet expires; the router at which it expires sends an ICMP *Time Exceeded* packet back to the source telling the source that the packet did not make it to the destination.

Traceroute uses the TTL field to elicit a response from a router at a defined distance (in hops) into the network. The response contains the address of an interface at the router at which the packet expired. To build up a view of the path between a source and a given destination, traceroute starts by sending a probe packet with a TTL of one, records the address of the responding router and then sends a probe with a TTL of two and so on until a response is received from the address that it was probing toward as shown in Figure 2.1. There are also other conditions which traceroute stops on depending on the implementation. For example, if a routing loop is detected or a series of hops do not respond to probes. Many probes must be sent into the network because traceroute needs to send at least one probe per hop in order to discover a complete path.

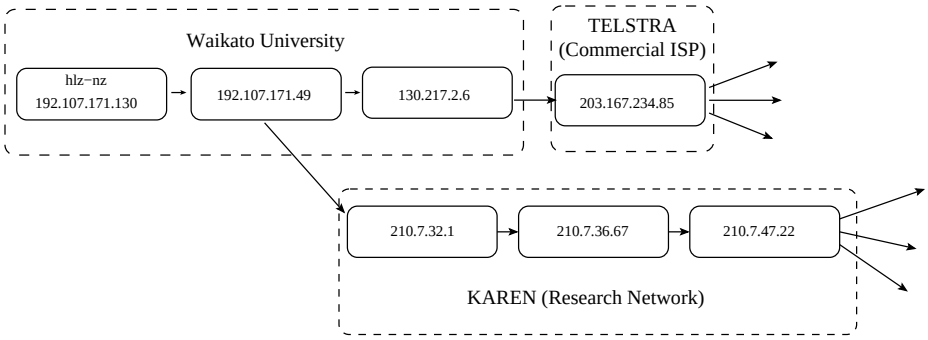



Figure 2.2: A real-world example of outgoing links from the hlz-nz Ark vantage point. The majority of destinations traced will traverse the commercial path, with a small number being routed over KAREN. These six routers are repeatedly traversed by Ark.

## 2.3 Efficient Topology Discovery

As the number of vantage points in a macroscopic Internet topology discovery project are scaled up - from a single vantage point tracing a path to a single destination - to a set of vantage points all tracing the paths to millions of destinations, the amount of time and resources used is also scaled up. To discover topology, traceroute must actively generate packets and send them unsolicited to destinations in the Internet. When a large-scale system is being used to trace a large number of destinations, these unsolicited packets can appear, to network operators, to be an attack on their infrastructure. This could result in complaints, or modification of firewall rules to prevent their routers from responding. Therefore it is in the interests of all parties involved to minimise the number of probes used.

Due to the tree-like structure of the Internet, traces to several destinations must repeatedly traverse the same outgoing links over and over. For example, the path out of The University of Waikato's network and then the local ISPs, as seen in Figure 2.2, is such that the first four hops in almost every trace are the same, with many traces having much longer sequences of hops in common. The effect of this is a large number of wasted probes; the first time a path is probed new topology is discovered, but with subsequent traces some probes are sent without gaining any new information.

This effect can also be observed when several vantage points are tracing toward a single destination. As the paths converge toward the destination, probes are sent

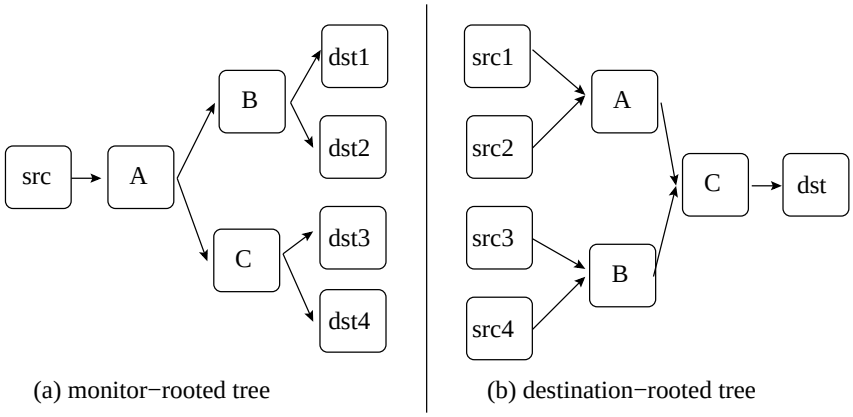



Figure 2.3: Internet path trees. (a) is an example of the tree which is seen when probing multiple destinations from a single vantage point. (b) is an example of the tree seen when probing a single destination from multiple vantage points.

without any new topology being discovered. Figure 2.3a is a simplified example of a tree of paths which are rooted at a single vantage point. As it traces multiple destinations, the paths diverge. Figure 2.3b demonstrates the tree rooted at a destination which has been traced by multiple vantage points.

Doubletree [9, 10, 11], a variation on the regular traceroute algorithm, addresses this problem and thus increases the efficiency of macroscopic Internet topology discovery. Doubletree assumes that Internet paths form two distinct trees; one rooted at the vantage point when probing multiple destinations, and the other rooted at the destination when multiple vantage points are being used. In order to reduce redundant probing, doubletree begins a trace by sending probes to a mid-point in the path to the destination. The mid-point is chosen to avoid the first probe reaching the destination most of the time. The authors of doubletree suggest a value for which the destination is not reached between 80% and 95% of the time. This maximises the efficiency while reducing the probability of appearing like a distributed denial of service attack on the destination which could occur if all vantage points send a probe which reaches the destination [11]. From the mid-point, the doubletree algorithm then proceeds to follow the algorithm of regular traceroute, increasing the TTL value of the probe packets, and enumerating each of the interfaces in the path.

The difference between doubletree and traceroute is that while probing, doubletree keeps a record of the interfaces encountered. This allows it to halt probing

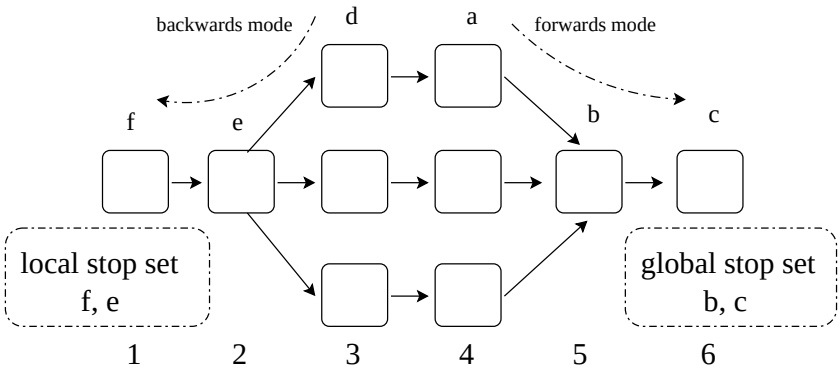



Figure 2.4: An example doubletree trace. Tracing begins at TTL 4. Assuming that the interfaces $b$ and $c$ are in the global stop set, forwards probing will discover $a$ and then $b$, at which point it will switch to backward probing as $b$ is in the global stop set. Once in the backward probing mode, $d$ will be discovered, followed by $e$, which, because it is in the local stop set, will cause probing to halt.

when a reply is received from an interface which has been discovered by another vantage point which has probed the same destination. This collection of previously encountered interfaces is known as the *global stop set*. The assumption is that the sequence of hops from an observed interface to the destination is the same for all vantage points. Once doubletree has found a stop condition for forward probing, whether that be by reaching the destination, discovering an address in the global stop set, or one of the other stop conditions that traceroute observes, it switches into a backward probing mode where it begins probing at a TTL just before the mid-point it started at. While in backward probing mode, it decrements the TTL by one with each probe, thus allowing it to effectively trace backward towards the vantage point. Tracing is halted when an interface is discovered which is a part of the *local stop set*. The local stop set is a record of all of the interfaces seen by a particular vantage point while probing. The assumption with the local stop set is that the sequence of hops from an interface back to a vantage point is the same for every trace from that vantage point.

Figure 2.4 gives an example of this process. In this example, the mid-point used is four. Probing begins in the forward probing mode, which discovers $a$ at TTL four. As $a$ is not in the global stop set, the TTL will then be incremented by one and $b$ discovered. Because $b$ is in the global stop set, forwards probing will halt and backwards probing begins at TTL three. The reply from TTL three will reveal

$d$. Because $d$ is not in the local stop set, the probe TTL will be decremented and $e$ discovered. As $e$ is in the local stop set, probing will halt.

The doubletree authors claim that the algorithm allows a reduction in measurement load of approximately 76% whilst maintaining link coverage of over 90% [11]. It should be noted that these results are based on simulations using skitter [14] data. This is revisited with data collected by our implementation of doubletree in Chapter 4.

## 2.4 Comprehensive Topology Discovery

The topology discovery algorithms and techniques described thus far have one common failing: they all operate on the assumption that there is a single path between a source and a destination. Internet network operators make use of a routing technology known as load balancing which allows routers to distribute traffic over multiple outgoing links. The result of this is the potential for there to be a number of different valid paths to any given destination which traceroute may not discover.

To understand the impact of this fully, we first need to describe the three methods that routers use to manage load balancing. Augustin [3] describes these as per-flow, per-destination, and per-packet. Per-flow load balancing forward packets in such a way that a flow of packets is forwarded out over the same link. A flow is based on several fields in the IP and either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) headers. These are the Source and Destination Addresses and Ports, Protocol and also a varying combination of the IP Type Of Service (TOS), ICMP Code and Checksum fields. Per-destination load balancing is a coarser version of per-flow load balancing where only the Destination Address in the IP header is used for load balancing. Per-packet load balancing is focused solely on distributing traffic evenly over outgoing links. Thus per-packet load balancing pays no attention to the information in the packet headers. In order to enumerate all paths between a source and a destination, the traceroute algorithm must be able to manipulate probe packets so that it can accurately probe all of the branches in the path caused by load balancing routers.

| $n$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 95% | 6 | 11 | 16 | 21 | 27 | 33 | 38 | 44 | 51 |
| 99% | 8 | 15 | 21 | 28 | 36 | 43 | 51 | 58 | 66 |

| $n$ | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|
| 95% | 57 | 63 | 70 | 76 | 83 | 90 | 96 | 103 | 110 |
| 99% | 74 | 82 | 90 | 98 | 106 | 115 | 123 | 132 | 140 |

Table 2.1: The number of probes that must be sent to a given TTL to rule out $n$ interfaces

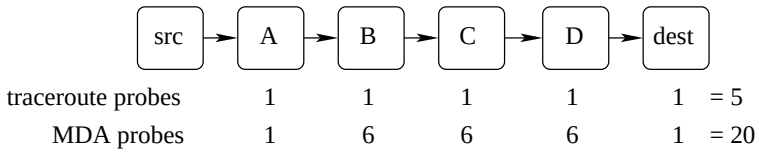



Figure 2.5: A simple path. Traceroute would use five probes to infer the topology whereas the MDA would uses twenty to find all interfaces. We assume that the source does not have any load balanced outgoing paths, and that the router with interface D will always choose a directly connected path, so that only one probe is required.

The MDA proposed by Augustin [4] is able to enumerate load balanced paths by varying the *flow identifier* (the fields a router consults when making a forwarding decision) of probe packets in a controlled manner such that there can be statistical confidence that the returned topology contains all of the possible paths. A large portion of our work is derived from the MDA. The MDA begins by assuming that at any given TTL there are two interfaces (that is, the router at the TTL before this has two outgoing links) and sends enough probes to test this hypothesis assuming even load balancing. If a second interface is indeed discovered, the algorithm then proceeds to test the hypothesis that there are three interfaces, and so on until the final hypothesis has been disproved at the specified confidence level. A portion of the look-up table used to determine the appropriate number of probes can be seen in Table 2.1. For example, to rule out that there are two interfaces at a TTL to 95% confidence, six probes must be sent, with the same interface returned for each probe.

The MDA must therefore send many more probes than regular traceroute. Figure 2.5 shows an example path in which there are five hops. Traceroute would send

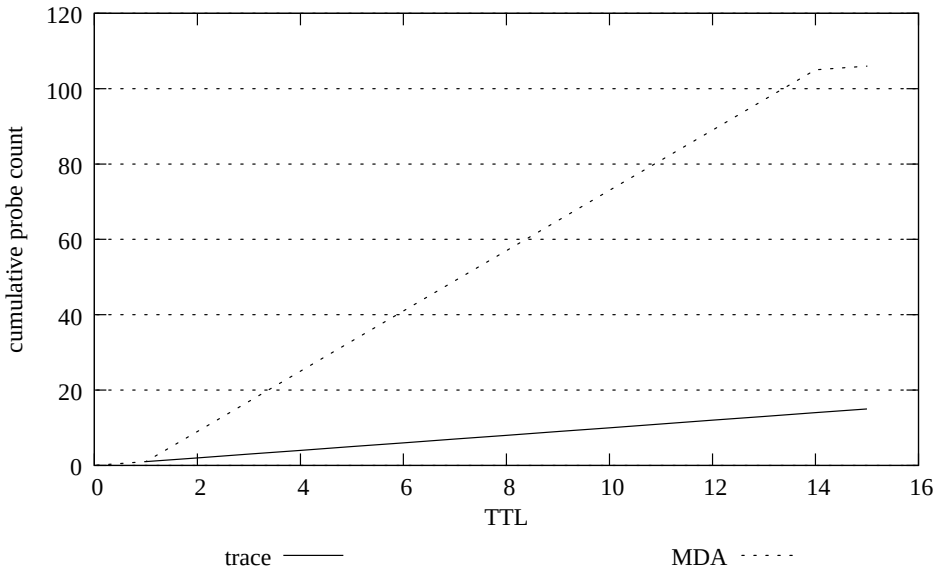



Figure 2.6: A best-case example of cumulative probe usage by traceroute and MDA traceroute for a path with 15 hops. The MDA traceroute is using a 99% confidence level and so must send at least eight probes to most TTLs.

five probes whereas the MDA would have to send at least 30 probes to discover the same path to 95% confidence. Figure 2.6 illustrates how quickly the number of probes used increases with longer path lengths. By the time a 15 hop path is reached, traceroute has sent 15 probes whereas MDA traceroute has sent at least 106 probes to rule out any alternate paths. This matter is discussed further in Chapter 3 where we present a version of this algorithm which is designed to be more economical than the original MDA specified by Augustin *et al.* [4].

## 2.5 Macroscopic Internet Topology Discovery

Having accurate knowledge about the structure at the IP level of the Internet is useful to researchers who can use it to model and analyse Internet routing. Collecting the data is a non-trivial task however. Most of the networks that comprise the Internet are maintained by private organisations, so there are few organisations that publish detailed information regarding the structure of their networks. Because of this, the structure must be inferred using tools such as traceroute. Since traceroute determines a single path from a source to a destination, a common way to gather an Internet-wide topology is to use a globally distributed set of vantage points [30],

each of which carries out traces to a representative sample of all of the end points in the Internet [7, 14, 24, 25, 31].

### 2.5.1 Coordination Methods

There are two main types of coordination methods; *team probing* and *cooperative probing*. Team probing works by splitting the destinations to be traced amongst the vantage points such that each vantage point only traces a portion of the overall list.

CAIDA's Ark infrastructure, for example, makes use of two teams of thirteen vantage points. Each team traces one random address in every routed /24 BGP prefix. The addresses to be probed are divided up amongst the vantage points. Thus the time required to trace the entire address space is approximately divided by the number of vantage points involved. It takes approximately 48 hours for a team of 13 vantage points to trace one destination in each routed /24 Internet Protocol version 4 (IPv4) prefix. Once this is completed, the process begins again with a fresh set of addresses generated from the prefix list. Ark has been running since 2007 and has, as of December 2009, collected over 5.18 billion traceroutes [15].

Cooperative probing, on the other hand, is designed so that all vantage points trace all destinations. The advantage of cooperative probing is that vantage points are able to share information regarding topology that has already been discovered, thus reducing the burden on other vantage points and the networks being examined.

## 2.6 Alias Resolution

Routers have more than one interface, as by definition, a router must be connected to more than one network. Each interface has at least one IP address assigned to it. This causes problems when discovering paths with traceroute, because traceroute discovers interfaces. In other words, maps derived from traceroute data may contain more nodes than actually exist in the Internet. *Alias resolution* is the process of folding the multiple IP addresses of a router into a single node in the map. Figure 2.7 shows how traceroute can infer a path which has more routers than actually exist, but once alias resolution is performed, interfaces can be assigned to the router topology.

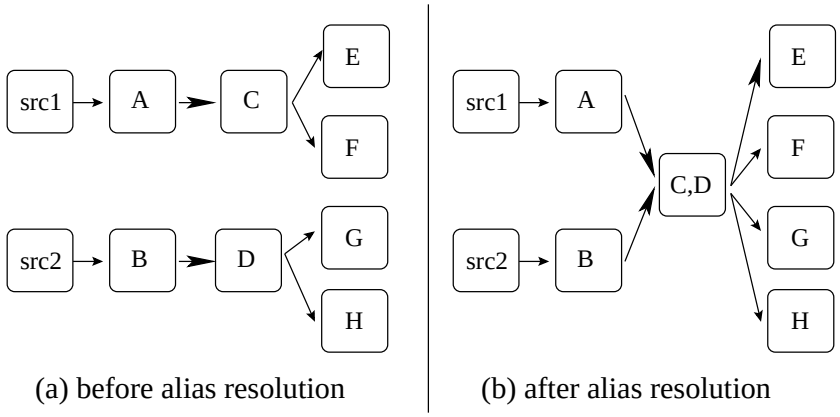



Figure 2.7: Alias resolution example. C and D are both interfaces on a single router. What appears to be two distinct networks before alias resolution turns out to be one interconnected network.

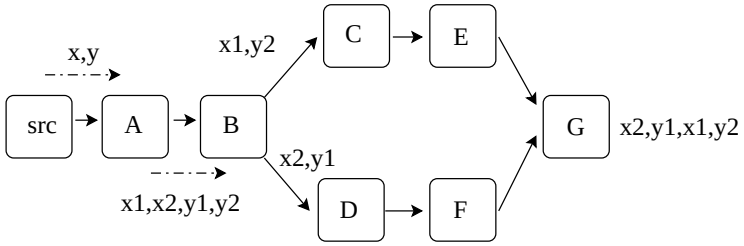



Figure 2.8: IPID usage. The packets identified by IPIDs x and y are fragmented by the router at A into x1, x2, y1 and y2. The load balancing router at B then forwards the fragments out links which have different speeds causing the fragmented packets to arrive at G out of order. The IPID allows the fragments to be uniquely identified and reassembled correctly into the original packets.

### Alias Resolution Methods

There are several methods for performing alias resolution [6, 26, 32, 33]. The Mercator [26] method relies on routers using the IP address of the outgoing interface as the source address for ICMP port-unreachable messages. This means that when two potential aliases are determined, packets can be sent to each of them, and if the source address of the replies is the same, then an alias has been found. This method is problematic as it assumes that there is only a single dominant route from all of the router's interfaces to a given destination, and that routers respond in this manner [33].

Another alias resolution method, Ally [33], takes advantage of how the Internet Protocol Identifier (IPID) field is implemented in routers. The IPID field is used in IP packets to uniquely identify a packet for re-assembly as illustrated in Figure 2.8.

Most router implementations use a single shared counter that increments the IPID field for each packet they create. Using this knowledge, when a pair of potential alias addresses are identified, packets can be sent to both and by comparing the IPIDs of the packets received back from each. If the IPIDs received back are sequential (or close to each other) and in the correct order, it can be assumed that the two IP addresses are aliases for the same router. This method is fairly reliable, provided that routers increment the IPID counter in a sequential manner. Some systems, the Linux kernel for example, set the IPID of all packets to zero. This makes it impossible to determine whether the addresses are aliases or simply routers which do not follow the convention of a single shared counter.

One other method for resolving aliases is to perform some parsing of the Domain Name Service (DNS) names assigned to a router interface [32] to extract information about the router that they belong to. This method takes advantage of the tendency for Internet Service Provider (ISP)s to name their routers in a way which identifies where they are located geographically within the network. For example, *sl-bb21-lon-14-0.sprintlink.net* and *sl-bb21-lon-8-0.sprintlink.net* are aliases for the same backbone router [32]. This technique has limited applicability as the software needs to be trained for each ISP's naming system. Also, some ISPs do not use a systematic naming system, do not name their routers within the DNS, or do not keep DNS records up to date, leading to false inferences [19].

Another method suggested by Spring *et al.* [32] is to use the maps generated from the traceroutes and two inference rules which allow alias addresses to be folded together. The first rule is that two adjacent addresses in the map are likely to represent adjacent routers rather than the same router. This is because if these addresses were in fact aliases, there would be a routing loop since one alias is forwarding to another alias on the same router. The other rule specifies that IP addresses immediately before a point where links merge are likely to aliases if the links are point-to-point, as there would not be one address connected to two different routers with a point-to-point link.

RadarGun [6] is an improvement on the Ally method, which attempts to model the IPID counter of a router by observing how it changes over time, and thus can cal-

culate the rate at which it is increasing (the velocity). This calculation is performed for each of the candidate interfaces. The velocity of a router's IPID is usually a straight line and so radargun computes the distance between all pairs of lines. Pairs which are close together represent IPIDs which were close together and so the interfaces are classified as aliases. Because RadarGun uses a model of the IPID counter, it is able to resolve aliases with far fewer probes than Ally for large graphs. We use RadarGun to resolve aliases on a set of interfaces identified by our large-scale testing described in Chapter 4 in order to classify the causes of missed topology.

## 2.7 Summary

Because network operators rarely publish detailed information regarding the structure of their networks, considerable effort must be put into gathering this information using alternate methods. Traceroute is currently the most widely used method for actively gathering information about the Internet at the IP level. Traceroute has two major drawbacks, it does not discover a comprehensive topology and it probes inefficiently, wasting resources. Doubletree attempts to improve the efficiency of traceroute by retaining information about previously discovered paths. MDA traceroute allows all load balanced paths to be discovered, improving the coverage of the tracing process. In the next chapter we present the economical MDA traceroute which attempts to discover all load balanced paths in an efficient manner.

