Implementing Internet topology analysis with emphasis on load balancers and using large numbers of vantage points.

A thesis
submitted in partial fulfillment
of the requirements for the degree
of
Doctor of Philosophy

at
The University of Waikato

by
Stephen J. Eichler

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Department of Computer Science
Hamilton, New Zealand
July 2016

Dedication

I would like to thank the following people and organisations for their assistance with this thesis:

Dr Richard Nelson from the WAND Group - my supervisor and Dr Ryan Ko my second supervisor. Dr Tony McGregor who was my chief supervisor when I began this research. Shane Alcock for critiquing my thesis and Dr Matthew Luckie for critiquing the introductory chapters.

Dr. Robert Durrant provided statistical assistance with understanding the theory behind stopping values.

The Waikato University scholarship office - for funding me through a PhD Research Scholarship.

Shane Alcock, Brendon Jones and Brad Cowie from the WAND group - for assistance with programming and problem solving.

Abstract

This thesis describes a study of load balancing and topology discovery in the Internet. Topology data was collected using variants of Traceroute including Traceroute MDA which provides topology information about load balancers.

Optimised Traceroute MDA was applied to investigating or updating basic statistics describing the behaviour and performance of load balancers and their populations in the Internet. Simulation studies were then performed to optimise the efficiency of distributed topology discovery algorithms including an algorithm for discovering load balancers, as well as one based on classic Traceroute.

Initial optimisation of the load balancer discovery tool was based on refining the parameters used to decide when probing should stop at each hop along the Internet path under analysis (stopping values). Internet data was then collected using the tool to attempt to confirm the refined parameters in practice.

Large repeated analyses were performed to quantify load balancer population parameters and change over time. These results included changing prevalence, turnover, load balancer diamond structure including a new statistic called literal width, fields outside the classic five-tuple that control load balancing and flow ID selection factors that affect the efficiency of discovering load balancers. Data was also collected on the occurrence of black holes in load balancers.

The classic Traceroute based tool Doubletree was simulated to test and improve efficiency, as was our new load balancer topology discovery tool Megatree. Doubletree was simulated with both a simple and advanced simulator and a simple simulator was used for Megatree. These topology discovery tools store information within vantage points and communicate

between vantage points to reduce traffic required for topology discovery. New approaches to using these tools helped improve efficiency when large numbers of vantage points were used.

Contents

List of Figures

List of Tables

List of Acronyms

ANOVA Analysis of Variance

AS Autonomous System

ASes Autonomous Systems

BGP Border Gateway Protocol

BISD Basic Internet Simulator Doubletree

BISM Basic Internet Simulator Megatree

CDF Cumulative Distribution Function

CDNs Content Delivery Networks

DDoS Distributed Denial-of-Service

EGP Exterior Gateway Protocol

ECMPs Equal Cost Multi-Paths

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IETF Internet Engineering Task Force

IP Internet Protocol

IGP Interior Gateway Protocols

IS0 Internet Simulator Zero

ISP Internet Service Providers

IS-IS Intermediate System to Intermediate System

MDA Multipath Detection Algorithm

MPLS Multiprotocol Label Switching

NCC Network Coordination Centre

OSPF Open Shortest Path First

PPS Probes Per Second

RBF Retouched Bloom Filters

RFC Request for Comments

RIP Routing Information Protocol

RIPE Réseaux IP Européens

RIS RIPE NCC Routing Information Service

SEM Standard Error of the Mean

TCP Transmission Control Protocol

TTL Time To Live

UDP User Datagram Protocol

VPs Vantage Points

VP Vantage Point

WAND Waikato Applied Network Dynamics

Chapter 1

Introduction

The Internet has become part of everyday life for many people and is now essential for many aspects of business and education. The Internet is structured as a network of networks based on routers, which have the role of electronically forwarding digital packets of data to another router or an end host. The router must decide which connected device to forward each packet to based on the destination address contained within the packet header. Routing algorithms are used both within and between networks to determine the paths that packets should follow. Because forwarding decisions are made on a hop by hop basis, no route overview is available and the basic packet forwarding structure of the Internet is not contained in a published master plan.

Because topology and path data is needed to help maintain and manage development of the Internet, it is necessary to develop tools to analyse existing network systems and provide path discovery information, as this information is not available. The Traceroute technique, developed by Van Jacobson [33] in 1987, is the basis of many of these tools. Traceroute identifies hops between a source and destination in the Internet to discover a path. The Traceroute algorithm requires the source to send probe packets to all of the nodes along a path to the destination. This process must then be applied to a set of sources and many destinations to be able to discover and map networks and, in particular, their routers and inter-router connections. Router interfaces occur at each end of an interconnection between two routers. Each router interface has an address and the address of incoming router interfaces are the basic units

that Traceroute discovers on most occasions[1].

There are a number of specialised ways in which Traceroute can be used to answer more complex questions about the structure of the Internet. Various systems have been set up to obtain topology information about the Internet, as summarised by Donnet *et al.* [24]. Among these is Atlas [64, 3] which uses a large number of vantage points to probe the Internet. Much of this thesis research is aimed at providing background information that is useful to systems like Atlas. Another Traceroute approach in conjunction with ping can be used to discover discontinuities in the Internet or blockages called black holes [35]. These are points in a path at which traffic that is meant to be forwarded is instead dropped by the router. Traceroute discovers the nodes prior to the black hole but when probe traffic attempts to traverse the black hole, no response is obtained to identify the next hop router interface, nor any hops after that.

1.1 Study of Load Balancers

The usual meaning of the term *load balancer* [10] [19] is a router that splits traffic at a *divergence point* so that it follows several paths until a *convergence point,* where again all traffic on the path passes through one router. Load balancers are used to achieve redundancy and to ensure that a network performs well. On occasions the term has been used to include the router and multipath network up to the convergence point, though this is not the primary meaning used in this thesis. The term *diamond* is used for that instead. We use load balancer to mean just the router that splits traffic, on the way to a convergence point. Load balancers result in multipaths and this feature can cause Traceroute to find nodes in different load balancer segments at different hops.

Equal cost multipaths are related sub paths where the routing protocol measures them as having the same overhead cost. This can mean

[1] Unless TTL limited record route option is used [58] [57] where the outgoing node is often discovered. Record route was originally provided as part of the Internet Protocol, however because of security concerns and the high computational overhead, its use has been limited [38]

that they are equally likely to be traversed, depending on how the network is configured. Load balancers lead to equal cost multipaths and improve Internet reliability, redundancy and performance by continuing to function when a non-terminal router in a diamond fails, and by sharing load across multipaths to improve throughput. Load balancers have thus been an important development for the effectiveness and growth of the Internet and this leads us to believe that study of the Internet structure associated with load balancers is important.

Load balancers have not been widely studied in large scale Internet topology discovery systems that run regular Internet probing cycles. However, discovering load balancers is well supported in the software tools that are available including variants of Traceroute. Though not run routinely, moderate scale data collection runs have been performed to quantify basic statistics about load balancing in the Internet [10] [9] [19] [63] [20]. Our initial studies suggest load balancers tend to be expensive, in terms of traffic, to discover so economies in this respect are of interest. More information about the rate of change and prevalence of load balancer populations would give insight into how often they need to be discovered and whether repeated large scale data collection cycles are worthwhile.

The efficiency of finding the successors of a load balancer (based on how the MDA port flow ID value is chosen for each known load balancer discovery method) is of interest, because of the possibility of using this information to reduce probe traffic when discovering load balancer topology. In this research a comparison of the efficiency of three port selection methods was performed: random, sequential and bit shifting sequences of port values for subsequent probes.

1.1.1 Load Balancer Discovery

The usual method for discovering load balancers is to use an algorithm derived from Traceroute called Multipath Detection Algorithm (MDA) [10]. A Traceroute derived algorithm called Paris Traceroute [8] was the first step towards MDA and Paris has MDA as one of its extra modes as standard. Paris Traceroute ensures that the topology discovered is part of the same continuous path, but does not discover entire load balancers.

In Paris Traceroute a special group of fields in the probe packet are controlled so that the load balancer consistently forwards probes to the same next hop towards the convergence point. Paris-MDA on the other hand goes a step further by varying the values of the special group of fields when the successor nodes of a load balancer are being discovered. A count is kept of the number of successors discovered and more probes are sent as more successors are found to ensure that all successors are likely to be probed. This thesis investigates the amount of probe packet traffic required to discover load balancers and how often load balancer populations change and in turn how often rediscovery of load balancers is needed.

Per-flow load balancing is one of three types of load balancing, whereby the router bases the decision about which load balancer successor to forward to on the classic 5-tuple. A related aspect that is of interest in understanding load balancers is the consistency of a routers response to the classic 5-tuple alone for per-flow load balancing. This research included a study into the possibility that there may be other fields that can influence traffic segregation at a load balancer.

Describing load balancers

Some shape parameters of load balancers have been described in previous research [10], including two measures of width (minimum and maximum) and a measure of length. These are interesting aspects of Internet topology.

Other parameters of interest include prevalence [9] and turnover [20] of load balancers. Prevalence of load balancers is their frequency in the population making up the Internet and turnover is the rate of disappearance and appearance of load balancers. These parameters give us further insight into Internet topology and the process of topology change.

Stopping Values

The research in this thesis included an initial study to provide the numbers of probes that need to be sent to confidently rule out existence of fur-

ther successors, when given numbers of load balancer successors have already been found. This number is called a stopping value.

Stopping values are set using statistical methods. The basic idea is that the more probes you send at a given hop after a load balancer, the more likely you are to have found all of the successors of the load balancer at that hop. An MDA confidence level is the probability that all load balancer successors have been found in an entire trace and is determined by the stopping values used. The stopping values for the higher confidence levels, that we required were not available [63] and needed to be obtained by running a published algorithm. We required 99%, 99.9%, 99.99% and 99.999%. The high confidence was required for the situation where it is necessary to say that one is extremely confident that all load balancer successors have been found. In particular, it was desired to determine if more successors were being missed than the standard 99% confidence MDA testing expected. Furthermore, if there is any question about the violation of underlying assumptions used by MDA, then using high confidence can help look for that. The main violation considered would be if successors are not found with equal probability and so require more than the expected quota of probes to be fully discovered.

1.1.2 Black holes in load balancers

As mentioned above, black holes are discontinuities in the Internet. There is little data available about the frequency or longevity of black holes in load balancer diamonds and indeed whether such discontinuities are a basis for concern. It seems likely that black holes would be resolved by the cost based algorithms that Internet load balancing routers use to partition traffic. Whether this happens in practice is important, because a black hole in a path that carries a share of load balancer diamond traffic, can cause that path to fail *e.g.* TCP streams tend to use the same segment of a load balancer diamond for all traffic to avoid packets arriving out of order. The fact that there has been little interest in black holes in load balancer diamonds is likely because the Internet is designed to adapt to this problem scenario, as it is to black holes occurring at any other location. Previous research shows that black holes regularly occur across

the Internet [35] [36] and it has been seen that in some cases it takes some considerable time for the black hole to be resolved. There have also been cases where the adapting fails to solve the problem [35]. One might therefore expect this to apply inside load balancer diamonds, as it does elsewhere. If a black hole were present in part of a load balancer diamond, it could cause a partial or complete blockage of some flows. It may also be that load balancer diamonds respond better to discontinuities within the parts that share traffic, as alternate paths are already available and in use. This research included a study of black holes in load balancer diamonds that used analysis of data from repeated MDA and basic Paris Traceroute analyses to a small number of destinations from multiple vantage points.

1.1.3 Simulation

As new tools that use more efficient ways to discover the Internet are developed, it becomes necessary to test them and find out if they are viable in practice. Simulation can be used to obtain an understanding of how an algorithm is likely to function. Also, simulation allows performance information to be obtained by running an analysis program on a single computer, or grid of computers, loaded with data collected from the real system. A simulation program is run on the data that imitates the behaviour of the program under test. Furthermore validation must be carried out to ensure that the imitated version of the original program behaviour is sufficiently accurate. The following discussion describes Doubletree, which does not involve measuring load balancing, but leads into a related new program called Megatree, which does.

One of the principle cost factors to be considered when creating a topology map is the number of active probes sent. An additional problem is that if one sub-network of the Internet is the target of highly repetitive probing, this may result in a complaint from a network administrator. It is therefore desirable to have strategies that reduce probe cost and undesirable repetition when making a topology map. One approach to optimising topology discovery is for Vantage Points (VPs) to store information about previously discovered paths and to use this to complete

partial traces without the need for the entire path to be fully rediscovered. An extension of this concept is to use a distributed approach and share information about what router interfaces have been seen between VPs to reduce the cost of probing new paths. One algorithm that does this is called Doubletree [27] and its behaviour has been partially characterised, but cost analysis has not been completed [47]. Doubletree uses extra traffic to communicate between VPs and share information about nodes seen in the destination associated portion of each trace, known as the global stop set.

Doubletree and Megatree are both based on Traceroute. Megatree is a new tool, created as part of the research for this thesis, that stores information about previously seen load balancers and avoids their rediscovery to minimise cost and repetition. Megatree stores information about previous traces, however unlike Doubletree, it stores data about previously seen load balancers rather than single nodes. If there are multiple paths or load balancers between a single source and destination that need to be fully discovered, then the cost is considerably greater than a single path. Megatree can result in savings based on reducing this greater cost associated with load balancers, by avoiding the rediscovery of load balancers when they are encountered again in subsequent traces in an analysis cycle. In this research Doubletree and Megatree are simulated to determine their cost benefit status using simple non event based simulation, which is without a time base. Doubletree is also simulated with a time base using Internet Simulator Zero (IS0) [47], which is based on timed events specifically allowing traces to be run simultaneously. Simultaneous analysis provides a more realistic simulation of Doubletree running on the Internet. The Doubletree research is used as a precursor to the load balancer based Megatree research.

1.2 The Question

Given that load balancing has been widely adopted across the Internet, how can we better understand topology and improve topology discovery with emphasis on load balancing?

- How can we detect load balancers efficiently and with high confidence?
- How can black holes in load balancers be detected, and how many are there?
- How can we make effective use of large numbers of vantage points in our study of topology?
- How can we do this using simple robust approaches?

The Internet is changing and evolving over time, so we expect to see trends in various aspects of topology including load balancer topology. It is important to understand such trends so that the Internet can be maintained efficiently and developed further in the future.

Load balancers add improved performance, reliability and redundancy to the Internet so understanding their distribution and structure in the Internet is important. There can be significant costs involved in measuring the distribution and structure of load balancers so efficient strategies can offer viable solutions. Known true confidence and high confidence approaches are important for gathering a wide range of accurate statistics about load balancers.

Discontinuities in load balancers may be hidden and should heal quickly. If they do occur measurably it is important to know how often and if they cause noticeable problems. This information would help determine whether it is necessary to regularly screen for discontinuities so that steps could be taken to correct the faults.

It is logical that more Internet vantage points will help analyse more of the Internet, but does this make analysis tools inefficient? It is important to know how available analysis tools perform when there are many vantage points so that users can make effective use of such systems.

There is a tendency for simple approaches to offer stability and efficiency along with a few drawbacks. In this research we see simple, robust and possibly natural approaches as desirable as long as the disadvantages are not too great. Efficiency is seen as important and does not

preclude comparison with more advanced approaches which may be at more risk of instability.

1.3 Overview of this Thesis

There are three chapters that lead in to this thesis. They are the introduction (Chapter 1), the background (Chapter 2) and related work (Chapter 3). The introduction presents a brief discussion of the areas of work for which research was carried out to build this thesis. The background introduces more detail to the topics of study and their foundations. Related work discusses the literature references relevant to the studies underpinning this thesis research.

A broad result area focuses on data collected from the Internet that is analysed to give information about load balancing. It has a preliminary phase where there is a component focused on application of an algorithm from the literature initially, which provides stopping values to use for data collection which are not presently available. The research determines if these stopping values match the rates at which successors are found when collecting data from the Internet, which is described in Chapter 4. Data collection on CAIDA and PlanetLab is described in Chapter 5. The focus of this section is to provide an upgraded practical MDA Traceroute test and then to use it to collect data for analysis of real Internet topology data containing a representative sample of load balancers.

After the preliminary phase load balancer research, in-depth analysis of the collected load balancer data provided information and statistics about the sample of load balancers that had been collected from the Internet. This includes load balancer prevalence in Chapter 6, efficient detection of per-destination load balancer diamond divergence points in Chapter 7, turnover in Chapter 8 and diamond structure in Chapter 9. Investigation of non five-tuple fields as possible controllers of load balancing for some routers is included in Chapter 10, along with investigation into finding black holes in load balancers in Chapter 11. Another experiment measuring the efficiency of finding load balancer successors when port flow IDs are selected using several different methods is presented

in Chapter 12.

There is a body of work focussed on simulation that once again uses topology data from the Internet, contained in Chapters 13, 14 and 15. Doubletree simulation includes two types of simulators and Megatree simulation includes the basic simulator only. The simulation is designed to give an indication as to whether the techniques Doubletree and Megatree are likely to be efficient for probing and studying the Internet.

1.4 Contributions of this Thesis Research

1.4.1 Preliminary tasks

Chapter 4 provides joint confidence estimates of stopping values for Traceroute MDA and tests them on the Internet using a single data collection. There are indications that the assumption that there is equal chance of probing each load balancer successor may have been violated in the Internet, meaning that more probes than expected are required to find successors at the desired level of confidence. In practice, this may mean that MDA analysis will become even more expensive than previously thought.

Chapter 5 involves collection of Traceroute MDA data sets on the Internet using several packet types and looking for the three load balancer types. This data will be made available for CAIDA to share with others.

1.4.2 Characterisation of load balancers

Chapter 6 provides updated statistics for load balancer prevalence helping to map change in the Internet. 'Percent load balancers of all interfaces seen' statistic figures are also provided to help improve the quality of information provided by these studies. Confirmation of previous findings in the literature are provided and no large changes are seen.

Chapter 7 introduces and provides analysis of a successful low traffic means to count load balancers. Greatly reduced traffic is possible if it is desired to only identify primary load balancing nodes of analysed diamonds. If there are researchers who desire to identify and count load balancer diamonds without investing in large amounts of traffic this ap-

proach may become viable.

Chapter 8 calculates several statistics to quantify load balancer turnover. Agreement is seen among these indicating a likely rate of turnover. This however is a lower rate of permanent change than is seen for short term change that may not be permanent. This result may affect the time frames used in Internet probing batches as concerns about high rates of short term change may be less of a concern than previously thought.

Chapter 9 provides updated statistics for diamond structure allowing change in the Internet to be seen. Previous findings in the literature were mostly confirmed. Literal width (a new statistic) is introduced and analysis demonstrates a small population of highly complex load balancer diamonds. Literal width provides a means to locate and study complex parts of the Internet.

1.4.3 Black hole detection

Chapter 11 provides analysis of the Internet for load balancers containing black holes. A small population of these are found with varying lifespans. There may be enough discontinuities in load balancers found to warrant searching for these across the Internet in the future.

1.4.4 Simulation

Chapter 14 provides simple analysis of Doubletree using non destination repeat data, and Traceroute MDA extracted data that benefits from both local and global stop sets. Sources windows give a benefit and there is a benefit from both the local and global stop sets when source and destination repetition is present in the many to few situation and cost analysis is carried out.

Chapter 15 introduces Megatree (a new algorithm) and provides cost analysis. Megatree shows useful savings when cost analysis is carried out in both directions of the data. Sources windows in the many to few direction are beneficial under cost analysis. If Megatree is sufficiently efficient, it may become viable for regular Internet analysis.

Sources windows is a new approach that helps distributed Internet

probers to efficiently communicate control information in a stepwise group probing strategy. It may be included with other improvements to Traceroute, as may Megatree.

Chapter 2

Background

2.1 Introduction

This chapter introduces Internet load balancing and the family of active probing tools that are based on Traceroute. Background is then given on the topics of load balancer prevalence, turnover and the diamond network structure that most load balancers create. Internet discontinuities (black holes) are then described, including the way black holes and load balancers interact.

2.2 The Internet

The Internet is a worldwide computer network of networks, consisting of routers and end-hosts. The Internet primarily uses a connection based service, the Transmission Control Protocol (TCP) with the Internet Protocol (IP), to communicate between machines. Connectionless service is provided by the User Datagram Protocol (UDP) and the Internet Control Message Protocol (ICMP) is used to send error messages. Fig. 2.1 shows the structure of network and transport layer headers [8] related to these packet types.

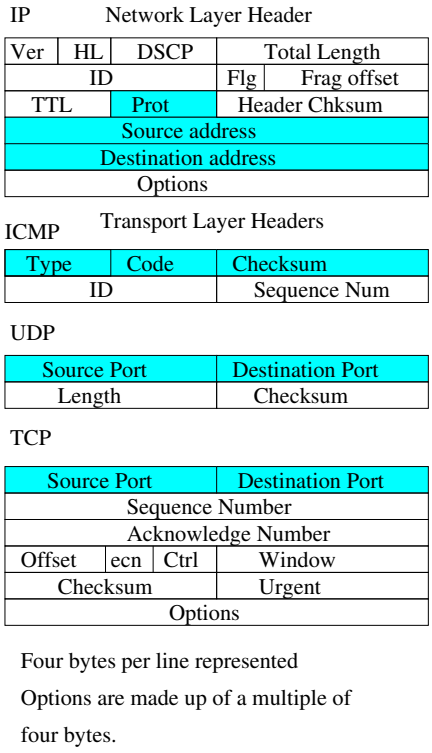

Figure 2.1: Diagram of network and transport layer headers.

2.2.1 Routing

A router is an Internet device that forwards packets of data between interconnected Internet computer subnets. Routers have the role of sending packet data traffic onwards towards their many and varied destinations.

The style of routing adopted by the Internet is known as hop by hop routing, because each router only needs enough information to determine the next hop to forward a packet to, based on the destination IP address of the packet. Exceptions to this are load balancers and Multiprotocol Label Switching (MPLS) tunnels [28]. Because there is no centralised map or database of Internet routing information, a hop by hop router must itself create the router forwarding table that it uses to make decisions about packet forwarding, which is a complex task.

The router forwarding table may include the following fields amongst others: network ID and next hop. When a packet arrives at a router,

the router compares the destination IP address to the forwarding table entries to determine the next hop to which the router will forward the packet. For example a network ID might be 111.34.3.0/24. This implies a network mask of the 24 most significant bits. The remaining bits may vary to identify IP addresses within the selected subnet. Typically all of the addresses in a sub-network defined in this manner will usually have traffic forwarded to the same next hop, which is specified in the same table entry. If, however, another related network ID is specified that has a longer prefix *e.g.* 111.34.3.32/28, and the destination of the packet falls into this smaller set of addresses, then this entry takes priority. Thus the packet is forwarded to the next hop listed for the smaller sub-network. This priority sequence is called longest prefix matching.

Some routers exist entirely within network domains (Autonomous Systems (ASes)). Others routers may also be within one domain and be connected to a router inside another network domain, serving as interconnections between domains. Such routers are known as border routers.

2.2.2 Autonomous Systems

An AS is often described as a network of routers controlled by a single company or technical administration. The purpose of an AS has been defined in RFC[1] 1930 [31]. The routers within an AS share routing information with each other using one or more Interior Gateway Protocols (IGP), as described in Section 2.2.3. An Exterior Gateway Protocol (EGP) is used to communicate between ASes.

The routers within an AS must generate related router forwarding tables using one IGP or more, and similarly border routers must also do the same using an EGP and one IGP or more.

2.2.3 Dynamic Routing

Unlike static routing where routers do not communicate network information, but use fixed routing information, which is obtained from net-

[1] A Request for Comments (RFC) may exist on one of many topics relating the configuration of the Internet and is a technical publication made by the Internet Engineering Task Force (IETF) and Internet Society that set standards for the Internet.

work administrators dynamic routing occurs when routers communicate with information about the networks that they are linked with. The routers use this information to create a router forwarding table. A mix of static and dynamic entries in a router forwarding table is possible.

Some common dynamic routing protocol types are distance-vector and link-state [61]. Distance-vector requires the exchange of routing information with neighbours. Routers build a table of prefix, next hop and cost and thus they do not know hops beyond the next hop. In link-state each router builds a full map of the network from combined information about each router's environment [50]. In both cases a cost type metric is used to determine the best path *e.g.* lowest cost. An IGP is one of several routing protocols used to share routing information within all or part of an AS. The routers in an AS supporting one IGP in a zone or subnetwork, are often called a routing domain. The link-state protocol Open Shortest Path First (OSPF) is the most commonly used IGP and is recommended for all ASes [50]. The link-state protocol Intermediate System to Intermediate System (IS-IS) is used by a handful of larger Internet Service Providers (ISP) and was not originally written for IP, thus to run IS-IS on IP requires an IP module to be run on the routers using it. Routing Information Protocol (RIP) is a distance-vector protocol and has largely been superseded by OSPF.

The EGP, Border Gateway Protocol (BGP), internally communicates reachability and routing information between peers and is a path-vector protocol. Path-vector means that each forwarding table entry includes an AS-path, whereas in distance-vector a distance is included in each table entry. An AS-path is a sequence of ASes to follow to the destination. Path-vector protocols are derived from distance-vector protocols, however they differ in two important ways. Though distance-vector implementations broadcast regular updates of information including routing table prefixes and cost metrics similar to OSPF, the path-vector protocol BGP only provides information about changes to neighbouring BGP routers. Secondly BGP has a mechanism to prevent loops[2] by checking the AS-path for repeated occurrences of the same AS, which has proven

[2]RIP does not specifically prevent loops although it can use some older mitigations.

to be very effective.

Under OSPF, cost factors are used in the creation of a routing table. These can include link transit time, link reliability, availability and traffic capacity. In OSPF the cost scheme is designed by the routing domain's administrator, often using a combination of these cost factors. Alternatively RIP chooses paths that traverse a minimum number of routers *i.e.* the cost is a distance measure.

The cost factors for multiple outgoing paths that lead to the same destination can be the same and traffic with the same destination address can be shared between these to create a load balancer. It is now common for load balancers to be implemented specifically by network administrators, to benefit from sharing traffic between alternate paths within or in some cases between ASes.

2.2.4 Load Balancing

Load balancers are routers that share traffic between different paths, between source and destination nodes [9]. A load balancing diamond occurs in a section of path and involves a divergence point (a load balancer) and convergence point through which all traffic from the connection again passes. A load balancing diamond is shown in Fig. 2.2. If a load balancer is present, there are multiple paths which divide the traffic load. The term *virtual path* can be used to refer to the set of multiple paths between a source and destination that a packet may follow [19]. In the Internet it has become more common for there to be multiple paths that packets travelling between the same source and destination follow. There is also a protocol called *Multipath TCP* [12] that uses multiple link level channels where available. Variation in link level paths is outside the scope of this study.

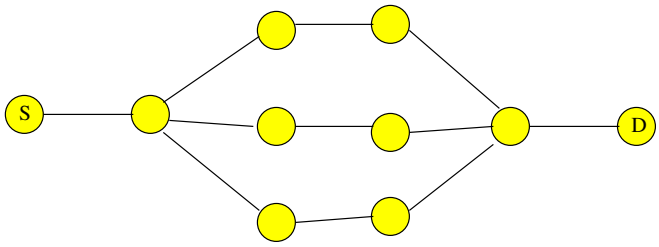

Figure 2.2: Diagram of a load balancing diamond between a source and destination.

Load balancers increase Internet reliability, redundancy and performance by sharing traffic between multiple paths, after the traffic is divided at a load balancer. The Internet is designed to drop one of these alternate sub-paths if it fails, thus making use of redundancy to improve reliability. The use of several sub-paths to share the load also helps improve performance. Load balanced paths are Equal Cost Multi-Paths (ECMPs) whereby each sub-path within one virtual path has equal cost, a metric that is coarsely determined and may not reflect differences in latency. However, the cost of travel through the same load balancer is usually equal for each sub-path connecting divergence and convergence points. ECMP routing involving load balancing is supported by the link-state IGP routing protocols, OSPF and IS-IS. The distance-vector routing protocol RIP supports ECMP, also. There is also a multipath option that supports load balancing for the path-vector protocol BGP.

Load balancing can occur across AS boundaries. Unlike Distance Vector protocols, BGP does not involve minimisation of hop count distance for each possible path, but rather minimises AS path length within other administrative policy restrictions. Multipath BGP creates multiple entries for path attribute ‘next hop’ in the router forwarding table. This is also classified as use of ECMP and on Juniper systems[6], for example, the multipath selection is also based on the IGP cost. Prevalence of cases of inter AS load balancing has been investigated [10]. These researchers reported that BGP multipath was activated in a small number of cases in their data set *i.e.* 14-20% of load balancers do not occur entirely within a single AS (domain).

There are three load balancer types: per-flow, per-packet and per-

destination, which use different methods to partition traffic between successors of the divergence point. The load balancing node in a per-packet load balancer randomly distributes packets to its successor nodes. This type of load balancer is relatively rare [10]. In per-destination load balancing, packets to the same destination will be sent to the same successor node. Per-flow load balancing is based on the classic 5-tuple. The classic 5-tuple consists of the IP fields source address, destination address and protocol as seen in diagram 2.1, and the other two fields come from the first four bytes of the transport layer header. For UDP and TCP this is the source port and destination port, and for ICMP this is code and checksum. In per-flow load balancing the forwarding of packets to successor nodes is the same if the fields of the classic 5-tuple are the same. However, part of the research of this thesis examines whether the classic 5-tuple applies in all cases, or for all routers, in terms of controlling which successor is forwarded to and which other fields can influence this decision.

The specific fields that act as keys for per-flow and per-destination load balancers may vary with the implementation of router software. For per-flow, these may sometimes be a subset of the classic 5-tuple. This can affect whether load balancers are determined to be both per-flow and per-destination or just one of these types. Though per-destination load balancers are also defined as per-flow load balancers, common tests for per-flow load balancers often do not coincide with this definition. This type of distinction is of interest among populations of routers because it provides more information about load balancer behaviour and how load balancers contribute to the improved performance of the Internet. The load balancing behaviour of Internet routers can be learned from their specifications as well as by making Internet measurements. Two of the more popular router brands are Cisco and Juniper. Cisco instructions make no reference to a per-flow mode for their routers [17]. They make reference to the hashing procedure that they use for per-destination: “For per-destination load balancing a hash is computed out of the source and destination IP address. This hash points to exactly one of the adjacency entries in the adjacency table, providing that the same

path is used for all packets with this source/destination address pair". Per-destination load balancing does not include the source address, but for a given source, per-destination load balancing will be seen. The Juniper documentation simply states the following: "You can specify what information the router uses for per-flow load balancing...". Juniper offer two per-flow modes [29], one where 3 fields of the five tuple are used: protocol, source address and destination address (layer 3 only). In the other mode all five fields are included in the flow ID *i.e.* source port and destination port are included for TCP and UDP, and for ICMP: code and checksum are included. This is the layer 3 and layer 4 option. The Internet is therefore diverse in term of load balancing methods and the load balancer statistics we choose to quantify may give some insight into the detail of this situation. Removing existing per-flow cases from per-destination counts to give a new statistic is a contribution of this thesis research.

The network portions connected to the successors of a load balancer do not necessarily have equal latency characteristics and if per-packet load balancing is used this may result in packet reordering [51] [34] [13]. This is particularly bad for TCP and may interfere with congestion control, which relies on correctly ordered packets to be efficient. This is because out of order packets can result in duplicate ACK TCP packets, which are a trigger for congestion control measures. Instead per-flow and per-destination load balancers are preferred because all packets belonging to the same TCP connection will traverse the same load balanced path segment, thus avoiding reordering. For this reason per-flow and per-destination load balancers are preferred by TCP end-users and in turn by network administrators.

2.3 Traceroute

Traceroute [33] [44] is a popular active probing Internet topology discovery tool. Traceroute, in its original form, sends UDP packets with the Time To Live (TTL) limited to the first hop at the beginning of the path, and then an incrementing TTL series to the end of the path. TTL

is decremented at each router the packet passes and so the hop count (of the probe) is determined by initial TTL. When the TTL reaches zero an ICMP 'Time-Exceeded' packet is returned to the sender identifying the node where the TTL expired, thus building up a picture of network topology listing inbound IP interface addresses.

Traceroute terminology:

- An 'end-host' is a computer that is connected to the edge of the Internet. End users interact with end-hosts.
- A Vantage Point (VP) is an end-host attached to the Internet, at a particular location, that runs a program that collects data about the Internet, which in our case is topology data.
- 'Active probing' is a technique used to infer Internet topology based on router adjacencies. The active probing program running on a vantage point sends probe traffic, or packets, that it creates itself, that seeks responses from devices that make up the network.

Traceroute suffers from two problems when used to analyse a network containing load balancers [1]: false links and missing nodes and links. This is because when classic Traceroute encounters a load balancer, no steps are taken to ensure the path taken through the load balancer is consistent. The values of the classic 5-tuple fields are not strictly controlled and, in particular, the port values vary. Progression to a particular successor after load balancer divergence is thus not controlled and may vary for per-flow load balancers. This means that different paths may be followed at different hop counts resulting in false links and some paths may not be followed at all resulting in missing nodes and links, as shown by Fig. 2.3.

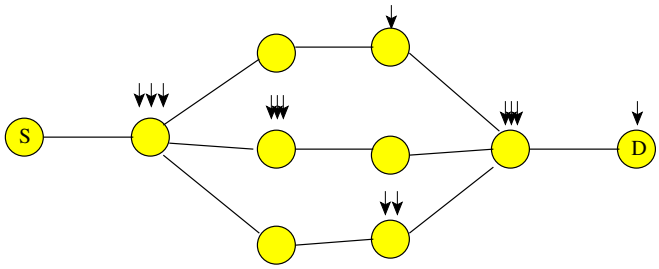

Figure 2.3: Diagram of possible random Traceroute behaviour at a load balancer in a path. The arrows show nodes discovered. At hop 2, the same node was found three times and at hop three the same node was found twice and the other once. These found nodes do not make up a segment that exists as a continuous path.

Traceroute only discovers topology in the forward direction from source vantage point to destination and not in the reverse direction. An example of the importance of this is in the detection of black holes, where it is helpful to know if a discontinuity is present in both the forward and reverse direction or not. It has been suggested that a protocol is required to recruit the cooperation of the destination to discover the reverse path and this has been attempted with spoofed probe packets [35]. This spoofing technique sets the source IP address of a probe packet to that of another vantage point, so that replies go there instead. However systems like Atlas [3] also offer promise in discovering reverse paths because a path could be chosen that has an Atlas vantage point at each end, thus requiring only forward discovery from each end. This is likely to be possible as Atlas is accumulating large numbers of vantage points, which would still allow a wide selection of paths and destinations to chose from.

Advances have been made in Traceroute technology to provide new derivatives that solve some of Traceroute's problems, including traffic wastage in repeating Internet discovery at certain locations, the discovery of non-existent paths and failure to fully discover load balancers.

There are a number of present and past systems that provide facilities for probing the Internet with Traceroute and providing structural information about the Internet, as summarised by Donnet *et al.* [24]. In some cases the downloading of data from the experiments of others who have successfully completed Traceroute probing, or other analysis, is pro-

vided. Two systems that are currently operational and run from relatively few vantage points are CAIDA Ark and PlanetLab. These are large scale systems that run regular Internet probing cycles. The CAIDA Ark [2] carries out regular Traceroute analysis to a large part of the Internet from several tens of vantage points. Some researchers are also permitted to collect data using a modest number of CAIDA vantage points and they may also make their data available to others. On Planetlab [60, 52] external researchers can use slices on shared Planetlab machines to run their own data collection. It is possible to use several hundred PlanetLab vantage points for data collection.

Some other more highly distributed data collection systems include DIMES [56], NETI@home [59], RIPE:Atlas [64, 3] and the now discontinued RIPE:Test traffic measurement service [4] and NLANR [48]. DIMES is a system that invites end users of the Internet to download the DIMES agent and allow a small amount of traffic quota to be consumed at that location. This can be used to discover Internet topology using Traceroute and Ping, which is a tool to send a packet to a destination requesting a reply, in order to determine if the destination was reached by the request packet. NETI@home collects network performance statistics from end-systems [59]. NETI@home does not use Traceroute or send any of its own traffic when collecting data. Instead, it monitors traffic to and from the end-host, obeying necessary privacy restrictions. RIPE NCC (Réseaux IP Européens (RIPE), Network Coordination Centre (NCC)) is building Atlas, an Internet probing system that uses a large number of vantage points installed by volunteer end-users. Traceroute and Ping are supported by Atlas probing devices, along with further common tests that quantify the performance of the Internet. RIPE:Test traffic measurement data is available from the RIPE:NCC [49]. The NLANR Network Analysis Infrastructure consisted of a passive monitoring project, an active monitoring project, and collection of network management and control data. Most of the data from the NLANR project is available from the University of Waikato Wand traffic archive and also from RIPE.

2.3.1 Doubletree

Doubletree [22] addresses the problem of wasted or repeated traffic associated with multiple traces from or to the same source or destination, as shown in Fig. 2.4. Wastage occurs in the diagram where there are two dotted lines together, indicating rediscovery of topology that is already known. Some is at the beginning of the traces and some is at the end. Doubletree remembers nodes that have been seen before in association with an end point and stops discovery when these are seen, relying on information already gathered to create the full topology picture. Information about previous traces that Doubletree stores in a source vantage point is called the local stop set and information shared between vantage points about nodes leading to a given destination is called the global stop set. Like Traceroute, Doubletree ignores load balancing leaving scope for it to be advanced in similar ways to which Traceroute has *i.e.* MDA Section 2.3.3, see Megatree Chapter 15.

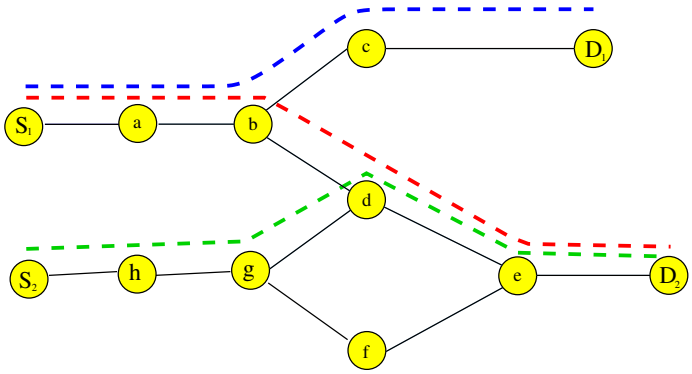

Figure 2.4: Diagram of Traceroute wastage.

Fig. 2.5 shows Doubletree starting in the middle of a trace (node d), stepping back and then stopping (node b) when a node in the local stop set is discovered. Doubletree then steps forward and stops when a previously seen node destination pair in the global stop set is encountered (node e). The blue and green traces completed before the red trace, in which Doubletree stopped and saved probe traffic. This saving must be balanced against the cost of sharing data between vantage points. Doubletree does not ensure that the paths discovered do not suffer from the

non-existent path problem shown in Fig. 2.3. In particular this is a consequence of combining information from different traces, even if the same path through load balancers can be followed within a trace.

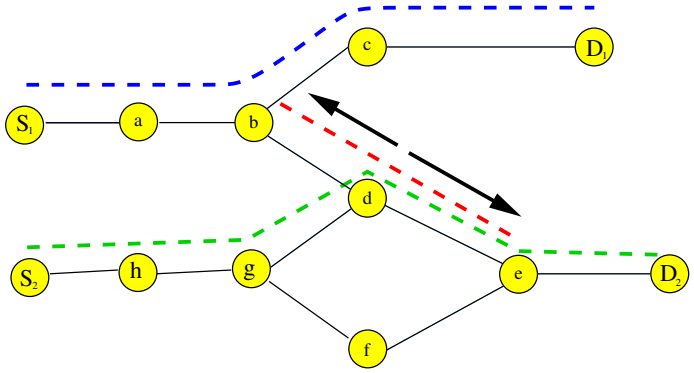

Figure 2.5: Diagram of Doubletree avoiding wastage.

2.3.2 Paris Traceroute

Paris Traceroute avoids topology map aberrations resulting from the presence of load balancers [8] [7], in particular false links. It achieves this by ensuring that packets in a trace are sent along the same path resulting in the path discovery shown in Fig. 2.6. To do this the classic 5-tuple is kept constant, whilst another field outside this tuple is varied to identify the sending packet. This field needs to be in the part of the sent packet returned with the ICMP 'Time-Exceeded' packet. This does not mean that the load balancer is identified or fully mapped, but it does mean that one of several true paths is usually found if a load balancer is present. The number of probes per hop can vary and is often set to one, if there is a reply from the hop router or end-host, for economy. The same is often true for some implementations of classic Traceroute as well.

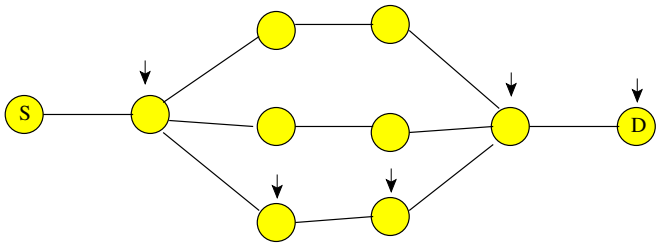

Figure 2.6: Diagram of Paris Traceroute behaviour at a load balancer in a path.

2.3.3 Multipath Detection Algorithm

A special mode of Paris Traceroute, MDA is a version of Traceroute that discovers load balancers [10]. To help find successors the flow ID is varied. The varied flow ID is normally one of the classic five-tuple fields for per-flow. The other fields of the five-tuple are kept constant, to allow diamond segments to be discovered. As more nodes are discovered at a given hop count, more probes are sent to be sure that all the nodes at that hop count are found, as shown in Fig. 2.7. When hop 2 is probed multiple probes can reach each successor of the load balancer and the sum of probes across successors equals the stopping value used, which is selected by knowing the number of successors already discovered. When probing hop 3 sets of flow IDs are used that probe the particular segment of the load balancer being discovered, each having its own stopping value. The predetermined values that determine when to halt MDA for the current node, given the number of successor interfaces found are called stopping values. MDA can be used for detecting all types of load balancers. MDA prevents the occurrence of missing nodes and links in addition to avoiding false links as in simple Paris Traceroute.

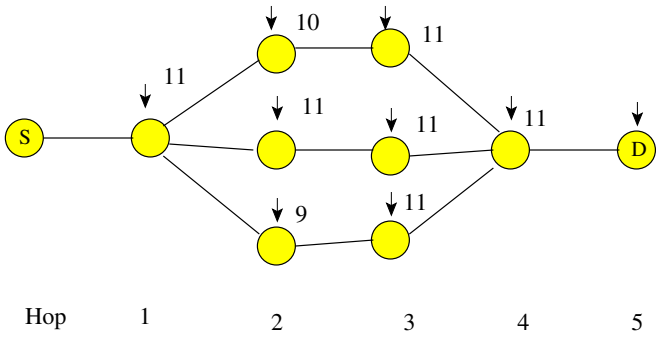

Figure 2.7: Diagram of MDA Paris Traceroute behaviour at a load balancer in a path. Multiple probes are sent to the successors of the load balancer. The counts show the number of probes that expire (TTL zero) at the given node.

As MDA examines a load balancer, flow IDs are collected that traverse particular arms of the load balancer and thus nested load balancers may be discovered using this approach. In order to detect per-packet load balancers the same flow ID is probed repeatedly when it is necessary to

test for this mode.

Bringing forward[3] is the process of finding probes and flow IDs that traverse a segment linked to a load balancer successor or that traverse a load balancer linked to a successor of a previous load balancer in a virtual path. Extra probes used by MDA beyond those prescribed by stopping values are a repeated flow ID to detect per-packet load balancing, and bringing forward. The bringing forward flow IDs are then used by MDA to probe a remaining segment or a nested load balancer.

One of the limitations of Traceroute and its descendants is that discovered nodes are identified as interfaces rather than router IDs [43]. This is because it is desirable to identify load balancing routers uniquely and to identify convergence points immediately once they are encountered. Fig. 2.8 shows these two consequences of this limitation. First we note that interfaces A and B are on the load balancing router and when discovered as load balancers give a different IP address or ID to the load balancer. This can be resolved using a technique known as alias resolution, however this involves extra traffic on the topology discovery analysis system. Because we wanted to use a simple approach, address the problem of minimising traffic usage and also to make the data collection cycles manageable we did not pursue this option, as a considerable amount of useful information is obtainable without it. The second is that if there is a single node (interface G) after the convergence point (interfaces C,E and F) then interface G will be reported as the apparent convergence point, one node later. This is still a workable situation, as in our initial studies we discovered many convergence points in our data, meaning that the occurrence of convergence points as new load balancers is not a high percentage.

[3] As described and implemented in Scamper code [42]

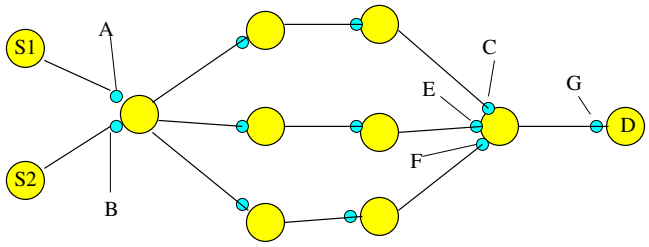

Figure 2.8: Diagram of MDA Paris Traceroute behaviour at a load balancer in a path, showing interfaces as small blue circles.

Most systems that map internet topology on a large scale do not map load balancers *e.g.* CAIDA Traceroutes. There is a significant overhead associated with carrying out MDA analysis compared to standard Traceroute. MDA analysis analysed in 2009 used a probe packet mean of 348 and maximum of 1334 [63] per path traced.

Once load balancers are able to be discovered using analysis tools, it is possible to quantify statistics that describe the use of load balancers in the Internet, such as load balancer prevalence and turnover.

Load balancer prevalence analysis has previously been performed to provide information about the percentage of traces analysed that contain a load balancer of the given type.

Load balancer turnover is a measure of change of the population load balancers on the Internet. A problem with repeated Traceroute analysis of these populations is that of route changes, as seen in [20]. In that paper, route age and route prevalence were graphed against residual lifetime of the routes. Sixty percent of routes (virtual paths) were shown to have a duration under one hour and 95 percent were under 100 hours. This statistic does not distinguish between changes within or between load balancers, so it would be useful to make this distinction. In particular, we chose to focus in part on the populations of load balancing nodes where traffic is split, as distinct from entire diamonds.

2.3.4 Diamond Analysis

Besides counting load balancers it is possible to measure their shape using diamond analysis. A diamond is a multipath structure on a network path [10] that is made up of one or more load balancers. It has a diver-

gence point and a convergence point, where in both cases all traffic on the path passes through one node. If alias resolution is not utilised and the convergence node is not a load balancer, then graphs of the diamond show convergence at the node immediately after the convergence node. Diagrams 2.2 and 2.9 show diamonds. Commonly used diamond metrics are minimum width, maximum width, symmetry and length. Minimum width is the number of link disjoint paths, which can affect the throughput of the load balancer. Length can bear some relationship to the actual distance served by a load balancing diamond, but is actually the maximum number of hops in any one path within the diamond. Symmetry may provide some indication about latency of different segments within a load balancer and hence the weakest or slowest link in that load balancer. A diamond is considered to be asymmetric when its convergence point can be reached with different hop counts [10].

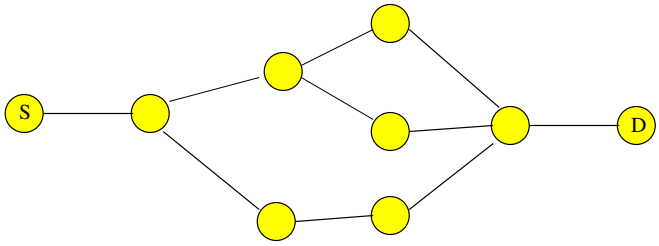

Figure 2.9: Diagram of a diamond in a path.

Following on from investigating load balancer shape, it is possible to quantify defects such as black holes that might occur within load balancers. It is known that many temporary local defects occur in the Internet from time to time, but the impact of defects inside load balancer diamonds has had little attention.

2.3.5 Black holes

Black holes are local discontinuities in the Internet. In particular, black holes involve traffic being discarded without the source being notified. When topology discovery is performed black holes only become apparent due to missing traffic. Many black hole scenarios involve a policy compliant physical path existing, but another path being used that fails to

deliver traffic [36]. In practise, existence of a physical path and a valid BGP path does not mean that traffic will always be delivered.

Because black holes make certain destinations unavailable from certain sources on the Internet for extended periods of time, they are a significant problem. This was the key motivation for the creation of black hole discovery projects such as Hubble [35], Lifeguard [36] and PlanetSeer [66].

Hubble [35] was a black hole detection system that ran on Planetlab from September 2007 for approximately five years, using repeated Traceroute analysis to detect black holes in the Internet. It was later replaced by Lifeguard [36]. Lifeguard also finds black holes using active probing of the Internet and then it offers a solution. This is to "poison" the sub-network or Autonomous System (AS) where the black hole is located, taking advantage of the loop prevention mechanisms in BGP. This stops traffic from using that sub-network or AS.

Hubble used failure of repeated pings every two minutes for 6 cycles to trigger targeted Traceroute analysis. This is consistent with the observation that short term problems such as those associated with BGP convergence are expected. For this reason, a black hole is defined as a reachability problem that has persisted for a minimum amount of time, usually some minutes. Hubble found that 80% of reachability events lasted more than an hour and 20% more than 10 hours.

Using OSPF as IGP, black holes in load balancers would be expected to exist for a matter of some seconds before the routing tables converged and adapted to the new situation *e.g.* a link has gone down. The RouterDeadInterval is typically set to 40 seconds after which if a Hello packet is not received from a neighbour, the connection to that neighbour is no longer advertised.

Black holes inside load balancers

Previous systems that looked for black holes in the Internet [35] did not look inside load balancers. In particular, the ping test used by Hubble would not fail consistently if a black hole occurred between diamond divergence and convergence points. This is because different ping probe

packets are likely to randomly pass through alternate load balancer segments and only sometimes encounter a black hole which exists in a particular segment. If a black hole in a load balancer segment persisted, it would be expected to randomly block some ICMP traffic transiting that load balancer. For TCP traffic where the classic five-tuple is fixed and for UDP where this is sometimes the case, users could occasionally experience outages because of black holes in per-flow load balancers. For per-destination load balancers less fields are involved in the load balancer forwarding decisions but the outcome is similar, as certain users accessing certain destinations via a given problem router could experience outages.

2.4 Other terminology used in this thesis

Unique load balancer A unique load balancer is a load balancing interface with a unique address. This means that when load balancing interfaces are counted the same interface is counted only once. However the same router could be counted more than once as it may be observed using more than one entry interface.

Distinct load balancer Distinct load balancers result from counting unique load balancers more than once when the successor sets vary *i.e.* there are no successor nodes in common. This repeats counting for cases where there is more than one diamond attached to the same load balancing node. Because there is no alias resolution the same repeats that happen for unique load balancer counting will still occur.

Non matching successor sets Two successor sets are considered not to match if they have no interfaces in common. Matching successor sets is one way of compensating for the lack of alias resolution, however not in the case of per-destination data collected with limited sets of destination IDs (see Chapter 7), as the found successor subsets are likely to be quite variable for the same load balancer.

Chapter 3

Related Work

3.1 Introduction

This chapter is focussed on reviewing the previously published research that underpins the work published in this thesis. A key focus of the research is the use of Traceroute MDA to study load balancers, which are often ignored by classic Traceroute based topology analysis systems. The reliability of this technique is reviewed, along with information on router load balancing behaviour. Also of interest is the study of algorithms for mapping the Internet that are focussed on reducing probing traffic. An important approach in this area is simulation. One advantage of simulation is that it is straightforward to test out new algorithms, compared with fully implementing the algorithm and using it to directly analyse the Internet.

Some chapters have their own related work section, and so this chapter contains reviews relating to several chapters.

3.2 Analyses with multiple packet types

Part of the research of this thesis is to collect topology data using Traceroute MDA, using UDP, TCP and ICMP probes. It has been reported [44] that Traceroute results with different probe packet types vary in terms of the numbers of nodes discovered. ICMP based probing tends to find more destinations *i.e.* the following percentages are cases where the reply from the destination is of the expected type: UDP-Paris 96%, TCP

port 80 94.6% and ICMP-Paris 98.8%. For load balancer detection analysis, UDP methods find a greater number of links, 91% of all links found across the three methods, TCP 89% and ICMP 77%. It was suggested that part of the reason was likely to be that ICMP is not load balanced as frequently as UDP or TCP. Section 6.1.1 in Chapter 6 refers to more publications on this topic.

3.3 MDA

In the research of this thesis Traceroute MDA is used extensively, thus it is of interest to consider pitfalls of this technique that other researchers have pointed out. In a paper by Marchetta *et al.* [45] it has been asserted that Paris MDA Traceroute is not entirely reliable. The argument is that in some cases instead of finding a load balancer in the forward direction, one exists in the reverse direction that exposes Traceroute to multiple IP addresses from a non load balancing node in the forward direction. RFC1812 states that the outgoing interface address of the ICMP reply is reported rather than the one that triggered the error. Some routers comply with this and some do not. The paper suggests that 14% of paths found to contain a load balancer do not. This possibility makes the interpretation of Traceroute results less certain and more complex. It also suggests that alias resolution could help with this problem in future work.

3.4 Doubletree

The scope of the research of this thesis includes direct probing analysis of the Internet using Traceroute and some derivatives. In particular, this includes evaluation of algorithms to bring about more network friendly and efficient topology discovery. Another derivative that we did not use directly, but rather performed simulation studies of, is Doubletree. In 2006 Donnet *et al.* proposed and evaluated Doubletree [22]. The authors compared it to Skitter and saw that it sent a third as many probes and thus showed the potential to be more efficient, if any extra costs were not excessive. On the other hand Skitter discovered slightly more nodes and links. Doubletree is also intended to be network friendly [26] by avoid-

ing unnecessary repetition of probing especially to end-hosts and therefore avoiding the appearance of Distributed Denial-of-Service (DDoS) attacks. To this end the authors also proposed limiting the number of vantage points per destination and dividing the vantage points into smaller groups that were the only ones to focus on a particular destination set.

The authors also investigated reducing the data size of the global stop set, which is the data that needs to be transmitted between vantage points when Doubletree operates. This included introducing Bloom filters and in particular Retouched Bloom Filters (RBF) [23]. Bloom filters are a compacted way to communicate global stop set information, reducing the data size by 10 fold. Bloom filters use a number of hash functions to set bits in a vector that represents the members of a set *i.e.* the global stop set consisting of (interface, destination) pairs. Bloom filters have the side effect that they result in false positives. The RBFs decrease false positives compared to regular Bloom filters and instead increase false negatives. False positives mean that Doubletree will stop when it should continue discovering a path, as the information implies that the (interface, destination) pair has been seen before. This is a serious flaw as it is important to discover all new topology on the paths chosen for analysis, and even at a reduced rate is a drawback to using the technique [23]. False negatives cause Doubletree to rediscover some of the same topology again. False negatives result in a small amount of wasted traffic, and as such are not a serious concern compared with false positives.

Another technique that can be used to reduce the global stop set size is by using address prefixes [25], such that the global stop set contains (interface, destination address prefix) pairs. However in this case useful reductions require large sized end network groupings or small sized prefixes *i.e.* a small number of bits specifying the subnet.

The obvious difficulty of using Doubletree in an environment where there are load balancers has been described previously [28]. If Doubletree traffic is able to go through the same load balancer repeatedly, different nodes will be recorded depending on which part of the load balancer traffic passes through, due to not fully observing the principle of destination based forwarding. It is possible that probing may not stop at

a load balancer that has been probed previously because a different path is followed. However, the divergence point and possibly the interface node after convergence will be seen if the load balancer is successfully probed. It thus seems likely that Doubletree will stop for load balancers, but will not see all of the internal topology of the load balancer. The authors state that with Doubletree 'forking paths are not explored and remain invisible to the system'.

In the research of this thesis Doubletree and a new derivative, Megatree, are assessed through simulation analysis.