# Chapter 3

# Economical MDA Traceroute

## 3.1 Introduction

This chapter describes enhancements to the MDA traceroute technique to make it more economical. The economical MDA aims to discover a comprehensive topology while minimizing the number of probes that it uses to do so. In order to do this, we build on two existing algorithms, doubletree and the MDA.

By sending multiple probes per hop like the MDA, we are able to discover load balanced paths. We also make use of stop sets and cooperation between vantage points in a similar fashion to doubletree in order to reduce the amount of probes needed. The assumptions made in the design of the doubletree algorithm are not applicable when considering load balanced paths. We therefore modify the local stop set so that it attempts to trace a path enough times to discover any per-destination load balancers before using an economical probing mode. We also implement the global stop set part of doubletree to reduce the amount of redundancy between vantage points.

In addition to describing the economical MDA that we have developed, we discuss an implementation of this algorithm which uses CAIDA's Ark infrastructure to conduct topology discovery. The implementation is able use a dynamic set of vantage points which it uses to probe a set of destination IPs in a cooperative manner. Our system shares global stop set data between vantage points and ensures that destinations are traced in a round-robin manner, where each destination is probed by a series of vantage points.

## 3.2 Motivation

Due to the manner in which MDA traceroute enumerates load balanced paths, a load balanced traceroute must send many more probes than a regular traceroute. While this is an inconvenience when tracing a single destination, if one wishes to use MDA traceroute in a macroscopic Internet topology discovery project such as Ark, the number of probes needed is almost prohibitive. Because load balanced traces use more probes, they therefore take more time to trace each destination. It takes an Ark team of vantage points approximately 48 hours to fully probe the routed /24 IPv4 address space. If MDA traceroute were to be used to trace the same destinations to 99% confidence, it would take approximately 26 days. A run time of this magnitude is impractical as routing changes could affect the topology. That is, a path traversed to a destination at the beginning of the run, may have changed by the end. If we then merge the traces to give a snapshot graph of the IPv4 topology, we run the risk of inferring invalid links because data inferred early in the process may no longer be valid due to routing changes. It is therefore important to reduce the amount of time MDA traceroute takes to run. Also, because MDA traceroute sends multiple probes for each hop, the number of unsolicited packets being sent into the Internet is far higher than with traceroute. This has the potential to look like a hostile act to network operators.

Thus, in order to be able to gather a comprehensive large-scale topology that we can have confidence in, a new method has been developed which improves the efficiency of the MDA traceroute process. We call this new method the Economical MDA Traceroute.

## 3.3 Overview

Economical MDA traceroute is based on the MDA discussed earlier. We have made two notable additions to this algorithm. These are the a global stop set based stopping criteria - similar to that which doubletree uses - and an adaptive local stop set which is able to reduce the number of probes needed when we have confidence about the initial section of the path.

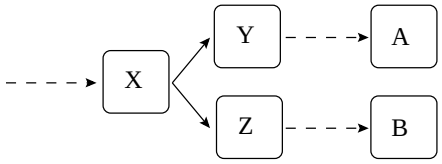



Figure 3.1: If X is a per-destination load balancer, the link X-Y will be missed by a trace to B. However, when tracing A, the link will be discovered as X is only recorded in the global stop set for B.

### 3.3.1 Global Stop Set

We make use of a global stop set for economical MDA traceroute in much the same way as doubletree does. It is a set of interface-destination pairs that are used to halt probing when a known section of a path is encountered. When a response is received from a node in the path, the global stop set is consulted to determine whether this interface has been encountered by another vantage point that also probed this destination. If there is no corresponding entry in the global stop set, an entry is created and probing carries on as usual. If there is an entry in the global stop set, then tracing along the *current branch* is halted. This is a slightly different behaviour than doubletree as it is possible for there to be other branches which need to be traced further. When the trace is completed, the global stop set is communicated to the other vantage points so that they can incorporate it into their global stop set.

Using a global stop set like this assumes that given a node which has been seen by another vantage point whilst tracing this destination, we have seen *all paths* from the node to the destination. We hypothesise that while doubletree has the potential to miss topology due to load balancing which occurs within the area hidden by the global stop set, economical MDA traceroute should not be affected if the types of load balancing (per-flow, per-packet, per-destination) have been adequately defined by Augustin [3]. This is because when tracing with MDA traceroute, we will see all links caused by per-flow load balancing with only a single trace - it takes regular traceroute many traces to approach this sort of coverage.

Because a global stop set entry is tied to a specific destination, per-destination load balancers should not affect the discovered topology either. Figure 3.1 shows the paths to two hypothetical destinations, the router at X is balancing packets on a per-

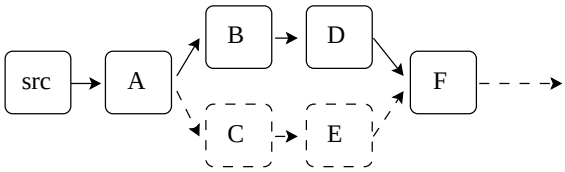



Figure 3.2: If A, B, D and F have been seen with previous probing, the local stop set can cause C and E to be missed if F is seen first.

destination basis. If MDA traceroutes are carried out to both A and B, then the two links X-Y and X-Z will be discovered. If we assume that X has been seen in the path to B by another vantage point, then when this vantage point reaches it, it will halt probing of this branch. However, when B is traced, X has not been seen in the path to B and so tracing will continue, leading to the discovery of X-Z. Thus, no topology is missed due to either per-flow or per-destination load balancing. The only type of load balancing that could cause topology to be missed due to the global stop set is per-packet. Per-packet load balancers cannot be reliably discovered by MDA traceroute as they pay no attention to the probe headers. Augustin however, states that per-packet load balancers affect only 2.1% of all paths [4] and so therefore should have a minimal impact on the link coverage of economical MDA traceroute. We investigate this further in Chapter 4.

### 3.3.2 Adaptive Local Stop Set

The assumption central to doubletree's use of a local stop set is that from any node in the tree of paths rooted at a vantage point there is only one possible path that leads back to the root. When we take load balancing into account, the situation becomes more complicated. If we assume that whenever we see an interface the remainder of the path back to the source is known, there is the potential for topology to be missed as illustrated in Figure 3.2. If probing were to begin at F, when E is found, the conventional local stop set would cause us to assume that the remainder of the path is known and halt probing. This assumption will cause the alternate path which traverses D to be missed. When tracing with MDA traceroute, all per-flow load balancers can be discovered with a single trace from one vantage point, however, per-destination load balancers can only be seen when tracing multiple destinations.

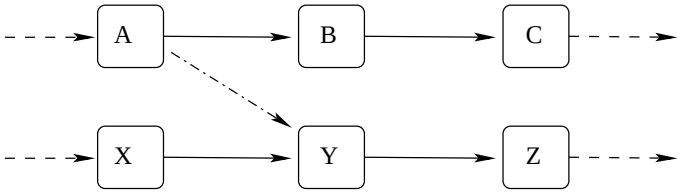



Figure 3.3: The paths A-B-C and X-Y-Z have been seen multiple times with previous probing, however a rare link between A and Y could be missed if back-links are not checked.

We therefore have re-designed the manner in which the local stop set is used so that it takes per-destination load balancers into consideration by ensuring that topology has been seen in the paths to multiple destinations before assuming that it does not need to be exhaustively probed.

To do this, we begin probing with a regular traceroute at TTL 1, assuming that we have seen this interface with prior probing. That is, we send only one probe per TTL to begin with. With each response, we check that there we still have reason to believe that all per-destination load balancers have been seen, and as such are not missing any topology. When we reach a point where a link has not been traversed enough to have confidence that there are not other per-destination links to be discovered, the algorithm switches into the full MDA traceroute mode, sending multiple probes per TTL and discovering all links. To enable this check to be made, we keep a record of the destinations that an interface has been seen in the path to and also which interfaces it forwards packets to.

We also keep information regarding the *back-links* from an interface. That is, the interfaces which forward packets to this interface. By keeping a record of the destination count, we are able to ensure that we have confidence that we have seen all of the per-destination links out of an interface. The back links allow us to be sure that we have seen this specific link with previous probing by checking that the interface which we traversed before discovering the current interface is present in the back links list. We therefore reduce the chance of missing rare per-destination links between two sets of previously seen topologies as illustrated in Figure 3.3.

Figure 3.4 is a hypothetical set of paths that have been traversed by the algorithm. The information kept for interface B says that B has been seen in the path to

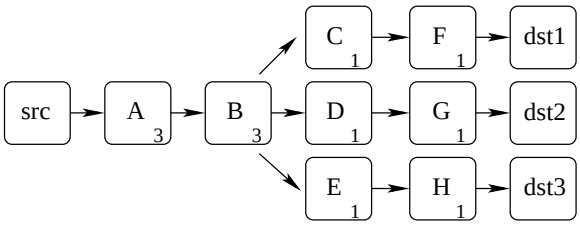



Figure 3.4: An example of a series of paths which have a common interface. B would have a destination count of three and a forward links field containing references to C, D and E

three destinations. The forward links list would contain references to the information held for interfaces C, D and E.

We call the information kept the Adaptive Local Stop Set (ALSS) to differentiate it from the local stop set which doubletree uses.

## 3.4 Algorithm

The economical MDA algorithm works by attempting to reduce the number of probes that must be sent by the MDA algorithm to discover a path. As demonstrated in Figure 3.5, the algorithm moves through two major states before using the original MDA algorithm to comprehensively trace the remainder of the path. These are the economical probing mode and the transition mode.

The economical probing mode is designed to use a minimal number of probes to traverse the initial hops in a path which have been seen in the paths to many previous destinations. Once the economical mode determines that a hop has not been seen in enough prior paths, the transition mode is used to determine a suitable starting TTL for the MDA. It does this by iteratively probing back toward the vantage point, looking for a TTL at which only one interface responds. Once in the MDA mode, probing continues as with the original MDA except we add an additional stopping condition. If an interface is discovered which is in the global stop set, then probing is halted. Upon completion of a trace, the discovered topology is then parsed and the ALSS is updated.

When an economical MDA traceroute run is started, the vantage point has no state information. To attempt to use the adaptive local stop set part of the algorithm

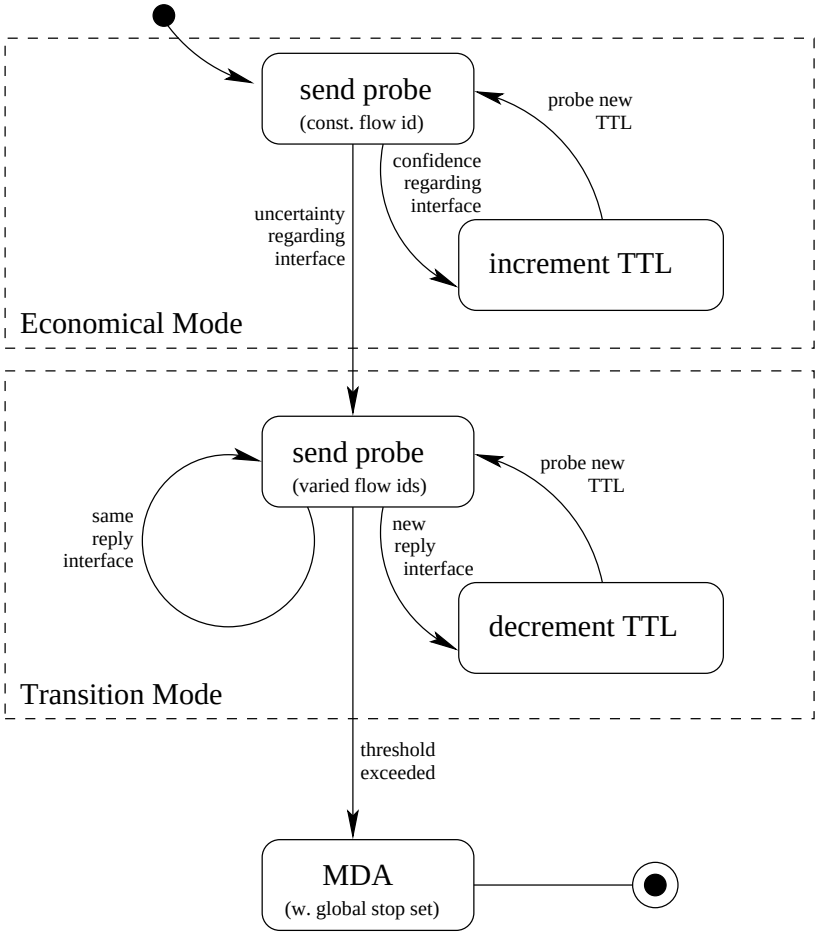



Figure 3.5: State diagram illustrating the operation of the economical MDA traceroute. When a trace is started in economical mode, each probe is sent with an invariant flow identifier. For each reply that is received, if there is uncertainty as to whether the discovered interface has been seen in enough paths to have observed all possible interfaces at the next hop, then the algorithm moves into the transition mode. Otherwise, the TTL is incremented and another probe sent. Once in transition mode, probes are repeatedly sent using the same TTL value but varying flow identifiers in an attempt to elicit a response from a different router at this TTL. If one is seen, then the TTL is decremented and another probe sent. Provided that no alternate reply interfaces are observed, probes are sent until the hypothesis that there are two interfaces at a TTL has been disproved (8 probes to 99% confidence). At this point, the regular MDA algorithm is used to trace the remainder of the path.

in this case is actually counter-productive as several probes would be used to determine that economical mode needed to be switched off at TTL one. Thus, when there is no state in the ALSS, traces are started with the economical mode already switched off. The effect of this is that there will be several traces, depending on how many traces a vantage point can conduct in parallel, at the beginning of a run which are essentially regular load balancer traces.

### 3.4.1 Economical Probing

Once state from at least one trace has been accumulated in the ALSS, new traces are started in the economical mode. The job of the economical mode is to efficiently probe a single path from the vantage point until it reaches an interface at which it no longer has confidence that all per-destination load balancing has been observed. It begins by sending a single probe to TTL 1. The ALSS is consulted and the information associated with the discovered interface is retrieved. As mentioned earlier, an ALSS entry for an interface contains a set of interfaces which represent the outgoing links, and a count of the number of destinations that it has been seen in the path to.

To be confident that continuing with economical probing will not cause topology to be missed, several criteria must be met. First, we must have already seen the back-link that discovering this interface creates. This means that the ALSS state for the previous interface must have this interface in its list of forward links. Having this check reduces the chance of missing alternate routes which are seen infrequently. Second, given $n$ forward links, to disprove the hypothesis that there is are $n + 1$ forward links, the destination count must be greater than the number of samples required[1] for $n+1$. For example, an interface with three interfaces in its forward links list needs to have been seen in the path to at least 21 different destinations to have 99% confidence that there is no more per-destination load balancing at this router. Several other checks are made at this stage also, such as ensuring that there are no loops in the path and that we have not reached the destination. The ALSS entries are not modified at any point during the economical probing mode because we are not

[1] See Table 2.1 for these values.

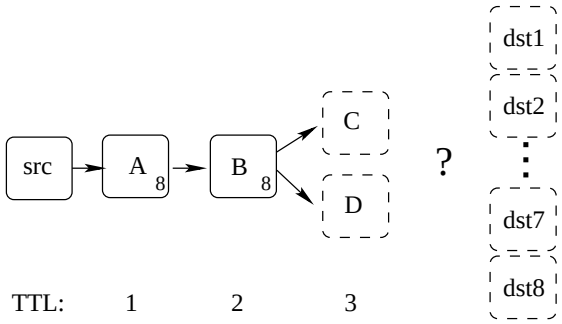



Figure 3.6: An example path where the per-destination adaptive prober can switch to an economical probing mode for TTL 3. The dashed lines indicate information contained in the stop set state (forward links and destination count).

exhaustively probing each hop and so do not have comprehensive knowledge of the path. That is, there may be alternate links previously discovered, but which are not re-encountered while in economical mode. If all of these conditions are met, then the algorithm records the current interface in a list of interfaces discovered during the economical mode. This is used for both the loop checking and back-link verification portions of the algorithm. If any of these checks fail, then we must transition back into the regular MDA traceroute algorithm in such a way that we can have confidence the topology being inferred is accurate.

Figure 3.6 is an example of how this process works. The interface labeled B has 2 forward links (B-C, B-D) and has been seen in the path to eight destinations. According to Table 2.1, B must have been seen in the path to at least fifteen destinations to rule out a third forward link. The destination count for B is eight, therefore we must transition out of the economical probing mode.

### 3.4.2 Transitioning

When we encounter an interface where we have not yet probed the number of destinations required to rule out per-destination load balancing, the economical mode is switched off and algorithm moves back into regular MDA traceroute. Before it does this, it first determines where in the path to begin. This is dependant on what has caused the economical mode to be terminated.

If there is not a back-link from the discovered interface to the previous interface, then we attempt to start the load balancer algorithm from the TTL at which the

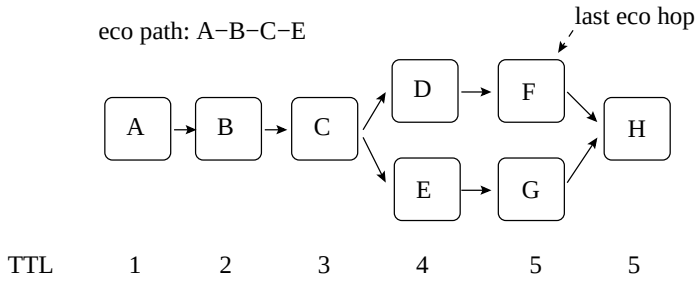



Figure 3.7: Alternate interface check. The economical mode is switched off at TTL 5 after discovering F. To be able to start the regular MDA, there must be no alternate interfaces. At most eight probes are sent to TTL 5 (assuming 99%). One probe will discover G, at this point the TTL is decremented by one and TTL 4 is probed up to eight times. The alternate interface D is discovered and so the TTL is decremented to 3. After re-probing TTL 3 eight times and only seeing C, the MDA can be started.

previous interface was discovered. We do this to allow the previously unseen link to be properly discovered.

To be sure that this can be done safely, the algorithm checks for the existence of any alternate interfaces at this TTL. This is done to ensure that we can be sure that a probe with any flow identifier will end up traversing this interface. If this is not done, then the topology inferred later in the path could be invalid.

To ensure there are no alternate interfaces, we send $n$ probes to the chosen TTL with varying flow identifiers, where n is the value which is needed to rule out two interfaces at a hop (5 for 95% and 8 for 99%). Provided that all probes return the same (original) interface, we set the MDA traceroute algorithm going. If alternate interfaces are discovered as demonstrated in Figure 3.7, the TTL is decremented by one and the check is carried out again. This process of checking the previous hop is continued until TTL three is reached. At this point, there are no gains in reducing the number of probes required to be made by continuing backward and so tracing is simply started from TTL one. This is because stepping backward to TTL two will cost as many probes to establish that there is only as single interface as it would to simply revert to the regular forward probing algorithm and send the usual one probe to TTL one and moving on to TTL two.

If the discovered interface has not been seen in the path to enough destinations, then tracing may be able to start from the current TTL. We again check for alternate interfaces before tracing continues.

If a loop is discovered, we attempt to salvage the trace by resetting to TTL one and switching the economical mode off. We do this because the economical mode only probes down one path, so it is possible for there to be an alternate path (already seen with previous tracing) that does not have a loop. The MDA algorithm probes both branches until it identifies the loop at which point it would halt probing the branch with the loop and only continue probing any other branches.

### 3.4.3 Collecting ALSS State

The ALSS is not modified during the economical phase of tracing, therefore it must be populated with data whilst in the regular MDA traceroute modes. Because the paths that can be discovered by MDA traceroute can contain multiple branches, our algorithm populates the ALSS once the trace has been completed and all of the links between branches inferred. The tree of links discovered is traversed and the ALSS state for each interface encountered is updated with the new information. A disadvantage to this method is if traces are being conducted in parallel, there are likely to be several traces started in parallel before the ALSS is populated.

### 3.4.4 Global Stop Set

Once the economical mode has been disabled and probing using the regular MDA commences, each interface that is discovered is checked against a global stop set which contains all of the interfaces that have been seen in paths toward this destination by each of the vantage points which have previously completed traces to it. If a discovered interface is in the global stop set, then we assume that the rest of the path has also already been seen and we halt probing on this branch.

This is a safe assumption provided that the majority of load balancing observed is per-flow or per-destination. MDA traceroute can, with one trace, discover all alternate paths caused by per-flow load balancing, therefore successive traces from different vantage points should not discover any new topology. Tracing continues down other branches of the path until a stop condition is met there also. This allows alternate paths which were not discovered by other vantage points to be seen.

### 3.4.5 Timeouts

Due to the unreliable nature of the Internet, replies are not always received for probes. The algorithm deals with probes that were not responded to in different ways depending on why the probe was sent.

If a probe times out while in the economical mode, we use a gap limit feature similar to that found in the traceroute used by macroscopic topology discovery projects. When a probe times out, we check that the number of probes which have timed out is less than the specified gap limit. If it is, then we simply increment the TTL by one and probe the next hop. Once the gap limit is reached, we decide the trace is not viable and terminate.

If a probe times out while checking for alternate interfaces, we simply treat it as a response which doesn't match the original interface as we cannot be sure that there is only one interface at this hop. That is, we decrement the TTL by one each time a timeout is received until we reach TTL 3, at which point we revert to regular MDA traceroute at TTL 1.

## 3.5 Implementation

This algorithm has been implemented as an optional part of the MDA traceroute implementation currently available in scamper [22]. Scamper is an open-source prober, written in C, which implements several well-known active measurement techniques. As scamper itself contains no mechanism for coordinating a set of vantage points, we also implement a coordination system, written in Ruby, to distribute global stop set data using Marinda [7].