3.5 Ingress point spreading

In 2014 a topology discovery system that combines 'Ingress Point Spreading' with 'Recursive Subnet Inference' was described [11]. Ingress point spreading is strategically choosing vantage points to find multiple entry points into an AS of interest, thus creating the opportunity to discover more of the internal topology of the AS. Recursive subnet inference performs a binary search over the address space of an Internet prefix, ignoring branches that fail to provide new topology data. The result is a 50% reduction in probe traffic and a small increase in discovered Internet vertices and edges compared with the CAIDA Ark system.

3.6 Building better maps

Claffy *et al.* reported on the state of the CAIDA Ark in 2009 [18]. The authors state that the Internet, a 'trillion dollar ecosystem', is a challenge to analyse to obtain its fundamental characteristics. The Ark performs regular Traceroute analyses using teams of monitor sites, in addition to providing access to the monitors for approved research projects by other researchers. Analysis covers a wide cross section of ASes. New monitors are continually being added to the network to improve the variety of vantage points. Tools are provided for alias resolution analysis along with an initial data set associated with related MIDAR research [37]. Ark data associated with AS-level Internet topology is also available. Some Ark services make use of tuple space, the use of simple ordered tuples to

invoke services such as Ping and Traceroute to particular destinations.

In 2012 Huffaker *et al.* made some comparisons between the CAIDA Ark with other systems [32]. When tested on an ISP for which networking ground truth was available, Ark performed well failing to find only 27% of routers, Dimes missed 37% and iPlane missed 67%.

In 2008 Gill *et al.* reported analysis of the creation of wide-area networks by large companies [30] such as Google, Microsoft and Yahoo. The authors say that reasons for doing so include business reasons, technical challenges and opportunity. In their research the authors perform Traceroute probing into these networks from a number of vantage points, and calculate number of Tier 1 hops per path, numbers of paths with no Tier 1 hops and connectedness. These networks on a large scale could affect the make up of the Internet.

3.7 Detecting outages

In addition to the information provided in Section 2.3.5, broader discussion of outages is provided here.

Internet outages or black holes have been studied in conjunction with MPLS [39] by Kompella *et al.*. A particular problem is black holes which are silent. These do not trigger any alarms or responses and the failure is often not routed around. In these cases packets are silently dropped. The system the authors use to test a network (for instance a section of a tier-1 ISP network) utilises active measurement between edge routers. After a problem is found, extra steps are required to interpret the specific nature of the outage.

Trinocular [55] [54] is a black hole discovery system that covers the edges of the Internet. The distinction is made between prefix based systems, like Hubble was, and this approach, which covers topology within the prefixes. Trinocular is designed to be parsimonious in its use of probing traffic. Trinocular only uses three vantage points and was originally tested on 2 days of Internet data. More recently it has been running on the Internet over time and its output data will be compared to other black hole discovery systems.

Labovitz *et al.* [40] studied RouteViews information collected from a large regional ISP in the USA. They analysed BGP updates to quantify stability and failures. A surprisingly large amount of route fluctuation was seen. Network failures were reported on the basis of type and frequency, including maintenance, power outage and fiber cut etc. Dainotti *et al.* [21] used a similar approach to 'outages or macroscopically disruptive events in other geographic or topological regions'.

3.8 Chapter summary

There are a number of areas of interest that have been reviewed related to how a distributed Internet probing system like Atlas might be implemented and perform. A main focus is how to avoid unnecessary traffic consumption and another is to be aware of drawbacks.

A focus of this thesis is to improve the base of knowledge available to designers of distributed approaches to discovering the Internet.

Part I

Preliminary tasks

Chapter 4

Stopping values

4.1 Introduction

A stopping value [7, 63] is the number of probe packets sent to the next hop after an interface that may be a load balancer in order to find the full set of successor nodes at a given degree of confidence. The number of packets sent depends on the number of successors already found belonging to the load balancer. The stopping value also depends on an assumption about the number of nodes that the virtual path is comprised of. The value chosen is typically thirty [63]. The other assumptions used in the paper [63] are as follows, referring to source node ‘s’ and destination node ‘d’:

- The in-degree of s and the out degree of d are both zero.
- If node v is in the virtual path from s to d and v is a node in an edge in the virtual path, then there is a path from s to v and from v to d.
- No routing changes occur during the topology discovery process of the virtual path.
- There is no per-packet load balancing.
- Successor nodes are chosen for traffic randomly with equal probability.
- All probes are responded to.

- Sending a probe has no carry over effect on later probes.

The method of Veitch *et al.* [63] models multipath routes and algebraically quantifies the probability of failure to find all paths or all successors of a load balancer. This algebra is converted to a computer program that models multipath discovery using the assumptions provided. This first principles method does not produce stopping values for many successors before running out of computing resources, particularly at higher confidence levels. High confidence levels cause the algorithm to run for longer before stopping values are discovered, as gradually increasing proposed stopping values are tested for their probability of failure and compared to the cut-off value. We produce an extrapolated set of stopping values from the ones that were published to upgrade Scamper to use joint probability.

The model used to predict stopping values [63] is a multiple comparisons approach as each found successor changes the stopping value for the algorithm and each step of this process invokes a new comparison. The per iter confidence parameter must shrink at each step when a further successor is added to keep the overall confidence parameter within its bound. The reason for this is the issue of multiple comparisons individually requiring higher confidence. This is similar in concept to post hoc [5] analysis where many tests are carried out once data is collected. The error that can occur if higher individual confidence is not used is a false discovery or family-wise error [62]. The correction used is similar in principle to a Bonferroni correction [65].

We also used a Monte Carlo random simulation method to generate a larger set of stopping values as a guide, using higher numbers of successors and higher confidence levels to confirm the extrapolated values. The same assumptions are used as listed above.

CDF analysis[1] is performed on data collected using a Scamper implementation of MDA to estimate stopping values in a real Internet situation where it is possible that some of the assumptions [63] might be violated.

[1]The cumulative distribution function of a real random variable is the probability that the value of the variable will be less than or equal to a given value of the variable that is used to evaluate the distribution function. The variable is graphed against the CDF function.

A graph is produced of probe counts against cumulative probability for each number of successors found. The cut-off probability for the given number of successors is applied to the graph data. This is the same 'alpha[k]' probability as used to predict stopping values using the Veitch algorithm and derived computer program. The probe counts used are the actual number of probes that it took to find the successors that were found. The cut-off chooses a number that is near to the largest number of probes that it took to find that number of successors.

4.1.1 Related work

In this research, we used MDA to map Internet topology including load balancers. In order to decide when to stop probing, MDA uses a statistical approach based on stopping values, which guarantees a desired level of confidence that all load balancer successors have been found. Augustin *et al.* [7] introduced the MDA and stopping values with a 95% confidence interval, in order to map load balancers as fully as practicable. The formula to determine these stopping values applied the confidence test to the given hop under test and not the entire trace. Standard Scamper [41], an Internet probing and analysis program by Matthew Luckie, which implements MDA, contains these values along with some 95% and 99% confidence stopping values, obtained by applying the method provided by Augustin.

In the MDA paper by Veitch *et al.* [63] algorithms are provided to determine confidence levels for entire traces and a table for 95% confidence only. We were interested in using 99% confidence for traces, and higher levels as well, as we needed to be able to say that all load balancer successors were found with extremely high surety, especially when testing to confirm that the above assumptions [63] about Internet multipath behaviour were not violated and that the algorithms performed as expected in practise on the Internet. Of particular interest is the assumption that successor nodes within a load balancer are equally likely to be traversed, as higher confidence is likely to detect violation if it occurs.

4.2 New stopping values

The Veitch research [63] provided 95% joint confidence stopping values, however we required 99% and higher. C code was written for the Veitch algorithm, which is provided in Appendix A. When the code was run, only up to the first nine stopping values in the series were able to be calculated because of a high computing overhead to this approach, see Table 4.1. See also the non parenthesised values in this Table for higher confidence levels of which only eight values could be collected on our system for higher joint confidence levels. The results for 95% agreed with those published by Veitch *et al.* [63].

				k					
Category	2	3	4	5	6	7	8	9	10
				n					
Veitch Alg. 99%	11	21	30	40	51	61	73	84	96
Veitch Alg. 99.9%	15	26	38	50	63	76	90	104	(119)
Veitch Alg. 99.99%	18	32	46	61	76	91	107	123	(139)
Veitch Alg. 99.999%	21	38	54	71	88	106	124	143	(162)

Table 4.1: Table of stopping values, which is the number of probes (n) to send to rule out a load-balancer having (k) successors. The categories are the new stopping values that we have predicted at 99%, 99.9%, 99.99% and 99.999% confidence. Parentheses show extrapolated results.

To produce the extrapolated sets of data, parabola fitting was used to predict extrapolated values based on the up to nine known values for each of the stopping value curves, Fig. 4.1. This type of curve is likely to cause the estimated values to be greater than the true values *i.e.* it offers a safety margin.

Our Monte Carlo simulation method sampled random numbers to choose successors with equal probability in a simulation of load balancers. Table 4.2 shows the predictions obtained. The number of cycles required in the analysis depended on the size of the probability involved (1-confidence_level). Appendix B shows a variable in this program called “allfound”, which is an indication of whether all successors for 30 load balancers were found in a simulation cycle. “afcount” is a count of cases of this variable being

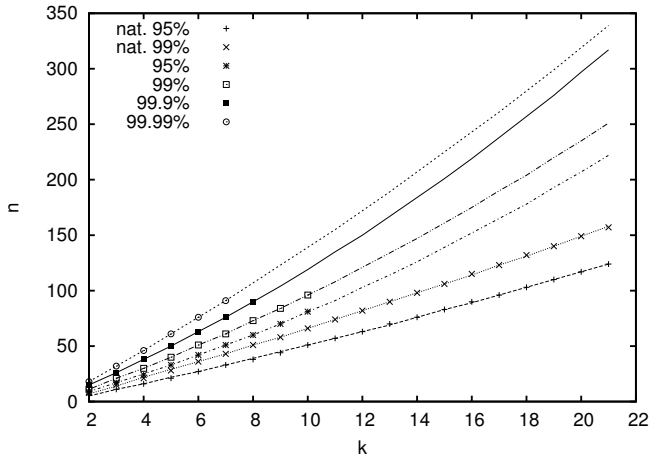

Figure 4.1: Graph of the number of probes (n) to send to rule out a load-balancer having (k) successors. The graph shows curves of predictions with Veitch computer algorithm predicted values or native scamper stopping values represented as points. The key comprises native scamper 95% and 99%, and then the updated Veitch predictions at 95%, 99%, 99.9% and 99.99%.

true and “cycles” is the number of simulations (minicycles) of 30 nodes in a path in one overall cycle. “Cycles” minus “afcount” needs to reach a similar generous level (we chose 500) for each confidence level in order for this approximation of precision to be good, as shown in Table 4.3. The standard error of the mean in the 95% case was 0.2 for the stopping values predicted. Because the number of mini-cycles was increased for the higher confidence cases, there is also expected to be a good level of repeatability.

This program does not have the universal bound on failure probability built in that the Veitch algorithm has and it is expected that the program will approximate the simple case described in the Veitch paper [63] where failure probability for all numbers of successors is constant. This case does not achieve its confidence goals for larger numbers of successors.

Though hash tables are likely to be used for load balancer forwarding decisions in real load balancers, our program should simulate a close approximation provided hashing is based on equal successor weighting. This analysis was based on virtual paths containing at most thirty nodes,

									k										
Category	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
									n											
mc95%	11	19	27	36	45	54	63	73	83	92	102	112	122	132	142	152	162	173	183	193
mc99%	13	23	33	44	54	65	77	88	98	110	122	133	144	156	167	179	192	203	216	227
mc99.9%	16	29	41	54	67	81	93	107	121	134	148	161	175	189	204	218	232	245	260	275
mc99.99%	20	34	49	65	80	95	112	127	143	158	175	190	206	221	239	255	272	289	305	322
ex95%										91	103	114	126	139	152	165	178	193	207	222
ex99%										108	121	134	147	161	175	190	204	220	235	251
ex99.9%									119	134	150	166	183	200	218	236	255	274	294	314
ex99.99%									139	156	173	191	209	227	245	264	284	303	323	344
ex99.999%									162	181	201	221	242	263	285	307	329	352	375	398

Table 4.2: Table of stopping values estimated from the Monte Carlo simulator and from extrapolation, which is the number of probes (n) to send to rule out a load-balancer having (k) successors.

Confidence	1-conf	cycles	(1-conf)cycles
95%	0.05	10000	500
99%	0.01	50000	500
99.9%	0.001	500000	500
99.99%	0.0001	5000000	500

Table 4.3: Table of cycle counts for random simulator stopping value predictor.

according to a bound that has been used in the previous research of Veitch *et al.* [63]. The Monte Carlo simulation used an over-simplified approach but was still expected to provide useful limits or approximations for stopping values. Table 4.4 shows the failure probabilities used by the programs and from these it can be predicted whether the stopping values from the Monte Carlo simulation are too high or too low. For 11 successors and below the random simulator gives higher stopping values than the Veitch algorithm and for 12 and above they are lower. This is because the Monte Carlo failure probabilities for 11 successors and below, are lower than Veitch making the stopping values higher *i.e.* less likely to fail to find successors. The negated argument also applies.

Table 4.5 shows our new estimates of stopping values at 99 %. The native values are from unmodified scamper and the updated values are our predictions based on the Veitch algorithm [63]. The Monte Carlo simulation program values were used as a comparison and tended to be slightly higher than those from the Veitch algorithm for $k < 10$, where k is the number of load balancer successors. The extrapolated values were deliberately set on the high side to allow a safety margin so that they

Successors	Veitch	Random simulation
1	0.001000	0.000335
2	0.000900	0.000335
3	0.000810	0.000335
4	0.000729	0.000335
5	0.000656	0.000335
6	0.000590	0.000335
7	0.000531	0.000335
8	0.000478	0.000335
9	0.000430	0.000335
10	0.000387	0.000335
11	0.000349	0.000335
12	0.000314	0.000335
13	0.000282	0.000335
14	0.000254	0.000335
15	0.000229	0.000335
16	0.000206	0.000335

Table 4.4: Table of failure probabilities (alpha[k] and alpha) for increasing successor count at 99% confidence. r = 0.9.

				k					
Category	2	3	4	5	6	7	8	9	10
				n					
Native scamper 99%	8	15	21	28	36	43	51	58	66
Veitch Alg. 99%	11	21	30	40	51	61	73	84	96
Random simulation 99%	13	23	33	44	54	64	76	88	99

Table 4.5: Table of stopping values, which is the number of probes (n) to send to rule out a load-balancer having (k) successors. The categories are 99% values: native scamper, Veitch Alg. derived updated values and Monte Carlo simulation program values. There are no extrapolated results in this table.

were unlikely to be low. Fig. 4.1 shows the results of the extrapolation process used to populate scamper. The points on the graph are those currently built in to scamper (native) or those obtained by running the Veitch algorithm. The Monte Carlo simulation values were used to confirm the extrapolated values between k of 11 and 21. The extrapolated values starting at k = 11 start off similar to the random simulator values and gently climb higher as it is known that they need to (as the Veitch

algorithm values also gently climb higher), through to k = 21. Thus we expect that the values beyond k = 21 are sufficient to achieve the desired confidence. These higher k values are less commonly encountered when probing the Internet, as we discuss in Chapter 9.

4.3 Experimental design and data collection

This Internet experiment is designed to test if the joint confidence stopping values incorporated into scamper find most load balancer successors as expected. Table 4.1 and Fig. 4.1 show the 99.99% stopping values that we used for collecting scamper topology data for probe count CDF analysis. Data collection for CDF analysis was run twice on 15 PlanetLab nodes in late 2013, collecting ICMP echo per-flow MDA data. There were 70000 destinations per VP, randomly selected from the 2013-04 MIDAR set provided by CAIDA. This set contained routers from across the Internet, as it was used for probing the entire Internet by CAIDA. The flow ID selection method for this analysis was by incrementing ICMP checksum. Fig. 4.2 shows the curves that were analysed (see Appendix C) to give the measured CDF stopping values using a probability threshold to determine the estimate of the stopping value. On the CDF plots a cut-off greater than 0.999 and equal to 1 - alpha[k] (See Table 4.4) was used to calculate results for joint confidence across the analysed nodes in the path. This approach uses similar probabilities to the Veitch algorithm [63] for predicting stopping values. The use of such a low cut-off relies on there being sufficient data in the remaining 0.001 or less, of the load balancer data population to give sufficient precision. A confidence level of 99.99% should make discovery of further successors unlikely, thus helping to validate the use of the CDF cut-off value. If there were further undiscovered successors then the predicted stopping values would have been higher. This CDF analysis ignored repeated occurrences of the same load balancing interface. Later CDF analysis was also performed with analysis of repeats to improve precision.

We preferred the analysis without repeats from the point of view of maximising the observed variability of load balancer behaviour. How-

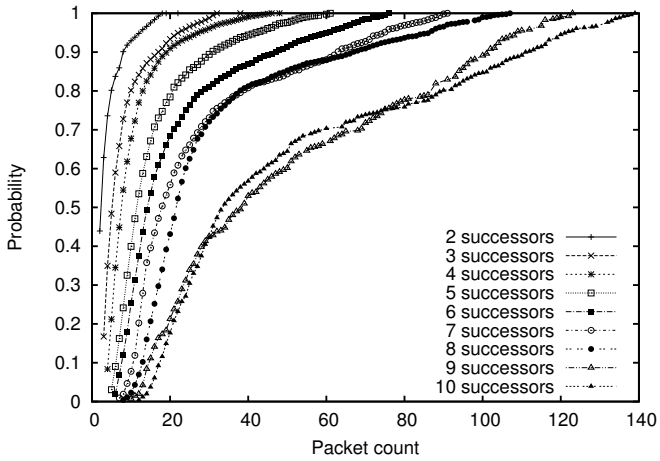

Figure 4.2: CDF graph of packet count for ruling out a load balancer having a given number of successors. This number of successors is shown in the key.

ever, the analysis with load balancer repeats was likely to give distributions with more load balancers beyond the cut off point in CDF graphs. Including repeats improves the prediction of the real world Internet occurring stopping values, because there are more data points.

4.4 Results and discussion

"Alpha k" is the conditional probability (conditional on k) of a type one error[2] or of failure to find a new successor. "Beta*" is the probability of failure to find a successor across the entire path. Equations 4.1 and 4.2 show the formulae that Veitch *et al.* used where k is the number of successors already found in an analysis and r is a value used to control the power series. The results of applying these formulae are shown in the Veitch column in Table 4.4. Here, "r" can be between zero and one (r used is 0.9), and "beta*" is one minus the confidence level for analysis of the whole path *i.e.* if confidence is 0.99 then "beta*" is 0.01. This creates a universal bound on the failure probability, where the value of r affects the steepness of the geometric progression and the initial value. Differences between the Veitch algorithm and the Monte Carlo simulation algorithm were due in part to the different ways in which the parameter "alpha k"

[2] A false positive *i.e.* that all successors have been found.

was calculated.

$$Alpha[k] = Alpha[1] * r^{(k-1)} \qquad r = 0.9 \tag{4.1}$$

$$Alpha[1] = (1 - r) * Beta^{*} \tag{4.2}$$

$$Alpha = 1 - (1 - Beta^{*})^{1/30} \tag{4.3}$$

In the Monte Carlo simulator the value for "alpha" is fixed for all values of k and is calculated using Equation 4.3. The Monte Carlo program achieves this value of alpha by running a loop for 30 nodes for 30 cycles each time executing an enclosed loop "n" times (the proposed number of packets to send) and generating a random number that selects the successor to which traffic is forwarded. "Beta*" is the probability of a type one error or failure to find a particular successor node in the whole path. A confidence of 0.99 or beta of 0.01 results in a confidence for one node of 0.9997 or an alpha of 0.0003 (probability of failure to find a new successor). The assumption used here is that there are not more than 30 nodes in a path and that "alpha" can be set to this desired failure probability for all values of k. The Veitch *et al.* formula for alpha also applies a bound but in a different way: setting values of "alpha k" that vary for different numbers of successors. This appears to be the main cause of difference between these algorithms as substituting the Monte Carlo alpha values into the Veitch *et al.* algorithm then produces similar results, confirming that the Monte Carlo program simulates the simple case described in the paper. Our stopping value results thus have a larger safety margin as they tend to be larger for up to ten successors, but they do not take advantage of the optimisation provided by the universal bound on failure probability. In particular we note that the Veitch "alpha k" values start at 0.001 for k=1 in the same example compared to our 0.0003. The other important difference is that the Monte Carlo random simulator relies on reducing the standard error of the results to a very low level (usually well below 0.4) so that the estimates are precise but still stochastic predictions of stopping values.

Table 4.6 shows the results of ICMP CDF analysis of warts[3] data collected using Scamper MDA at 99.99% confidence. The higher confidence value will allow us to consider nearly all successors when evaluating the lower confidence level of 99% and experimentally observing stopping values. The estimates of stopping values differ from the expected levels by exhibiting a higher value than expected, corresponding precisely with the 99.99% levels used to collect the data. The same pattern occurred even when analysis with repeats data was used, which had a greater volume (10 fold). The reason for this appears to be that there is a long tail (as seen in Table 4.7) containing data analysed without load balancer repeats. The tail stops at the stopping value for the value k = 2 at 99.99% joint probability. One would have expected the tail to stop sooner than the data collection at 99.99%, and this suggests that even higher joint confidence data could be of interest to collect to see if this long tail eventually stops.

These results suggest that the incremental flow ID selection used may not be resulting in random selection of successors in a sub population of load balancers. If this result is confirmed, higher stopping values may be needed for a given confidence level when it is intended to find most successors.

				k					
Category	2	3	4	5	6	7	8	9	10
				n					
Meas. cdf	17	32	46	61	76	91	107	123	139

Table 4.6: Table of stopping values, which is the number of probes (n) to send to rule out a load-balancer having (k) successors. The categories are the measured stopping values at confidence 99% from warts CDF data collected from the Internet at confidence 99.99%.

[3] Warts is the format of topology data collected by Scamper

Probes	count	cdf
2	13234	0.4393
3	5700	0.6286
4	3241	0.7361
5	1973	0.8016
6	1076	0.8374
7	667	0.8595
8	1255	0.9012
9	507	0.9180
10	344	0.9294
11	272	0.9385
12	289	0.9480
13	297	0.9579
14	254	0.9663
15	268	0.9752
16	253	0.9836
17	273	0.9927
18	217	0.9999*
19	1	0.9999
22	2	1.0000

Table 4.7: Table of population data for load balancers with two successors. Data was collected using ICMP probes and a joint confidence of 99.99%. * shows the cut off point.

Chapter 5

CAIDA and Planetlab data collection

5.1 Introduction

This chapter describes the data collection process for the analysis presented in Chapters 6 - 9 of this thesis.

In this research Traceroute MDA was run using the version built into scamper [41]. The stopping values used by scamper MDA were updated to joint confidence of finding all successors based on the work in the previous chapter. Per-destination modes were also provided.

5.2 Experimental design and data collection

Four data collection runs were performed, with each run consisting of UDP, TCP and ICMP Traceroutes. Table 5.1 shows the dates for each collection run. Each collection run tested for the three load balancing methods previously discussed: per-packet, per-destination and per-flow.

Data was collected in 2013, from June through to the end of Septem-

Run ID	date	start day	week
Run 1	18 June	day 1	(week 1)
Run 2	4 July	day 16	(week 3)
Run 3	3 August	day 46	(week 7)
Run 4	10 September	day 84	(week 13)

Table 5.1: Table of run dates for the scamper data collections in 2013.

ber. The data was collected using the facilities of the CAIDA Ark [2] and Planetlab [60, 52]. Customised scamper MDA was run from 21 Planetlab vantage points and 22 CAIDA vantage points. TCP and UDP probes were used on CAIDA and ICMP probes were used on Planetlab. Each vantage point ran traces to 70000 addresses randomly selected from the 2013-04 CAIDA MIDAR run. This set contains 1.2 million non end host router addresses. The same addresses were used from the same node for subsequent runs to increase the chance of seeing the same load balancers.

This quantity of analysis was a trade off between the number of nodes CAIDA were able to loan us, the packet rate permitted and the time that a data collection cycle needed to complete in. We chose to balance our use of PlanetLab with our usage of CAIDA, thus not increasing to larger numbers of vantage points, in order to maintain balance in the results.

The probe rate used on Planetlab was limited to 300 Probes Per Second (PPS) and scamper windows were limited to 50. Scamper windows are similar in behaviour to threads and reflect the number of concurrent traces that can be analysed. Two vantage points had maximum probes set to 65000 rather than the usual 15000.

The probe rate on CAIDA was limited to 200 PPS and maximum probes to 15000. In this case the two vantage points that had maximum probes set to 65000 were limited to 300 PPS. Scamper ran with unlimited windows on the CAIDA nodes.

The minimum probe wait time was 150 ms within a trace. In practise the actual gap was usually more than a second because of the number of concurrent data collection windows that ran. The data collection cycles took about two weeks on CAIDA and one week on Planetlab. Alias resolution was not performed as this would have increased the workload. There are other scalable algorithms that run faster than our system, such as fastmapping [19], however our emphasis on measuring stable permanent turnover (or non fluctuating changes) of load balancers required a different design.

Differences reported in the following chapters as significant have been analysed using Analysis of Variance (ANOVA) at 99% confidence unless otherwise stated. On bar graphs the valid comparisons based on the

error bars are between major ticks on the x-axis, unless otherwise stated.

5.2.1 Per-Destination

A cautious approach was taken with the collection of per-destination traces based on Planetlab rules regarding scanning random addresses. The PlanetLab Acceptable Use Policy has network usage rules which include the following: do not do systematic or random port or address block scans. We may have been slightly over cautious but this also provided an opportunity to collect data addressing a slightly different question about load balancers.

The approach is to restrict the set of destination IDs (destination address set) to seven values for one destination or one per-destination Traceroute trace. These are referred to as reduced destination ID MDA analyses or reduced destination IDs.

If we want to know the population of load balancer diamonds and in particular the identity of the primary load balancing interface, we can greatly reduce the amount of traffic required compared to finding all successor interfaces and successors in nested load balancers within the diamond. The question is then: has that one primary load balancing interface been found in a diamond and what are the dynamics of this population? The confidence of achieving this with seven destination IDs is good when compared with the probability of finding all successors, which is poor (we expect to find no more than three or four and at most seven). Without using joint confidence, there is better than a 95% chance of finding 2 successors (which identifies the load balancer), and for more successors the confidence of finding the load balancer improves. Clearly this data set has a completely different emphasis to the rest of the MDA traces, but it helps to address what can be achieved realistically when large amounts of the Internet are mapped and traffic usage and collection time are likely to be serious constraints. Detailed information about the Internal structure of load balancing diamonds is not available using this approach, however it is still possible to study populations of diamonds based on the primary divergence point.

Scamper was customised to perform low traffic per-destination load

balancer analysis. The initial stage of an existing algorithm was used [10] to create a low packet overhead analysis of per-destination load balancers. The last three bits of the destination address were varied to create the destination ID. This created a /29 block of addresses (less one) that were probed for each trace destination.

Part II

Direct analysis of load balancer data

Chapter 6

Load Balancer Prevalence in the Internet

6.1 Introduction

Population statistics of load balancers for different load balancer types and different probe packet types are useful because these statistics can aid in understanding Internet topology and topology change over time.

Population analysis of load balancers has previously used counts of load balancer types across paths [9]. This means that the same load balancer can be counted many times if it is close to a vantage point and that there is no scale factor indicating how the results relate to all of the nodes encountered. Counting the unique occurrences of load balancing interfaces as a proportion of observed interfaces avoids this undesirable repetition and gives a more relevant sense of magnitude. Here we report path percent figures to compare with previously published results as well as interface percent.

Contributions of this work:

- Updated and new measurements of populations of the different load balancer types, for the three probe types (UDP, TCP, ICMP).
- Improved on the standard load balancer population measurement by using a proportion of the total population of interfaces.