### 3.5.1 Scamper

We implement economical MDA traceroute as two optional parameters to the MDA traceroute already implemented in scamper. These are the ALSS and the global stop set.

The ALSS portion of the algorithm is enabled by specifying a name for the

stop set. This allows concurrent but distinct experiments to be run without the stop sets overlapping. The stop set is implemented as a splaytree. The nodes in the stop set are structures which contain the IP address of the node, the number of destinations it has been seen in the path to, and a list of stop set entries to which this node forwards packets to (forward links). We are able to make the back-link check without explicitly storing any additional data. Given the link $x \rightarrow y$, where we want to check that $y$ has a back-link to $x$, we retrieve the ALSS record of $x$ and verify that $y$ appears in the forward links array of $x$.

The global stop set part of the algorithm is enabled by passing in a list of interfaces which are to be used as the global stop set for this trace. This list is managed by the control mechanism and passed to scamper as command parameters. The global stop set is implemented as an array of IP addresses which are specific to a single trace. These are not shared between traces.

When a trace is started, if the ALSS is enabled, the ALSS is checked to ensure that there are nodes in it. If there are not, then there is nothing to be gained by using the economical mode and so the original MDA is used. When this trace completes, because the ALSS is enabled, the topology discovered by the trace is parsed and converted into ALSS nodes and inserted into the splaytree. Now that there are nodes in the stop set, proceeding traces are started in the economical mode.

Scamper makes use of call-back functions to handle replies to probes sent. This is implemented using modes; the code that is executed when a reply is received is dependent on the current mode. We therefore implement our algorithm as a series of modes. To begin with, we are in the economical mode. When a reply is received in the economical mode, the interface that the reply is received from is looked up in the ALSS. The ALSS record retrieved is then consulted and the conditions described in Section 3.4.1 are checked. If they are met and economical probing can proceed, the TTL is incremented and another probe sent. Otherwise, the mode is changed to the transition mode in order to check for alternate interfaces. A different flow identifier is used and another probe sent. When a reply to this probe is sent, the transition code deals with checking that the probe is received from the same address every time. If the addresses match then a different flow identifier is used and another

probe sent. This process continues until the required number of probes have been sent or a reply is received from a different address. If different reply addresses are seen then the TTL is decremented and another probe is sent. This process continues until a suitable TTL is found or the trace is terminated as described in Section 3.4.2. Provided a TTL is found for which there is only one address seen, the mode is changed to the first mode that the original MDA uses. A probe is then sent such that when the reply is received, the code for the original MDA handles it.

We have altered the code for the MDA to allow the use of a global stop set. Each reply address that is received is looked up in the global stop set. If there is a corresponding entry, then probing along the current branch of the path is halted; other active branches continue to be probed however. We also add code to populate the ALSS with the new information gained with each trace. When a trace is completely finished, we traverse the links discovered and for each address, retrieve the relevant ALSS entry, update the destination count, and add the forward links from this address.

### 3.5.2 Marinda based Coordination Mechanism

Because scamper runs independently on a vantage point, we implement our coordination mechanism using the shared memory implementation, marinda, that CAIDA uses to coordinate its Ark infrastructure [7]. Marinda allows a high-level approach to be taken to coordination. It is written in Ruby and abstracts away issues such as race conditions and loss of connectivity between the vantage points and the control server. This is useful as it allows different projects to be implemented and deployed with a minimum of repeated effort.

Marinda uses tuples to share data. A tuple is an ordered set of objects, implemented as an array in marinda. Arbitrary tuples can be written into a region of shared memory, or *tuple space*, within marinda, and then retrieved using basic pattern matching. For example, the tuple

$$TEST, item1, item2$$

could be retrieved by requesting a tuple which matches the pattern

$$TEST, *, *$$

where * is a wildcard.

We implement our coordination system as a set of clients running on the vantage points, responsible for driving scamper and returning global stop set information, and a central control server which sends tasks to the vantage points. All of the vantage points, along with the control server, connect to marinda. The control server inserts a tuple for each destination to be traced into the tuple space. These tuples are considered low-priority and so are only taken if there are no high-priority tuples available. We separate the tuples into high-priority and low priority so that once and address is traced by one vantage point, the others trace it shortly after. The vantage points then each retrieve enough of these tuples to occupy scamper. Because scamper is a parallelised tracer, it requires many tasks to be maximally efficient. Once scamper completes a trace, the tuple is updated with global stop set information returned by scamper and inserted back into the tuple space as a 'done' tuple. The control server, watching for these done tuples, retrieves it, determines which vantage point should probe it next by using an alphabetically ordered list of vantage points that are yet to trace it, and inserts it back into the tuple space as a high-priority tuple. Vantage points retrieve high-priority tuples using pattern matching so that they only receive tuples designated for them. For example, the vie-at vantage point would request a tuple the matched the following pattern:

$$TASK\_HP, *, *, vie\_at$$

This request could retrieve the tuple:

$$TASK\_HP, 130.217.230.17, trace, vie\_at$$

This tuple instructs the vantage point to issue a task to scamper requesting a classic traceroute ('trace') to 130.217.230.15.

Once a destination has been traced by all vantage points, the control server removes it from the tuple space. After all destinations have been traced, the control server sends a notification tuple to each of the vantage points informing them that the run has completed.

To reduce load on marinda, the trace data collected is stored locally on the various vantage points and manually collected upon completion of the run.

## 3.6 Summary

In this chapter we present economical MDA traceroute, our variation on MDA traceroute which is designed to reduce the number of probes required to discover load balanced paths with confidence. We also outline our implementation of this algorithm and a control mechanism which allows it to be tested on the Ark infrastructure. We evaluate the economical MDA traceroute in Chapter 4 and discuss potential shortcomings and improvements.

# Chapter 4

# Method Comparisons

## 4.1 Introduction

Because the main objective of this thesis is contribute to making real-world macroscopic Internet topology discovery systems more efficient and comprehensive, it is important that the techniques devised are tested in a manner which closely reflects an actual implementation. We test the implementation of the economical MDA traceroute algorithm presented in Chapter 3 alongside a complete implementation of the doubletree algorithm. We test these in concert with classic traceroute and MDA traceroute to give a benchmark against which we can compare the doubletree and economical MDA traceroute methods.

This chapter begins by discussing the metrics used to compare the performance of the topology discovery methods tested. We describe two metrics - probe count and link count - which we use for overall performance comparison between methods. We also describe several metrics for evaluating the performance of the stop sets used by both doubletree and economical MDA traceroute.

Once the evaluation metrics have been discussed, we present the results obtained from conducting a large-scale coordinated experiment using CAIDA's Ark infrastructure. We investigate variances in discovered topology between the methods and explain why some methods perform better than others. In doing so, we observe a type of load balancing - per-source/destination - which was not considered by Augustin *et al.* [3]. Per-source/destination load balancers take both the source and destination addresses into consideration when forwarding a packet. In addition to

this, it appears that the assumptions behind doubletree and economical MDA traceroute's use of a global stop set may be flawed. The global stop set is the cause of a large fraction of the links which these algorithms fail to discover. We investigate the causes of this and determine that aliases and per-source/destination load balancers form the majority of the cases where topology is not discovered. We suggest altering the global stop set to consider per-source/destination load balancing.

## 4.2 Efficiency Metrics

Because the various techniques that we are dealing with in this thesis have different goals and so gather topology in different ways, some thought must be given to establishing a set of efficiency metrics that allow for robust and unbiased comparison of both variations of methods (classic and doubletree traceroutes) and indeed between different methods (MDA and classic traceroute). We use two basic metrics to compare the overall performance of each method. The first is probe count, which gives measures the relative efficiency of the method. The second is link count which we use to give an indication of the amount of topology that each method is able to discover. In addition to this, we also investigate the effectiveness of the stop sets that both doubletree and economical MDA traceroute use. In order to do this we determine the topology that doubletree has failed to discover and attempt to determine the cause of this. For the ALSS, we use the length of path that the economical mode was able to remain on for. We use this metric as, for each TTL that is probed using the economical mode, several fewer probes are sent when compared to the original MDA.

### 4.2.1 Probe Count

The most basic indicator of an algorithm's efficiency is the number of probes that it uses. A probe being a packet sent into the Internet in order to discover some information. Probe count is a particularly good indicator of efficiency as a probe is of finite size and so takes a finite amount of time to send. Therefore, the more probes that are used, the longer the method will take to run. In addition to time, the

amount of data sent into the network also increases proportionally with the probe count.

Probe count by itself however is not enough to robustly compare methods. To be able to use probe count alone to compare methods, all of the other variables must be the same. The methods must have traced the same number of destinations from the same number of vantage points, using the same probing structure (team or cooperative probing). In order to reliably compare methods that conduct probing in different manners, the probe count metric must be enhanced. In order to do this, we impose a set of conditions which allow different methods to be compared using the probe count value.

The first condition we impose on this comparison is that all methods have traced the same set of destinations. We impose this condition because traces to two different destinations can have completely different characteristics. For example, one of the destinations may be located closer to the vantage point than the other in terms of hops and so will take cost fewer probes. The only way to minimise this problem is to have all methods tracing identical destination lists. The second condition is that either both methods used team probing or both methods used cooperative probing. If team probing is used, each vantage point should have traced the same set of addresses with both methods. If a cooperative probing structure is used, then all vantage points must conduct traces using all methods to all destinations.

With these conditions in mind, we can then measure the efficiency of each method by calculating the number of probes used per destination. By using a cumulative probe total, we can look at the probe count for the last trace for each method and see the total number of probes which were sent during the course of the experiment. Figure 4.1 shows an example graph of this metric. In this example, we can see that the MDA traceroute uses many times more probes than regular traceroute. Also, the increase in probes is linear because neither of the methods make any attempt to be efficient.

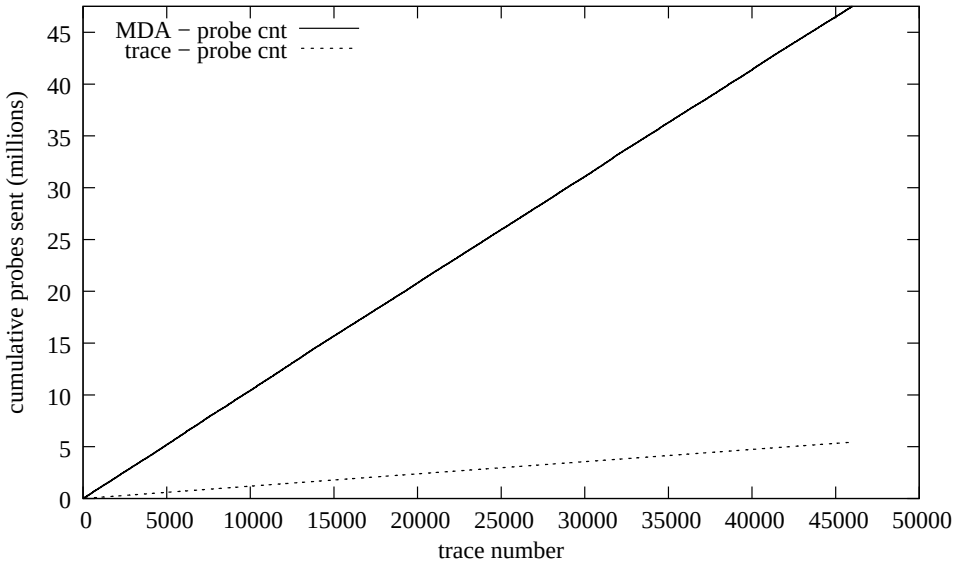



Figure 4.1: An example probe count graph showing probe usage as the experiment is run. The data is a sample from the large scale experiment described in Section 4.3

.

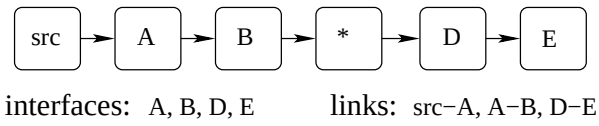



Figure 4.2: An example path from which we can extract 4 interfaces (A, B, D and E) and 3 links (src-A, A-B, D-E)

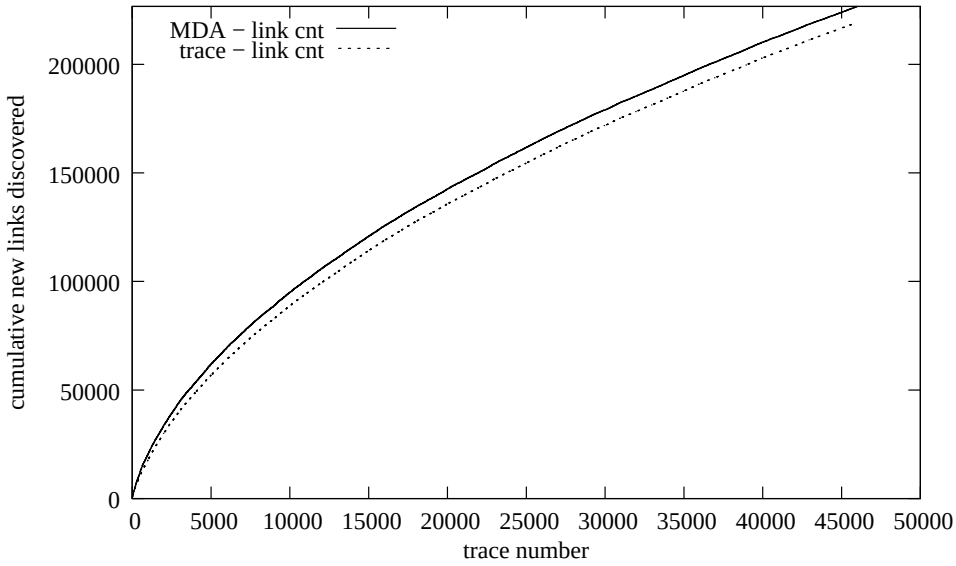



Figure 4.3: An example link count graph showing link discovery as the experiment is run.

### 4.2.2 Topology Coverage

While the probe count goes a long way in determining how efficient a method is, there is little point in using a method which uses very few probes but also misses large amounts of topology. We therefore need to view the probe count in the context of another metric which can give an indication of the coverage of the method. There are two statistics that can be used to measure the amount of topology that a method discovers. These are interface and link counts. The number of interfaces discovered by a trace is the number of unique IP addresses that are seen. The link count value is the number of links that can be inferred from the interfaces discovered. For example, given the path discovered in Figure 4.2, the interface count would be 4 (A, B, D and E). The interface labelled with a '*' was unresponsive during probing and so we cannot infer any topology from it. The link count would be 3 (src-A, A-B, D-E). Again, as we did not discover the interface between B and D, we cannot infer any links between the two nodes. Because we are only looking at net gains in topology, we count only the globally unique links or interfaces in each trace. That is, the links in trace $n + 1$ that are also in traces 1 to $n$ are only counted once.

As with the probe count we use a cumulative count per trace to plot the topology discovery over time. Figure 4.3 is an example link count graph for the same

set of traces that Figure 4.1 was generated from. By looking at these two graphs in the context of each other, we can see that although the MDA traceroute uses nine times more probes than regular traceroute in this situation, the gain in topology discovered is relatively small. However, as we have mentioned, one of the failings of current macroscopic traceroute projects is that they over-sample the edges of the Internet closest to the vantage points [20, 2, 28, 34, 13]. Classic traceroute is approaching the coverage of MDA coverage because it re-probes the nodes close to the vantage points over and over, thus inadvertently discovering load balanced links. We therefore argue that the majority of the extra links that MDA traceroute discovers are located toward the end of the paths traced, thus reducing the sampling bias prevalent in current macroscopic traceroute data. So although the gain in topology is relatively small, we argue that the gains are in the areas of paths that traceroute has been criticised for under-sampling. We thus provide a more comprehensive topology which is less influenced by sampling bias.

### 4.2.3 Stop Set Effectiveness

Because both of the efficient methods (doubletree and economical MDA) use stop sets, it is necessary to have a way to identify which of the stop sets (local or global) are reducing the probes used and also identify the situations which cause missed topology.

#### Doubletree

We begin by looking at the stop sets used by doubletree. If we compare doubletree and traceroute using only the probe and link count metrics then we are unable to determine where the increase in efficiency is coming from or where topology is being missed. We therefore must separate the trace into two parts, the interfaces discovered by backward probing and the interfaces discovered by forward probing. This is done by simply taking the value that was used as the initial hop distance (first hop) for the vantage point in question and splitting the trace at that point. The interfaces up to, but not including, the interface at the first hop TTL are assumed to have been discovered by backward probing, the remainder by forward probing.

| ttl | traceroute | doubletree | diff |
|---|---|---|---|
| 1 | 128.223.157.3 | | x (in lss) |
| 2 | 128.223.3.8 | | x (in lss) |
| 3 | 128.223.3.9 | | x (in lss) |
| 4 | 207.98.68.181 | | x (in lss) |
| 5 | 207.98.64.162 | | x (in lss) |
| 6 | 207.98.64.10 | | x (in lss) |
| 7 | 207.98.64.137 | | x (in lss) |
| 8 | 63.211.200.245 | 63.211.200.245 | + (exact match) |
| 9 | 4.68.105.12 | 4.68.105.12 | + (exact match) |
| 10 | 4.59.232.54 | 4.59.232.54 | + (exact match) |
| 11 | 207.88.13.142 | 207.88.13.142 | + (exact match) |
| 12 | 65.106.1.45 | 65.106.1.45 | + (exact match) |
| 13 | 65.106.0.174 | | x (in gss, san-us) |
| 14 | 207.88.83.194 | | x (in gss, san-us) |
| 15 | 66.239.36.10 | | x (in gss, san-us) |
| 16 | * | | x (star) |
| 17 | * | | x (star) |
| 18 | * | | x (star) |

Figure 4.4: difference between doubletree and traceroute traces to 12.172.35.5 from eug-us. An ‘x’ indicates an implicit match (the interface is seen elsewhere), a ‘+’ indicates an exact match. In this trace doubletree is able to discover the same topology as traceroute.

Because doubletree is designed to be a more efficient version of traceroute, we compare the interfaces discovered by doubletree to those discovered by traceroute. We do this in a hop-by-hop manner. For each hop doubletree misses an interface at, we attempt to locate the missing interface elsewhere in the trace data. For interfaces missed during backward probing, we look only at the doubletree data for the vantage point in question. For interfaces missed during forward probing, we look at the doubletree data for all of the vantage points. It is possible for an interface which is determined to be missed by backward probing to be discovered coincidentally by another vantage point and so not contribute to the overall missed interface count. However, with this metric we are testing the assumptions behind the local stop set and so do not include these coincidental matches. Figure 4.4 is a representative 'diff' between doubletree and traceroute traces to one destination. In this case, doubletree starts with a first hop value of 9 and probing forward, discovers 4.68.105.12, 4.59.232.54 and 207.88.13.142 which were not in the global stop set. It then discovers 65.106.1.45 at TTL 12, causing forward probing to be halted as it is in the global stop set. In this case we can see that the interfaces that doubletree does not discover due to the global stop set stopping probing are indeed already discovered by the san-us vantage point. It then switches to backward probing, discovering 63.211.200.245 at TTL 8 which has been seen by this vantage point previously and so is in the local stop set. The trace is then halted completely. Again, the interfaces not explicitly discovered by doubletree are able to be found in other traces from this vantage point.

Perhaps more important than identifying which interfaces have been missed is identifying the reason that doubletree has not discovered them. We do this by using MDA traceroute data which was collected in parallel with the traceroute and doubletree data. We use the MDA traceroute data as a 'ground truth' from which we can make inferences about the causes of missed interfaces. Given an interface which has been missed by doubletree, we look up the relevant MDA traceroute traces and attempt to determine whether the miss was caused by load balancing, and if it was, the type of load balancing in place. We attempt to make this determination based on heuristics about how the different types of load balancing work. If we

| ttl | traceroute | doubletree | diff |
|---|---|---|---|
| 1 | 128.223.157.3 | | x (in lss) |
| 2 | 128.223.3.8 | | x (in lss) |
| 3 | 128.223.3.9 | | x (in lss) |
| 4 | 207.98.68.181 | | x (in lss) |
| 5 | 207.98.64.162 | | x (in lss) |
| 6 | 207.98.64.10 | | x (in lss) |
| 7 | 207.98.64.137 | | x (in lss) |
| 8 | 63.211.200.245 | 63.211.200.245 | + (exact match) |
| 9 | 4.68.105.30 | 4.68.105.30 | + (exact match) |
| 10 | 4.69.132.18 | 4.69.132.18 | + (exact match) |
| 11 | 4.69.132.54 | 4.69.132.54 | + (exact match) |
| 12 | 4.68.107.104 | 4.68.107.104 | + (exact match) |
| 13 | 4.71.40.2 | 4.71.40.2 | + (exact match) |
| 14 | 64.78.230.215 | | o (gss miss) |
| 15 | 64.78.193.134 | | x (in gss (scl-cl)) |
| 16 | * | | x (star) |
| 17 | * | | x (star) |
| 18 | * | | x (star) |

Figure 4.5: difference between doubletree and traceroute traces to 216.17.122.184 from eug-us. An ‘x’ indicates an implicit match (the interface is seen elsewhere), a ‘+’ indicates an exact match and an ‘o’ indicates a missed interface. In this trace, the doubletree global stop set assumptions have not held up and caused the interface 64.78.230.215 to be missed.

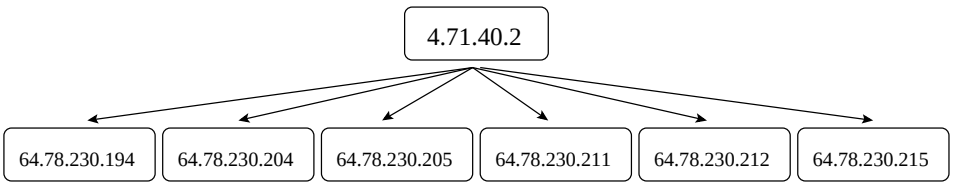



Figure 4.6: The interfaces that MDA traceroute discovers in links from 4.71.40.2

consider the trace shown in Figure 4.5, doubletree has missed the interface at hop 14 (64.78.230.215). By not discovering this interface, we also do not discover the link 4.71.40.2→64.78.230.215. We hypothesise that if there is a link 4.71.40.2→$x$, where $x$ is not 64.78.230.215, then it is likely that $x$ is the interface discovered by doubletree from another vantage point, thus causing the link we observe in this trace to be missed. Figure 4.6 shows the links from 4.71.40.2 that MDA traceroute discovered.