6.1.1 Related work

This research includes studying load balancer prevalence or population abundance of different types of load balancers. Augustin *et al.* reported application of MDA to collect data from the Internet [9]. They did not use ICMP probes to gather information as some UDP per-flow load balancers do not segregate ICMP traffic and therefore do not appear as load balancers [7]. They also did not use TCP probes because they found that TCP probing triggers Intrusion Detection System (IDS) alarms. We were nevertheless interested in statistics generated by data collection with all three probe packet types and also in how the the Internet has changed since this paper was published. The results from the 2007 dataset demonstrated the following pattern of load balancer type: per-flow 39%, per-destination 70% and per-packet 1.9%. In their 2011 paper [10] Augustin *et al.* reported data from 2009 data sets: per-flow 50-55%, per-destination 75-83% and per-packet less than 1.0%.

A study by Flach *et al.* [28] reported an increase in load balancer occurrence, up to 73% of paths from 39% reported by Augustin *et al.*. The Paris Traceroute method used for gathering the data presented in the Flach paper is described as finding all load balancers. It should however be noted that the 39% referred to by Augustin was for per-flow load balancers and another figure of 70% was given for per-destination load balancers, as part of the same analysis. This suggests a value slightly higher than 70% for paths containing any load balancer for the Augustin paper. This means that a large change in load balancer populations may not have been observed.

6.2 Experimental design and data collection

The run 1 data from chapter 5 was used for load balancer prevalence population analysis. Scamper warts data for each trace was analysed for the presence of the load balancer type in question to produce counts of percent paths containing that LB type.

It should be remembered that the data collected for per-destination used limited numbers of probes. Although, where diamond divergence

points have greater than 0.95 probability of being found, the probability of finding nested load balancers is much lower, because a nested load balancer receives only a subset of the probes for a given hop.

6.3 Results

Fig. 6.1 shows the frequency of paths containing a load balancer of each type as measured with each probe protocol. For UDP probing the results are similar to those previously published [10]. For per-flow our 58%±6 (± 95% confidence interval) compares to 50% previously, for per-packet our 5%±1.5 compares to 1% previously and for per-destination our 76%±4.2 compares to 83% previously. These differences exceeded 95% confidence intervals (though per-flow did not exceed 99% confidence), however this only suggests that change has occurred because the system used to collect the previous data was not identical to ours. Chapter 7 contains results used to validate the use of these special per-destination results here. For ICMP probing and per-flow load balancing our 32%±6 compares to 28% previously. There appears to be a rise in per-flow load balancing whereas our value for per-destination load balancers is lower than previously reported.

A comparison of normal (up to 128 address) and reduced destination IDs (7 address) per-destination MDA Traceroute data collection was performed. We observed that normal data collection gave a percentage paths result 5% lower than reduced destination IDs though this was not a significant difference statistically.

Our per-packet results vary considerably at different vantage points (as shown in Fig. 6.2), as a spread of ‘percent of paths’ values is observed for each packet type. There is bimodality and a long tail, which means that some vantage points are located close to per-packet load balancers. ICMP behaviour differs from the UDP and TCP behaviour, which is consistent with ICMP being collected on PlanetLab and the others being collected on CAIDA. Fig. 6.3 is a CDF graph of occurrence of load balancers at different distances from the vantage point, where the same load balancer may be counted more than once when it is found as the first

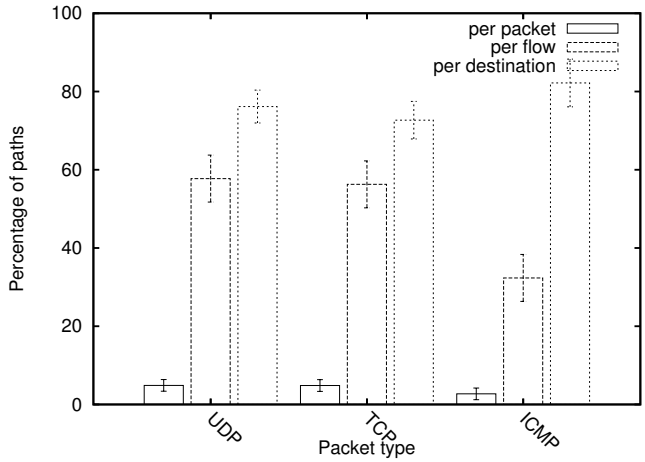

Figure 6.1: Graph of the percentage of paths containing a load balancer of the given type. Categories are UDP, TCP and ICMP and the key contains load balancer type. Error bars show 95% confidence intervals.

load balancer of its type in a path. Taking the first occurrence of a given type of load balancer in a path means that there is the same number of underlying data points as paths that are found to contain load balancers of the given type. However it should be noted that with this approach there is a bias towards smaller load balancer TTLs. Though this graph is consistent with some per-packet load balancers being close to vantage points, the other load balancer types appear to be even more problematic in that load balancers close to VPs are repeatedly found causing a higher percentage of paths to be reported as containing such load balancers.

Another statistic for load balancer characterisation is the proportion of uniquely identified load balancer nodes as a percentage of uniquely identified nodes, across a set of traces. This is the proportion of nodes that are load balancers. This statistic can not be used with reduced destination IDs per-destination data but we expect that the usual percentage paths statistic is valid and may be used, as per-destination diamonds are identified and counted as usual. Fig. 6.4 shows this statistic where ‘dmd’ (diamond) means the load balancers counted are divergence points for entire diamonds and ‘npf’ (not per-flow) means the previously seen per-flow load balancers were subtracted from the per-destination load balancer set. Per-destination load balancers theoretically include

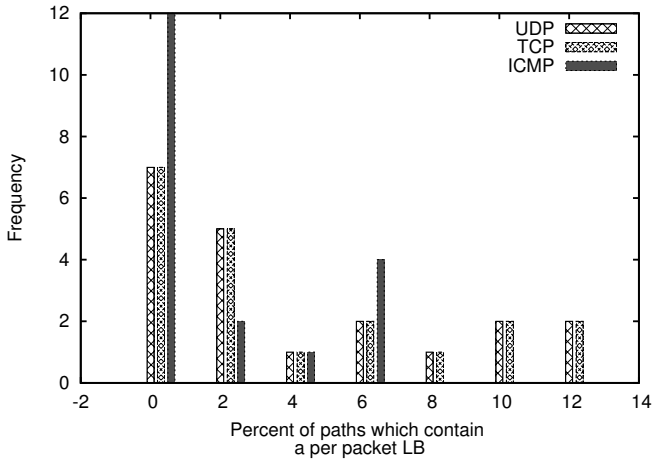

Figure 6.2: Graph of frequency distribution of the percentage of paths analysed in run 1 containing a per-packet load balancer for each vantage point.

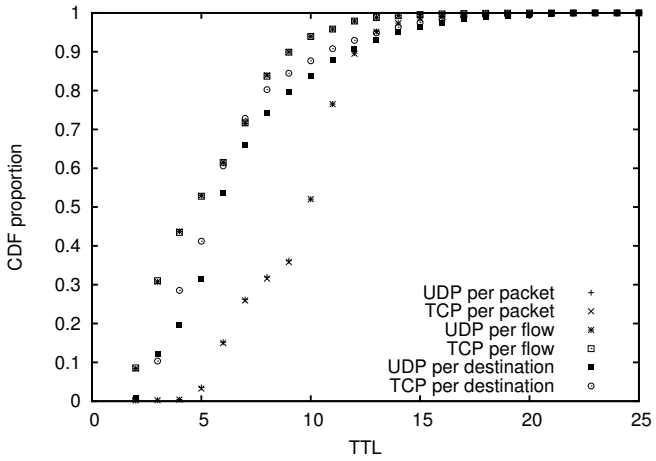

Figure 6.3: Cumulative distribution graph of TTL or hop count of the first load balancers found in a path.

per-flow load balancers, as the per-flow five-tuple includes the destination address. Though, strictly speaking, per-destination load balancers are per-flow load balancers we make this distinction to see if the sets of per-flow and per-destination load balancers in practise include each other. We see a reduction (30% for UDP and TCP) from per-destination 'dmd' to per-destination 'npf dmd' *e.g.* for UDP 3.3% of nodes were per-destination 'dmd' but for per-destination 'npf dmd' 2.2% was observed. A smaller reduction is seen for ICMP (12% reduction, from 3.5% to 3.1%).

This shows that 70% of per-destination diamond divergence points are not found as per-flow load balancers.

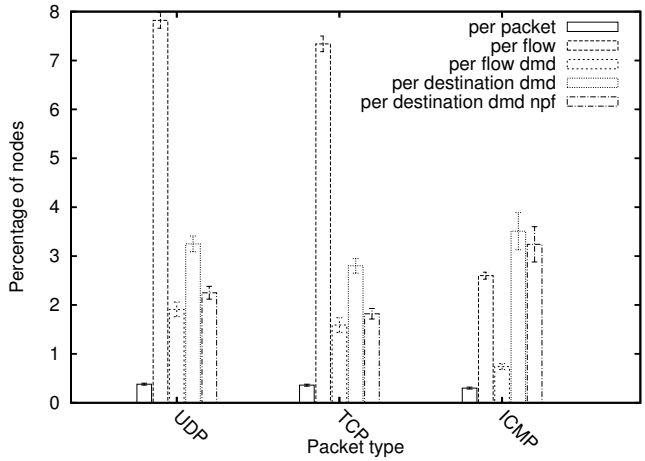

Figure 6.4: Graph of the percentage of nodes that are a load balancer of the given type. Categories are UDP, TCP and ICMP and the key contains load balancer type. Error bars show standard error of the mean.

The per-packet results in Fig. 6.4, show very little difference between probe types (99% confidence), where ICMP is slightly lower than the other two probe packet types. For per-flow load balancing ICMP is less than half of the other two methods, which differ slightly with TCP lower than UDP. This finding is consistent with differences previously reported [44]. A similar result was observed for per-flow diamond load balancers, although these percentages are much lower than the combined per-flow result. For per-destination diamonds ICMP and UDP are significantly different from TCP, where TCP is much lower than the others at greater than 99% confidence. For per-destination 'npf' diamonds, all three probe types were different from each other with TCP lowest and ICMP highest. Table. 6.1 shows results for per-flow load balancer plus internal diamond nodes, which are interfaces contained within a load balancer. ICMP is easily significantly lower at half the size of the other two.

We see ICMP load balancer populations being represented differently to the largely more similar UDP and TCP. A key question is: are some UDP per-flow load balancers behaving as ICMP per-destination 'not per-flow' load balancers? We compare the ICMP data from PlanetLab with the UDP

LB type	UDP	TCP	ICMP
per-flow internal	31.4%±1.2	30.2%±1.2	14.9%±1.2

Table 6.1: Table of load balancer prevalence statistics where internal diamond nodes are included. 99% confidence interval of the mean is shown as plus or minus.

data from CAIDA. The number of cases where there were load balancers in both sets across run 1 was 10,654, showing us that that there are many cases where this duality exists. It should be noted here that 'not per-flow' ignores per-flow load balancers of the same probe type, where the UDP source port or ICMP checksum is varied rather than the destination address. This means that there are many cases where ICMP used destination address rather than checksum to control segregation across successor nodes, whereas UDP used source port rather then destination address for the same load balancers.

To give some indication of coverage of the Internet a graph of vantage point count versus count of load balancer successor sets is shown in Fig. 6.5. Some deceleration is seen suggesting that an increasing proportion of load balancers found have already been seen or covered. An analysis using CAIDA and PlanetLab traceroute data to determine the rate of discovery of new nodes as opposed to load balancers showed that the curve has much further to climb before it truly flattens out. Table. 6.2 shows percentages of load balancers found uniquely on average across vantage points. The smaller this figure is, the more of the Internet there is that has already been seen by other vantage points, *i.e.* the better our coverage of the Internet.

Probe packet and LB types	% unique
UDP per-flow	2.9%
TCP per-flow	3.2%

Table 6.2: Table of percentages of load balancers found uniquely at each vantage point on average.

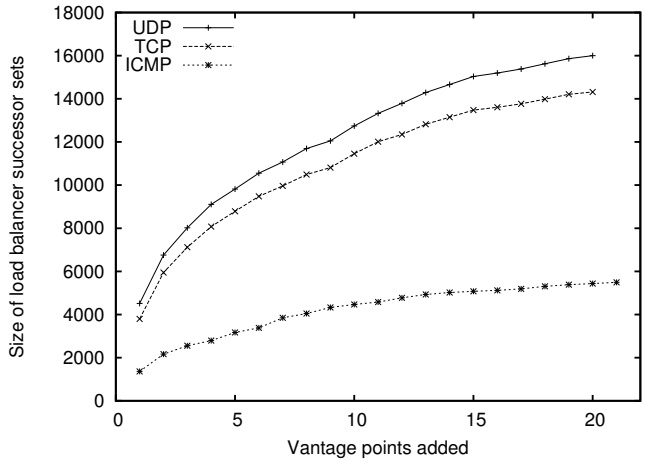

Figure 6.5: Graph of the size of non matching successor sets versus the number of vantage points whose data has been included.

Chapter 7

Efficient analysis of per-destination load balancer divergence points

7.1 Introduction

This experiment addresses the issue of counting per-destination diamonds without expending the probe traffic required to fully map the internal structure of the diamond. If counting diamonds without mapping them completely is desired, this could be a useful step towards discovering per-destination load balancer diamonds without the expense of full analysis.

Two new scamper modes were created. One performs full analysis of per-destination load balancers, allowing up to 128 different destination IDs. The second, introduced in Section 5.2.1, implements increased efficiency analysis with a reduced mapping of diamond internal structure allowing only seven destination IDs, which gives a greater than 95% probability of finding load balancers without using joint probability adjustments. The key emphasis of the second is to save on traffic and to focus on finding diamond divergence point interfaces.

7.2 Experimental design and data collection

The per-destination and per-flow data from each CAIDA run in Chapter 5, Table 5.1, was used in this experiment where the following were

collected:

- Counts of paths with per-destination load balancers,
- Paths with per-destination load balancers that have not been seen previously as per-flow,
- Counts of unique per-destination load balancers.

A scamper driver was written to perform both the efficient per-destination collection and full per-destination collection to provide a comparison of these modes. Run dates for full ICMP per-destination data collection and the compared ICMP run are shown in Table 7.1.

Run ID	date
Run 5	16 October 2013
Run full destination IDs	1 February 2014
Run comparison	1 April 2015

Table 7.1: Table of run dates for the ICMP per-destination scamper data collections on PlanetLab.

7.3 Results and discussion

Fig. 7.1 shows the number of traces that contained a per-destination load balancer where the same 70000 destinations in total were used in each cycle for data collection. The term “reduced” means that the efficient method is used, and “full” means that full traffic mode is used. The term “(all)” means that no cases have been removed from the per-destination load balancer set, and “(not per flow)” means that cases previously found as per-flow load balancers have been removed.

For Runs 1-5 we see a 10% reduction within each data collection cycle when per-flow load balancers in the per-destination data are excluded. This means that 10% of per-destination load balancers are forwarding on both source port and destination address as these are the fields that were varied in this analysis. The term per-flow suggests that forwarding decisions based on the destination address might occur as well as

ports, however this was not observed frequently. There appears to be a slight downward trend for the "reduced destination IDs (all)" results, however this is not significant (P=0.09 for regression). "Full destination IDs (all)" are not significantly lower than "reduced destination IDs (all)". This suggests that the reduced destination IDs method is producing useful per-destination load balancer count data.

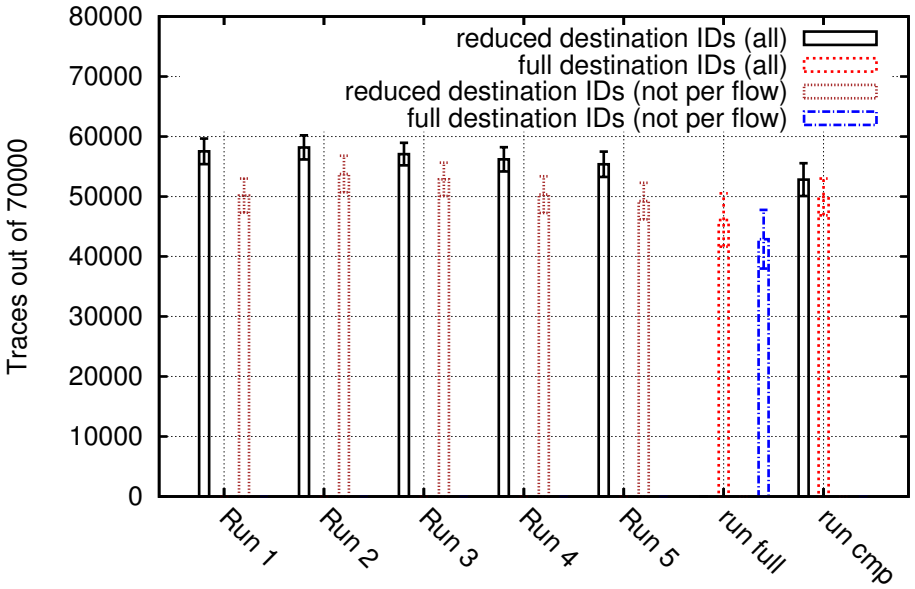

Figure 7.1: Graph of paths out of 70000 that contain a per-destination load balancer. SEM is shown.

Fig. 7.2 shows the numbers of unique per-destination load balancing nodes found. There were about half as many found under the reduced destination IDs method, however the numbers of paths found were similar. This could be because diamond internal load balancers, *i.e.* load balancers within the diamond but not including the initial divergence point load balancer, are not found under this method.

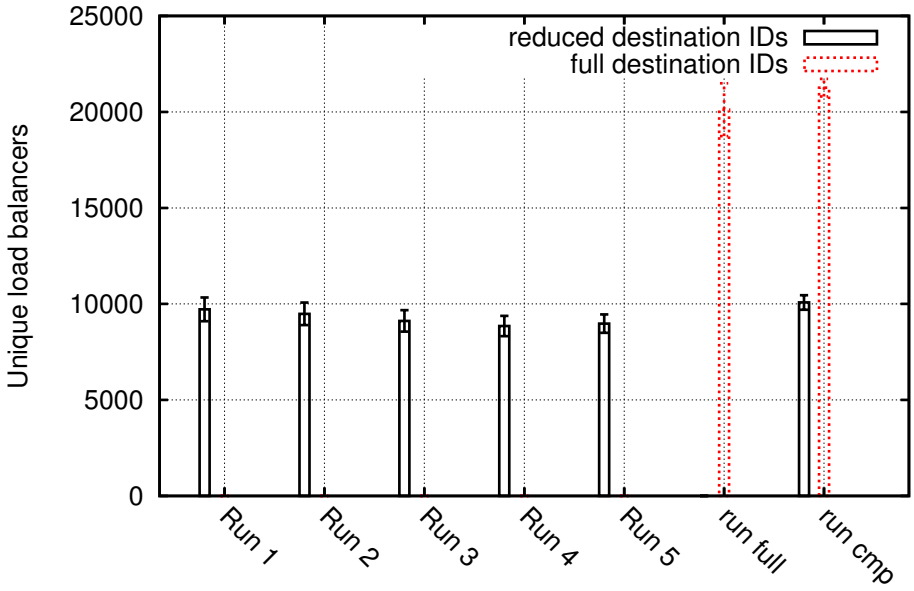

Figure 7.2: Graph of count of unique per-destination diamond divergence points. SEM is shown.

Fig. 7.3 shows that the reduced destination IDs method uses between 10% and 20% the amount of probes as used with full destination IDs. It is likely that this saving can be improved on and still give similar performance, as discussed in Section 16.2.2.

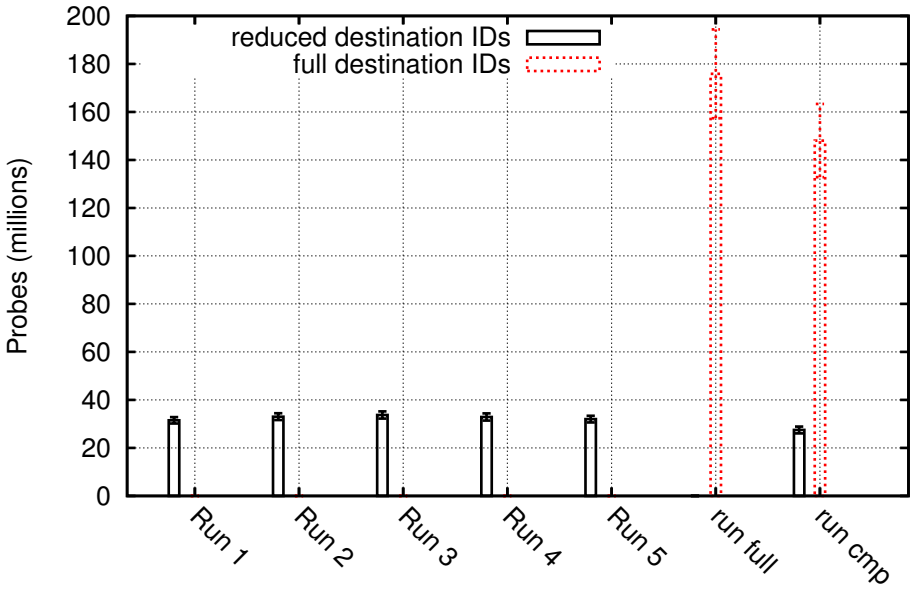

Figure 7.3: Graph of trace probe counts. SEM is shown.

Chapter 8

Load Balancer Turnover in the Internet

8.1 Introduction

We study changes in the populations of load balancers in the Internet, because this is another obvious and interesting characteristic of load balancer behaviour. This includes quantifying the disappearance of load balancers and the creation of new ones, along with changes in the internal structure of existing load balancer diamonds.

The issues addressed in this chapter can be encapsulated in the following questions: Are the same nodes still behaving as load balancers from one run to the next? Do they have the same set of successors? Have there been any changes to the entire diamond structure? These questions are important because load balancing improves the performance and reliability of the Internet, and understanding trends in the implementation of load balancing helps us to understand what the make-up and impact of load balancing will be in the future.

8.1.1 Related work

In a related area of research Cunha *et al.* [20] analysed route duration, route age and route prevalence. Route duration was demonstrated to be less than 100 hours for 95% of virtual paths and less than 1 hour for 60%. Route age and route prevalence were related to route residual lifetime, which is the remaining life of a route from the current time. A route

age of 20 hours related to a median residual lifetime of 40 hours and route prevalence of 24 hours related to a median residual lifetime of 20 hours. The associated graphs indicated that younger routes (route age) had shorter lifetime remaining and routes that spend a lesser percentage of the time active (route prevalence) are also likely to have a shorter lifetime. In another paper the same authors used Fastmapping to count internal changes of load balancer diamonds [19]. This revealed that 10% of load balancer diamonds experienced internal change, *i.e.* a change in the group of nodes between the divergence and convergence points, within 3 days and 4.6% within one day. Fastmapping is comprised of frequent mapping with Tracetree (using 28 minutes as a cycle duration) and daily mapping with MDA to 1000 varying destinations.

8.2 Experimental design and data collection

Runs 1, 2, 3 and 4 were collected as described in Chapter 5 and comparisons between run 1 and the other runs were performed using a warts analysis program that had been written to quantify the following metrics of load balancer turnover:

- percentage of distinct load balancers in common between two runs,
- percentages of unique per-flow and per-destination ‘npf’ (not per-flow) load balancers in common between two runs,
- percentage of per-flow full matches of immediate successor sets between two runs,
- percentage of diamonds exhibiting an internal change between two runs.

8.3 Results and discussion

A key consideration when quantifying load balancer turnover is the frequency of non load balancer route changes as a route change could bypass a previously seen load balancer. Counts of paths not exhibiting

route changes are graphed in Fig. 8.1 for UDP, TCP and ICMP packet types. The UDP and TCP means are not noticeably different from each other, however the number of route changes seen with ICMP was significantly lower and there were significant decreases over time. Cunha *et al.* showed long lived paths persisting for tens of days even though there was a high rate of short term change [20]. The rates associated with long lived paths are consistent with what we have seen. There is still an important rate of route change and therefore the analysis of load balancer turnover is less simple than it otherwise would be. It also may be that the very high rates of route change reported previously by Cunha *et al.* are too high, according to the work of Marchetta *et al.* [45].

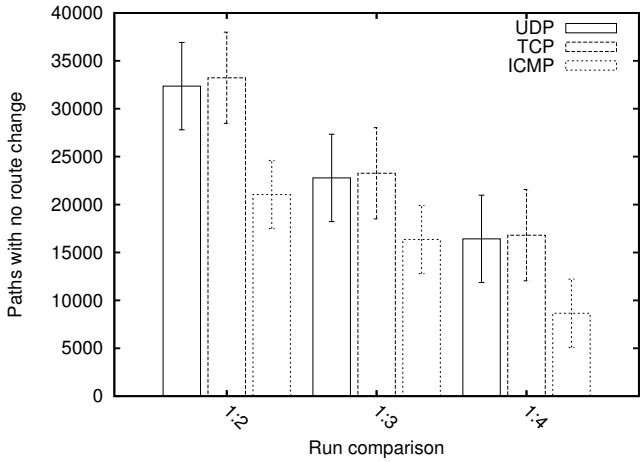

Figure 8.1: Graph of paths with no route change between runs in non load balancer nodes, out of 70000 paths. 99% confidence intervals are shown.

Fig. 8.2 shows the percentage of distinct load balancers in common between run 1 and the subsequent runs *i.e.* load balancers in common relative to the number of load balancers observed in run 1 as a percentage. This intersection decreases significantly over time for all three probe types. This means that the number of original load balancers remaining in the population is falling over time, as identified by load balancer divergence IP address and immediate successor set, and that the population is changing.

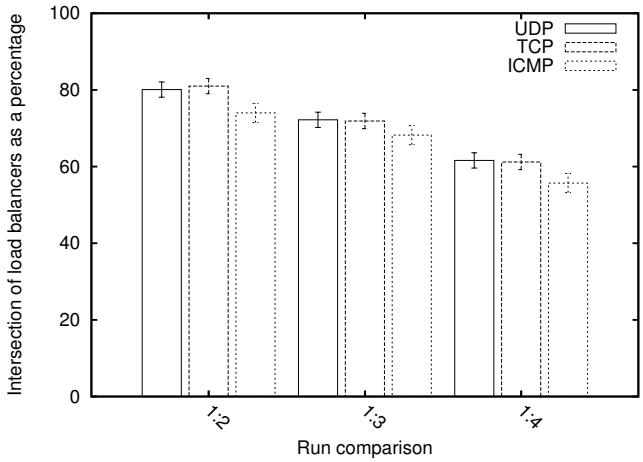

Figure 8.2: Graph of intersection of distinct per-flow load balancers as a percentage between runs. 99% confidence intervals are shown.

Fig. 8.3 shows the percentage of unique load balancer per-flow and per-destination (not per-flow) in common between run 1 and subsequent runs. There are significant decreases over time for each packet type and load balancer type. For validation of the use of per-destination data refer to Chapter 7 where we show that the diamond divergence point is likely to be found in this data. This means that the number of original load balancers is decreasing, as identified by divergence point IP address.

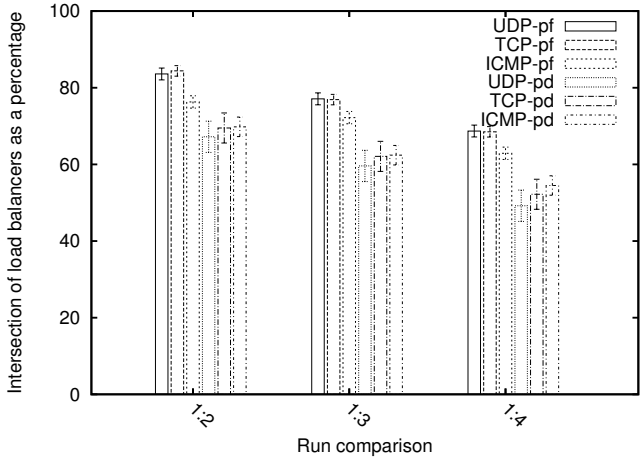

Figure 8.3: Graph of intersection of unique per-flow and per-destination 'npf' load balancers as a percentage between runs. 99% confidence intervals are shown.