In order to classify the type of load balancing that is causing this link to be missed, we first need to perform alias resolution on the six interfaces to ensure that we are dealing with distinct routers. As routers have more than one interface associated with them, it is common to see multiple interfaces in the IP topology discovered by traceroute which in reality, all belong to the one router. To resolve these aliases, we use the RadarGun [6] technique discussed in Section 2.6 and implemented in scamper.

Once we are confident that there a six distinct routers at the same hop that doubletree missed the interface at, we attempt to classify the type of load balancing in effect by considering which vantage points discovered which interfaces. If every vantage point discovers all of the possible links, then we say that there is per-flow load balancing. Because the MDA traceroute algorithm can manipulate probes such that it discovers per-flow load balancing, it follows that each vantage point is able to discover the complete topology.

Per-flow load balancing is not in effect in the example case we illustrate in Figure 4.5 as each vantage point only discovers one outgoing interface from 4.71.40.2. Because of this, we suggest that there is a type of load balancing which is based on the source and destination address tuple. We believe that this is the type of load balancing that Augustin *et al.* [3] classify as per-destination. Taken from one vantage point where the source address is constant, it would appear that it is solely the destination address that the router is using to vary the path. However, as we see with six vantage points, it appears that by varying the source address and keeping the destination address constant, we also see these variances in the path. To classify an interface as missed due to per-source/destination we use a heuristic that if each van-

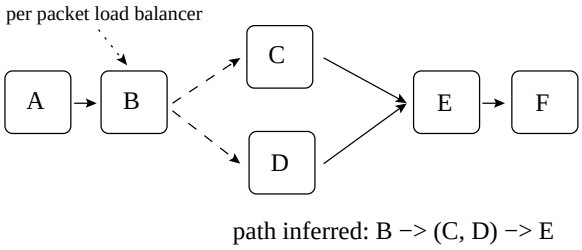



Figure 4.7: The result of probing past a per-packet load balancer. We cannot have confidence about which flow identifiers are forwarded to C or D, but because the paths re-converge at E, we can group C and D into a clump and treat it as a single node for probing past.

tage point discovers only one interface and there is more than one unique interface discovered by the various vantage points then there is per-source/destination load balancing. For a link $x \rightarrow y$ to have been missed due to per-source/destination load balancing, where $y$ is the interface that doubletree has missed, the MDA traceroute data from each vantage point must show only one link which originates at $x$. Also, at least one other vantage point must discover a different link. We only require two unique interfaces rather than one per vantage point as it is possible to see routers which have only two outgoing interfaces towards a destination. This case we would see the two interfaces repeated across the set of vantage points but still with each vantage point only discovering one of the two interfaces. If it were per-flow load balancing, all of the vantage points would see both of the interfaces.

The other type of load balancing that could cause doubletree to not discover some topology is per-packet. As our implementations of traceroute and doubletree only send one probe per hop provided a response is received, it is possible that a per-packet load balancer will cause traceroute to see a different path to doubletree. To classify an interface as being a per-packet load balancer, we once again use the MDA traceroute data. In our implementation of the MDA traceroute algorithm, if a potential per-packet load balancer is identified in the path, the algorithm attempts to probe past it by treating the ambiguous links as a clump. Figure 4.7 shows a hypothetical per-packet load balancer. If the paths which lead up to the load balancer are symmetrical, then the algorithm is able to probe past it, however, it cannot accurately enumerate the links out of the load balancer so it forms a clump. These clumps can indicate whether a router is doing per-packet load balancing.

Once we have processed all of the missed interfaces in the doubletree data, we can determine the relative impact each cause has on the doubletree data and develop methods to mitigate their effects.

**Economical MDA Traceroute**

As with doubletree, we are interested in quantifying how well the economical MDA traceroute algorithm is performing by checking if, and how much, topology being missed. If there is missed topology, we attempt to determine the portion of the algorithm causing the topology loss.

Because the ALSS has not been previously tested, we ensure that it is reducing the number of probes used. We do this by checking the TTL at which the economical mode is switched off. Because the economical mode generally only sends one probe per TTL, compared to at least eight with regular MDA traceroute to 99% confidence, the further that the economical mode can get into the path, the less probes that are sent overall. We use these TTL values to plot a graph which illustrates the number of hops the economical mode is used for, on a per-trace basis. We expect this length to increase over time as the ALSS grows and more confidence about the topology is gained.

In addition to this, we also plot the number of probes used in the economical mode. A probe count which is similar to the number of hops that the economical mode is switched on for indicates that the algorithm does not often need to step backward to find a hop at which there are symmetrical paths leading to it as discussed in Chapter 3.4.2. For example, a trace which reaches TTL 10 in economical mode, but then must step back to TTL 4 before switching the economical mode off will use up to 66 probes, compared to only 18 for a trace which is able to switch the economical mode off immediately at TTL 10.

As we did with doubletree, we analyse the topology generated by economical MDA traceroute to determine which links are missed. We divide each trace into the links discovered while using the ALSS to guide probing and those discovered while using the global stop set. This allows us to see where the majority of topology is being missed.

| name | location | organization type | doubletree firsthop |
|---|---|---|---|
| hlz-nz | Hamilton, NZ | university | 12 |
| eug-us | Eugene, USA | university | 9 |
| nap-it | Napoli, Italy | university | 13 |
| san-us | San Diego, USA | research | 8 |
| scl-cl | Santiago, Chile | network infrastructure | 7 |
| vie-at | Vienna, Austria | community network | 6 |

Table 4.1: The Ark vantage points used in our large-scale testing.

## 4.3 Large-Scale Cooperative Testing

### 4.3.1 Methodology

In order to carry out our large-scale experiments using the cooperative probing methodology that doubletree requires, we make use of CAIDA's Archipelago (Ark) measurement infrastructure. Ark is a set of over thirty globally distributed vantage points which are used to collect the IPv4 Routed /24 Topology Dataset. For our experiments we use a subset of these vantage points. Table 4.1 lists the vantage points that we use, along with their location and organisation type. The Ark platform is well-suited to carry out these experiments as it has been designed to allow new measurement types to be deployed and tested with a minimum of effort. Ark uses a tuple space coordination system, marinda [7], which allows the vantage points to communicate at a high level.

Marinda is a coordination mechanism which allows arbitrary tuples to be stored and retrieved with ease. We use marinda to issue tracing tasks to the vantage points and to pass stop set data on to successive vantage points, this is outlined in Chapter 3.5.2. Marinda inherently prevents one tuple from being retrieved simultaneously from two vantage points, so doubletree always probes in a round-robin fashion, ensuring that only one vantage point will trace a destination with an empty stop set. The original doubletree authors used windows of destination addresses to keep vantage points busy [11]. We eliminate the need for vantage points to manage windows of addresses by dividing tasks into two categories; high and low priority. The

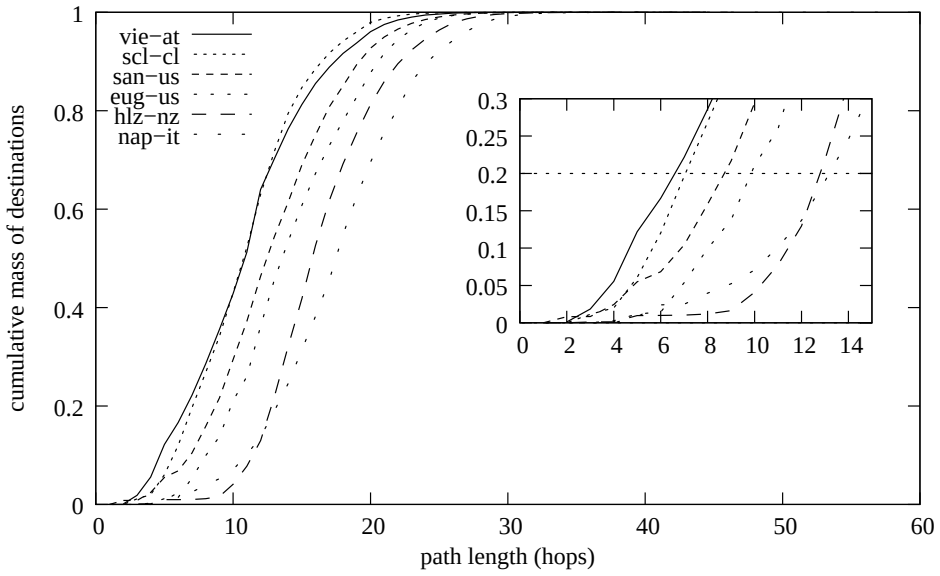



Figure 4.8: The cumulative mass of path lengths for each Ark vantage point used in the large-scale experiments. The horizontal line at $y = 0.2$ indicates the first hop values selected for doubletree tracing. These values mean that the destination will, with 20% of traces, be reached by the first probe.

low priority destinations are those which have not yet been traced by any vantage point. A vantage point first checks for any high-priority destinations assigned to it, if there are none, it takes a low priority destination and proceeds to trace according to the method described in the tuple. Once the trace has completed, the tuple is updated with the vantage point's name and new stop set information. It is then inserted back into marinda as a done record. The control server collects these done records and determines which vantage point should probe it next. The record is inserted as a high-priority tuple directed at the next vantage point in the list. Because high-priority tuples are taken first, destinations are probed by all vantage points in quick succession. This minimises the effect of routing changes. Also, because we pass the global stop set on a per-destination basis, the need for windows of destination addresses is removed.

The tunable parameter to doubletree is the first hop value. The first hop value is the distance into the path at which doubletree begins its forwards probing. The first hop value is chosen individually for each vantage point such that 80% of the time, the destination is not reached with the first probe [11]. To calculate this value, we use existing trace data from each vantage point and plot a cumulative mass graph

of observed path lengths as seen in Figure 4.8. From this we can determine the appropriate first hop value for each vantage point by reading off the path length where the cumulative mass is 0.2. These values are shown in Table 4.1.

We carry out traces to 46,000 destinations for each of the methods being tested. This number was derived from prior knowledge about the time it takes, on average, to carry out MDA traceroutes. We decided on an estimated three-day run time. Three days provides a compromise between keeping the tracing time short, so that the effects of routing changes are minimised, and gathering a comprehensive data set. The addresses we select are generated randomly from a RouteViews BGP prefix table. We select 46,000 prefixes at random, and for each prefix generate a random IP address covered by it.

For each destination in the list, we conduct four traces. The first is regular traceroute. We use ICMP-echo request packets as these have been shown to have the highest response rate [23]. We also set the maximum number of unresponsive hops (gap limit) to three, as this helps to speed up traces for which the destination is unreachable. The second method we use is doubletree traceroute. We use the same parameters as for regular traceroute. Third is MDA traceroute, for which we use a 99% confidence level, a gap limit of three and a minimum inter-packet time of 150ms. The fourth method is the economical MDA traceroute, using the same parameters as the MDA traceroute.

### 4.3.2 Results

We began our experiments on December 30, 2009. The run took approximately 72 hours, with almost 74 million packets sent in total.

Before we can analyse the data gathered, we first must merge the topology from each vantage point as both the doubletree and economical MDA traceroute methods attempt to not re-probe topology which has already been discovered by another vantage point. We do this by extracting all of the links that each method discovers in the path to a destination and removing any duplicates. We only consider links from one interface to another, that is, links which have stars (unresponsive hops) in them are discarded. If one vantage point discovers $a \rightarrow b \rightarrow c$ and another discovers

| vantage point | traceroute | doubletree | MDA | eco MDA |
|---|---|---|---|---|
| eug-us | 910614 | 429611 | 7387756 | 2971282 |
| hlz-nz | 1028975 | 294221 | 8436059 | 2403428 |
| nap-it | 1038420 | 298587 | 8635434 | 2596954 |
| san-us | 853177 | 372781 | 7200280 | 2975541 |
| scl-cl | 773296 | 327035 | 6948846 | 2845561 |
| vie-at | 871827 | 411568 | 7433549 | 2684581 |
| total | 5476309 | 2133803 | 46041924 | 16477347 |

Table 4.2: The number of probes sent from each vantage point.

| vantage point | traceroute | doubletree | MDA | eco MDA |
|---|---|---|---|---|
| eug-us | 93171 | 51469 | 99535 | 58975 |
| hlz-nz | 97590 | 33079 | 104344 | 41231 |
| nap-it | 94582 | 30923 | 100181 | 38012 |
| san-us | 97590 | 53899 | 103318 | 59932 |
| scl-cl | 98170 | 50229 | 104825 | 57908 |
| vie-at | 103735 | 47438 | 112050 | 55356 |

Table 4.3: The number of links discovered by each vantage point.

$c \rightarrow d \rightarrow e$, our final link list will contain $a \rightarrow b$, $b \rightarrow c$, $c \rightarrow d$, and $d \rightarrow e$. For the probe count metric, we simply sum the number of probes that each vantage point used for the given method. Table 4.2 contains a list of these raw probe count values for each method and vantage point. Table 4.3 shows the number of links discovered.

### Link Coverage

We use a per-trace link count to allow us to determine performance over time of each method tested. Figure 4.9 shows the cumulative number of unique links discovered by each trace. These results are unexpected as we see the link discovery of regular traceroute is a close second to MDA traceroute. One possible reason for traceroute almost matching the link coverage of MDA traceroute is that, due to having several vantage points, each tracing the same destination, we effectively send six probes to every hop with traceroute. Thus potentially discovering alternate paths without explicitly re-probing each hop.

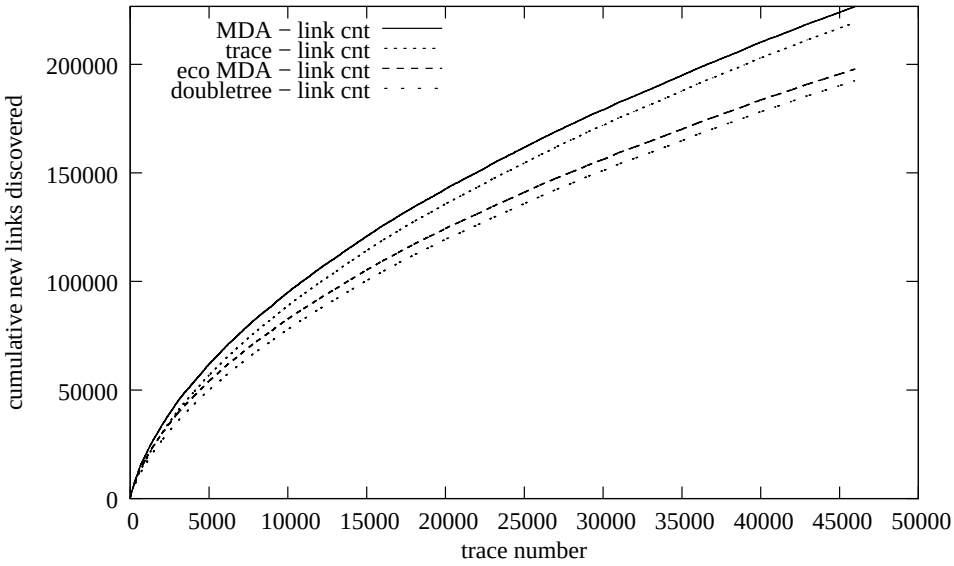



Figure 4.9: Per-trace cumulative link discovery. 46,000 destinations.

**Missed Topology**

The difference between the number of links discovered by traceroute and doubletree is larger than expected, as is the difference between MDA traceroute and economical MDA traceroute. We attempt to determine the cause of this gap by classifying links missed by economical MDA traceroute into links missed due to the ALSS and links missed due to the global stop set as this indicates an assumption has not held up. Figure 4.10 shows the number of links missed by a representative sample of vantage points. It appears that it is the global stop set which is causing the majority of links to be missed. For example, the global stop set causes 6,726 links to be missed by hlz-nz whereas the ALSS only causes 877 links to be missed. Given that the links missed by the ALSS only represent 1.49% of the total links discovered by eug-us, we focus our attention on determining why the global stop set missed 11.4% of links.

In order to do this, we turn to the doubletree data which exhibits much the same missed-link characteristics as we saw with the economical MDA traceroute. Figure 4.11 shows the results of our attempts to classify the missed interfaces based on the type of routing which immediately precedes them.

By far the most biggest cause of missed topology are aliases. Aliases comprise around 45% of the causes of missed doubletree topology in our experiment. When

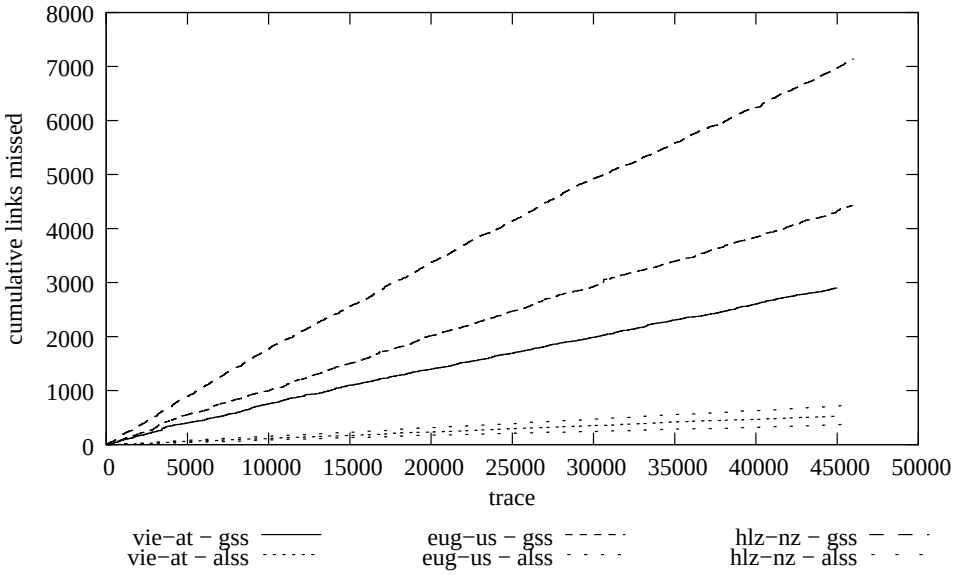



Figure 4.10: The number of links economical MDA traceroute missed on a representative sample of vantage points. The links have been separated into the stop set which caused the miss.

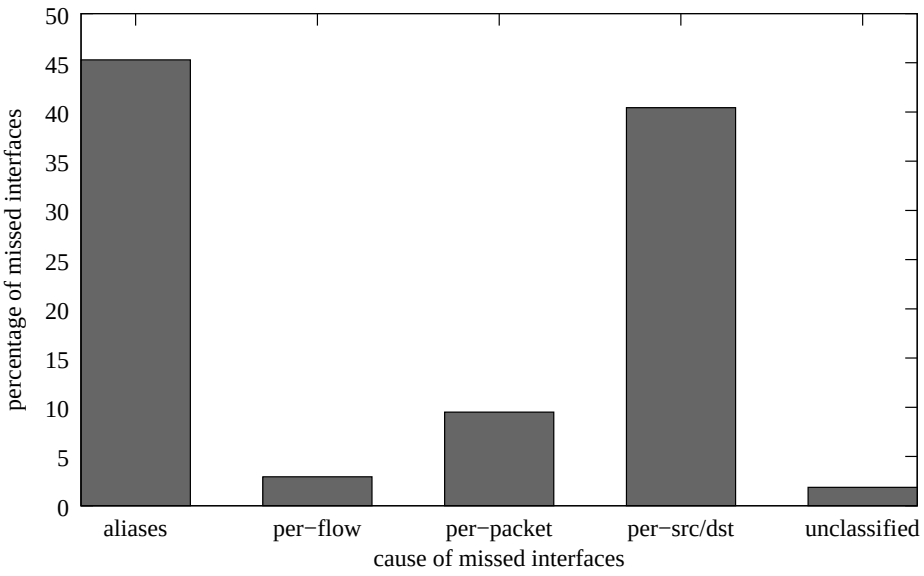



Figure 4.11: Path characteristics which cause doubletree to miss interfaces while probing. The majority of missed interfaces are due to aliases and per-source/destination load balancers. Aliases and per-source/destination load balancers can only be discovered by re-probing the path with different vantage points.

one router replies with more than one address, a path collected with traceroute could see a different topology to one collected with doubletree traceroute. In fact, two consecutive doubletree traces might see a different topology. Whether or not this is a cause for concern is entirely dependent on the end goal of the topology mapping project. This is because two traces where there is a difference caused by aliases are describing the same router-level path. If the topology at the interface level is important, then doubletree would benefit from sending multiple probes per TTL, however, this impacts the efficiency of the algorithm.

The next largest cause of missed topology are per-source/destination load balancers, making up 40% of the missed interfaces. These links are missed because a global stop set entry from another vantage point has caused tracing to halt. Because the other vantage point has a different source address, the per-source/destination load balancer has routed the probe packets on an alternate path, thus invalidating the doubletree assumption that paths form trees converging on a destination from multiple vantage points. This a concern for doubletree as regular traceroute mitigates this effect somewhat by repeatedly probing each hop from multiple sources. Each source will potentially see a different outgoing link and thus improve the overall coverage. As doubletree only requires one vantage point to see an interface before it is used in the global stop set, the alternate paths are missed. A possible way to improve doubletree's coverage with respect to per-source/destination load balancers would be to alter the global stop set such that an interface needs to be seen from a certain number of vantage points before it is to halt future probing.

With around 10% of misses, per-packet load balancers are the next most prevalent. Because per-packet load balancers pay no attention to the IP header fields, a different interface can be seen with each probe sent. In this case, we miss topology for much the same reason as we do with aliases. For example, two vantage points A and B trace toward the same destination, B stops after discovering an interface previously seen by A. However, there is a per-packet load balancer later in the path which causes the path seen by traceroute to differ from the path observed by A. As with other types of load balancing, the only way to see the alternate paths caused by per-packet load balancers is send multiple probes. Again, if we ensure that a

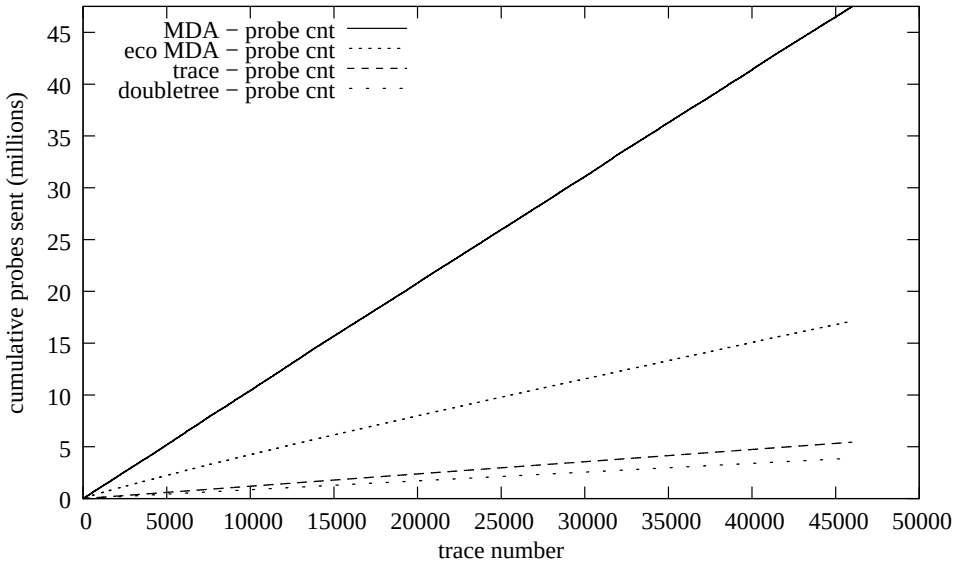



Figure 4.12: Per-trace cumulative probe usage. 46,000 destinations

number of vantage points have seen an interface in the path to a destination before allowing a trace to be halted, we should discover more of these per-packet load balanced links.

Per-flow load balancers make up less than 5% of the misses we observed. This is most likely due to the Paris traceroute implementation used by both traceroute and doubletree in these experiments. Paris traceroute maintains a constant flow identifier such that per-flow load balancers always forward probe packets out the same interface. The remainder of the interfaces for which we have no MDA traceroute data are labelled as unclassified.

One of the most significant causes of missed interfaces in doubletree is also applicable to economical load balancer traceroute. Per-source/destination load balancers are not affected by repeated probing, we see only one outgoing interface for each vantage point which probes them. Therefore if economical MDA traceroute stops earlier in the path, the alternate interfaces are missed.

### Probe Usage

Once we have checked the topology gathered and determined the reasons for any shortcomings, the next step is to ensure that the efficient algorithms have indeed resulted in a reduction in the number of probes sent. We plot the cumulative number

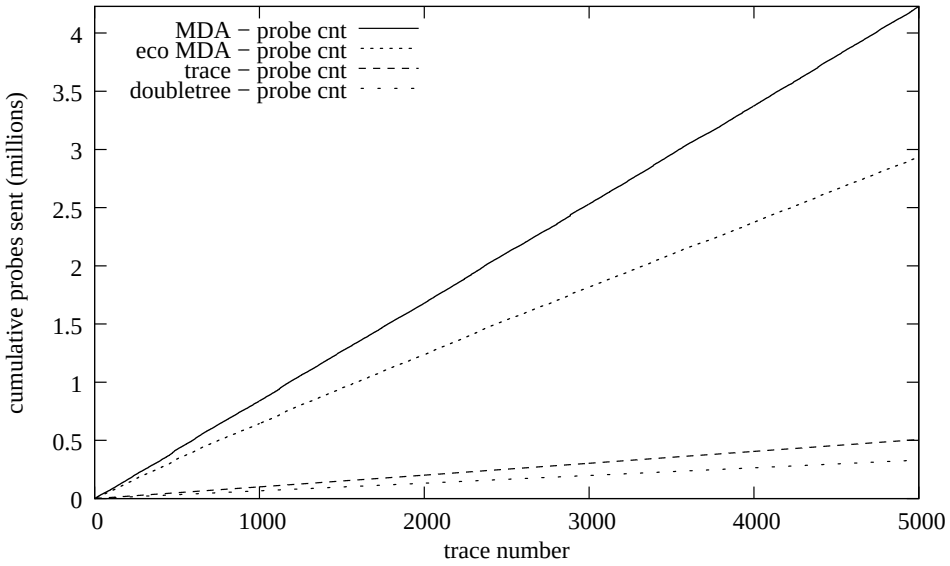



Figure 4.13: Per-trace cumulative probe usage. The global stop set was disabled in both doubletree and economical MDA traceroute.

of probes used to trace each destination in Figure 4.12. The reduction in probes between the standard methods and their efficient counterparts can be seen. The top line, MDA traceroute, sends over 46 million probes, whereas economical MDA traceroute only sends around 16.5 million. This represents a 64% reduction in probes. We see a similar reduction in probes used between doubletree and traceroute. Traceroute sends nearly 5.5 million probes, whereas doubletree only sends just over 2 million; a 61% reduction in the number of probes used.

To empirically determine the extent to which the global stop set is causes links to be missed, we re-run the experiment to a 5,000 destination subset of the original list. We choose 5,000 addresses to decrease the run-time and because in the original graph we can see that the lines are already cleared separated by the 5,000 trace mark. For this run, we disable the global stop set features of both doubletree and economical MDA traceroute. Figure 4.13 shows much the same distribution of probe counts as seen in the first experiment, with doubletree and economical MDA traceroute both using slightly more probes as is expected. In Figure 4.14 however, we can see the improvement in link discovery by both methods. Link coverage by economical MDA traceroute has increased to over 95%, again demonstrating the need to re-think the way that the global stop set is used.

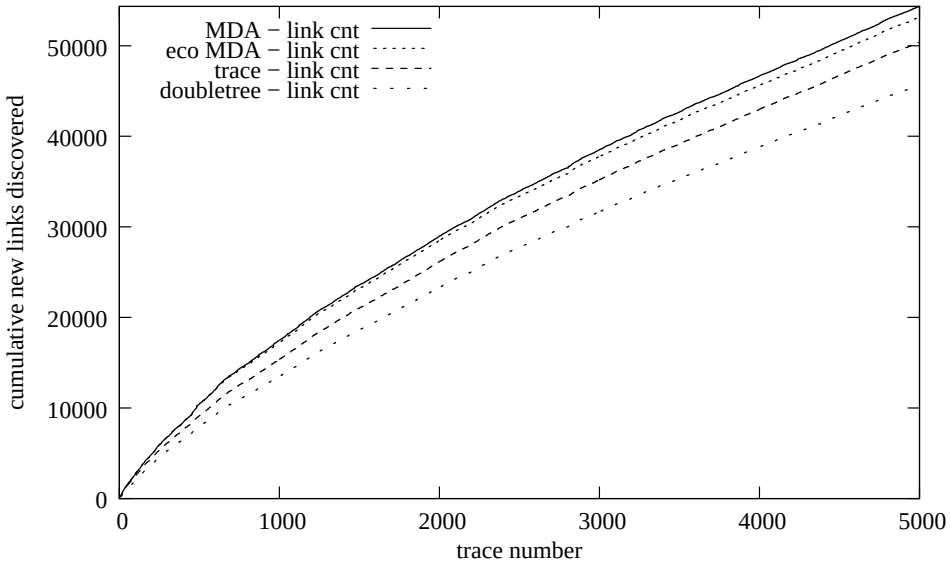



Figure 4.14: Per-trace cumulative link usage. The global stop set was disabled in both doubletree and economical MDA traceroute. Note that the eco MDA line is much closer to the MDA line than in the graph where the global stop set was used.

**ALSS Performance**

To check the performance of the ALSS in economical MDA traceroute, we plot both the number of probes used by the economical mode and the TTL at which the regular MDA was engaged. Figures 4.15 and 4.16 are a the probe usage results from san-us and vie-at vantage points which are representative of the others. The slight increase in probes over time is expected as the economical mode gathers more knowledge of the topology and is thus is engaged for longer lengths. This is seen in Figures 4.17 and 4.18, which show the TTL at which the economical mode finished. As anticipated, the TTL increases over time, with the peak lengths being reached after approximately 20,000 traces.

## 4.4 Per-source/destination Load Balancing

It is worth noting our discovery of a type of load balancer that Augustin *et al.* did not explicitly consider in [3]. We see evidence of routers which forward packets based solely on their source and destination addresses. While probing with MDA traceroute from a single vantage point, only one outgoing link is seen from a per-source/destination load balancer, however, when we combine the data from multi-

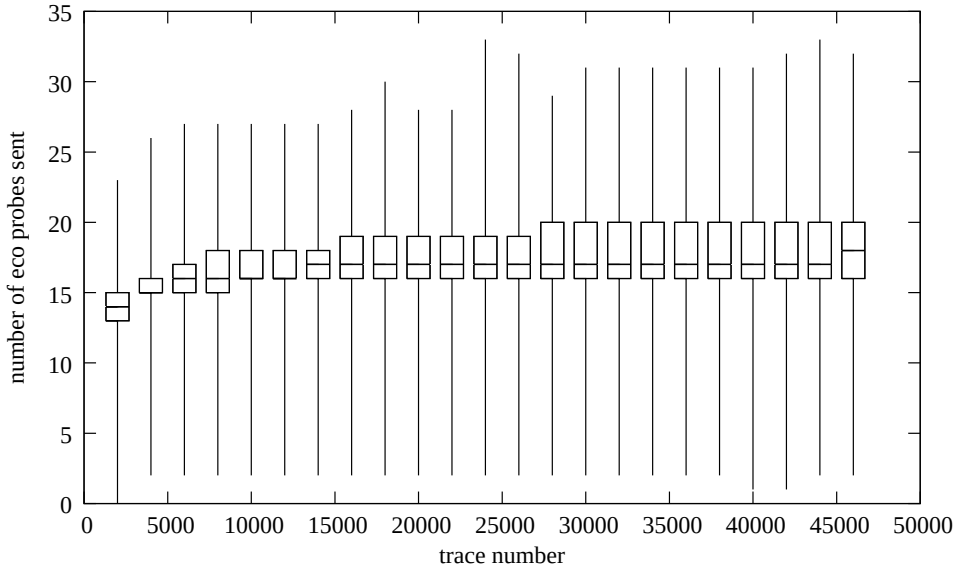



Figure 4.15: san-us. Box and whisker plots showing the minimum, lower quartile, median, upper quartile and maximum number of probes sent by san-us while in economical probing mode. Traces are grouped into 2,000 trace bins. The number of probes used by the economical mode increases over time indicating that the economical mode is remaining enabled further into the path.

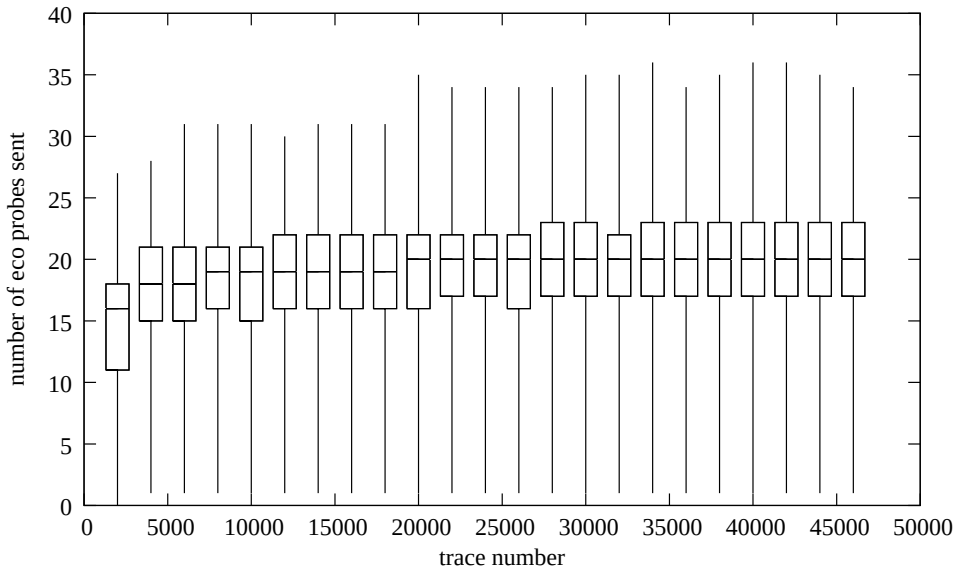



Figure 4.16: vie-at. Box and whisker plots showing the minimum, lower quartile, median, upper quartile and maximum number of probes sent by vie-at while in economical probing mode. Traces are grouped into 2,000 trace bins. The number of probes used by the economical mode increases over time indicating that the economical mode is remaining enabled further into the path.

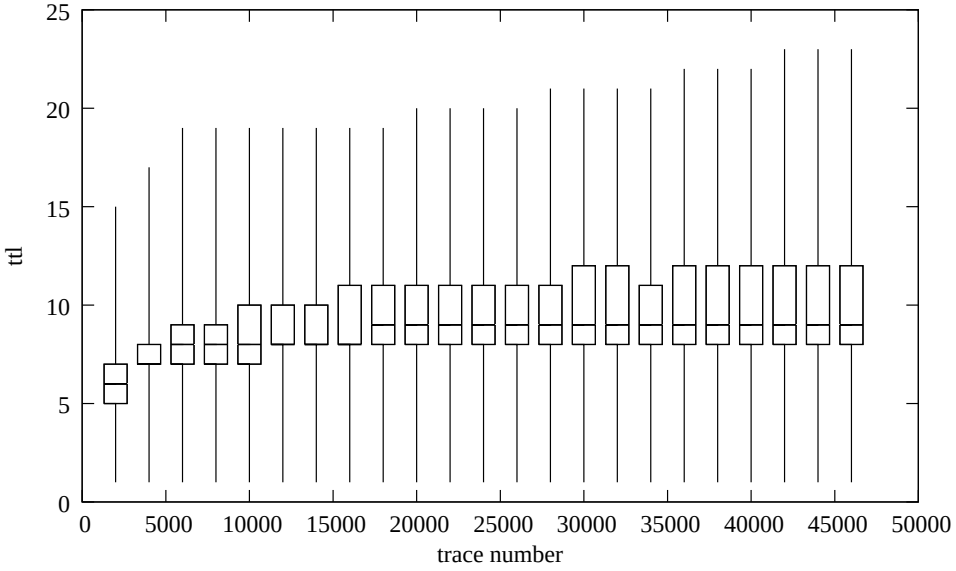



Figure 4.17: TTL at which the economical mode is switched off for the san-us vantage point. A higher TTL indicates that the economical mode was able to get further into the path, thus conserving more probes.

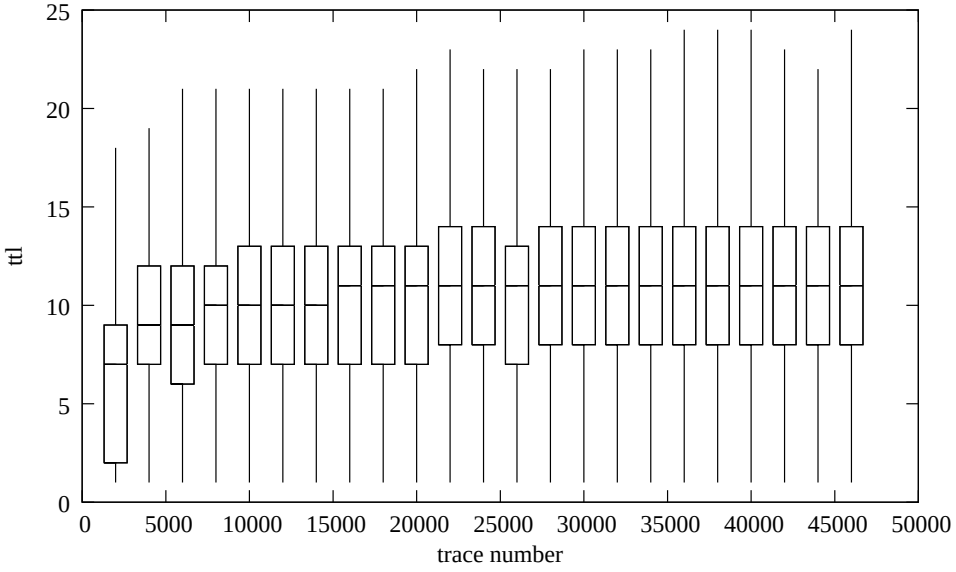



Figure 4.18: TTL at which the economical mode is switched off for the vie-at vantage point. A higher TTL indicates that the economical mode was able to get further into the path, thus conserving more probes.

ple vantage points, we see multiple unique outgoing links. It is unknown whether these are a type of load balancer not encountered by Augustin, or a more accurate classification of the load balancing that they refer to as per-destination. A per-source/destination load balancer would appear to be a per-destination load balancer if data from a single vantage point is considered. It appears that because these routers forward based on the source address as well as the destination address, the only way to be confident that all forward links have been discovered is to conduct probing from several vantage points.

Because per-source/destination load balancers are one of the leading causes of missed topology when using a global stop set, we propose a modification to the global stop set and associated forward probing algorithm, which would allow these alternate links to be discovered. We suggest that when probing with a large set of vantage points, an interface must be seen by a number of these vantage points before it is included in the global stop set. This is similar to the method that MDA traceroute uses to enumerate per-flow load balancers, except that we are using multiple source addresses to vary the fields consulted by the router when making a forwarding decision. We suggest using a table of values like the MDA does which increase depending on how many outgoing links are seen. For example, a per-source/destination load balancer with two outgoing links, would need to be probed by 15 vantage points to rule out a third link.

## 4.5 Summary

We have presented our results from comparing four macroscopic Internet topology discovery methods, classic traceroute, doubletree, MDA traceroute and economical MDA traceroute.

In real-world testing, while the local stop set portion of doubletree performs well and discovers upward of 99% of the topology that classic traceroute does, the global stop set is causing around 10% of topology to be missed. We see similar results from testing of economical MDA traceroute. The ALSS is performing well, while the global stop set causes large numbers of links to be missed.

We classify the interfaces that the global stop set is causing doubletree to miss. We conclude that in order to better discover interfaces caused by aliases and per-source/destination load balancers, the behaviour of the global stop set needs to be modified. We suggest that the global stop set is extended such that an interface must been seen in the path to a destination by a number of vantage points before it can be used to halt tracing. This will allow other vantage points to discover interfaces which the first vantage point to trace the destination may have missed.

# Chapter 5

# Challenges In Cooperative Probing

## 5.1 Introduction

Following our experiences with the large-scale testing described in Chapter 4, we discovered that due to several contributing factors, the various vantage points used were not able to work at even rates. We observed that vantage points which had low memory and slow CPUs would cause the coordination mechanism to become saturated with tasks that the slow vantage points could not act upon fast enough. To begin with we mitigated this effect by improving the efficiency of the coordination code itself, but we also considered a more scalable approach which involves manipulating the order in which vantage points are allocated traces such that the slower vantage points are given tasks with larger global stop sets and therefore more chance of being able to stop probing earlier in the path. We demonstrate that marginal gains can be made when using our optimisations on a dedicated measurement platform such as Ark. Also, we show that there is the potential for more significant gains to be had when using these optimisations in a system which has a more varied set of vantage points, such as a end-user based tracing project.

## 5.2 Motivation

The cooperative probing methodology which we use for doubletree and economical MDA traceroute is useful because it allows the vantage points involved to share information about topology they have discovered, reducing the amount of work

each has to do. However, for maximum efficiency, two vantage points cannot trace the same destination simultaneously. Because of this, a destination is probed in a round-robin fashion, such that once a vantage point probes a destination, the stop set is then passed to the next vantage point in the sequence which proceeds to trace it.

This causes a problem when not all vantage points are able to complete traces at the same rate. If there is even one vantage point which does not complete traces at the same rate as the others, all other vantage points are slowed down while they wait for the slow vantage point to complete traces that they are waiting for. Hence, toward the end of a cycle, the faster vantage points will be blocked, waiting on the slower vantage points to complete traces.

A slow vantage point can be caused by a number of factors. By virtue of being in geographically distinct locations, the vantage points tend to have varying path lengths. For example, the hlz-nz vantage point located in Hamilton, New Zealand, has a median path length of twenty hops, whereas sjc-us, located in San Jose, California, has a median path length of fifteen [8]. This means that hlz-nz takes longer to complete traces than sjc-us because, in general, each path that hlz-nz traces has more hops in it, thus taking longer to reach each destination.

If we consider the six vantage points used in our testing, Figure 5.1 shows the distribution of the number of hops in each path when tracing with vanilla traceroute. We use a count of observed hops to enable comparison to doubletree, as doubletree does not necessarily trace the entire path to each destination. A hop count will give an indication of the amount of work that each vantage point has to do relative to the others. We can see that there is variation between the vantage points, as described earlier.