The mean of distinct per-flow UDP load balancers found over 70000 traces was 12500. For unique load balancers the mean was 6300. We found that 63% of unique load balancers had a single successor set, 18% had two, 7.5% had three and 11% more than three successor sets. If a unique load balancer has more than one successor set it is counted once only, however a distinct load balancer count would count the load balancer once for each successor set, thus explaining the difference between the presentation of data in Fig. 8.2 and Fig. 8.3.

Fig. 8.4 shows the percentage of fully matching immediate successor sets in common between run 1 and subsequent runs. A decline over time is observed at a similar rate seen for turnover of load balancer nodes.

The number of different successor sets found over time for the UDP probing case showed an increase at 95% confidence: run 2 produced 4381 sets and run 4 produced 4533 successor sets. This suggests that the population of load balancing diamonds is increasing.

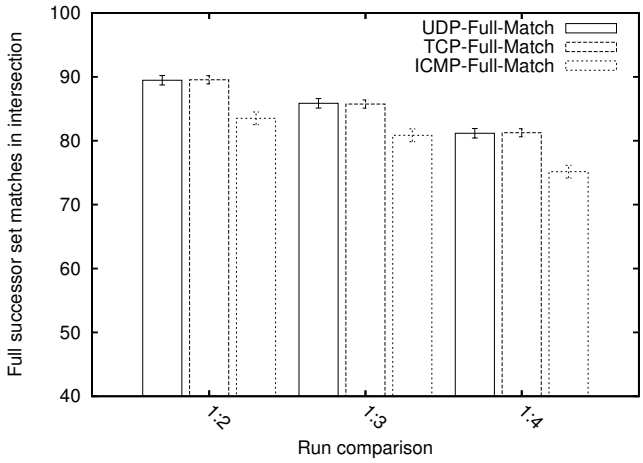

Figure 8.4: Percentage of per-flow full matches between immediate successor sets found in the intersection set between runs. 99% confidence intervals are shown.

Fig. 8.5 shows the percentage of diamonds exhibiting an internal change. Increases which are significant at 99% confidence are seen for each probe type at each step over time. ICMP gives a significantly higher value than the others in each case. This suggests skew in the smaller population of load balancers that share ICMP packet load towards those

that are changing internally. The steady climb in the proportion of the population of diamonds that have changed is similar to the rate of load balancer divergence point turnover discussed earlier. There appears to be a steady rate of change of the internal structure of load balancer diamonds.

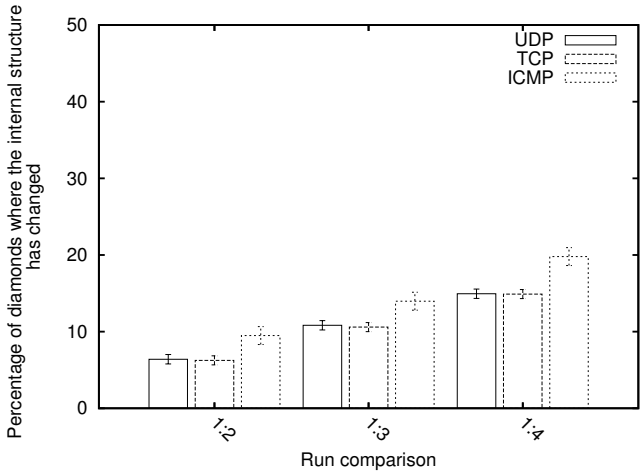

Figure 8.5: Percentage of per-flow changed diamond member interface sets between runs. 99% confidence intervals are shown.

When a node exists as a load balancer in one run but is not a load balancer in another run then it is called a collapsed load balancer. Whether the load balancer is created or destroyed from one run to the next is interpreted as the direction: forward or reverse collapse. Fig. 8.6 shows the number of collapses out of 70000 traces (for an average VP). Differences between forward and reverse are mostly small and there appears to be an upward trend over time. The slight upward trend does not correlate strongly to the time passed, suggesting that reversible fluctuations are being observed. The levels of collapse affect about 10% of unique load balancers.

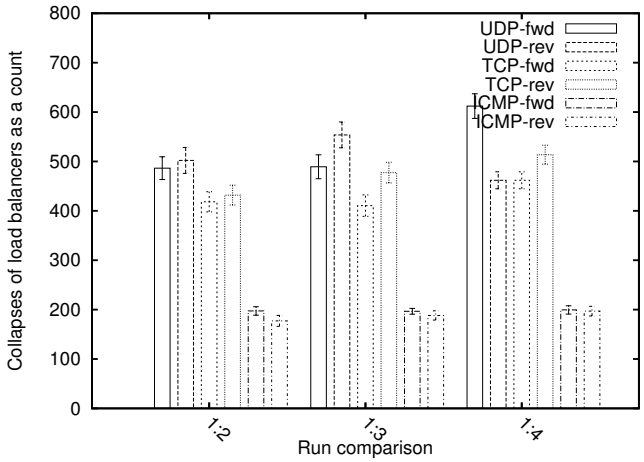

Figure 8.6: Graph of forward and reverse collapse counts for UDP, TCP and ICMP. SEM is shown.

For distinct and unique intersection sets, along with successor set matching and internal diamond structure, changes in the order of 1.4% per week are seen between runs as shown in Table. 8.1. Changes between run comparisons relate to time periods of 30 and 38 days, comparing run 1 with 2 and 3 and then run 1 with 3 and 4. With regard to intersection analysis, change (turnover) corresponds directly to load balancers disappearing from the run 1 set and also new load balancers appearing in the later set. For successor set analysis, change (turnover) relates to reduction in fully matched sets. Lastly, for cases where diamonds are found for the same load balancer associated interface, cases of changed internal diamond structure follows a similar pattern except an increase is seen. The observed value for changes in internal diamond structure is lower than previously published [19] and once again may be caused by the occurrence of non permanent virtual path fluctuations. However, in this experiment we see some agreement between a variety of statistics relating to load balancer turnover. ‘Matching per-flow’ and ‘internal per-flow’ are lower than ‘distinct per-flow’, ‘unique per-flow’ and ‘unique per-dest’ suggesting that internal changes in existing load balancers are less common than changes in the load balancer divergence point population.

Figure data type	1:2-1:3 udp	1:2-1:3 tcp	1:2-1:3 icmp	1:2-1:3 all	1:3-1:4 udp	1:3-1:4 tcp	1:3-1:4 icmp	1:3-1:4 all	Average all
distinct per flow	1.84	2.12	1.35	1.77	1.95	1.97	2.3	2.08	1.92 (0.13)
unique per flow	1.52	1.75	0.96	1.41	1.55	1.55	1.71	1.6	1.51 (0.12)
unique per dest	1.77	1.73	1.73	1.74	1.92	1.82	1.46	1.73	1.74 (0.06)
matching per flow	0.84	0.89	0.62	0.78	0.86	0.83	1.05	0.91	0.85 (0.06)
internal per flow	-1.03	-1.01	-1.05	-1.03	-0.76	-0.79	-1.07	-0.87	-0.95 (0.06)

Table 8.1: Table of turnover values from the figures in this chapter. Percent per week change. Standard error of the mean is shown in parentheses.

Chapter 9

Diamond structure

9.1 Introduction

Load balancer diamonds improve the reliability and performance of the Internet, but not all diamonds use the same configuration. Load balancer configuration is part of diamond structure. Diamond structure also includes load balancer nesting. The configuration of a diamond can affect its contribution to improving reliability and performance, thus obtaining some measures of configuration in conjunction with diamond population frequencies is likely to provide insight into how the Internet should develop in the future.

Though diamond structure has been analysed before [9], we were interested in studying this because load balancing is likely to be an area of growth and change due to its relative newness. Diamond width is determined by counting the number of interfaces at a particular hop count within a load balancer diamond. Minimum width is actually the number of link disjoint paths in a diamond, which occasionally can be less than the successor count of the divergence point, if there is a high degree of nesting. Maximum width refers to the maximum number of interfaces that can be reached in a diamond where interfaces are counted at each hop. If there is more than one diamond the largest width is reported as the maximum. Diamond length without alias resolution refers to the maximum hop count between divergence and convergence at the IP level. For example, a diamond of length 2 is likely to correspond to a pair of routers with multiple links between them. Symmetry refers to the variability of

path length for all paths between the divergence and convergence points for the diamond. As diamond structure has been investigated before, we were interested in how the Internet has changed and also to introduce a new metric called literal width, which helps reveal large highly nested diamonds.

9.1.1 Related work

Augustin *et al.* presented a previous study of diamond structure in [9]. They found that 99% of per-flow UDP probed paths had a minimum width of 5 or less. 85% of per-flow UDP probed paths had a maximum width of 5 or less. Augustin *et al.* also reported that the maximum value obtained for either width is 16. 90% of paths had a length of 2 and fewer than 1% have a length greater than 8. The authors also found that minimum width and length are positively correlated. This work compliments [9] by investigating the same metrics for a more recent dataset to see how the Internet has changed.

9.2 Experimental design and data collection

The Traceroute MDA data from run one was used for diamond analysis. UDP, TCP and ICMP per-flow data was analysed to find diamonds and examine their structural properties.

We developed software to count cases of different diamond maximum and minimum widths and lengths and to determine symmetry in these cases. To do this we had to first recognise diamonds. This was done by assembling topology information and collating IP addresses of nodes and their successors. This focussed on finding primary diamond divergence and convergence points.

We also developed a new metric, literal width which is the total number of unique paths in the virtual path between diamond divergence and convergence. Fig. 9.1 shows a diamond with minimum and maximum width of three and a literal width of nine. A diamond with a large literal width relative to its maximum width is likely to consist of nested and repeated load balancing structures. Mapping such diamonds will require a large

amount of probe traffic to ensure all possible paths are seen.

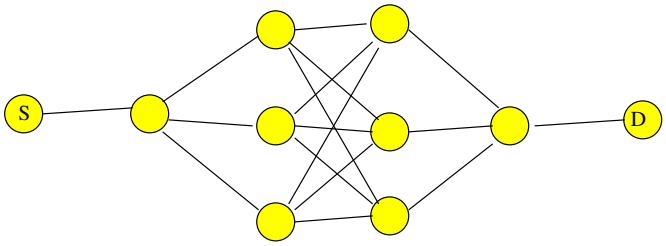

Figure 9.1: Diagram of a load balancer with a larger literal width than maximum width.

9.3 Results and discussion

Complete diamonds with a discovered convergence point are present in 65% of unique load balancers for UDP and 73% for TCP where a given load balancing interface was analysed once. The missing convergence points are most likely because some traces end inside load balancers making it impossible to find a convergence point. Complete diamonds were found in 57% and 56% of UDP and TCP paths respectively, which is similar to the results in Fig. 6.1. Use of the MIDAR address set may be a partial cause of failure to find convergence points, however we were strongly encouraged to use these addresses by CAIDA to avoid unnecessary traffic to end hosts.

In the following figures UDP, TCP and ICMP load balancing behaved in a very similar fashion. The results were accumulated across all of the vantage points. Fig. 9.2 shows diamond length. Augustin *et al.* [9] saw longer diamonds than we did, with a maximum length seen of 20. The number of paths containing asymmetric diamonds with a maximum length greater than 10 is much higher than the corresponding number of symmetric paths. The population of load balancers with length greater than 10 is small.

Augustin *et al.* observed that 37% of UDP per-flow diamonds had a length of 2. We saw that 66% of UDP per-flow diamonds had a length of 2. For per-flow diamonds with greater than length 8 we saw 0.3%. These results suggests that smaller diamonds are occurring in greater

proportions than previously. It may be that network administrators are now getting useful performance, efficiency and redundancy benefits from increased use of shorter load balancer designs.

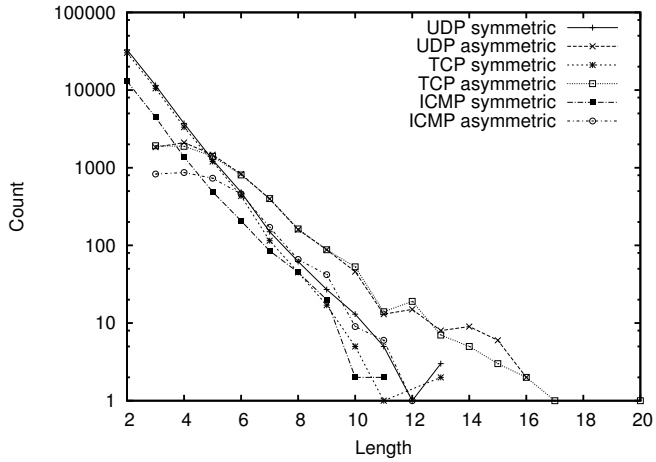

Figure 9.2: Distribution of observed load balancer diamond lengths.

Fig. 9.3 and Fig. 9.4 show diamond minimum width. The greatest minimum width seen in previous research is 16 [10]. This gives an indication of the range of successor counts seen in the Internet. The barrier of 16 successors seems to be gone for some load balancing routers. Augustin *et al.* found that a minimum width of two was seen for 55% of UDP per-flow load balancers. We saw that 60% had a minimum width of 2. This suggests there may be a slight increase in the use of very narrow diamonds.

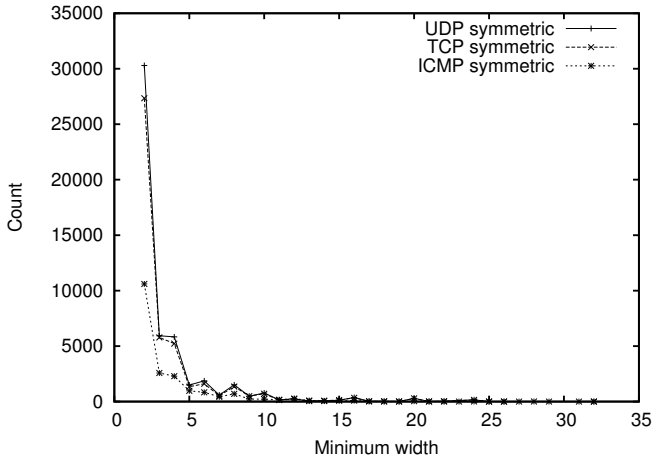

Figure 9.3: Distribution of symmetric diamond minimum width.

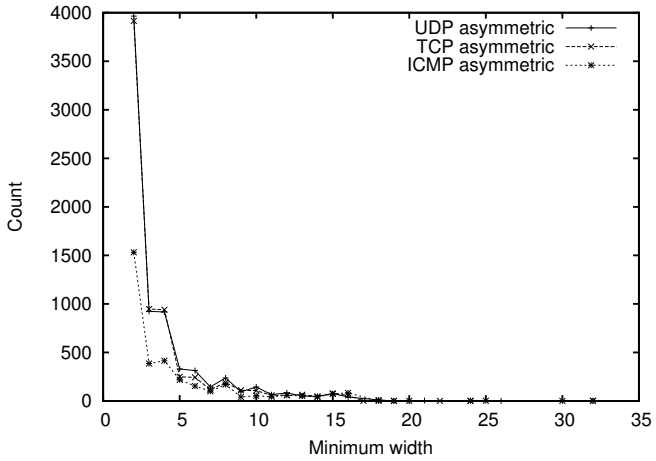

Figure 9.4: Distribution of asymmetric diamond minimum width.

Fig. 9.5 and Fig. 9.6 show diamond maximum width. Some greater minimum and maximum widths than have been previously reported have been observed [10] [9]. For minimum width a maximum of 16 was seen previously and we saw 32. For maximum width a maximum of 30 was seen previously and we saw 63. Augustin *et al.* found that a maximum width of two was seen for 24% of UDP per-flow load balancers. We saw that 46% of UDP per-flow load balancers had a minimum width of 2. Augustin *et al.* reported that 85% of UDP per-flow diamonds had a maximum width or widest hop distance of 5 or fewer. We saw that 78% of UDP

per-flow diamonds had a maximum width of 5 or fewer. This indicates a slight trend towards using more nested load balancers.

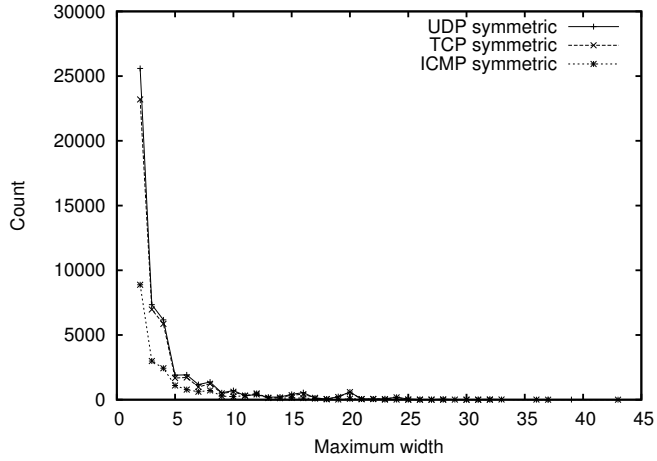

Figure 9.5: Distribution of symmetric diamond maximum widths.

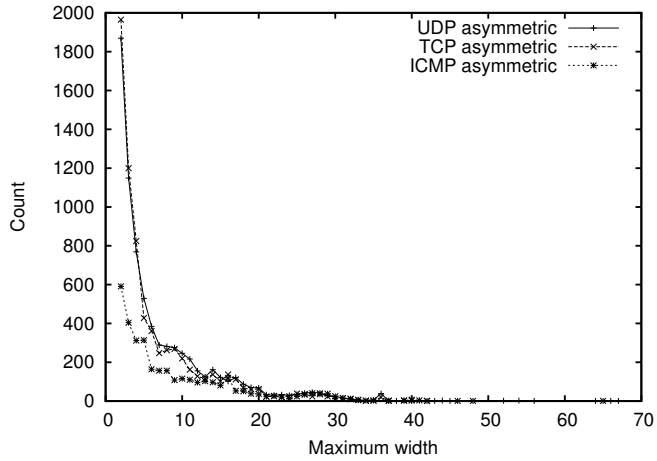

Figure 9.6: Distribution of asymmetric diamond maximum widths.

Fig. 9.7 and Fig. 9.8 show diamond literal width. It can be seen that there is a tail of literal width values over 50; however these larger values make up a small proportion of the total. Our investigation of these cases shows that the amount of traffic required to probe these diamonds is very large and sometimes exceeds the maximum of 65000 probes.

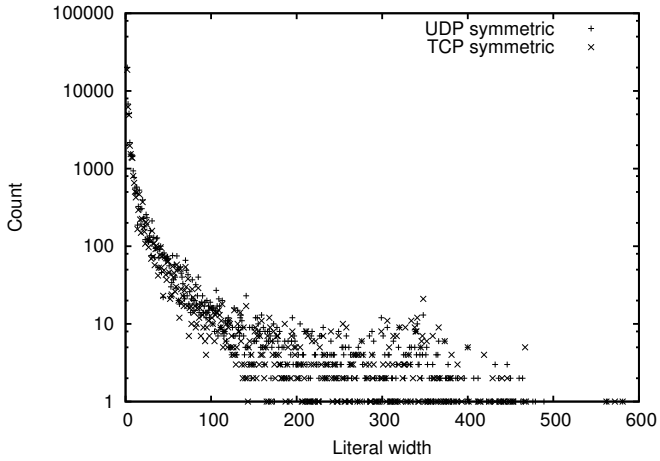

Figure 9.7: Distribution of UDP or TCP symmetric diamond literal widths.

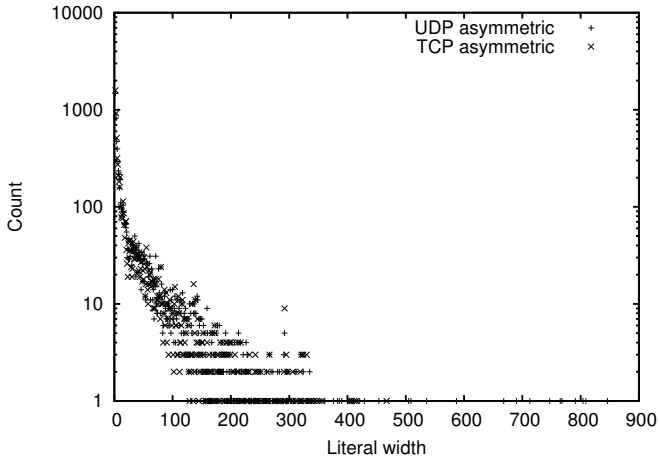

Figure 9.8: Distribution of UDP or TCP asymmetric diamond literal widths.

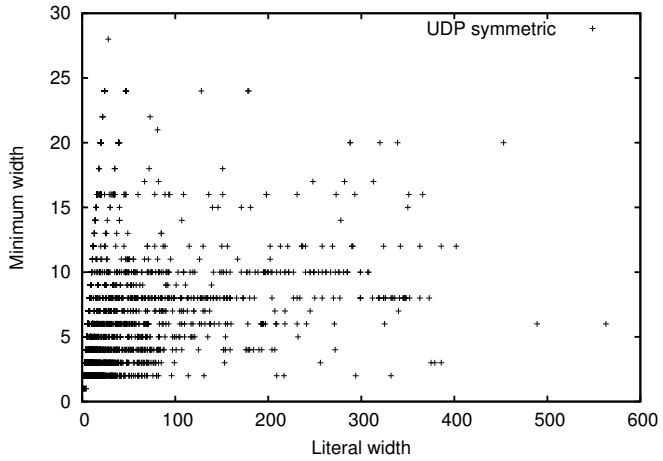

Figure 9.9: Minimum width versus literal width for symmetric UDP diamonds.

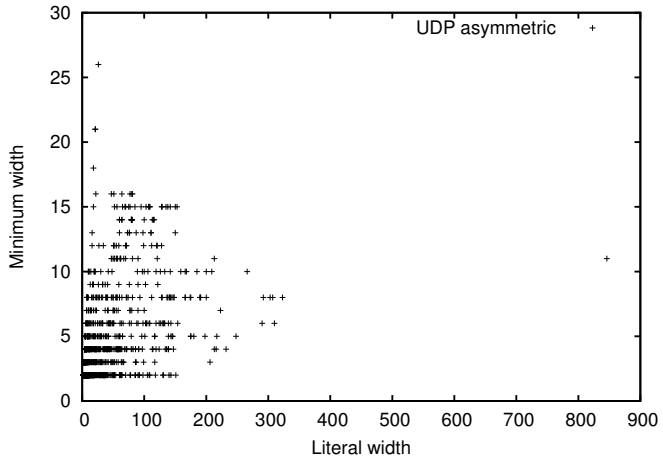

Figure 9.10: Minimum width versus literal width for asymmetric UDP diamonds.

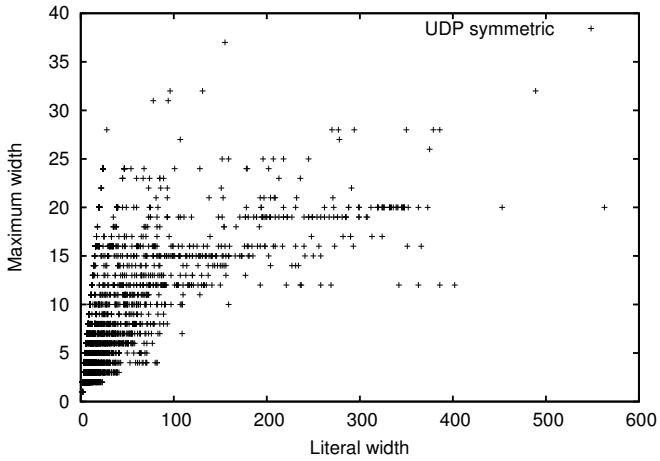

Figure 9.11: Maximum width versus literal width for symmetric UDP diamonds.

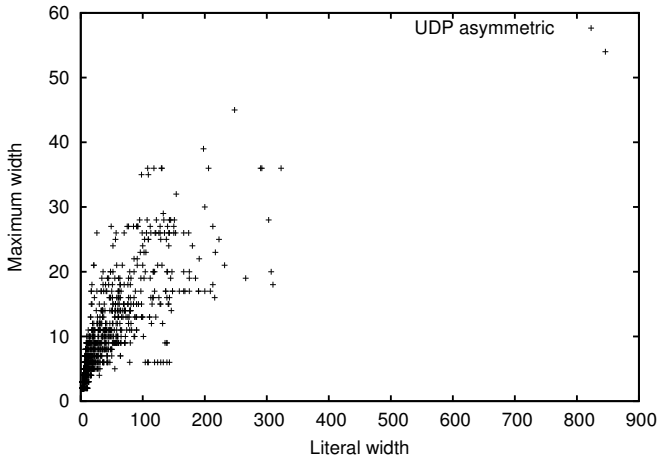

Figure 9.12: Maximum width versus literal width for asymmetric UDP diamonds.

Fig. 9.9 and Fig. 9.10 show minimum width versus diamond literal width for symmetric and asymmetric UDP diamonds respectively. Minimum width equals literal width for 60% of symmetric diamonds and 27% of asymmetric diamonds. Fig. 9.11 and Fig. 9.12 show maximum width versus diamond literal width for symmetric and asymmetric UDP diamonds respectively. Maximum width equals literal width for 63% of symmetric diamonds and 36% of asymmetric diamonds. These results suggest that simple unnested diamond designs are the most common and

also that asymmetric designs are more likely to contain load balancer nesting.

Chapter 10

Analysis of fields other than the classic five tuple

10.1 Introduction

The classic five-tuple is usually considered to be responsible for the choice of successor in per-flow load balancers. However, are there some routers that are influenced by other fields in the packet being forwarded? There may not be an obvious benefit to using fields outside the classic five-tuple, when per-flow load balancing is designed to stream TCP connections through the same segments of load balancers. However, there may still be cases where certain brands of routers, for no obvious reason, respond to non five-tuple fields when forwarding packets to load balancer successors.

10.2 Related work

A paper by Augustin *et al.* [7] states that some load balancers ignore ICMP packet contents when defining a flow. Another of their papers discusses Paris Traceroute [8], but leaves further exploration of load balancer fields for later research. According to a later paper of theirs [9], the MDA algorithm varies load balancing fields to cause segregation of probe traffic to the set of load balancer successors. Another field was varied that supposedly does not influence load balancing, but is used to identify individual packet ICMP time exceeded messages. This was because the IP header and the first 8 bytes of the payload are returned

from the original probe packet, which allowed that packet to be identified. However, consider a case where a router is not limited to just the 5-tuple fields when performing per-flow load balancing. Therefore, it is important to understand what the effect of modifying various non five-tuple fields will have on Traceroute-based mapping techniques. If a load balancer is found that responds to fields outside the classic five-tuple we called it a pseudo per-flow load balancer.

10.3 Experimental design and data collection

Table 10.1 shows the fields used as pseudo flow ID and packet ID for this experiment. Pseudo flow ID is the field that is varied to attempt to cause segregation and packet ID is the field that is varied to recognise the packet fragment that is returned in the ICMP time exceeded packet that returns to the source.

Bit shifting a field involved setting the field to one and then shifting the bit across 15 steps of increasing significance based on the increasing value of the flow ID. After 16 positions the bit started again at position 1. This provided enough flow IDs to easily detect a primary diamond divergence point.

In the UDP IP HeadLength RR case there are only two states: with and without the IP option record route. This is a very limited situation, but it appeared that some insight into router behaviour might still be gained from this approach.

If there is a segregating effect from the packet ID, even though the packet ID field is chosen to reduce the possibility of this, per-packet load balancing will usually be seen rather than pseudo per-flow load balancing. The possibility of choosing a segregating field as packet ID is a potential problem with this experiment. However we anticipate that it would be safe to use the field that has been used for packet identification when conventional per-flow analysis has been performed in previous research (ICMP: ICMP sequence number, UDP and TCP: IPID), as there is a history of sensible results that are unaffected by the degradation that would occur if this field caused extra segregation of traffic across load

Mode name	Pseudo flow ID	Packet ID
ICMP ID	ICMP ID	ICMP seq
ICMP seq	ICMP seq (BS)	IPID
ICMP IP Total Length	IP TotLen	ICMP seq
UDP IP Total Length	IP TotLen	UDP sum
UDP IPID	IPID (BS)	UDP sum
UDP IP HeadLen RR	IP HL	UDP sum
TCP seq	TCP seq (BS)	IPID
UDP checksum	UDP checksum (BS)	IPID

Table 10.1: Table of scamper modes for field analysis. In the IP HL mode the IP option used is Record route - RR. BS means that two modes exist, the second being a bit shifted version of the first.

balancers. This is true for most routers because we do not see the higher levels of per-packet load balancing that would result if many routers were affected. Per-packet load balancing can be detected by repeating a particular flow ID while changing the new packet ID. If the packet is not forwarded to the same successor, then the packet ID field is being used to load balance.