Along with path lengths, another cause of difference in tracing speed between vantage points is resource limitations. Several vantage points in the Ark infrastructure are several years old and have relatively slow CPUs and limited memory as shown in Table 5.1. Early versions of the cooperative probing mechanism were significantly affected by inefficient memory usage and threading implementations in Ruby. This resulted in vantage points with limited resources struggling to keep

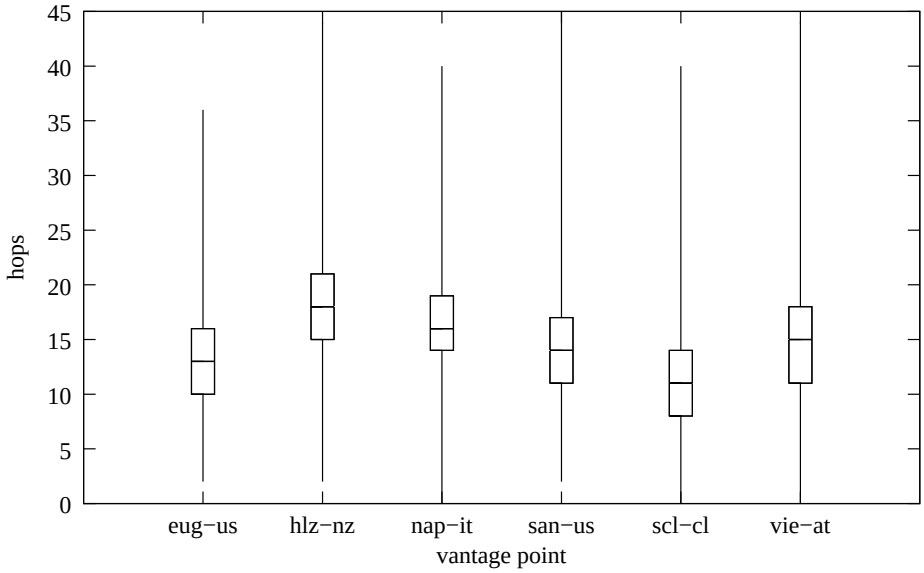



Figure 5.1: The minimum, lower quartile, median, upper quartile and maximum hop counts for traces using classic traceroute. Note that the hlz-nz and nap-it vantage points have longer paths than, for example san-us and scl-cl. This means that, overall, traces from hlz-nz will take longer to complete than from scl-cl.

| name | CPU | RAM | OS |
|---|---|---|---|
| hlz-nz | Pentium II (400MHz) | 362MB | FreeBSD 7.1 |
| eug-us | Celeron (2.4GHz) | 108MB | FreeBSD 6.2 |
| nap-it | Dual Pentium (2.0GHz) | 1985MB | FreeBSD 6.4 |
| san-us | Dual Pentium (2.5GHz) | 1926MB | FreeBSD 7.1 |
| scl-cl | Pentium III (1.4GHz) | 993MB | FreeBSD 6.2 |
| vie-at | Core 2 (2.4GHz) | 256MB | Ubuntu 4.1 |

Table 5.1: Specifications for the Ark vantage points used. These are significantly different in terms of CPU power, memory available and OS used.

up with the others. hlz-nz, for example was slowed down enough that Marinda, the shared memory system which allows communication between vantage points, became overloaded and caused the entire experiment to fail.

Also, doubletree is suited well to a system which utilises consumer machines such as traceroute@home [10]. It would be advantageous for such a system to allow end-users, who are taking part voluntarily to specify how much data the tracing can use. For example, in New Zealand, DSL users have a capped amount of data, so a participant on a small data plan may set the rate at which probe packets are sent at a much smaller amount than a research organisation. This has the effect of altering the amount of time that each trace takes to complete, thus the faster members would be waiting for the slower ones to complete traces.

## 5.3 Optimisation Method

In order to counter this, a system must be devised so that slow vantage points can still make the most of the advantages allowed by doubletree, but do not force other vantage points to sit idle whilst they complete traces. The system that we propose uses logic on the control server such that it keeps track of how long each vantage point is taking to complete traces. It can then use this information to control the order in which destinations are allocated to vantages points.

For example, if we have four vantage points, A, B, C and D. A and B are able to complete 10 traces a second. C, 5 traces a second, and D, 1 trace per second. In this situation, when A, B and C are finished with all of the available fresh destinations, D will still have a large number locked. To overcome this, the control server must keep watch and ensure that D never takes addresses that have not been traced by all of the other vantage points. Similarly, C must only be given an address once both A and B have traced it. A and B should be free to request fresh addresses whenever they run out of work as they will complete traces before the others.

Our control mechanism uses two different priority levels for destinations. Destinations which have not been traced previously are designated as low-priority and can be selected by any vantage point which has run out of high-priority jobs. The

high-priority jobs are designated for a specific vantage point. To optimise the ordering, we keep track of the time between making a destination available for tracing and receiving a completion notification. We then use this time to update a rolling mean value for the vantage point which just completed it. Thus, over time, we gain an estimate of the various speeds at which the vantage points are completing traces. When a trace is to be allocated to a specific vantage point, rather than simply using a round-robin process, we select the next fastest vantage point. The effect of this is that by the time a slower vantage point receives the destination to trace, the global stop set will be larger, and so the slow vantage point should have to do less work, thus increasing its speed.

## 5.4 Results

### 5.4.1 Ark

We begin by testing our optimisation methods using the same six Ark vantage points that we use for the large scale testing described in Chapter 4. We run two doubletree experiments, each to the same 46,000 destination list. The first experiment we run is regular doubletree with no additional optimisations. That is, when a destination has been traced, it is passed the next vantage point in alphabetical order.

Figure 5.2 shows the number of probes that each vantage point used to trace the destination list. When compared to Figure 5.3, we can see that there is only a marginal improvement in terms of even distribution of effort between vantage points. We can see that scl-cl and vie-at have sent more probes to become marginally more even with the other vantage points. We also see that nap-it has had a slight reduction in probes used to fall into line with hlz-nz and san-us.

Figures 5.4 and 5.5 illustrate the over time that each experiment took. We again, see a negligible improvement; scl-cl takes slightly longer to complete. Overall, the experiment completes around three minutes quicker with optimisation turned on (a 1% improvement).

Figure 5.6 helps explain why we see only minimal differences between the vantage points in terms of probes and time. We can see that the distribution of hops

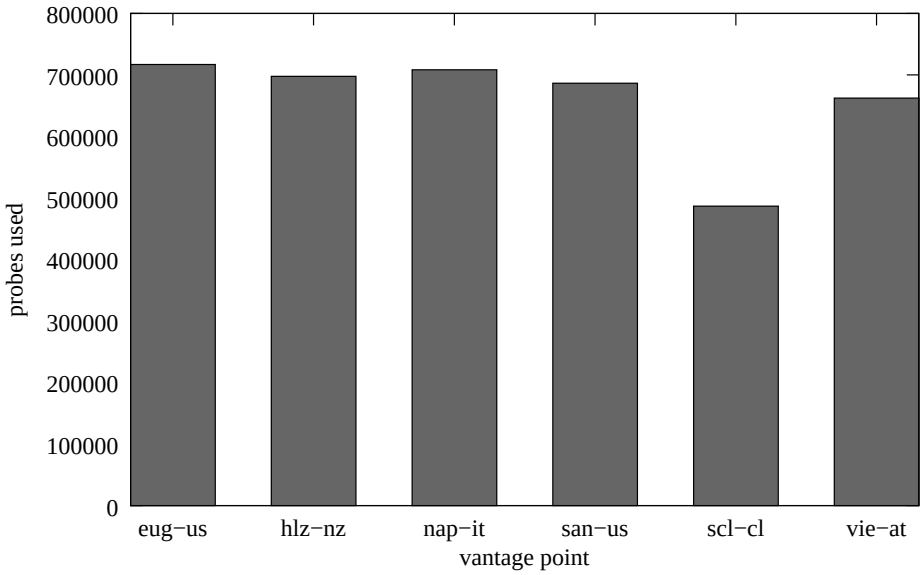



Figure 5.2: The number of probes that each vantage point uses to conduct doubletree traces to 46,000 destinations. The order in which vantage points trace a destination has not been optimised for these results. Note that the distribution of probes is similar to the distribution of hop counts in Figure 5.6.

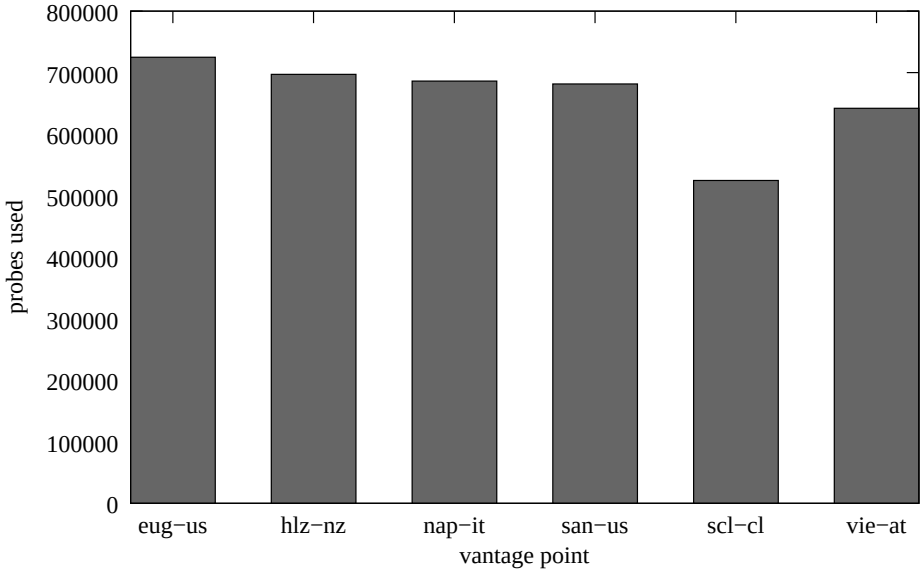



Figure 5.3: The number of probes that each vantage point uses to conduct doubletree traces to 46,000 destinations with the ordering of vantage points optimised. There is a marginal gain, the overall probes used is reduced by a fraction of a percent. The number of probes used by hlz-nz, nap-it and san-us has been smoothed slightly so that each is using almost the same number of probes.

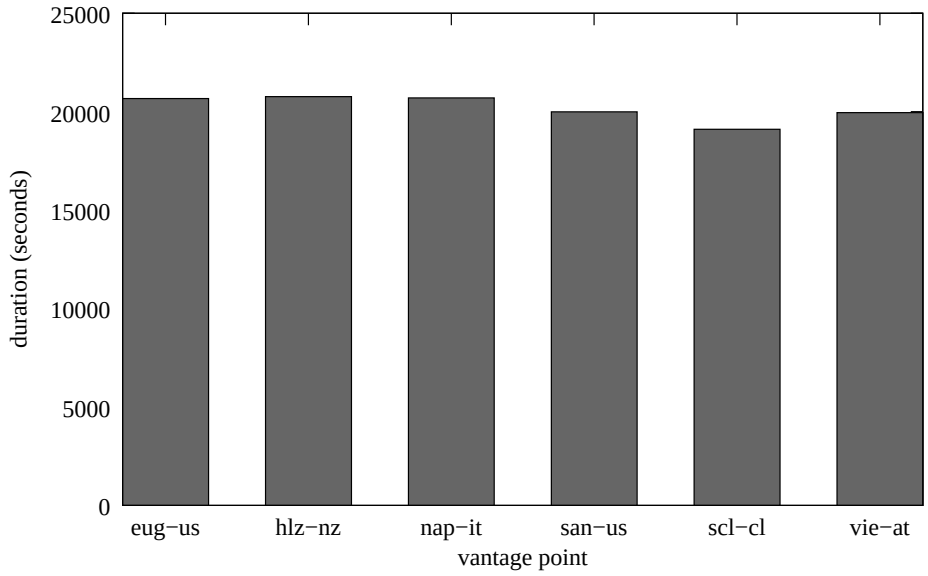



Figure 5.4: Without optimised ordering. The overall amount of time taken to conduct doubletree traces to 46,000 destinations. All vantage points take roughly the same amount of time to complete their traces.

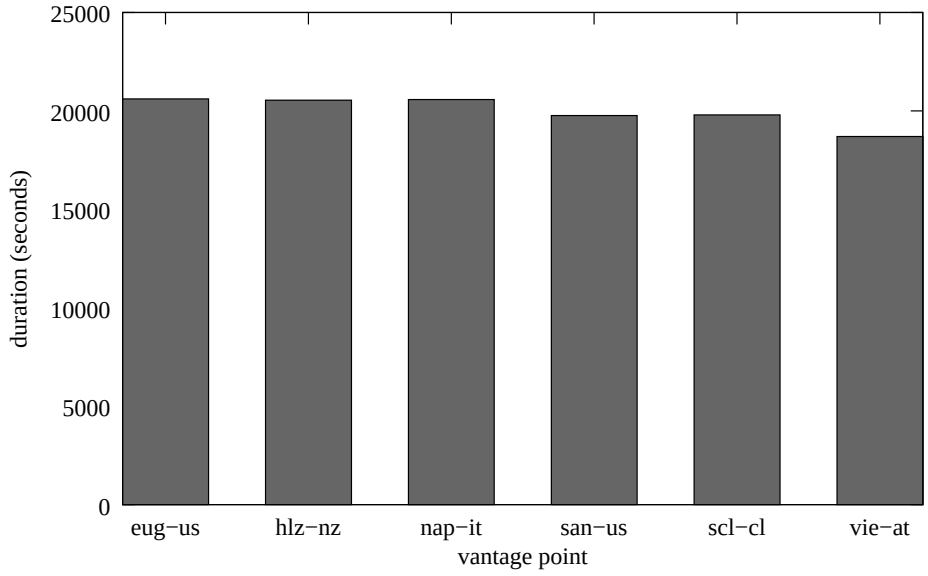



Figure 5.5: Optimised vantage point ordering. The overall amount of time taken to conduct doubletree traces to 46,000 destinations. Optimisation has, as with the probe usage, reduced the differences between vantage points marginally.

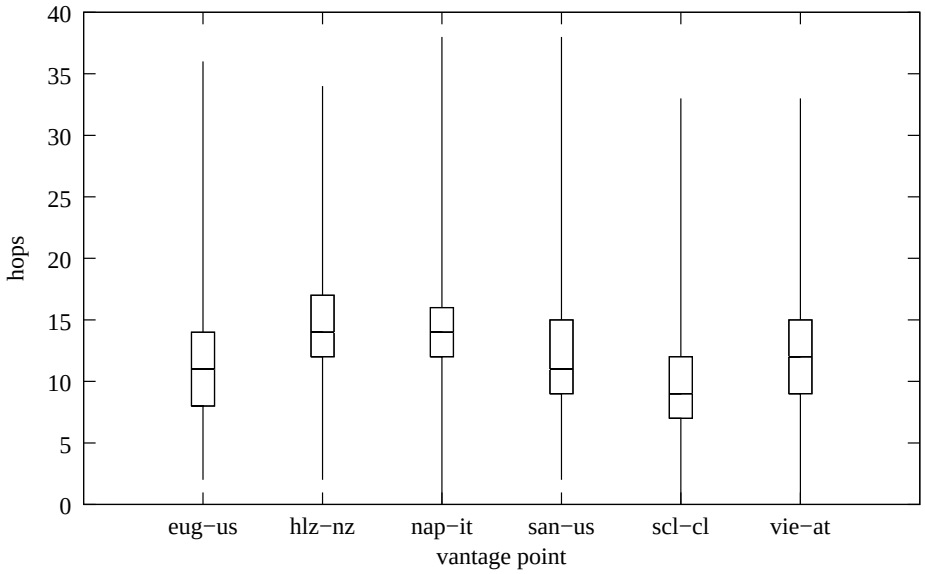



Figure 5.6: The minimum, lower quartile, median, upper quartile and maximum hop counts for traces using doubletree traceroute. Compared to Figure 5.1, the variation in hop counts is much smaller. This impacts the extent to which optimisation of vantage point ordering is effective.

observed by doubletree is much less varied than those observed by vanilla traceroute in Figure 5.1. Many of the hops which cause some vantage points to have longer paths are seen in the majority of traces from that vantage point. Therefore we see less variance in the number of hops visited by each vantage point when using doubletree because these common nodes are not re-visited. Because there is only a small variance in hops visited, there is less to be gained by optimising the order in which vantage points are assigned traces.

### 5.4.2 Varied Probing Rates

In order to test the optimisation method on a system that has vantage points which probe at varying rates, we limit the rate at which three of our vantage points can send probes. We reduce the probes per-second rate on hlz-nz, nap-it and scl-cl from 50, to 25, effectively halving the speed at which they can complete traces. We then re-run the two experiments using a 10,000 destination sub-set of the original list. We use a smaller list because the optimisations are done on a per-destination basis, and therefore is independent of the number of destinations traced.

Both Figure 5.7 and Figure 5.8 show much the same distribution of probes used

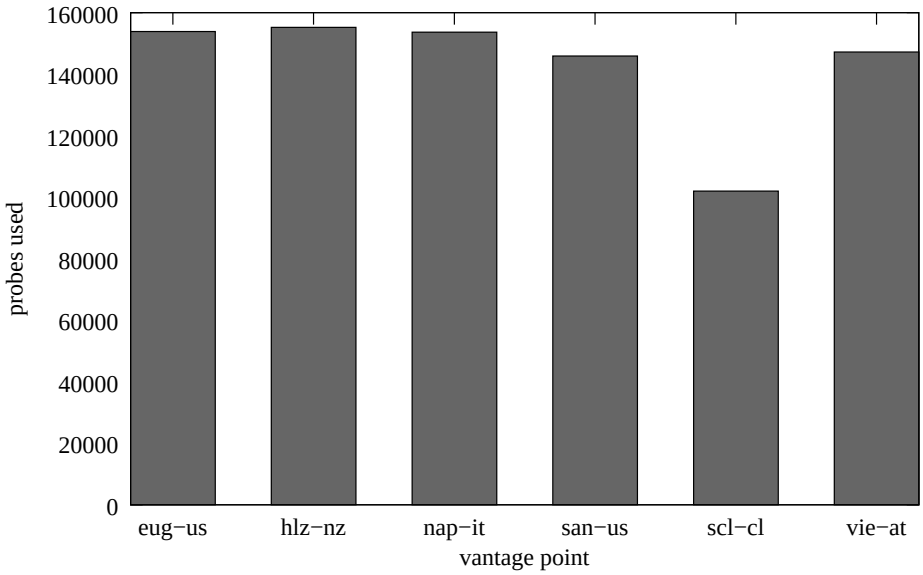



Figure 5.7: No optimisation. The number of probes used to conduct doubletree traces to 10,000 destinations. hlz-nz, nap-it and scl-cl have been artificially slowed to 25 probes per second, half the speed of the other vantage points.

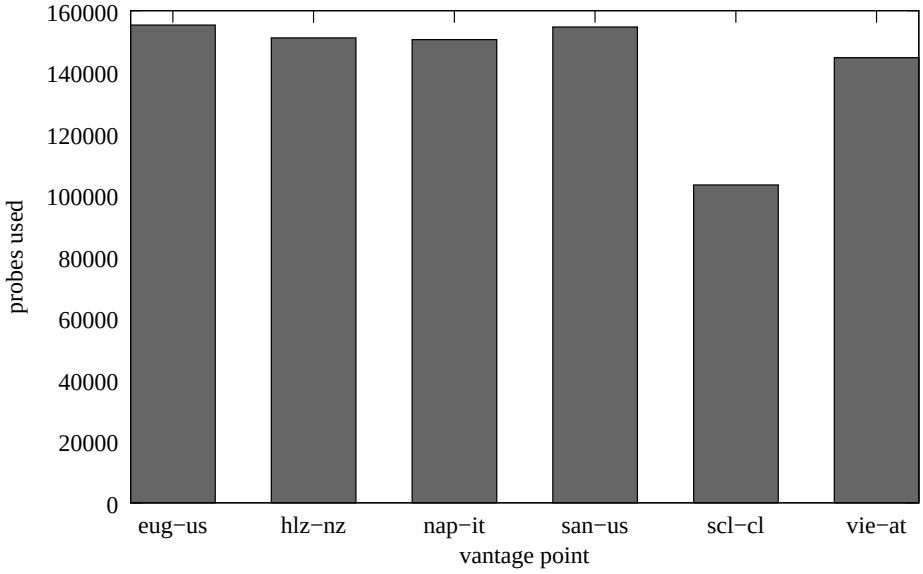



Figure 5.8: With optimisation. The number of probes used to conduct doubletree traces to 10,000 destinations. hlz-nz, nap-it and scl-cl have been artificially slowed to 25 probes per second, half the speed of the other vantage points. There is no significant difference in probe usage between optimised and non-optimised traces.

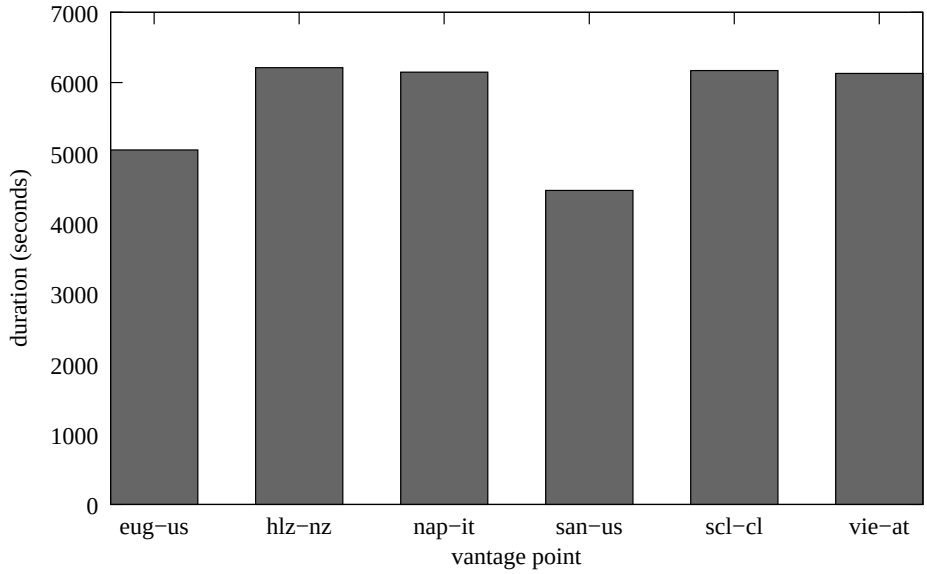



Figure 5.9: No optimisation. The overall time taken for each vantage point to conduct doubletree traces to 10,000 destinations. hlz-nz, nap-it and scl-cl have been artificially slowed to 25 probes per second, half the speed of the other vantage points. This is evident in their longer trace times.

as we saw with all the vantage points running at equal speeds. This is because although the vantage points are tracing at differing speeds, they must still trace all destinations and so the number of probes used will not vary greatly.

When we look at the amount of time taken by each vantage point however, we can see the advantages of optimising the ordering of the traces. Figure 5.9 shows the duration for each vantage point when optimisation is not used. The vantage points which have been slowed down are clearly visible. The anomaly is vie-at which also has a longer run time. This is most likely due to the traces being assigned in alphabetical order such that it has to wait for traces from all of the slow vantage points. Figure 5.10 shows that optimisation has indeed been beneficial. Both scl-cl and vie-at have had their durations significantly reduced, at the expense of eug-us and san-us, thus bringing all the durations closer to each other. We also see a cumulative reduction in run-time of around 8%.

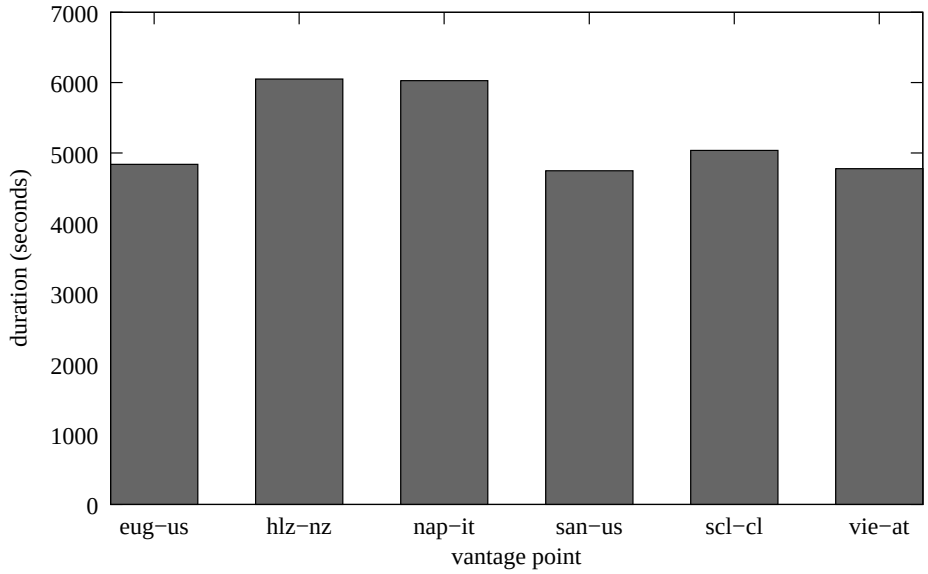



Figure 5.10: With optimisation. The overall time taken for each vantage point to conduct doubletree traces to 10,000 destinations. hlz-nz, nap-it and scl-cl have been artificially slowed to 25 probes per second, half the speed of the other vantage points. The advantage of optimising the ordering of the vantage points can be seen. Overall time is reduced by almost 10% and the time taken by each vantage point is smoothed in relation to the other vantage points.

## 5.5 Summary

Because we see less variance between vantage points of the number of hops visited, there is limited scope for optimisations to be advantageous in situations where all vantage points are able to conduct traces at the same rate. However, even though we observe minimal improvements when using optimisation on the standard Ark vantage points, the only cost for such optimisations, provided the assumptions made by doubletree about the paths forming trees hold up, is in calculating and storing a mean trace time for each vantage point. We therefore recommend using taking vantage point speed into consideration even when using a set of vantage points which are able to trace at similar rates.

We demonstrate that there are gains to be had by using optimisation in a system where there are vantage points with varied tracing speeds. For both of these experiments, better results may be observed by making use of a larger set of vantage points. This would allow the differences between assigning traces to vantage points in alphabetical order and in speed order to be more clearly seen.

Also, it may be beneficial to consider a different method of tracking vantage point speeds. For example, a weighted rolling mean similar to that used by TCP [17] could be used to favour newer data points.

# Chapter 6

# Stop Set Considerations

## 6.1 Introduction

In implementing doubletree in a large-scale topology discovery system which is designed to repeatedly trace a representative portion of the entire IPv4 address space, we must be careful that the state that is collected within the local and global stop sets remains accurate. For example, when routing changes occur, the state held in the stop set may not match the actual topology and as such cause doubletree to halt probing erroneously. To counter this, we consider the most appropriate time to empty the stop sets, thus removing potentially stale topology information. We briefly consider the global stop set, but determine that clearing the global stop set at the end of a cycle is sufficient. We then consider the local stop set, for which there is no clearly defined interval to clear it. We conduct simulations to attempt to empirically determine an optimum interval. Although we see a small decrease in topology discovery with longer clearing intervals, this is in most cases around 2% over a month of data.

## 6.2 Stop Set Clearing Intervals

The local stop set is a per-vantage point set of all the interfaces which have been discovered while in the backward probing phase, whereas the global stop set contains a set of interface-destination pairs which describe the interfaces seen in the paths to a destination. As these sets are used to decide where in a path to halt probing, it is

important that the information contained is up-to-date and accurate. If there is data about topologies that have since changed, this can cause doubletree to halt probing prematurely and therefore not discover topology. The more often that the stop sets are cleared, the less effective doubletree becomes at reducing the number of probes sent. On the other hand, clear the state too infrequently and changes in topology will be hidden by data in the stop sets.

### 6.2.1 Global Stop Set

Because the two sets hold different information, it makes sense that they are considered separately and cleared independently of each other. The global stop set is the simplest to consider; the global stop set holds the collective knowledge about the interfaces that comprise the paths that converge on each destination. Therefore it makes sense that when a dynamically generated destination list is being used for probing, this knowledge is discarded at the end of a cycle. The chances of an identical destination being randomly generated again within a short enough window such that there has been no major routing changes is fairly small [27].

### 6.2.2 Local Stop Set

The local stop set on the other hand, has no clear cut point at which it should be cleared. Because the local stop set is specific to a vantage point it should be kept for as long as the paths closest to the vantage point remain stable. While the paths leading away from a vantage point are stable, there is little chance of the doubletree algorithm stopping backward probing and therefore missing topology. This chapter reports on a study into the amount of topology which is missed over time by doubletree based on how long the local stop set is kept for. Using existing data from the IPv4 Routed /24 Topology Dataset [16] we have simulated the backward probing phase of the doubletree algorithm running on each vantage point in team one.

## 6.3 Simulation Design

In this simulation, we take existing Internet topology data and use this to simulate the backward probing phase of the doubletree algorithm. We then compare the results from the doubletree probing to the original data in order to see the extent of any missed topology. We repeat this experiment 30 times, each time with a different local stop set life-time. We have chosen to simulate this scenario as the length of time necessary to conduct such an experiment in the real-world would be prohibitively long (nearly three years) and changes in Internet routing would mean comparisons would be flawed.

The topology data we use is sourced from the CAIDA IPv4 Routed /24 Topology Dataset [16]. We use the data collected by the 13 team one vantage points during the month of June, 2009. This amounts to over 104 million traces in total. As a starting point for doubletree, we use firsthop values tuned to the individual vantage point. These are the same firsthop values that we use in our real-world doubletree runs. See Section 4.3.1 for a description of how these values were obtained. For each run through the data, we record the links that doubletree discovers so that they can be compared to the trace data. We use stop set-lifetimes that increase exponentially to give a fine-grained coverage of the smaller intervals and a more coarse coverage of the longer intervals. The formula we use is $n^2 * 60$ where $n$ is the run number. We begin with an interval of 60 seconds ($1^2 * 60 = 60$). That is, the local stop set is emptied every minute. The next interval is 240 seconds ($2^2 * 60 = 240$) and so on up to 54,000 seconds (15 hours). Along with generating data about the links that doubletree discovers, we also extract the links that trace discovers.

## 6.4 Results

Figure 6.1 shows the number of interfaces which were missed by the various vantage points used in the simulation when the local stop set is not cleared for the duration of the simulation. This simulates the base case where we never clear the local stop set. It appears that some vantage points are missing a far higher number of interfaces than others. When we look at Figure 6.2 however, we see that the total size of

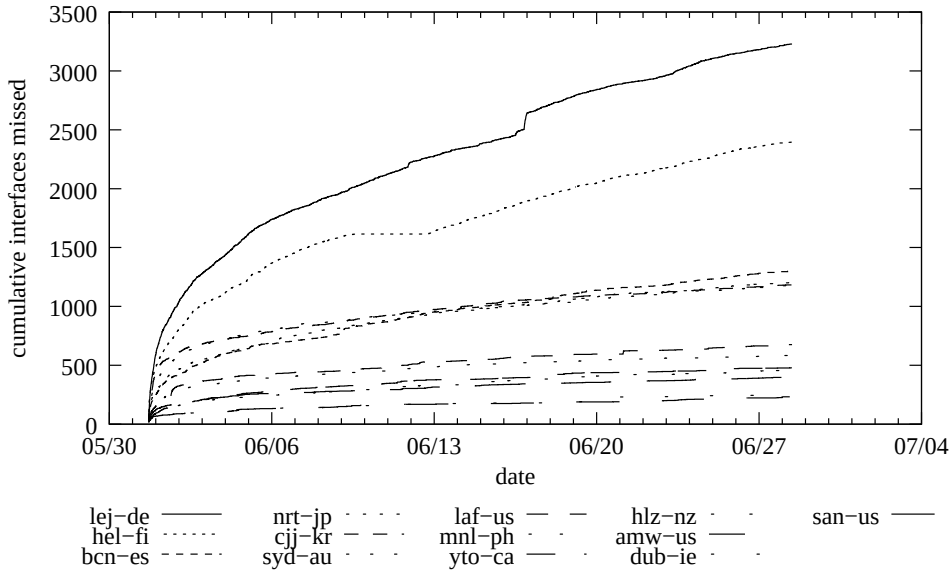



Figure 6.1: Cumulative count of interfaces that are missed due to doubletree's local stop set path assumptions. Simulated using Ark Team 1 traceroute data from June 2009.

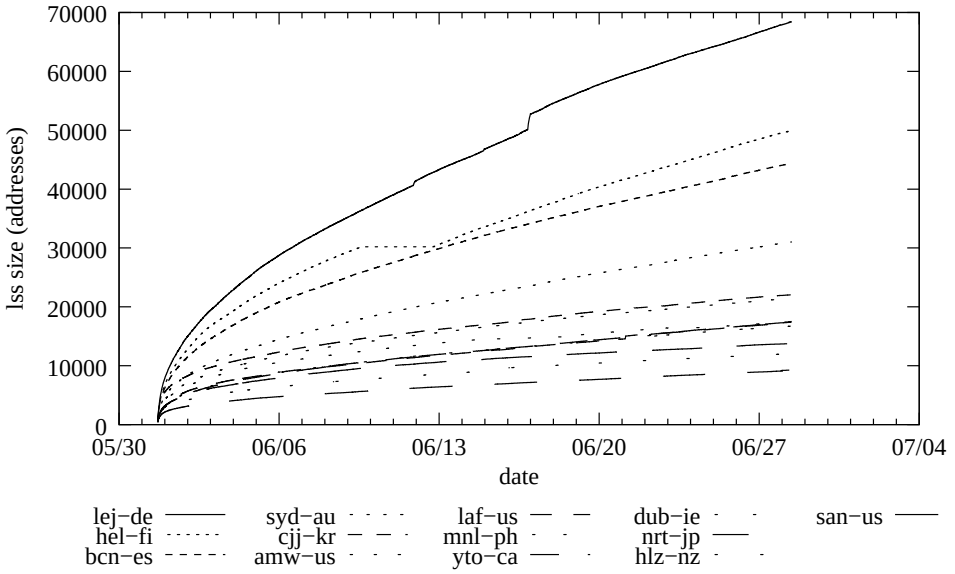



Figure 6.2: Number of interfaces held by the local stop set of each vantage point over time.

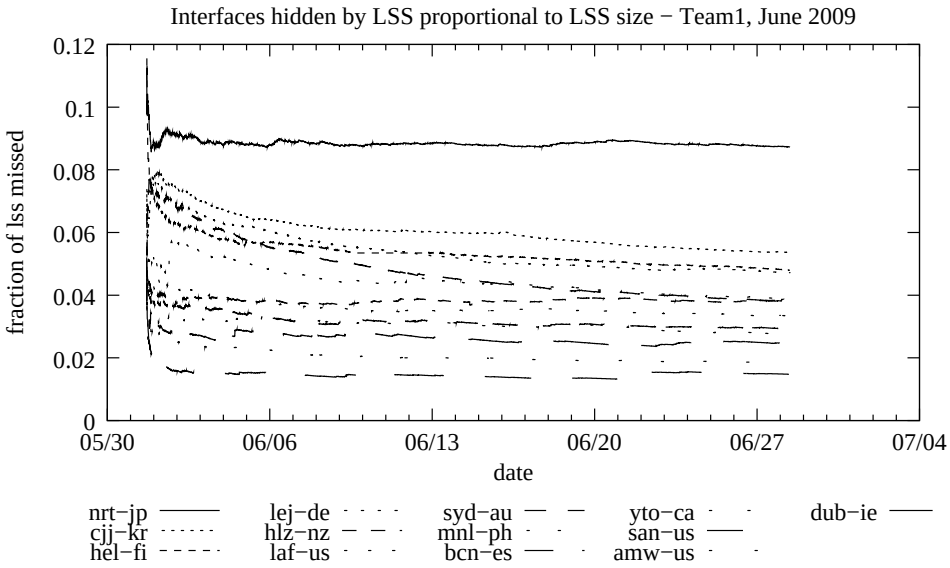



Figure 6.3: The proportion of interfaces missed in relation to the total number of interfaces held in the local stop set for each vantage point.

each vantage point's local stop set also varies. This is most probably due to varying amounts of diversity in the paths prior to the first-hop value used by doubletree. That is, some vantage points a higher out-degree for nodes within the first $n$ hops, where $n$ is the doubletree first-hop value. If we take this variance into account by plotting the number of interfaces missed by each vantage point in proportion to its local stop set size, we get Figure 6.3. From this graph we can see that all of the vantage points in the simulation discover well over 90% of the interfaces with some discovering over 98%. There does not appear to be an increase in the proportion being missed over time. Thus implying that not clearing the local stop set may not cause the topology loss feared. These results are supported by results from our large scale testing outlined in Chapter 4. We discuss this further and suggest an appropriate interval at which the local stop set should be cleared in Section 6.5.

Figure 6.4 shows the proportion of links that doubletree discovers when compared to traceroute for each of the clearing intervals in the simulation. We see that with the exception of nrt-jp, there is not a large variance between the different clearing intervals. This is again the case in when we consider the number of probes used in Figure 6.5. We can see that the probe count is virtually unaffected by the changes in clearing intervals.

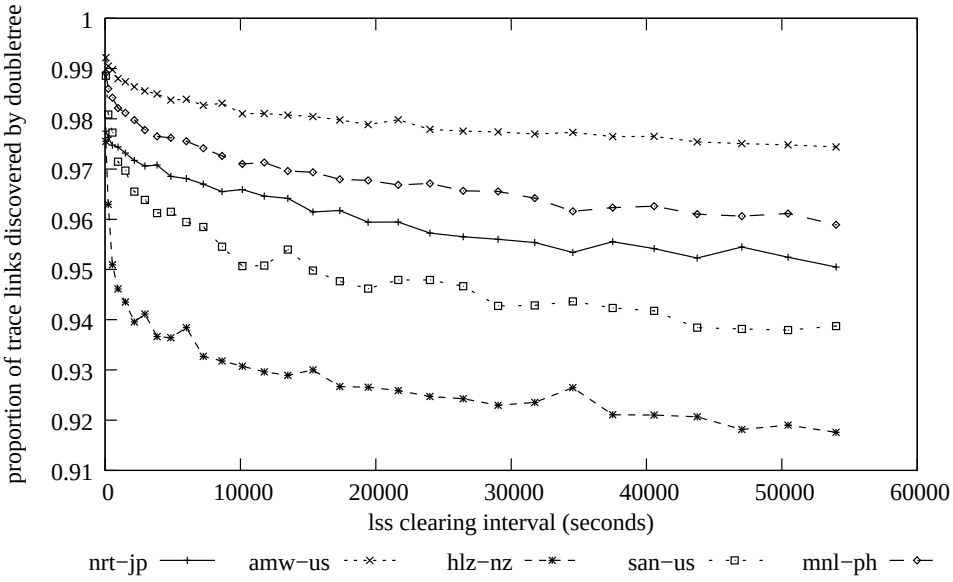



Figure 6.4: The proportion of links discovered by doubletree during backward probing compared to traceroute for a representative selection of vantage points. Only hops up to the doubletree first hop value are considered.

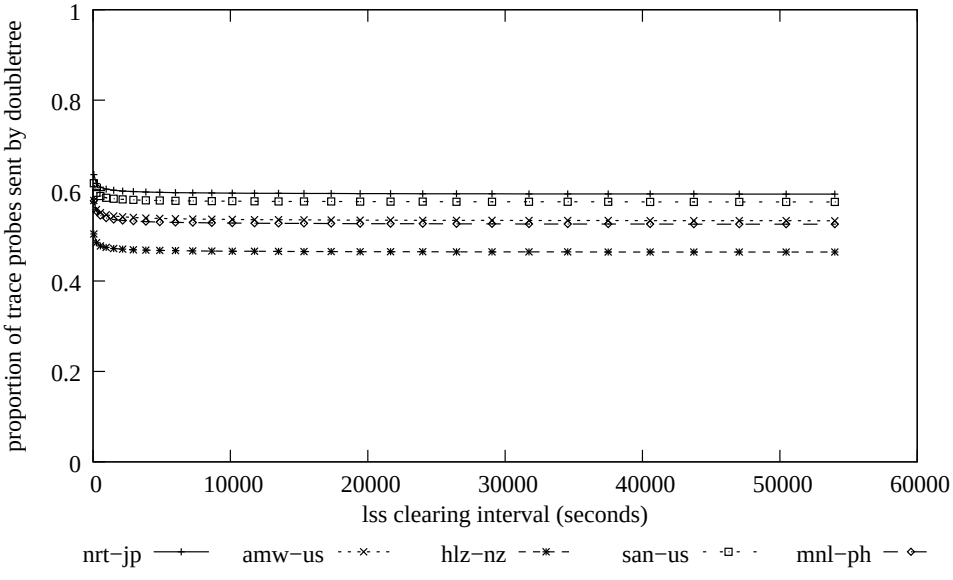



Figure 6.5: The proportion of probes used by doubletree compared to traceroute for a representative selection of vantage points.

## 6.5 Discussion

When we consider the base case where we never clear the local stop set, we see that with the exception of a couple of vantage points (lej-de and hel-fi), the number of interfaces missed does not rise quickly as seen in Figure 6.1. For most vantage points, interfaces are missed regularly at the beginning but this rate drops off over time. Once we take into consideration the total size of each vantage point's stop set as in Figure 6.3, we see this trend more clearly. The number of interfaces missed drops quickly in the first few hours/days and then stabilizes.

When we look at the data from the entire simulation, we see that the number of links missed by doubletree generally increases over time (Figure 6.4). This growth is very slow however, and after a month of tracing, the simulation shows that in most cases less than 1,000 interfaces are missed. This implies that the effect of routing changes on doubletree's topology discover is minimal and leaving the stop set uncleared for an entire month would not affect the data gathered too much.

The effect of varying the interval at which the local stop set is cleared has even less of an effect on the number of probes used. When we look at 6.5 we see almost no decrease in the number of probes used when the stop set is cleared every 54,000 seconds (15 hours) compared to when it is cleared hourly. The only effect seen is when the stop set is cleared very quickly such that doubletree does not have enough knowledge about the topology to probe efficiently.

From these results, we see that the interval at which the local stop set is cleared has little effect on both the topology discovered and the number of probes used. Another aspect to be considered is the memory requirements of the local stop set. As the size of the stop set increases, so will the memory usage of the probing process. If it is running on machines with limited resources, this could present a problem, so some care must be taken such that the stop set is cleared regularly enough to prevent the probing process from exhausting the available resources. As with the global stop set, we decide that the most appropriate time for the local stop set to be cleared is at the end of a tracing cycle. This allows the data gathered to be used as a discrete set. Again, care must be taken if the cycle is going to take a sufficiently long amount of time that routing changes or memory usage will affect the results.

An unexpected result from this simulation is that we see remarkably better link coverage by doubletree in simulation than we see in the real-world experiments described in Chapter 4. The cause for this is presumably load balancing routers. In this simulation, we are treating the traceroute data as a ground truth regarding the topology. However, the traceroute data is incomplete due to its inability to completely discover load balanced paths. Due to the low number of interfaces missed by this simulation, we conclude that the majority of missed interfaces in real-world doubletree runs are caused by load balancing and not by routing changes.

# Chapter 7

# BGP Guided Probing

## 7.1 Introduction

The various methods for increasing the efficiency of a macroscopic traceroute project discussed thus far have all attempted to reduce the number of probes sent by maintaining state at an interface level about paths that have been observed with previous probing. Another potential way to reduce the number of probes required to trace a path is to use BGP routing information specific to each vantage point [11]. We firstly use simulations to determine whether it is feasible to use BGP routing data coupled with existing traceroute to accurately estimate the length of an IP path to a destination by learning the mean width of the ASes in the path. We then use another simulation to determine how many times an AS should be traversed before the width learned can be used to direct further probing. This is an exploratory examination of the area and we leave running a full simulation to determine link coverage for future work.