For the IP header length varied case using record route, an option length of 31 bytes was used allowing up to 7 addresses with three bytes for type, length and pointer. RFC791 states the following [53]: 'If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route.' It should be noted that the data contained in this option gradually changes as more addresses are added along the path. This option data is attached to the end of the IP header.

A scamper routine was written for each field and probe packet type that we intended to investigate. A scamper driver was written to run these fields analyses in two batches at different times to spread load over time. The same 70000 destination data files were used on each batch, but the destination addresses differed between vantage points. There were however some repeats of the same destination across vantage points. (See Fig. 10.1 to see characteristics of the data sets.) The driver also tested for the presence of pseudo per-flow load balancers and repeated these traces. Warts data was collected and a warts analysis program was

used to gather statistics about the segregation behaviour of these fields.

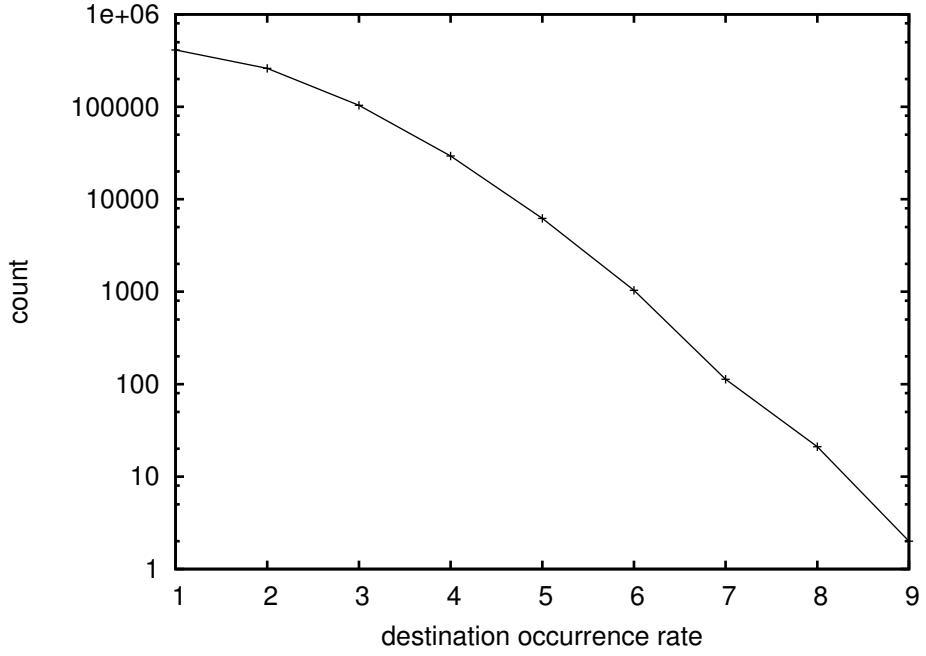

Figure 10.1: Repeated destinations across vantage points during data collections.

Steps were taken to avoid false positive results. Primarily this was the automatic repeat of traces where pseudo-per-flow load balancing behaviour was found. If a per-packet load balancer happened to segregate like a per-flow load balancer by chance in the first case, the same behaviour was extremely unlikely to be repeated. Furthermore, repeats of segregating behaviour on paths to the same destination from different vantage points were counted. This should make repeated chance positives less likely, as repetition is the key to detecting false positives.

10.4 Results and discussion

Initial results are shown in Table 10.2. The fields are the ones shown in Table 10.1 and their abbreviations are shown in Table 10.3. The columns of Table 10.2 are as follows:

- Column ‘ppk_tr’ is the number of traces that demonstrated per-packet load balancing.

- Column ‘pflow_tr’ is the number of traces that demonstrated pseudo-per-flow load balancing.
- Column ‘com’, or common, is the number of cases where the same node has been detected as pseudo per-flow and per-packet.
- Column ‘pflow_tr_rep’ is the number of cases repeated successfully by the algorithm when pseudo per-flow load balancers were found.
- Column ‘dest_pf_rep’ is the number of times that the same load balancer was found by multiple vantage points when probing to the same destination.
- Column ‘com_rep’ is the number of cases where the previous column is found to intersect with the per-packet set of load balancers.

The ‘pflow_tr’ column indicates where pseudo-per-flow load balancing was found in one test in a path, not including repeats, whereas ‘pflow_tr_rep’ ignores results where the repeat test failed. ‘com’ counts the paths where the positive result coincides with a previously seen per-packet load balancer. This value can be subtracted from ‘pflow_tr’ to give a more reliable set of positive results than ‘pflow_tr’. ‘dest_pf_rep’ results should be more reliable than ‘pflow_tr’ and usually more reliable than ‘pflow_tr_rep’ as they rely on repeated observation (many repeats) of pseudo-per-flow behaviour for a given load balancer. It is therefore surprising that the ‘com_rep’ column still finds these previously seen per-packet cases. ‘com_rep’ may be subtracted from ‘dest_pf_rep’ to give what should be a higher reliability set. The columns of Table 10.2 show a decrease in set size (apart from the record route case) for increasing repetition and removal of previously seen per-packet load balancers. ‘dest_pf_rep’ minus ‘com_rep’ shows similar results for each mode except record route. Because of the lack of clearly more frequently load balancing fields observed here and a suspicion that random events were still clouding the results, we conducted an involved analysis of hits and misses in the search for load balancers to better interpret the data collected in this experiment. The idea of this approach is that we were most interested in reliable and consistent responses from load balancers and

we were suspicious of load balancers detected on a few occasions out of many attempts.

field	ppk_tr	pflow_tr	com	pflow_tr_rep	dest_pf_rep	com_rep
icmpid	2741	360	59	262	148	28
(std err)	(619)	(65)	(4)	(47)	(25)	(3)
icmpseq	2737	329	60	244	126	29
(std err)	(619)	(62)	(4)	(45)	(22)	(3)
icmpseqbs	2737	334	59	242	128	29
(std err)	(618)	(64)	(4)	(45)	(24)	(3)
iplen	2734	337	59	240	126	29
(std err)	(620)	(64)	(4)	(44)	(23)	(3)
tcpseq	2733	341	59	252	134	29
(std err)	(620)	(67)	(4)	(46)	(24)	(3)
tcpseqbs	3345	432	62	386	202	32
(std err)	(627)	(66)	(4)	(77)	(39)	(3)
udpchk	3207	428	61	367	191	33
(std err)	(620)	(65)	(4)	(72)	(37)	(4)
udpchkbs	3205	426	61	370	193	33
(std err)	(621)	(63)	(4)	(74)	(37)	(4)
udpipid	3220	432	62	371	193	32
(std err)	(620)	(65)	(4)	(74)	(38)	(4)
udpipidbs	3216	417	61	374	194	32
(std err)	(621)	(63)	(4)	(76)	(38)	(4)
udplen	3203	421	62	373	193	32
(std err)	(620)	(64)	(4)	(76)	(39)	(4)
udpiphlrr	21902	19500	441	*	31733	352
(std err)	(2741)	(3438)	(49)	(*)	(4176)	(42)

Table 10.2: Table of analysis of non five-tuple fields and cases of load balancing by these. Standard error of the mean is shown in parentheses.

In the UDP IP HeadLength RR case load balancers are often found to be per-packet or pseudo per-flow because of the limited number of flow IDs. Many load balancers that we know not to be per-packet are detected as such by the record route method. Record route is a difficult field to analyse.

Fig. 10.2 is a graph of misses and hits for each field analysis. A miss occurs when a node has been seen as a pseudo per-flow load balancer in one trace but does not qualify as a load balancer in another trace (*i.e.* to another destination or from another vantage point). A miss could include

Mode abbreviation	mode description
icmpid	ICMP ID
icmpseq	ICMP sequence number
icmpseqbs	ICMP sequence number bit shifted
iplen	ICMP IP total length
tcpseq	TCP sequence number
tcpseqbs	TCP sequence number bit shifted
udpchk	UDP checksum
udpchkbs	UDP checksum bit shifted
udpipid	UDP IP ID
udpipidbs	UDP IP ID bit shifted
udplen	UDP IP total length
udpiphlrr	UDP IP header length record route

Table 10.3: Table of scamper mode abbreviations.

a case where the node is traversed but in a different direction to the load balancer. A hit is when a particular node is detected as a pseudo per-flow load balancer. A hit means that a particular mode has been detected as finding pseudo load balancing behaviour. The 'lim' or limited cases (limmiss or limhits) exclude cases where the node has behaved previously as a per-packet load balancer. Once again the UDP IP HeadLength RR case is unusual and is consistent with finding many load balancers. This case is hard to interpret because there are only two flow IDs in the analysis *i.e.* with and without the IP option. Because of this the algorithm seems to find common load balancers, probably per-flow, as both per-packet and pseudo-per-flow load balancers.

One concern is that if there are many more misses than hits, the few hits that are observed may be some kind of statistical aberration or noise. Most of the analyses show a similar pattern in this regard besides the UDP IP HeadLength RR case. The one exception is ICMP ID but the increase over the other cases in limited hits (though statistically significant) is only small compared to the variation between the values for the other fields.

Text Table 10.4.1 shows pseudo per-flow load balancers which were found twice (successfully repeated) in the same destination and trace set and then found in another trace set. The bold text at the beginning of

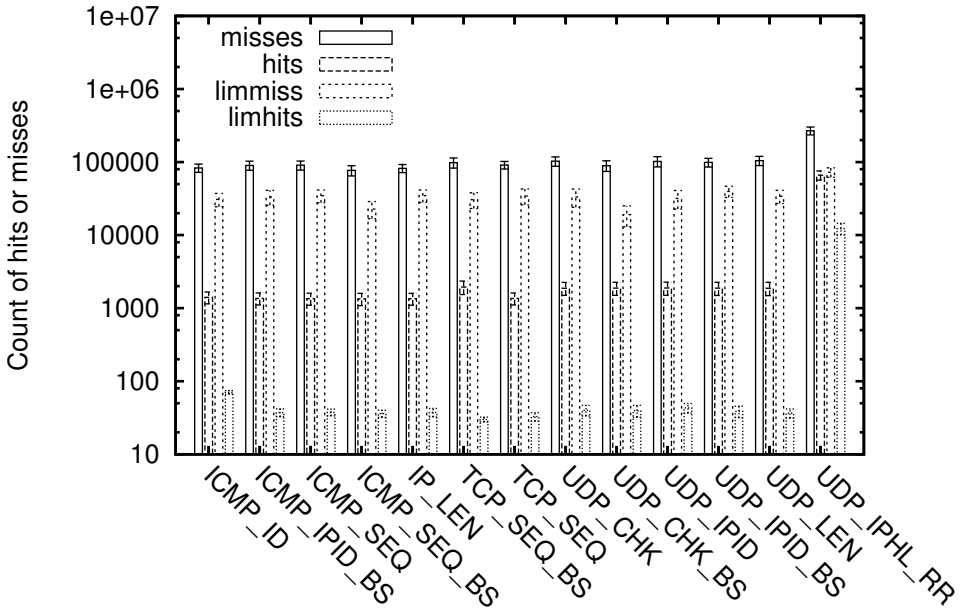

Figure 10.2: Hits and misses observed for each evaluated field. Error bars are standard error of the mean.

each section shows the mode being evaluated and the parentheses show the full count of replicated pseudo per-flow load balancers for that mode. Because there is a small probability of this occurring by chance in a per-packet load balancer, all cases of per-packet load balancers across all trace sets, except udpiphlrr, were subtracted from these sets and the results are shown in Text Table 10.4.2. The numbers are greatly reduced but one case still stands out. Udpipid has 90% of cases in common with udpipidbs (447). A further three modes share a few replicated cases with these two: udpchkbs (494), udpchk (510) and udplen (505). This suggests that there may be a very small number of routers that respond to these non classic five tuple fields when segregating traffic between load balancer successors.

icmpid (239) icmpseq 167 tcpseq 169 udpipidbs 159 udpchkbs 160 udpiphlrr 34 udplen 164 udpipid 165 tcpseqbs 157 udpchk 162 iplen 172 icmpseqbs 171

icmpseq (217) tcpseq 166 udpipidbs 164 udpchkbs 164 udpiphlrr 34 udplen 168 udpipid 165 tcpseqbs 156 udpchk 163 iplen 172 icmpseqbs 173

tcpseq (219) udpipidbs 164 udpchkbs 160 udpiphlrr 35 udplen 166 udpipid 168 tcpseqbs 161 udpchk 165 iplen 170 icmpseqbs 169

udpipidbs (230) udpchkbs 183 udpiphlrr 35 udplen 183 udpipid 187 tcpseqbs 168 udpchk 175 iplen 166 icmpseqbs 166

udpchkbs (251) udpiphlrr 38 udplen 186 udpipid 188 tcpseqbs 173 udpchk 182 iplen 163 icmpseqbs 161

udpiphlrr (1803) udplen 37 udpipid 35 tcpseqbs 34 udpchk 39 iplen 32 icmpseqbs 35

udplen (230) udpipid 186 tcpseqbs 174 udpchk 178 iplen 171 icmpseqbs 171

udpipid (252) tcpseqbs 175 udpchk 182 iplen 173 icmpseqbs 168

tcpseqbs (222) udpchk 167 iplen 164 icmpseqbs 160

udpchk (222) iplen 164 icmpseqbs 162

iplen (216) icmpseqbs 174

icmpseqbs (220)

Text Table 10.4.1: Number of pseudo per-flow load balancers found using multiple pseudo flow ID fields.

icmpid (12) icmpseq 0 tcpseq 0 udpipidbs 0 udpchkbs 0 udplen 0 udpiphlrr 0 udpipid 0 tcpseqbs 0 udpchk 0 iplen 0 icmpseqbs 0

icmpseq (5) tcpseq 1 udpipidbs 1 udpchkbs 1 udplen 1 udpiphlrr 0 udpipid 1 tcpseqbs 1 udpchk 1 iplen 2 icmpseqbs 4

tcpseq (4) udpipidbs 0 udpchkbs 0 udplen 0 udpiphlrr 0 udpipid 0 tcpseqbs 0 udpchk 0 iplen 0 icmpseqbs 1

udpipidbs (12) udpchkbs 4 udplen 4 udpiphlrr 2 udpipid 9 tcpseqbs 1 udpchk 3 iplen 1 icmpseqbs 1

udpchkbs (13) udplen 5 udpiphlrr 2 udpipid 5 tcpseqbs 1 udpchk 4 iplen 1 icmpseqbs 1

udplen (7) udpiphlrr 2 udpipid 4 tcpseqbs 1 udpchk 4 iplen 2 icmpseqbs 1

udpiphlrr (1184) udpipid 2 tcpseqbs 0 udpchk 2 iplen 0 icmpseqbs 0

udpipid (15) tcpseqbs 1 udpchk 3 iplen 1 icmpseqbs 1

tcpseqbs (7) udpchk 1 iplen 1 icmpseqbs 1

udpchk (8) iplen 1 icmpseqbs 1

iplen (4) icmpseqbs 2

icmpseqbs (10)

Text Table 10.4.2: Number of pseudo per-flow load balancers found using multiple pseudo flow ID fields, with known per-packet load balancers removed.

Fig. 10.3 shows counts of cases with a particular number of pseudo per-flow hits (non per-packet or limited) where no misses occurred. All the cases listed have zero misses and so are likely to be reliable positives, especially if there is a reasonable number of pseudo per-flow hits. Two hits was common, however there were cases where there was a larger number of hits counted. These are cases where there is a consistent pseudo per-flow response from a load balancer.

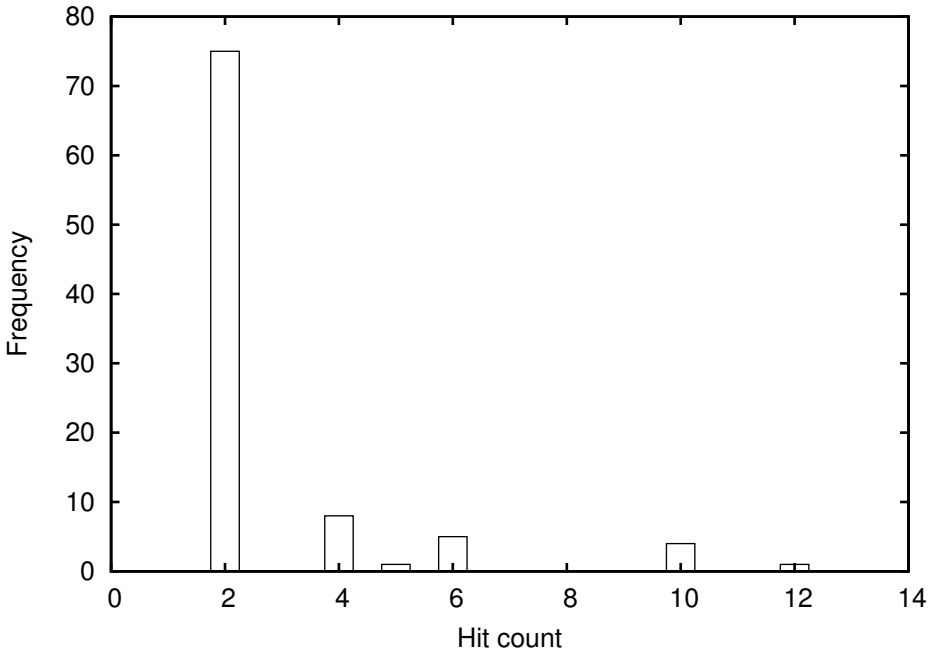

Figure 10.3: Distribution of pseudo per-flow hits where there were zero misses. Axes are the number of hits versus the frequency of that rate of hits.

Table 10.4 shows the number of combinations of total non per-packet hits and vantage points, where there were zero misses. The four instances of ten ‘limhits’ and the one instance of twelve all relate to the same pseudo per-flow load balancer under different analysis modes. Similarly there is another pseudo per-flow load balancer that has zero misses across eight fields analysis modes with smaller numbers of ‘limhits’ in each mode but a total of 35 ‘limhits’.

Freq.	limhits	VPs
72	2	1
1	4	1
7	4	2
1	5	2
1	6	2
4	6	3
4	10	5
1	12	6

Table 10.4: Table of frequency of combinations of total non per-packet hits and the number of vantage points that observed the hits.

These cases of a consistent load balancing response, with zero misses in vantage points where hits were found, more strongly suggest that there may be a small population of routers that forward traffic in response to variations in fields other than the classic five-tuple. The fields most commonly involved in multiple load balancing responses from routers are UDP_IPID, UDP_IPID_BS, UDP_CHK, UDP_CHK_BS, UDP_LEN, ICMP_SEQ_BS and IP_LEN.

Chapter 11

Detection of black holes in load balancers

11.1 Introduction

This chapter presents the results of a study into the prevalence of discontinuities (*i.e.* black holes) in load balancers.

One of the questions that arises out of doing research on load balancers and the application of systems like ATLAS is: Do discontinuities occur in load balancers and if they do are they sufficiently common that systems like ATLAS should try to detect them? Routing protocols are designed to respond to situations where routes become unavailable and find alternatives at low cost, but how long can this response take and are there any cases where the response fails to fix the discontinuity? The focus of this experiment is on finding examples of discontinuities inside load balancing diamonds between the divergence and convergence points.

11.2 Related work

11.2.1 Hubble

A logical extension of the use of algorithms to map the Internet is to use them to find localised points of discontinuity or black holes, because the mapping probe traffic will stop at such a place and there will be no ICMP time exceeded message returned to the probe. Katz-Bassett *et al.*

explored this topic in 2009 with a system called Hubble [35]. Hubble used regular pings (every 2 minutes) to a large set of addresses and noted when a previously responsive ping failed. After several failures this then triggered Traceroute analysis to the area in question to localise and confirm the problem.

11.2.2 Lifeguard

Lifeguard [36] is a successor to Hubble that also uses regular pings to detect and isolate a black hole. Lifeguard then uses historical measurements to identify problem candidate locations in the Internet. Lifeguard uses techniques based on reverse traceroute to determine the direction of failure. Lifeguard then homes in on the location of the problem by identifying routers in the area that are working. Once the problem is located, BGP route 'poisoning' is used to divert traffic around the problem AS.

11.2.3 Load Balancers

Both Hubble and Lifeguard aim to find black holes in the Internet as a whole. Because of our interest in load balancers specifically and because no existing research has attempted to do so, we have investigated the prevalence and properties of black holes in load balancers.

We aim to investigate if black holes in load balancers are extremely short lived as expected, according to IGP adaptation and route table reconvergence algorithms, if it is detectable at all. We note that black holes found by Lifeguard are not always short lived and that IGP adaptation and dynamic routing would have been expected to prevent long lived black holes. We also aim to determine the lifespan and frequency of black holes in load balancers, if they can be found at all.

11.3 Experimental design and data collection

From 20 vantage points on PlanetLab scamper was run with two drivers. Driver one ran scamper 'tracelb'[1] MDA analysis once initially and then 8 cycles of scamper 'trace' Paris Traceroute without MDA one each hour, followed by a final repeat of MDA analysis. Each non MDA Paris run had a randomly chosen flow ID. If a non MDA trace was found to stop short of the end of the trace based on the results of the initial MDA run, driver one sent the destination address and flow ID information to driver two. The first MDA analysis was used for comparison with non MDA runs and the later MDA runs were used to confirm that the topology had not changed by comparing them with the first run.

Driver two waited for targets to be provided by the first driver and then ran 25 cycles of scamper 'trace' (the MDA mode of 'trace' is turned off), one each hour, followed by a final scamper 'tracelb' MDA analysis. Because scamper 'trace' probed each hop once and subsequent runs were an hour apart for a given destination, there would have had to have been a very large number of targets before processing delays affected the time before MDA analysis in driver two began. If a hop failed to respond two further attempts were made. When determining when to stop a trace on finding a non responsive hop, two more hops were checked before testing the destination for a response and ending the trace. If a non MDA trace run by driver two stopped before the point that the initial MDA trace reached, then the trace was found to be short in a secondary analysis performed once all the data was collected.

Driver one could probe at up to 200 PPS and driver two 100 PPS. However, the drivers did not always probe at these rates due to variations in the time taken for the targets to become available and the driver two probing cycle to begin. Contributing to this was variation in the initial MDA analysis carried out by driver one and variation in the time that discontinuities actually occurred or were seen. The address lists were limited to 20000 to avoid delays in the hourly cycle both in driver one and two. Correct timing of non MDA Paris Traceroute traces was later

[1] 'tracelb' and 'trace' are syntaxes of modes used to run scamper, where lb means load balancer.

confirmed by checking the timestamps of the traces.

Three runs were performed on Planetlab, over the course of two weeks per run although most vantage points had finished their analysis well before this time. Data sets of 20000 addresses were randomly selected from the CAIDA MIDAR data set and the address selection was repeated for each new run to give different address sets each time. After the first run, the number of vantage points was increased. This was because Planetlab vantage points often become unavailable and so a larger starting set was preferable.

11.4 Results and discussion

Table 11.1 shows the dates on which black hole in load balancer detection was initiated from Planetlab. It also shows the number of vantage points used for the collection of the data.

Run No	Date	VPs
run 1	2015-02-12	7
run 2	2015-02-28	21
run 3	2015-04-29	14

Table 11.1: Table of black hole data collection dates and vantage point numbers.

To ensure that there were no delays between non MDA Paris Traceroute runs due to the maximum traffic limit being reached, we compared trace timestamp information. The actual time intervals between the Paris Traceroute non MDA analyses for the runs was determined. We observed quite precise timing of the trace data collections, especially for the targeted analyses. For driver one 95% of traces were started within a minute of one hour from the previous trace. The longest delay observed was eight minutes and this occurred very infrequently. The performance for driver two was better than this as it had fewer traces to run in comparison to driver one. This is interpreted as validation that the experiment operated correctly and that significant delays due to maximum traffic limits did not occur.

Table 11.2 shows the mean and standard error for the number of targeted traces, total detections of load balancers including repeats and total occurrence of unique load balancers within vantage points. These statistics are calculated on a per VP basis. Many load balancers are repeatedly analysed, so it would be possible to detect the same black hole within a load balancer more than once if that load balancer was one that was repeatedly analysed. Table 11.3 shows the uniquely detected load balancers within each run. Run 2 had more repetition across VPs. This information is intended to help define the context of the results on how many black holes have been detected. In particular, it is useful to know how many unique load balancers have been analysed compared to the number of black holes found.

	Mean	Std error
Number of triggered analysis sequences	1346	322
Total detected occurrences of load balancers	17960	1164
Total occurrences of load balancers unique within VPs	909	39

Table 11.2: Black hole data collection statistics on a per vantage point basis.

Run No	Number LBs
run 1	8354
run 2	4474
run 3	5550

Table 11.3: Uniquely discovered load balancers during each run.

Table 11.4 shows the discontinuity results for the three runs across all of the VPs included. It shows all instances of a repeating trace destination where short non MDA Paris traces were found. The table represents all of the cases across the destinations probed where a black hole was found to occur. The lines of data include the number of short traces ending in a load balancer within a set of non MDA Paris traces compared to the initial MDA analysis, the number of occurrences of this number of short traces and the number of gaps among the traces in this count. The gap counts the number of complete traces between the first and last

short trace. This indicates intermittency of the discontinuity found. The trace count of data line 2 in run 1 indicates adjacent traces, or a discontinuity lasting about two hours. For the next two destinations (data lines 3 and 4), one showed intermittency and for the other the discontinuity was continuous the entire time. For runs 2 and 3 the observed discontinuities were mostly intermittent with varying discontinuity lifetimes within the day long range of this analysis.

Twenty black holes being detected in 18378 load balancers tested is proportionally low (0.1%), especially considering that the same black hole could be seen more than once. 75% of the black holes in load balancers existed for 6 hours or longer. When approximately compared with the strike rate of Hubble these results appear to be low: 1500 black holes per day using 2000000 ping destinations implies 0.0007 black holes per address per day for Hubble, compared with 20x3=60 black holes per day using 42x20000=840000 addresses implies 0.0000714 black holes per address per day, approximately 10% of the amount of Hubble for black holes in load balancers.

This technique relies on virtual paths not fluctuating frequently, however if fluctuations are found they are likely to be temporary discontinuities as the technique relies on finding short paths and checks MDA traces against each other for validation. To improve the reliability of this approach it would be desirable to monitor black hole results in real time and carry out targeted Traceroute analysis from several vantage points to confirm the discovery of a black hole.

Run No	Results
run 1	short traces 1 count 1 gaps 0
run 1	short traces 2 count 1 gaps 0
run 1	short traces 21 count 1 gaps 4
run 1	short traces 25 count 3 gaps 0
run 2	short traces 1 count 1 gaps 0
run 2	short traces 5 count 1 gaps 12
run 2	short traces 6 count 1 gaps 18
run 2	short traces 8 count 1 gaps 14
run 2	short traces 9 count 1 gaps 11
run 2	short traces 12 count 1 gaps 12
run 2	short traces 21 count 3 gaps 11
run 2	short traces 22 count 3 gaps 4
run 2	short traces 23 count 3 gaps 6
run 2	short traces 24 count 2 gaps 2
run 2	short traces 25 count 9 gaps 0
run 3	short traces 1 count 1 gaps 0
run 3	short traces 8 count 1 gaps 10
run 3	short traces 16 count 2 gaps 18
run 3	short traces 24 count 2 gaps 2
run 3	short traces 25 count 9 gaps 0

Table 11.4: Table of black hole data collection results. ‘Count’ is the number of short traces ending in a load balancer. ‘Gaps’ counts the number of complete traces between the first and last short trace.