## 7.2 Overview

If a vantage point has access to the routing information specific to its location within the Internet, it is able to determine, with no probes, the AS path from it to a destination. With this information, the algorithm is able to determine the ASs in the path which have not been traced previously and can direct probes to them. We can use this ability to create a prober which specifically targets sections of paths which

| vantage point | bgp source | date | file name |
|---|---|---|---|
| ams-nl | RIPE RIS | 01/06/2009 | bview.20090601.0759.gz |
| nrt-jp | Route-Views | 01/06/2009 | rib.20090601.0000.bz2 |

Table 7.1: Sources of BGP routing data used in our simulations.

have not been traced previously. We would see similar benefits to doubletree at the edges of paths, but we would also be able to skip path segments mid-path, which doubletree can not do. This would mean that topology which may have been hidden by doubletree's stop sets is discovered.

In order to be able to probe a specific AS in a path, the algorithm must be able to determine the widths of each of the preceding ASes in the path. We use the term AS width to describe the number of interfaces that are seen in a traversal of an AS. If the width of each AS in the BGP path can be determined, the algorithm can then effectively skip sections of the path which have been seen with prior probing.

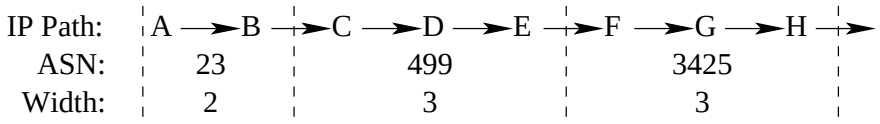
To establish whether we can use BGP data to estimate path lengths, we take existing Ark trace data and use it to learn the average width of all of the ASes in the paths. From these widths, we then attempt to predict the overall length of the path to each destination by taking the sum of the widths of each AS in the advertised BGP path.

## 7.3 Simulation Design

We test our algorithms in simulation by using a month of data from two vantage points in CAIDA's Ark infrastructure. We use the ams-nl and nrt-jp vantage points as they both have publicly available BGP routing data, as defined in Table 7.1.

We run two simulations. The first is to determine whether routing data can be used to accurately predict path lengths and the second is to determine the appropriate number of times that an AS must be traversed before using the calculated width to influence future tracing.

Our first simulation involves using the real-world trace data to learn average widths for the various ASes which are then used to estimate the total lengths of the




IP−AS Table

| IPs | ASN |
|---|---|
| A,B | 23 |
| I,J,K,L | 334 |
| C,D,E | 499 |
| X,Y,Z | 3160 |
| F,G,H | 3425 |

Figure 7.1: An example of how the simulator determines AS widths from an IP path and routing data. Each interface is looked up in the routing data to determine which AS it belongs to. Adjacent IP addresses which belong to the same AS are counted and a width for that AS is determined. In this example, the addresses C,D and E all belong to the AS 499, thus making our first width estimate for AS 499 three hops.

paths to each of the destinations in the data set. We first convert the trace data into a series of IP paths. Using these paths, we attempt to learn the AS widths. To do this, we traverse each path and, by using the routing data, look up the AS number of each interface in the path. When we cross an AS boundary, we update the width value held for the AS. An example of this is demonstrated in Figure 7.1. Each IP in the path is converted to its respective AS number and a width for each AS is calculated. In this example, the IP addresses represented by A and B both belong to AS 23, so the width estimate for AS 23 would be two hops. Because the width of an AS may vary slightly from trace to trace we have our algorithm learn the mean width. Once all of the paths have been parsed, we have a look-up table of AS numbers and their respective widths. Again using the destinations from the Ark data, we look up the corresponding BGP route using the longest matching prefix. From this we get a sequence of AS hops. For each AS in the path, we look up our table of widths. We take the sum of each AS in the path, up to, but not including the final AS, for which we add half of the recorded width. This is because we expect, on average, to only traverse half of the final AS before reaching the destination. We then compare these estimated lengths to the actual path lengths to determine how accurately we can predict the width of each AS.

We then attempt to empirically determine the number of times an AS should be

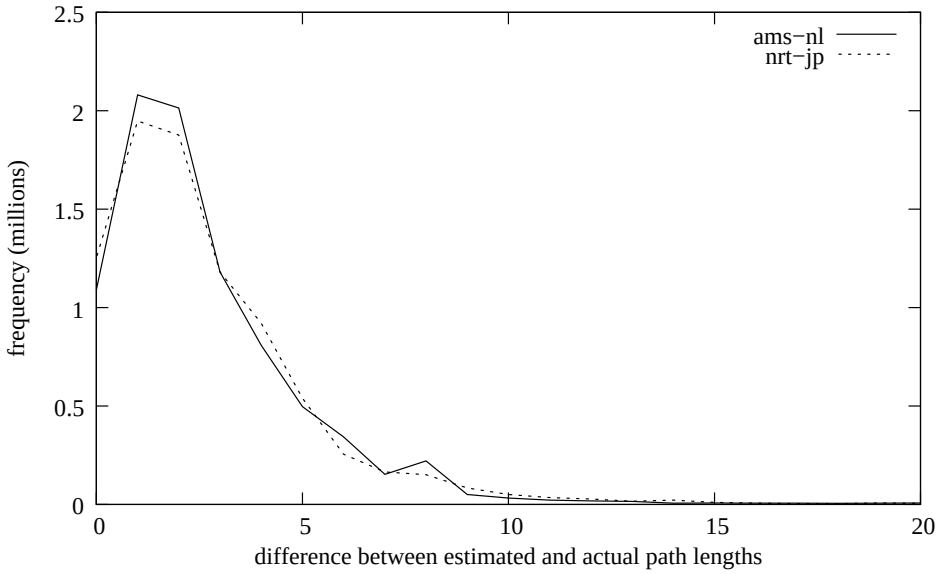



Figure 7.2: The frequency of differences between estimated and actual path lengths when simulated with a month of ark trace data from the ams-nl vantage point.

traversed before using the calculated width to influence future tracing. We run a simulation which varies the number of times that an AS must be traversed before using its width information to skip over it in future traces.

For each trace that was used in the simulation, we look up the AS path from the routing information. We then iterate through each AS in the path and determine whether we have traversed the AS enough times to 1) have an accurate width value and 2) reduce the chance of an alternate path through an AS being missed. If we do have an accurate width value, the TTL that traceroute probes is increased by the sum of the determined AS widths. If there is not enough data to make an accurate width determination, the TTL is left unchanged and tracing continues as with normal traceroute until the next AS in the path is reached and the test is performed again.

## 7.4 Results

The results from initial testing using BGP data to predict path lengths is encouraging. Figure 7.2 shows over 1 million path estimated precisely. Also, the majority of estimates are less than five hops different to the actual path length.

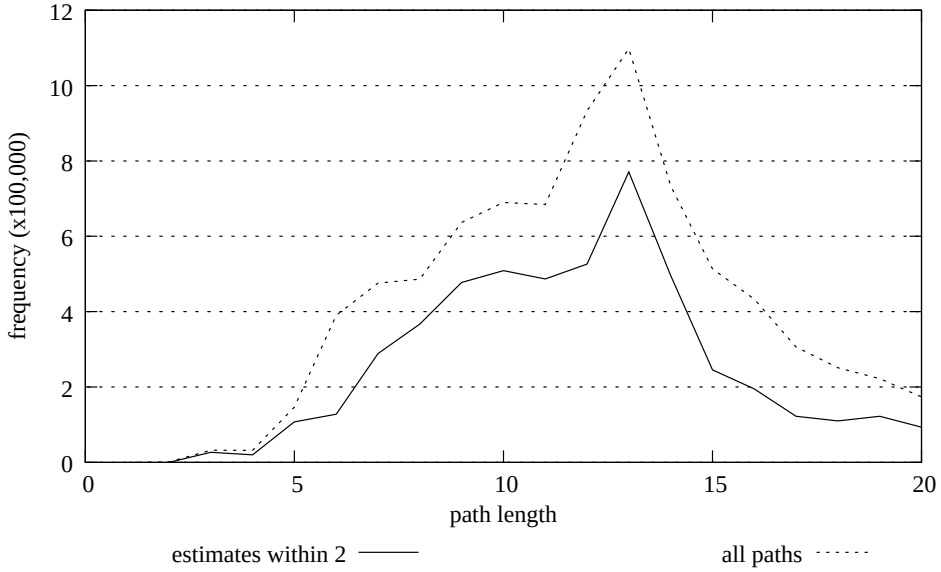



Figure 7.3: The number of paths where the difference between the estimated path length and the actual path length is less than two. Simulated using data from the ams-nl Ark vantage point

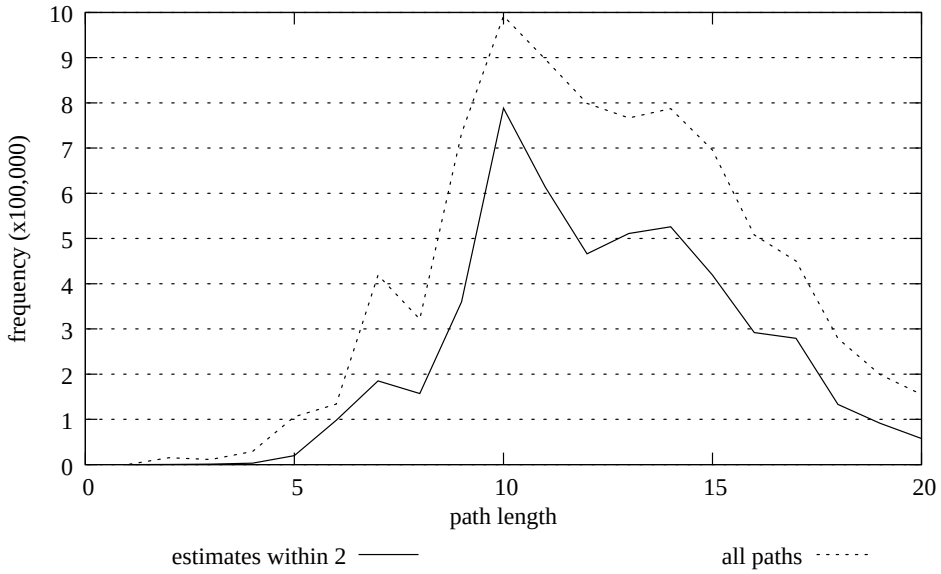



Figure 7.4: The number of paths where the difference between the estimated path length and the actual path length is less than two. Simulated using data from the nrt-jp Ark vantage point.

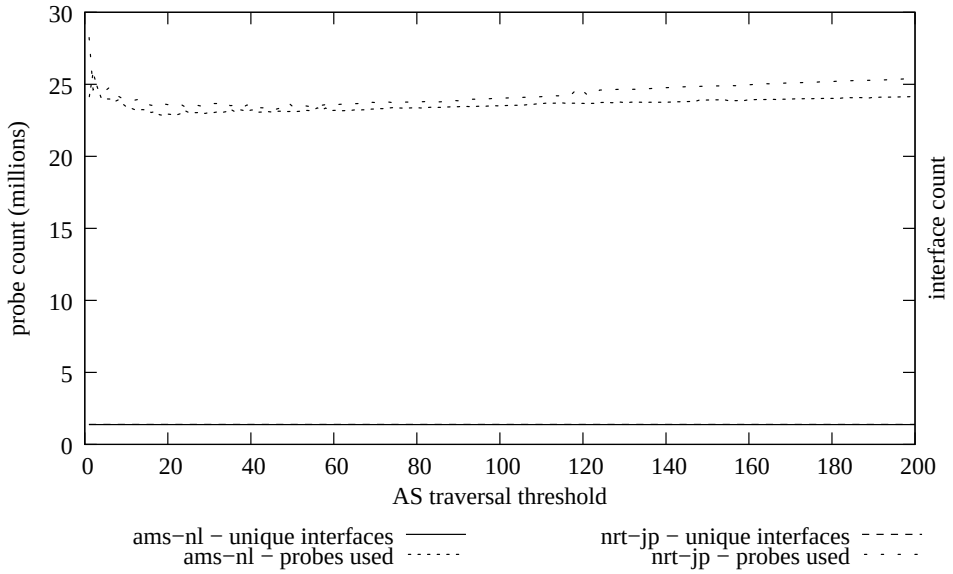



Figure 7.5: Results of using BGP data to skip previously visited ASes. The AS traversal count is the number of times that an AS must be traversed before it can be skipped in tracing. Varying the confidence value has negligible impact on the number of interfaces discovered (the ams-nl interface line is obscured by the nrt-jp interface line). A confidence value of approximately 20 appears to be best for reducing the number of probes.

Figures 7.3 and 7.4 shows the results when we plot the number of traces where the estimated path length was within two hops of the actual path length. We can see that the distribution closely follows that of the actual path lengths. We see that, especially for the ams-nl data, short paths (up to five hops) are accurately estimated. Also, for the majority of path lengths, the estimated lengths are within two hops of the actual path length.

If we look at the data from the simulation of the AS traversal threshold, illustrated in Figure 7.5, we can see the effect that varying the threshold has on both the probes used and the interfaces discovered. Foremost, we see that the effect on the number of interfaces discovered is negligible such that on this graph, the interface count is a horizontal line. Both vantage points have interface discovery counts which are within 20,000 of each other. Given that interface discovery is not impacted, the coverage of this algorithm is not in question. The other factor to consider is the number of probes used. The probe count for each vantage point is shown toward the top of the graph. For very low traversal thresholds, there is a larger number of probes used. This is likely due to the initial path length estimates

being markedly different to the actual AS widths resulting in regular traceroute must be used more frequently, thus not probing as efficiently. Also, as the threshold increases, the number of probes used also increases. This is due to having to use the regular traceroute algorithm more before the traversal threshold is reached. We determine that 20 traversals ensures maximum efficiency as both higher and lower traversal thresholds result in higher probe usages.

## 7.5 Summary

This chapter demonstrates that by using BGP routing data, it is possible to predict the IP path length to a given destination accurately. This is useful in active measurement projects as this data is essentially free. That is, obtaining this knowledge requires no active probing, thus if we can reduce the number of traceroute probes we send by making use of this data, we can reduce the impact that our unsolicited traffic has on the Internet. Also, because this data can generally be obtained far quicker than traceroute data can, we can improve the speed with which we can trace large numbers of destinations.

We also determine approximately 20 traversals to be a reasonable number of times to observe an AS before using its width in future tracing. 20 traversals ensures maximum probe efficiency as demonstrated in Figure 7.5.

The limitation of this method for improving traceroute efficiency is that each vantage point must have access to the BGP data for its routing. Having access to local routing data means that the routes used to select which sections of the path to trace will be more accurate than those gathered from other sources. Future work should look at the effect that using routes gathered from public sources such as RouteViews [1] in place of local routing information has on the accuracy of data gathered and on the number of probes used. There would need to be investigation into which route was the best to use. This would allow vantage points which do not have access to local BGP data to use this technique also.

# Chapter 8

# Conclusion

## 8.1 Contributions

This thesis provides several contributions to the field of macroscopic Internet topology discovery. We present a variation on the MDA, economical MDA traceroute which is designed to discover comprehensive topologies while minimising the number of probes used. We find that in real-world testing, we are able to maintain over 90% link coverage while reducing the number of probes used by around 60%. We are able to show that the ALSS is causing a reduction in probes, whilst maintaining link coverage of over 95%.

We discover that the assumptions behind the global stop set are flawed and cause our doubletree implementation to miss around 10% of interfaces. Upon classifying these missed interfaces, we discover that the topology loss is largely due to aliases and per-source/destination load balancing. Per-source/destination load balancing is a type of load balancing not explicitly considered by Augustin *et al.* in [3]. We determine that per-source/destination load balancers are not affected by manipulating the flow identifier as per-flow load balancers are. Alternate links in a path to a single destination are only seen from differing sources.

Our investigation into optimising the order in which tasks are given to vantage points shows that, in the Ark infrastructure, we are able to make minimal gains at no extra cost. However, in an environment where the vantage points are probing at significantly different rates, we reduce the overall time taken by nearly 10%. This will be useful in a massively distributed environment where the probers are

end-users with varying link speeds and data plans.

We also investigate the impact that the doubletree local stop set has on discovered links. We simulate clearing the local stop set at various intervals and observe the number of links doubletree misses. We determine that there is minimal difference in link discovery across the clearing intervals and conclude that the decision as to when to clear the local stop set should be made based on an acceptable memory overhead for storing the stop set on the probers.

Our work with BGP data shows that IP path lengths can be estimated using AS paths obtained from routing information. Based on this work, we simulate using BGP data to guide traceroute probes to previously undiscovered sections of paths. We simulate using different AS traversal thresholds to determine that an AS should be traversed approximately 20 before using the width estimate to guide future traces in order to gain maximum efficiency.

## 8.2 Future Work

There are a number of areas of our work which we believe deserve further research. The results from the global stop set investigation we have conducted, indicate that the assumptions Donnet *et al.* originally made do not hold up in today's Internet. We suggest that the global stop set be extended so that several vantage points must discover a link before it can be used to halt probing to a destination. This would allow better coverage of aliases and per-source/destination load balancers, the most prevalent causes of topology loss when considering the global stop set.

Another improvement which could be made to the economical MDA is to re-enable the economical probing mode mid-path when confidence in the path segment is regained. This would allow a further reduction in the number of probes sent. This would need to be analysed as to whether the potential topology loss is worth the efficiency gain.

To extend the preliminary work we have done using BGP data, we suggest that a BGP aware MDA is developed such that probes can be directed to ASes in which have not been comprehensively probed with previous tracing. The system would

need to peer with a local BGP router and obtain routing information with which it could learn AS widths as we have described. From there it can analyse the AS path to a destination and determine the sections which need to be probed using the MDA. Using the MDA would require fewer traces to target each AS as most of the load balanced links could be discovered with one trace.

To further improve on our work into optimising the ordering of vantage points, we suggest using a large set of vantage points which are situated on consumer machines. This will allow a more accurate experiment to be conducted, and the impacts of optimisations better observed. We also suggest research into a more sophisticated method for tracking the speed of each vantage point. For example, a weighted moving mean may help by favouring newer data points.

# Bibliography


[1] University of Oregon route views project. `http://www.routeviews.org/`.

[2] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: or, power-law degree distributions in regular graphs. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing (STOC)*, pages 694–703, 2005.

[3] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 153–158, 2006.

[4] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the Internet. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 149–160, 2007.

[5] Brice Augustin, Timur Friedman, and Renata Teixeira. Multipath tracing with Paris traceroute. In *Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, pages 1–8, 2007.

[6] Adam Bender, Rob Sherwood, and Neil Spring. Fixing Ally's growing pains with velocity modeling. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 337–342, 2008.

[7] CAIDA. Archipelago measurement infrastructure. `http://www.caida.org/projects/ark/`.

[8] CAIDA. Summary statistics for all archipelago monitors. `http://www.caida.org/projects/ark/statistics/all_monitors.xml`.

[9] Benoit Donnet, Timur Friedman, and Mark Crovella. *Improved Algorithms for Network Topology Discovery*, pages 149–162. Lecture Notes in Computer Science. 2005.

[10] Benoit Donnet, Bradley Huffaker, Timur Friedman, and kc claffy. Implementation and deployment of a distributed network topology discovery algorithm. Technical report, CAIDA, 2006.

[11] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *Proceedings of the 2005 ACM SIGMETRICS International conference on Measurement and modeling of computer systems*, pages 327–338, 2005.

[12] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262, 1999.

[13] Abraham D. Flaxman and Juan Vera. Bias reduction in traceroute sampling - towards a more accurate map of the Internet. In *Proceedings of the 5th International Workshop on Algorithms and models for the Web-Graph (WAW)*, pages 1–15, 2007.

[14] Bradley Huffaker, Daniel Plummer, David Moore, and kc claffy. Topology discovery by active probing. In *Symposium on applications and the Internet (SAINT)*, pages 90–96, 2002.

[15] Young Hyun. Personal communication, 2009.

[16] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, and Matthew Luckie. The CAIDA IPv4 routed /24 topology dataset – 20090601 to 20090630. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml`.

[17] Information Sciences Institute. Transmission control protocol. RFC 793, 1981.

[18] Van Jacobson. traceroute. `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`.

[19] Ken Keys. Internet-scale IP alias resolution techniques. In *ACM SIGCOMM Computer Comunication Review (CCR)*, 2009.

[20] Anukool Lakhina, John W. Byers, Mark Crovella, and Peng Xie. Sampling biases in IP topology measurements. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 332–341, 2003.

[21] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *ACM SIGKDD*, pages 177–187, 2005.

[22] M. Luckie. scamper. `http://www.wand.net.nz/scamper`.

[23] Matthew Luckie, Young Hyun, and Brad Huffaker. Traceroute probe method and forward IP path inference. In *IMC*, pages 311–324, 2008.

[24] Harsha Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: an information plane for distributed services. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 367–380, 2006.

[25] A. McGregor, H-W. Braun, and J. Brown. The NLANR network analysis infrastructure. *IEEE Communications Magazine*, 38(5):122–128, 2000.

[26] Jean-Jacques Pansiot and Dominique Grad. On routes and multicast trees in the Internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 28(1), 1998.

[27] Vern Paxson. End-to-end routing behavior in the Internet. *SIGCOMM Computer Communication Review (CCR)*, 26(4):25–38, 1996.

[28] Pedram Pedarsani, Daniel R. Figueiredo, and Matthias Grossglauser. Densification arising from sampling fixed graphs. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 205–216, 2008.

[29] John Postel. Internet control message protocol. RFC 792, Internet Engineering Task Force, 1981.

[30] Y. Shavitt and U. Weinsberg. Quantifying the importance of vantage points distribution in Internet topology. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, 2009.

[31] Yuval Shavitt and Eran Shir. Dimes: let the Internet measure itself. *SIGCOMM Computer Communication Review (CCR)*, 35(5):71–74, 2005.

[32] N. Spring, M. Dontcheva, M. Rodrig, and D. Wetherall. How to resolve IP aliases. Technical report, University of Washington, 2004.

[33] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

[34] Guoqiang Zhang. Measuring the impacts of sampling bias on Internet as-level topology inference. In *Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing (CMC)*, pages 181–188, 2009.