Chapter 12

Efficiency of finding load balancer successors under three different port selection modes

12.1 Introduction

Load balancer discovery is expensive in terms of probe packet traffic and so is not usually considered feasible for regular Internet discovery. One approach we considered to improve Traceroute efficiency was to choose flow IDs that more efficiently access all successors of a load balancer. To be successful this would require a means to cause the built in hash function used by the load balancer successor segregation algorithm to step through the set of successors for the majority of load balancers. This was likely to be difficult as different router vendors, models and even router model software versions can use different hash functions, but it would still be useful as the possible benefits of success justify the effort.

12.2 Related work

In another paper Cao *et al.* proposed a number of hashing options [15] and investigated their efficiency at distributing flows across equal cost paths. The type of approach actually used in routers could give insight into the best and most efficient way to choose flow IDs.

12.3 Experimental design and data collection

The three methods that we used vary the source port in UDP packets. The first method is to increment the source port in the top half of the two byte range, where the process ID is used to set the start value[1]. The second method is bit flipping which stemmed from the belief that some router hashing functions involved in load balancing might be based on values of individual bits in the source port value. In the bit flipping method the process ID based starting value is used and then based on a number counting up from zero that controls flow ID, bits in this number are flipped in sequence from minor through major. When the flow ID counter goes above 16 another bit is flipped based on how many multiples of 16 the counter contains. The third method is random selection of source port across the whole non zero range of the port.

First the two new modes were added to scamper then a scamper driver was written to perform the three analysis modes consecutively. The data collection was run using the CAIDA facilities from 12 October 2013 for a week, using a joint confidence of 99%. Seventeen vantage points were used to collect data using 70000 randomly selected addresses from the CAIDA MIDAR set.

12.4 Results and discussion

The two metrics used in this analysis are probe count for a given number of successors found and load balancer count accumulated at a given successor count. The probe count metric measures the efficiency of finding successors in terms of probes sent and the second is the efficiency of finding load balancers given a sequence of successor counts.

Fig 12.1 shows that the native source port incrementing mode found successors using the fewest probes. The best comparison for this test was found by comparing load balancers with at least eight successors. This was because there was a full set of results for each load balancer (*i.e.* each load balancer tested contributed data for successor counts 2-8) and this provided a balanced comparison where each point on the graph

[1]The is the method used in Scamper by default.

contained data from the same set of load balancers. Using an unbalanced comparison by using the load balancers with fewer successors resulted in a lot of noise in the results.

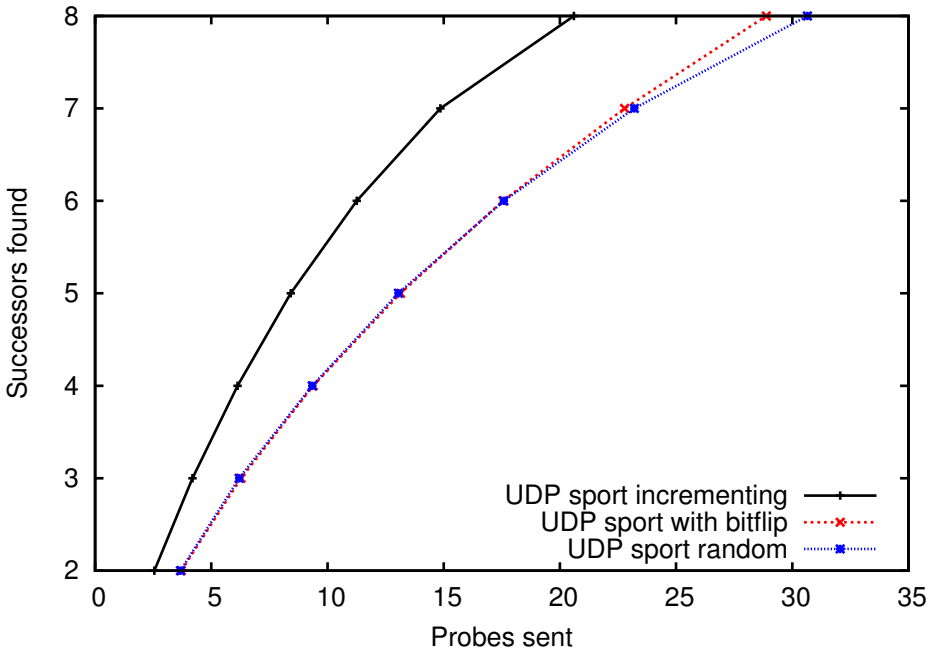

Figure 12.1: Average probes sent and successors found for different source port assignment modes. LBs were analysed only once. Data used contained at least 8 successors.

Fig 12.2 shows the accumulation of unique load balancers with increasing successor count for the three modes tested. The accumulation is the set count of load balancers with successor counts ranging from 2 up to the current value on the X axis. This is a bit like a CDF analysis. Scamper MDA gives lists of load balancer successors, so the scamper warts analysis program reports cases of two or more successors, once for each load balancer interface IP address. The random source port mode found more load balancers than the other two modes. The bit flipping mode gave similar results to the random mode for a smaller numbers of successors and then found the smallest number of load balancers. The native incrementing mode found less load balancers at the lower successor counts and then for larger numbers of successors found a number of load balancers between the other two modes.

This graph does not count probe traffic and so focusses on load balancers discovered with a given number of successors instead. The ability to discover a load balancer is different from measuring the amount of traffic to discover each successor once the load balancer is detected and so different parameters are quantified by each analysis. We see that native incrementing mode in the first case is best and most efficient user of traffic and in the second case is second best discoverer of load balancers. However random is also a contender, as it is best at coverage and close to second in probes sent.

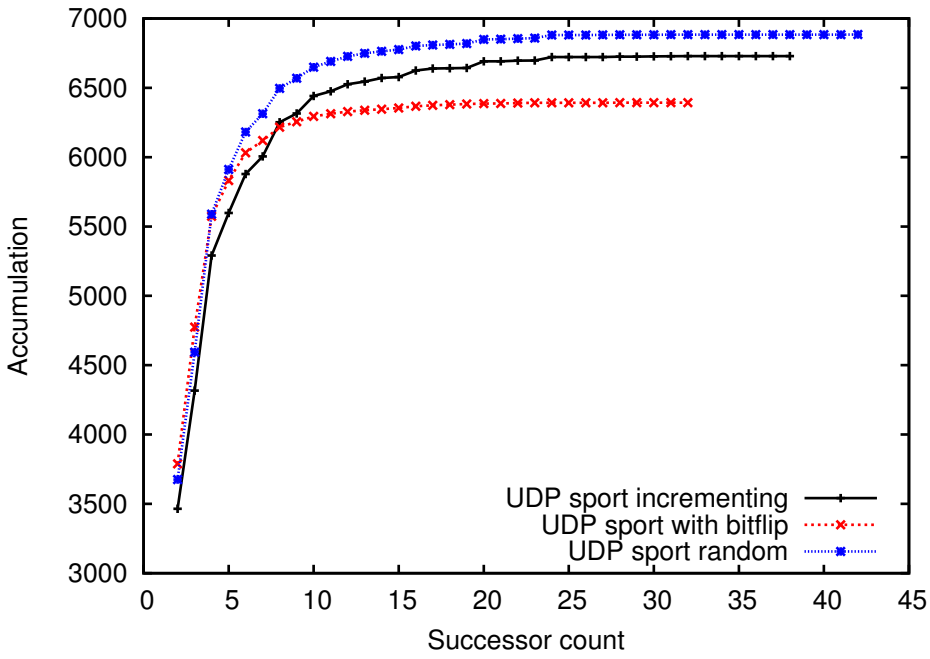

Figure 12.2: Successor count vs. load balancer interface accumulation for different source port assignment modes.

Fig 12.3 shows the distribution of total probe counts used by traces in the three modes and the fitted native scamper source port incrementing mode. The graph is a Cumulative Distribution Function (CDF). No obvious differences are seen, however the shared shape is of interest. It shows that half of the traces required 250 probes or less and 82% of the traces used less than 1000 probes. In cases where there is complex load balancing the probe counts greatly increase beyond these levels. This causes the distribution to have a long tail which however stops at 65535

probes as this is the standard maximum limit for scamper Traceroute MDA. The fitted curve is based on Pareto CDF, however it uses a first constant that varies slightly from one, as shown in Equation 12.1.

$$cdfprobability = 1.01446 - (99.2006/probes)^{0.68754} \tag{12.1}$$

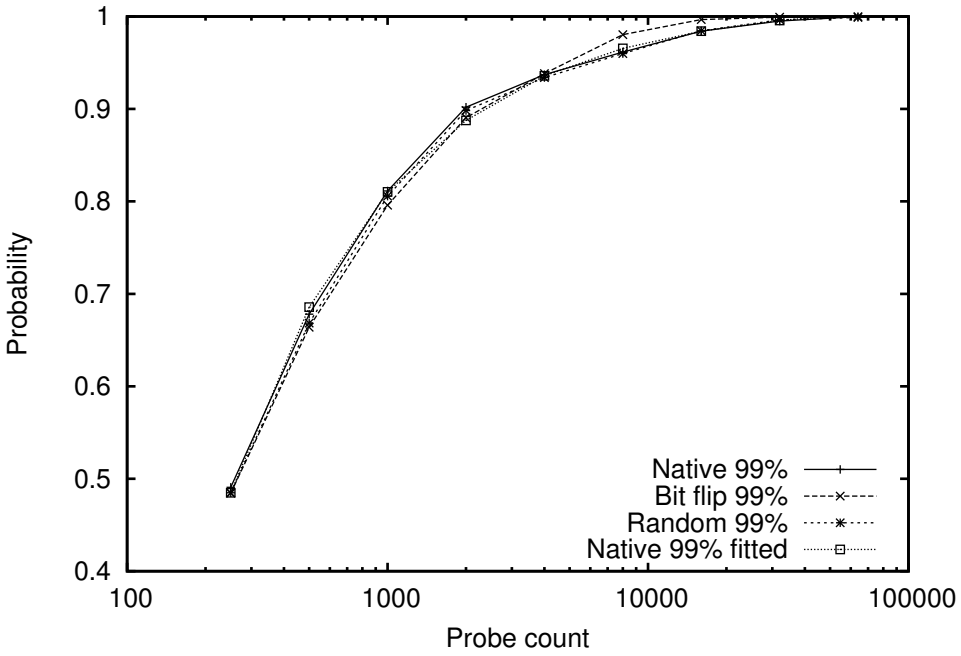

Figure 12.3: CDF of total trace probe count. Data is from UDP probing in all three modes tested. Fitted scamper native source port incrementing mode is also plotted.

The goal was to use traffic more efficiently, which suggests that the native incrementing mode is best, however finding all discoverable load balancers is also important. Native incrementing mode was not far behind the best case in terms of load balancer discovery. As the native incrementing mode was the first method used during data collection it is not likely to have been affected by previous traffic from one of our vantage points through the same path reaching a traffic limit (possibly a limit on the amount of ICMP TTL expired packets allowed).

The native incrementing mode is the most efficient in terms of probe cost but is less capable of getting complete load balancer coverage. If

full coverage is desired, random mode is a better choice, but this method will introduce a greater probe cost.

Part III

Simulation

Chapter 13

Doubletree using event based simulator IS0

13.1 Introduction

Doubletree is a derivative of Traceroute that is designed to reduce probing where there are repeated pieces of topology, such as near probing vantage points and also near repeated destinations. Doubletree and its ability to perform in systems like Atlas have been analysed previously by others[46] [47] using simulation, however the costs incurred by the extra communication of Doubletree between VPs were not evaluated. It would be desirable to further adapt Doubletree to the many sources scenario and perform full cost analysis, as approaches like Doubletree show potential to lead to more efficient topology discovery systems.

Doubletree reduces link topology rediscovery near sources and destinations, as discussed in Sections 2.3.1 and 3.4. But what is the cost of transmitting the global stop set when there are large numbers of vantage points like Atlas will have? Can steps be taken to reduce the amount of control traffic required by Doubletree. In this research, sources windows have been designed to help make Doubletree feasible when there are large numbers of vantage points. These efficiency measures are described later in this section.

The simulator used by McGregor[46] [47] was called IS0. The discreet events used in IS0 primarily are based on sending packets to certain hops in a path and receiving response packets from those hops. Information

about previously seen nodes from simulating these Traceroute analyses is stored in local and global stop sets. Sharing global stop sets between VPs is the origin of the control traffic cost. The detailed information needed to perform simulation is obtained from Traceroute analyses, in this case the warts data was publicly provided by CAIDA.

In this experiment 'few sources' refers to the vantage points that the data was collected from. 'Many sources' to 'few destinations' is the reverse direction analysis of data from the actual destinations in the data, back to the vantage points, which become the few destinations. The phrases 'few to many' and 'many to few' are used to indicate few sources to many destinations and many sources to few destinations, respectively.

The discreet event based Internet simulator of McGregor [46] [47] IS0 was used to perform cost analysis in some previously used Doubletree and Traceroute simulation modes, as well as being adapted to implement sources window analysis in the many sources to few destinations direction of analysis. This was of interest because of our desire to understand how to efficiently make use of a system with 100000 vantage points, such as Atlas may approach.

13.1.1 Computer simulation

A computer simulation is based on the construction of an abstract model of a system under study, in our case the Internet, and running a simulation based on this model on a computer. In order to create an abstraction some assumptions may be necessary about the usual behaviour of the system under study, including some approximations to reduce the size of the model. The model includes a process interacting with the system and in our case we are simulating Traceroute and Doubletree running on the Internet or a subset of the Internet. For example one of the characteristics of Traceroute and Doubletree is they provide information at the interface level rather than the router level. Here we make the assumption that knowing many paths of interfaces provides sufficient information about routes traversed in the Internet to learn about the usefulness of Traceroute and Doubletree when used to analyse the Internet. We also assume that these paths can be used in the reverse direction without ac-

tually collecting data about reverse paths. The tools at our disposal do not allow us to easily analyse reverse paths, however highly distributed systems like Atlas promise this possibility in the future.

13.1.2 Simulation variability

All topology discovery simulations presented in this thesis gain the model of the Internet from data sets collected using Traceroute or a derivative of Traceroute. This means that the simulations are only deterministic for a given Internet structure *i.e.* for a specific Traceroute data set. It seems that there are two approaches to dealing with this. One approach is to carry out repeated analyses across multiple data sets and average the results. The other approach is to use a large data set for a single analysis and achieve an averaging effect that way. The latter approach is not unprecedented [47], and it has been used in this research because of simplicity and because of the way IS0 has been designed. IS0 is intended to be a step 'towards Internet scale simulation' [46] as opposed to a sample.

13.1.3 IS0

In 2011 McGregor introduced IS0 an Internet Traceroute and Doubletree simulator [46] [47]. In this work it was demonstrated that redundant interface discovery rates can be optimised by carefully choosing the initial TTL and that Doubletree gives useful savings in probe traffic. McGregor also noted that costs in terms of communicating the global stop set should be determined using simulation. If Doubletree were found to be sufficiently efficient, it may be more likely to be adopted for routine topology discovery. Also of interest was the detail of how a system with many vantage points might function optimally as initial analyses of cost appeared to give excessive estimates for a standard approach as used by McGregor in native IS0.

IS0 topology

IS0 stores a topology memory map file that stores real world topology data in hash tables. The branching nature of the Internet is preserved, but with emphasis on Interfaces rather than routers. In particular routing information is stored in addition to nodes and links, using tables of next hops to given destinations for each node. In addition, in the absence of reverse path information, symmetric reverse paths are used.

IS0 ignores paths that contain loops or contain too many non responding nodes. Alternative paths resulting from route changes during data collection or load balancing are maintained in the model. Alternate paths remain associated with the source and destination combinations that discovered them. As a result source data must be stored with next hop data to maintain alternate paths.

The amount of data used by the simulator must be manageable and represent the Internet. Using the full data set (trace set) made the simulator run for very long periods of time *i.e.* more than three weeks for one simulator factor set. The simulations select end points that are balanced for the AS that they come from. Originally one destination per AS was programmed, but this was modified to allow more destinations per AS. The process of trace selection counts the number of addresses included from each AS and limits the number included to 20 in each case. Selecting data across ASes is likely to help maintain representativeness of the data set. Trace selection uses RIPE NCC Routing Information Service (RIS) data containing route dumps that have had AS and IP address range information extracted. Using these mitigations the simulator ran for several days for each run, thus making sets of runs with varied parameters feasible.

A Doubletree run needs a controller node which is used to receive and send shared global stop set information from and to vantage points. In particular, paths for sending data to and from the controller node to vantage points need to be created[1]. This is done using a breadth first search algorithm to search the available Internet topology from the source to

[1] Because the topology is derived from Traceroutes rather than Doubletree, not all of the required paths between VPs and the controller may exist.

the destination. The search provides information that allows key links to be added, including reverse links, to allow traffic between the provided source and destination nodes. A breadth first search is more likely to make a path that crosses existing paths to lead to the destination, rather than following them.

IS0 is event driven where various hooks are triggered as a trace is simulated in Doubletree. This includes dispatching a packet and forwarding it from hop to hop until the TTL expires. There are 25 API event hooks and 6 of these relate to packet events: newPacketHook, packetQueuedHook, packetArrivedHook, packetDropHook, ttlExpiredHook and changePacketTypeHook. Other categories of events include 'simulation start and termination', 'hash management' and 'reporting'. Though these events take place in their proper sequence, they are not usually recorded for a large run as this consumes a lot of disk space. When debugging is necessary small scale runs can be performed with debugging and event collection turned on.

IS0 data files

Firstly the perl script get-topo.pl is run on zipped Traceroute warts files to produce files called hops-dests and leaves. Understanding the overall structure of these files helps one to understand the function of the simulator and how it stores the topology information in memory that is used for simulation during a run.

Hops-dests file Each line in the hops-dests file is preceded by a single character to indicate the type of entity described in the line. The file always begins with a single "L" line which lists the total number of nodes, sources and destinations described in the file. A node is described by a line starting with "N" which contains node address and latency, followed by a series of "D" and "S" lines. A "D" prefixed line contains destination address, next hop address, serialisation rate and trace flag. An "S" prefixed line contains source address, destination address, next-hop, serialisation-rate and trace-flag. The "S" lines differ from "D" in that they associate a next hop on the way to a given destination with a particular

source address, thus making the next hop dependent on packet origin. This approach results in a more faithful representation of the behaviour of the Internet.

Leaves file Each line in the leaves file is preceded by a single character to indicate the type of entity described in the line. The file always begins with a single "N" line which lists the total number of sources, destinations and ASes described in the file. After this there is a "C" line followed by lists of "S" then "D" lines in the quantities indicated in the "N" line. The "C" line contains the controller address. "S" prefixed lines contain a source address, the AS number of the source, the number of destinations analysed and then a list of these destination addresses. Finally the "D" prefixed lines contain a destination address, a destination AS, the number of sources that analyse this destination and the list of sources.

Trace list files The perl script make-tr-list.pl is used to generate trace lists from the hops-dests and leaves data. The script takes command line arguments for direction-type, timing-type and probing-type. For my analysis, there were two more added to this list: sources-windows-type and destinations-per-AS. This file contains a single "N" prefixed line which counts the number traces in the file. The controller address is "C" prefixed. The rest of the lines are numbered by trace ID (the running number assigned to a given trace), start time, source address, destination address and trace-flag. If the trace-flag is set, packet events will be recorded in the trace file produced by the simulator.

IS0 simulation runs

A simulation run begins by creating the memory map (hash tables of topology data from the hops-dests file and the leaves file) if it is older than the binary simulator file, older than input data files or non existent. This memory map of topology information is loaded into memory and used to route packets along the paths specified in the trace lists in either

Traceroute or Doubletree mode. The creation of the memory map also includes routes for Doubletree control packets. These routes for control packets are created using a breadth first search to create paths to and from the controller IP address. For each run the simulator will create a variety of output files that describe the results of the simulation. The output files are described below:

Log files The .log file contains a list of parameter settings from the simulation program. Counts of completed probers and times of packets in flight are listed. A summary is included which counts the number of traces and then counts the various packet types in the simulation. These packet types include probe counts along with counts of packets to and from the controller node. There is also a total packet count.

Stats files The .stats file describes the behaviour of each individual node over the course of the simulation. Relevant statistics included in the file are numbers of times nodes are discovered and profiles of numbers of nodes found at a range of hop counts. There is also a summary listing the numbers of stop set nodes, stop set hits, total packets transmitted and total non-controller packets transmitted.

Progress files The .progress file contains the following fields on each line, which is written every 10 seconds during the simulation: secondsElapsed in the execution of the simulator, simTime (stage of program in artificial time units), events (number of events so far), packets in flight of various types (control or probe), number of active probers, total number of hops taken by probe packets, total hops taken by stop set information packets and memory in use.

Trace files The .trace file contains information about the Traceroute analyses. Simulation time and packet number are included along with a description of the stage that the given Traceroute analysis is at. Start and end of analysis are reported along with information on each hop that probing packets reach.

IS0 validation

Two routines for validating IS0 are provided: check-paths.pl and check-links.pl. Each .trace file available is checked to ensure that the data in the .trace files matches the data in 'hop-dests' file.

Check paths Firstly information about the hops leading to many destinations is loaded from 'hops-dests'. Next counting of hash keys is performed made up of hop node or node where TTL expired, each associated with a particular source and destination address. The number associated with the key indicates if an expected case was not found or an unexpected case was found, when the data is compared to trace data. Some expected errors resulted from limiting the AS count to 20 where an expected key was not discovered.

The trace flag in the trace list files is usually set, for a small group of traces as the analysis can take a long time to run with larger samples. A sample subset is satisfactory for validation as the algorithm for processing traces is still tested in this scenario, otherwise similarly all traces of a smaller trace set simulation can be analysed by setting a global data record flag in a reduced rerun to achieve a similar result.

Check links IS0 has a compiler flag that can be used to enable validation of link queue sequencing.

To perform this validation the 'hops-dests' file is used as a source of link and timing information. For each line of each .trace file the results of the simulator are checked to ensure that each link is not queued for and left at the same time and that the time spent in queues is sufficient according to the input file. Included with timing is a check for the correct sequence of links by comparing the data in the input file with the .trace set.

13.2 Experimental design and data collection

The topology data used by IS0 needs to correspond with the real Internet as much as possible in order to produce valid results. In particular the

relative size of the data set compared to the Internet is important. Team Traceroute data collected by CAIDA is used [16] for this purpose. One full team includes a destination from each IPv4 /24 across the Internet. We were unable to use a full team because of the RAM capacity of the machines that we had available for running the simulations, so we use about one third of a team set. There were three days over which the team set was collected, so our data was taken from the first day. CAIDA state that 'The current list of routed IPv4 prefixes was created using RouteViews BGP tables' and that 'we dynamically pick a new random address in each /24 prefix for every new cycle of probing'. This gives maximum chance of producing a representative sample. Using this technique it is also likely that the scale is sufficient to give adequate coverage of the core of the Internet.

Part of a Traceroute team data set taken from 23 vantage points collected by CAIDA on 17/6/2013 were used to simulate Doubletree. This data set was downloaded from their data repository and consisted of 2.2 million destinations. The amount of input data was chosen to maximise the amount of Internet topology data that could be processed by our simulation host. In particular, the memory map topology file needed to be able to be completely loaded into memory and the host was unable to support a memory map file larger than 58 GB. This meant that only one simulation was run at a time and so the automation for simultaneous simulation built into IS0 was not used. Instead .sh scripts were used that contained the simulator run configurations in a run sequence.

This analysis has a limitation, in that there are no destination address repeats in the Traceroute data obtained from CAIDA. This matters because the utility of the global stop set data depends on this. However there is a chance to use the global stop set when the direction is reversed to 'many sources to few destinations'.

The sources windows scenario applies to the many to few direction of analysis and involves grouping sources into windows of a certain size, selected from the following: 0, 1, 2, 10, 100, 500, 3000, 18000. The sources in each window perform their Traceroutes simultaneously. Each window only starts when the previous window has completed. When the

current window's sources complete their Traceroutes, global stop set data collected from all sources in one window and all of the previous windows is transmitted to all sources in the next window before they start. Now that the previous window has completed the next window of sources starts probing from each new source simultaneously. This approach is designed to manage and limit the amount of control traffic needed when using large numbers of vantage point sources. The amount of probe traffic is managed as usual for Doubletree, especially if there are repeated destinations from one sources window group to another where traffic reductions usually occur. A key feature of the sources windows is that control packets are only sent to a subset of all available sources at any given time, rather than to all participating sources.

The timing factor indicates the number of stages, including a final stage where all remaining traces are evaluated simultaneously. 1-stage is when all trace analyses occur simultaneously. In 2-stages and 10-stages each vantage point probes one trace per stage until the final stage when all the rest of the traces are probed together. Global information becomes available to the next stage, with time of availability dependent on queuing delay settings. Staggered is when no simultaneous trace analysis occurs within a vantage point. Other factors are starting TTL and queuing delay.

13.2.1 Validation

Validation of the trace lists was performed by comparing trace lists orientated in the same direction, *e.g.* many to few, to check that no traces were repeated and that all the trace list files had the same set of traces, just with different timing. This is expected because the perl program that generates the trace lists is intended to produce the same set of traces with varied timing, depending on the different levels of the timing factor imposed.

Validation to check paths was attempted on a .trace file from a simulation where sources windows was 500, maximum destinations per AS was 20, initial TTL was 8, queuing time mode was 1000 time units and control packets were sent after traces were completed. This is one of the runs that was used in the results section and included the changes that had

been made to IS0 over the course of this research.

The “check links” program confirmed the integrity of this full data set, however “check paths” ran for a period of several weeks without producing a result and was then stopped to analyse its behaviour. There were bugs in the path validation routine which had to be fixed to be able to validate my simulations.

After these changes the test completed, except that it reported that some paths were undiscovered. This was to be expected as the algorithm that creates the trace lists uses a selection of the available traces, up to 20 per AS.

13.3 Sources windows and AS counts

When writing the modifications for IS0, there was a choice between implementing a new timing factor level called sources windows, or using the most logical existing one: staggered. The staggered setting for timing was used for sources windows. Sources windows mode resembles staggered analyses in groups, because groups of vantage points are processed sequentially, one after the other.

The simulator was configured by creating trace lists that had groups of simultaneous traces of the size of the specified window. Two new parameters were required in the make-tr-list.pl and simulator command lines to support the sources windows feature. These are sources window size, with zero indicating that the native mode with no sources window is to be used, and maximum destinations per AS, which was set to one in the native configuration. A value of twenty for maximum destinations per AS was used for these simulations. The sources window parameter had values of 0, 1, 2, 10, 100, 500, 3000, 18000.

13.4 Results and discussion

Total packet count is used a marker of efficiency in these analyses. It may however be that controller traffic is less of a real cost than probe traffic if higher capacity networks can be used for communication with vantage points in practice. It is also expected that coverage of topology

achieved by Traceroute and Doubletree in the simulator is likely to be fairly constant under variation in the available parameters and a constant topology set, and is not likely to be useful as a measure of efficiency.

The trace count in these results for full simulations was 143,190 traces per simulation.

The investigation of sources windows from many sources required an optimisation of the start TTL based on measurements using native Doubletree (sources window zero) and a sources window setting of 500 sources per window. This was because it was necessary to provide a setting for start TTL in the simulator, and a value had not been established for this. The control queueing delay used was 1000 time units. The maximum number of sources per AS was 20. The direction was many to few. Reasonable choices for some factors were chosen for this optimisation that would later be verified as that factor in turn was optimised. This strategy was used to avoid the large number of simulations that would result if all of the different factor levels were used in every combination available. The factor levels for these simulations are shown in Table 13.1. Fig. 13.1 shows that start TTL of 15 produced no advantage for Doubletree, however 12 showed an improvement and a start TTL of 8 showed a useful saving of 44%. Fig. 13.2 shows the distribution of path lengths in the data set. This suggests that the great majority of paths analysed with start TTL of 8 have a number of nodes after the eighth hop, making the global stop set more effective in its use. The loss of effectiveness of the local stop set in this case is expected as each source has only one trace.

Simulation	Probe mode	start TTL	Sources Windows
dt-ttl8-sw0-st	doubletree	8	0
dt-ttl12-sw0-st	doubletree	12	0
dt-ttl15-sw0-st	doubletree	15	0
traceroute	traceroute		
dt-ttl8-sw500-st	doubletree	8	500
dt-ttl12-sw500-st	doubletree	12	500
dt-ttl15-sw500-st	doubletree	15	500

Table 13.1: Table of factor levels for Fig. 13.1. All have timing set to staggered and control packet queueing delay set to 1000. Direction is many to few. All of the Traceroute modes give the same results. Sources windows of zero means that IS0 behaves in its native fashion.

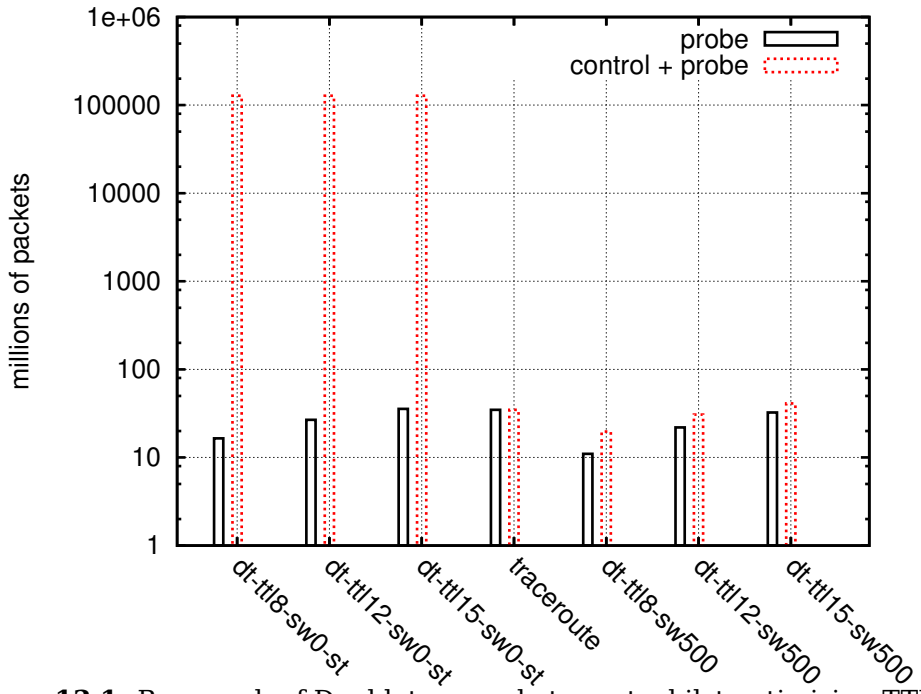

Figure 13.1: Bar graph of Doubletree packets sent whilst optimising TTL, control applied after sources windows size 500 if present, many to few, 19000 ASes and up to 20 traces/AS

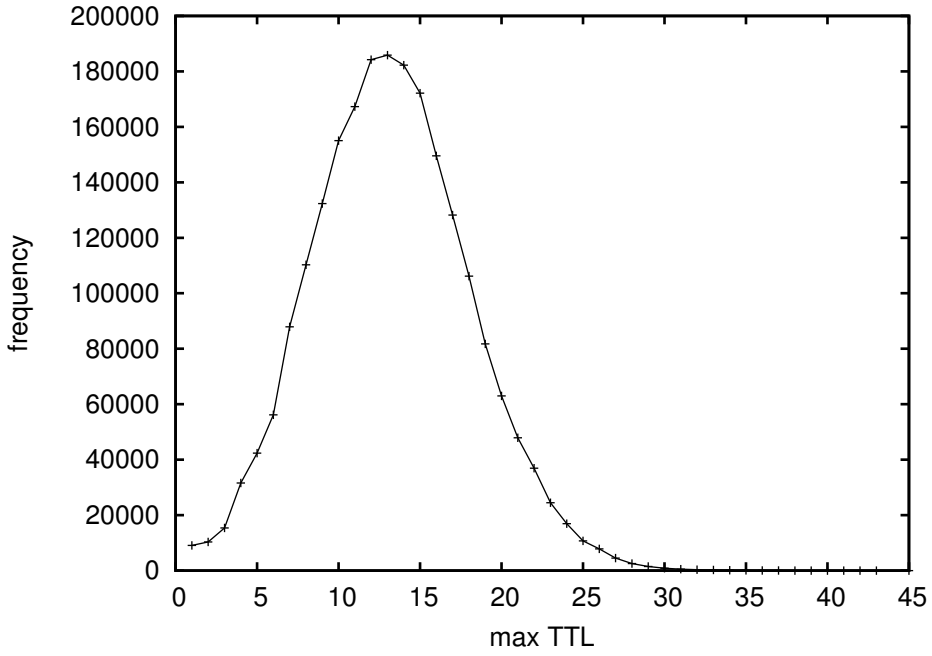

Figure 13.2: Distribution of path lengths in data set

Subsequently a greater variety of sources windows settings was investigated (as shown in Fig. 13.3) for the optimum start TTL of 8. The control queueing delay used was 1000 time units. The maximum number of sources per AS was 20. The direction was many to few. The configuration data on the X axis shows the mode, Traceroute or Doubletree, the TTL start value and the sources window setting, (see Table 13.2). In the case of sources window zero, where sources window analysis is not implemented, 1 stage, 2 stages, 10 stages and staggered are analysed. These are levels of the timing factor which determines how many traces are carried out sequentially at each vantage point before the remainder are carried out together. For control plus probe traffic sources windows of 500 or less gave the best improvement over Traceroute of 44%. Without sources windows vast amounts of control traffic were required.

Simulation	Probe mode	Timing	Sources Windows
dt-ttl8-sw0-s1	doubletree	1 stage	0
dt-ttl8-sw0-s2	doubletree	2 stage	0
dt-ttl8-sw0-s10	doubletree	10 stage	0
dt-ttl8-sw0-st	doubletree	staggered	0
traceroute	traceroute		
dt-ttl8-sw1	doubletree	staggered	1
dt-ttl8-sw2	doubletree	staggered	2
dt-ttl8-sw10	doubletree	staggered	10
dt-ttl8-sw100	doubletree	staggered	100
dt-ttl8-sw500	doubletree	staggered	500
dt-ttl8-sw3000	doubletree	staggered	3000
dt-ttl8-sw18000	doubletree	staggered	18000

Table 13.2: Table of factor levels for Fig. 13.3, Fig. 13.7, Fig. 13.8 and Fig. 13.9. All have starting TTL set to 8 except Traceroute and control packet queueing delay set to 1000. Direction is many to few. All of the Traceroute modes give the same results. Sources windows of zero means that IS0 behaves in its native fashion.

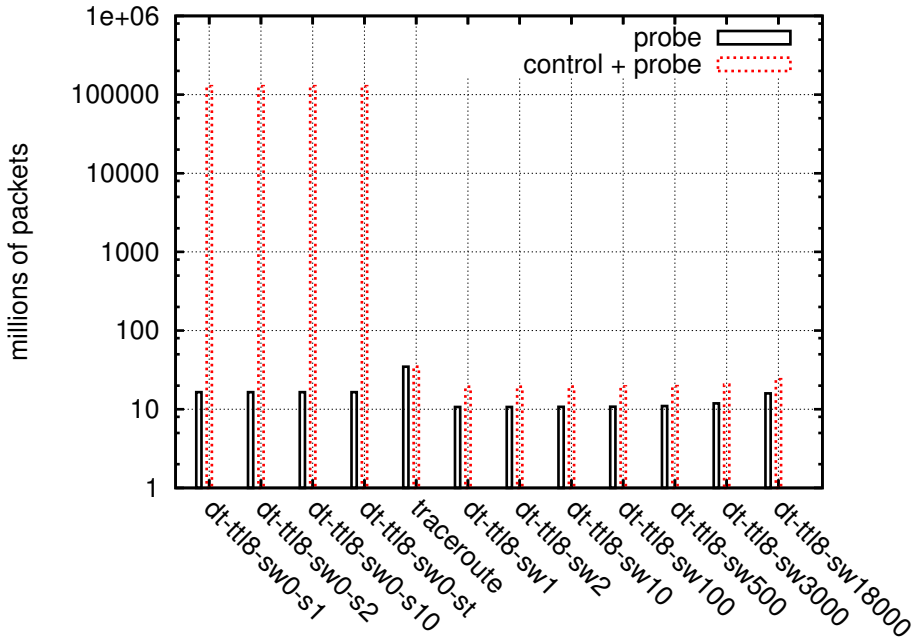

Figure 13.3: Bar graph of Doubletree packets sent, control applied after sources windows if present, many to few, 19000 ASes and up to 20 traces/AS

Table 13.3 shows simulations designed to identify the effects of varying the control packet queuing delay. In practice this parameter is largely a consequence of the structure and traffic conditions of the parts of Internet relating to the controller and also of how the control node is configured, as these are the physical features of the Internet most closely related to controller traffic. It is useful however, to know if longer delays have a detrimental effect and how severe this effect might be. For this comparison a start TTL of 8 and a sources window of 500 were used. The maximum number of sources per AS was 20. The direction was many to few. Delays occur on the paths to and from the controller, which shares the global stop set. Previously the setting for the control packet queuing delay for both sending to controller and receiving from controller has been set to 1000. 1000 is also midway on the scale used in IS0 taken in power based steps (ignoring zero): 0, 1, 10, 250, 1000, 5000, 10000, 15000. Fig. 13.4 shows the effect of varying control queue delay on the start TTL optimised cases. Among the sources windows 500 cases shown

no change due to varying queueing delay is observed.

Simulation	Probe mode	Control queueing delay
traceroute	traceroute	
dt-ttl8-sw500-0T	doubletree	0
dt-ttl8-sw500-10T	doubletree	10
dt-ttl8-sw500-100T	doubletree	100
dt-ttl8-sw500-1000T	doubletree	1000
dt-ttl8-sw500-10000T	doubletree	10000

Table 13.3: Table of factor levels for Fig. 13.4. All have timing set to staggered (required for sources windows). Doubletree has TTL set to 8 and sources windows set to 500. Direction is many to few.

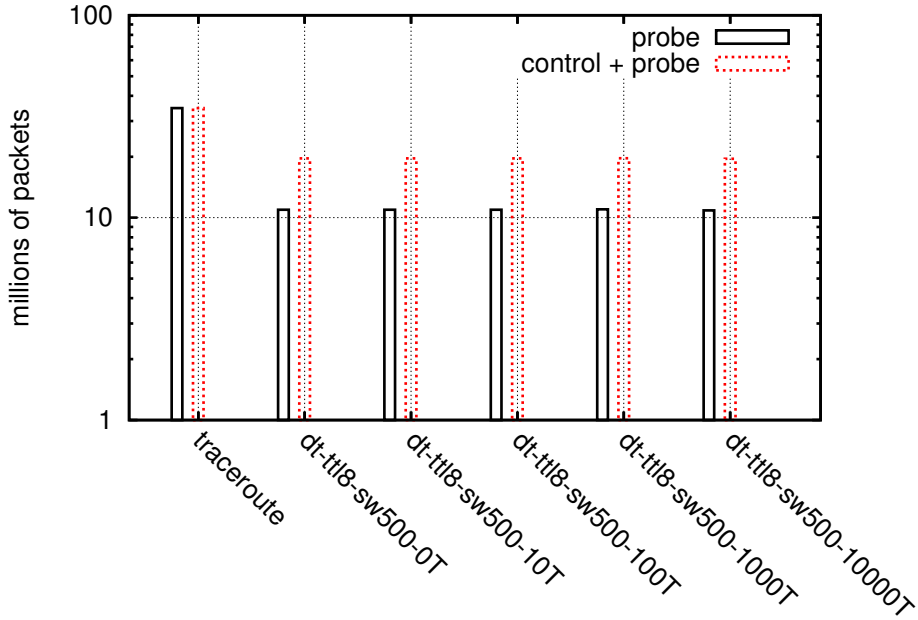

Figure 13.4: Bar graph of Doubletree packets sent whilst optimising control packet queue time, control applied after sources windows size 500 if present, many to few, 19000 ASes and up to 20 traces/AS

The basic analysis used natively in unmodified IS0, which varies the number of stages (Levels of the timing factor), was performed using the optimised TTL value of 8, see Table 13.4 for factor levels. This experi-

ment is a point of comparison with previous work on Doubletree including cost analysis. The control queueing delay used was 1000 time units. The maximum number of sources per AS was 20. The direction was many to few. Sources windows were deactivated with a value of zero. Fig. 13.5 shows the results for TTL 8. These modes of Doubletree were hugely expensive in terms of control traffic and demonstrated that there was significant room for improvement.

Simulation	Probe mode	Timing
traceroute	traceroute	
dt-ttl8-1stage	doubletree	1 stage
dt-ttl8-2stage	doubletree	2 stage
dt-ttl8-10stage	doubletree	10 stage
dt-ttl8-st	doubletree	staggered

Table 13.4: Table of factor levels for Fig. 13.5. All have TTL set to 8 except Traceroute and control packet queueing delay set to 1000 (does not apply to Traceroute). All of the Traceroute modes give the same results. Direction is many to few.

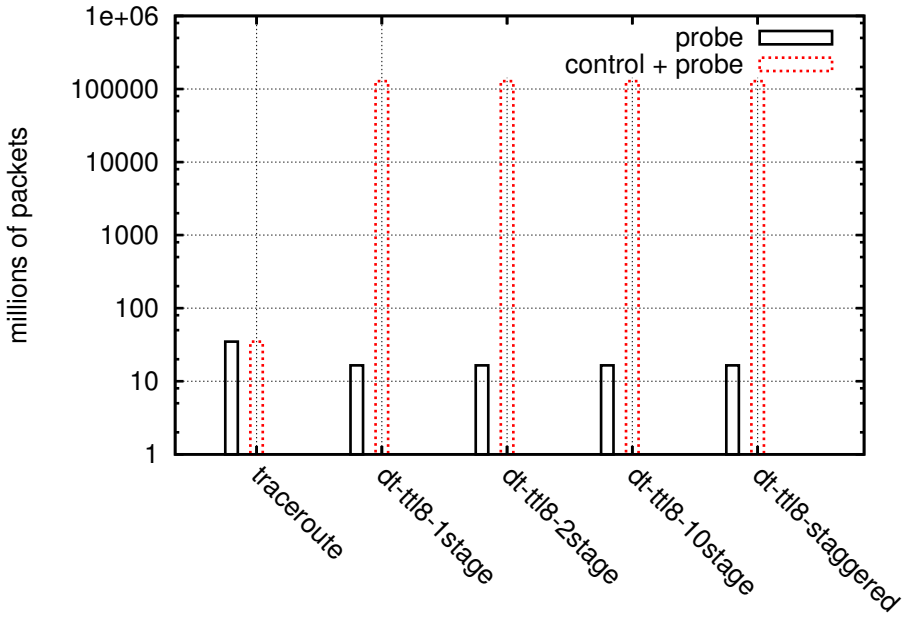

Figure 13.5: Doubletree packets sent whilst varying the number of stages, many to few, 19000 ASes and up to 20 traces/AS

Also the basic (native) analysis in IS0 for few to many Doubletree and Traceroute is shown in Fig. 13.6, (see Table 13.5 for factor levels). IS0 was unmodified for this except for the use of cost analysis. The control queueing delay used was 1000 time units. The maximum number of destinations per AS was 20. For Doubletree the start TTL was 8. In this case there is little control information as there are few vantage points operating to share global stop sets. Use of the local stop set gives an improvement of Doubletree over Traceroute, however the global stop set is of no advantage with this data set as there are no repeated destinations.

Simulation	Probe mode	Timing
tr-few-1stage	traceroute	1 stage
tr-few-2stage	traceroute	2 stage
tr-few-10stage	traceroute	10 stage
tr-few-staggered	traceroute	staggered
dt-few-1stage	doubletree	1 stage
dt-few-2stage	doubletree	2 stage
dt-few-10stage	doubletree	10 stage
dt-few-staggered	doubletree	staggered

Table 13.5: Table of factor levels for Fig. 13.6. All have control packet queueing delay set to 1000. Direction is few to many. This is the IS0 native analysis.

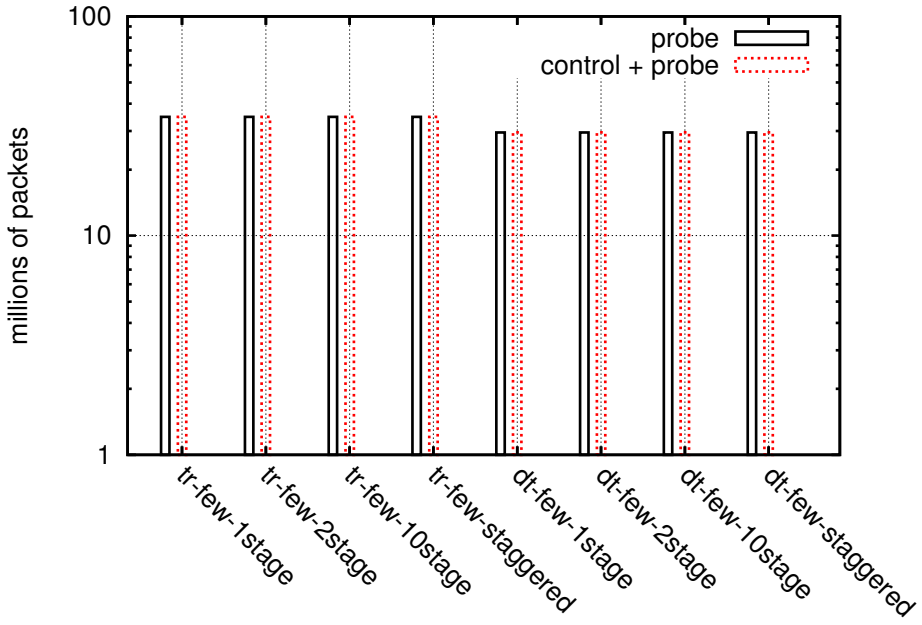

Figure 13.6: Doubletree packets sent whilst varying the number of stages, few to many, 19000 ASes and up to 20 traces/AS

Other data available from IS0 includes link usage, which is shown in Fig. 13.7. Both metrics (maximum packets sent on a link and maximum concurrent packet sent on a link) are higher in the native sw0 (sources windows deactivated) case. Maximum packets on a link is slightly higher

for the sources window activated cases than the Traceroute case. For maximum concurrent packets on a link the packet load is directly related to the window size. Presumably this is related to controller traffic as this is where all probers in the window send traffic. In future work, it may be better to look at the 95th percentile rather than the maximum to avoid bias towards controller traffic.

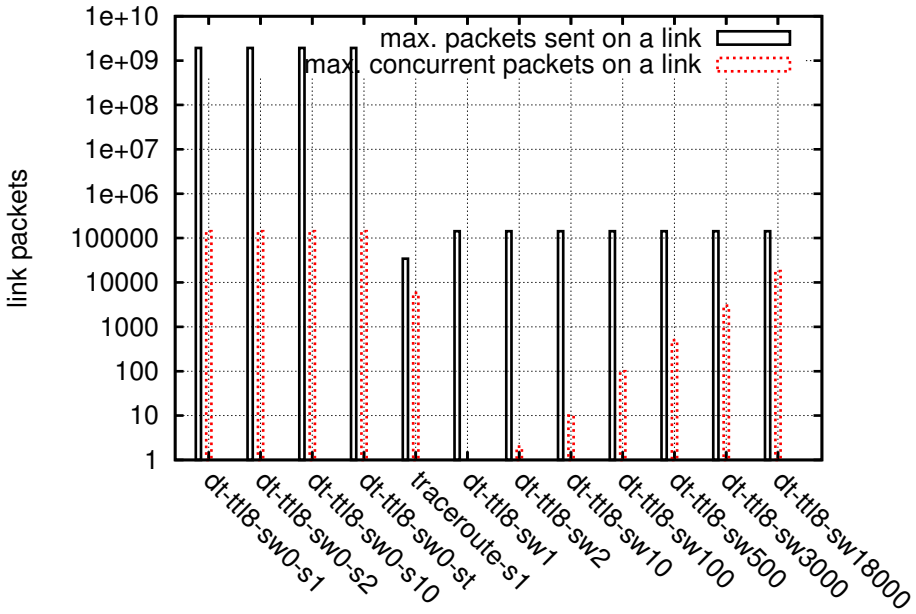

Figure 13.7: Maximum packets sent on a link and maximum concurrent packets sent on a link during simulations.

Fig. 13.8 shows global stop set hits. The various stages settings (Levels of the timing factor) of the native modes in the many to few cases show a larger number of look ups because the traces are all processed simultaneously and therefore changes are relatively slow to appear in the global stop set, given there is a control traffic delay of 1000 time units. The look ups then are less in number and increase with larger sources window sizes as more traces are processed simultaneously. Similarly more nodes are found in the global stop sets in the sources windows cases as there is a greater chance for the node to have been seen previously. The ratio of these two numbers gives the percent global stop set hits shown in Fig. 13.9. The percent local stop set hits are all zero. The stop set sizes

are 318,000 for the global stop sets and 1,100,000 for the local stop sets and do not vary greatly across the modes tested. The local stop sets are large and there are no stop set hits because there are no repeated sources.

The results show that sources windows can greatly reduce the amount of control traffic required by Doubletree. However, we also find that excessively large sources windows should be avoided as they can lead to control traffic congestion and wasted lookups in the global stop set.

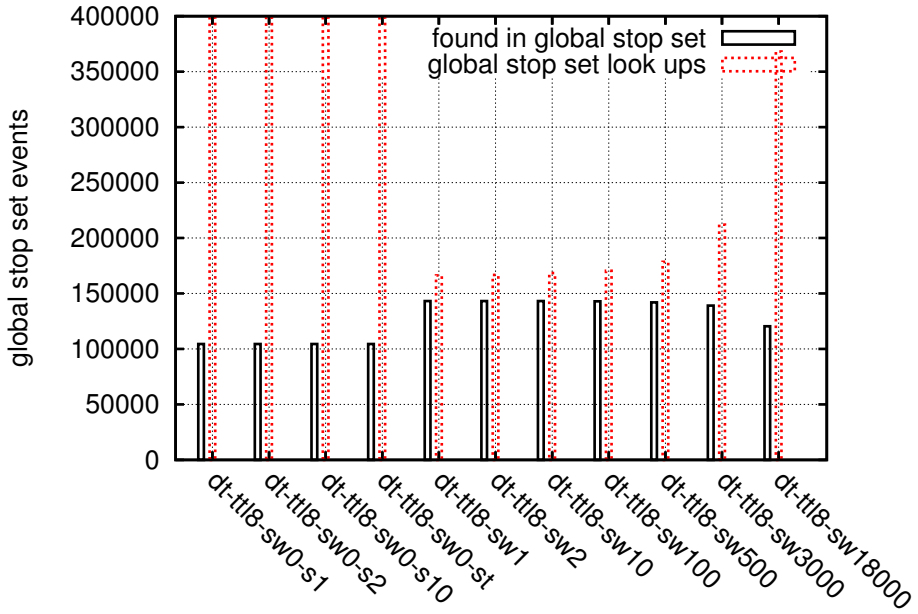

Figure 13.8: Global stop set nodes found and total global stop set lookups.

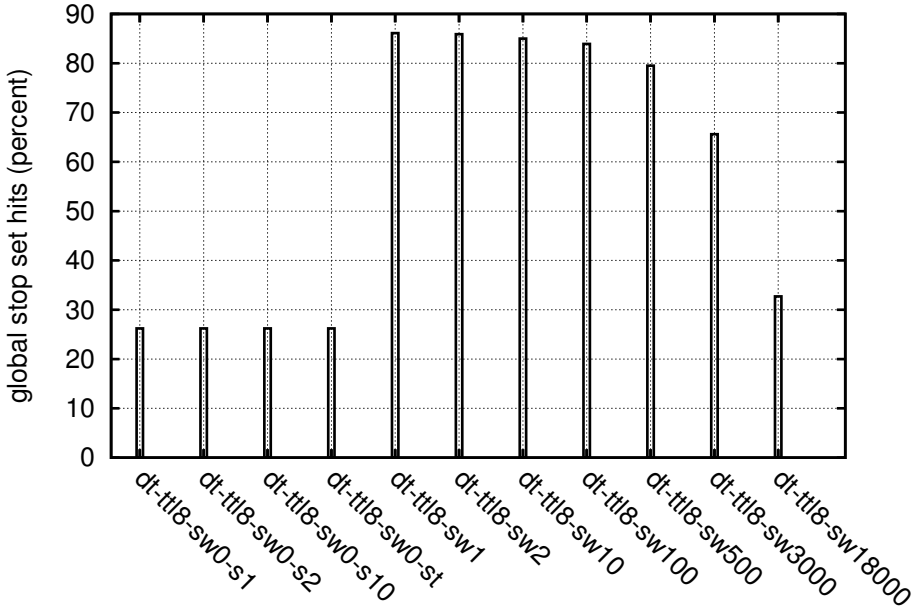

Figure 13.9: Global stop set hits as a percentage of all attempts to find an address in the stop set.

Chapter 14

Doubletree using a trace based simulator

14.1 Introduction

Previous work on Doubletree has suggested that it would be worthwhile to perform cost analysis of Doubletree [47]. In that previous work it appeared likely that under the unmodified structure applied using IS0 described in chapter 13, Doubletree would be traffic greedy. This experiment aims to determine whether an improved structure could be applied that would result in an advantage for using Doubletree over Traceroute. We also aim to identify the circumstances under which Doubletree could provide other improvements over Traceroute beyond traffic volumes, such as degree of repetition of sources and destinations.

If our experiment demonstrates that Doubletree provides sufficiently reduced traffic volumes in response to improved configuration, then we can be confident that distributed Internet active probing systems will be more likely to adopt it or something similar.

14.1.1 Trace based simulator, BISD

This simulator is a simplified process compared to IS0 where the unit of analysis is the trace collected for a particular (source address, destination address) pair. The simulator was developed as part of this research into distributed active probing in the Internet and is named Basic Internet Simulator Doubletree (BISD). The input to BISD is trace data from

scamper warts files collected from a number of vantage points simultaneously. Traces from different vantage points are not processed simultaneously but one trace at a time giving alternating vantage points with each trace. The simulator is derived from the program for the regular analysis of scamper warts data, which was modified to alternate vantage points, and to sequentially apply various factor level settings when each trace is analysed.

Per trace processing consists of trace data being compared to the data in the accumulating global and local stop sets, as Doubletree is simulated. The analysis counts the number of probes that are used, until a previously seen node is reached. Topology information from earlier in the run about the nodes that occur after the stop point allows the entire path to be inferred. The stop point is always associated with a repeated source or destination, and is found as Doubletree probes outward from the start TTL in the middle of the path.

There are three factors that are Direction, Timing and Control. Direction indicates whether trace data is used in the forward or reverse direction *i.e.* reverse direction uses the destination as source, thus there are many sources and few destinations in this case. The timing factor indicates whether simultaneous or sequential analysis of traces occurs, along with some options that group traces into windows that are analysed sequentially. Each window acts as a stage such that control traffic is sent to the vantage points only after completion of the window. When there is analysis of a set of stages, the control factor determines that control information (global stop set) is communicated to the vantage points after each stage. Each stage is one trace at each vantage point until the final stage which performs the rest of the traces. When staggered analysis or staggered groups are analysed, the control factor determines when control information is sent to the vantage points: either after each trace or group of traces, or after a certain number of packets or after each one percent of the traffic in a run cycle. When sources windows are analysed control information is sent to the new window of vantage points after each window of vantage points performs its traces.

This trace or non packet event based simulator uses sequential par-

allel access to traces collected from tens of vantage points. This means that it processes one trace from each vantage point sequentially and then moves on to another round from each. This is an approximation for the order in which the data was actually collected, though the precise sequence is not vitally important to the analysis. This is because the first encounter with a particular node causes later cases to be flagged as previously seen on the path to a given destination. Changing the order of node occurrence still results in similar traffic savings. The main purpose of the data sequence is to correctly calculate and read global stop sets during simulation. The simulator performs a sequence of factor level analyses for each trace, accumulating results in arrays of variables accessed via a number representing the analysis mode or combination of factor levels. Because there are so many modes, the program is divided into sections that compile at different times depending on the switches that are selected. This means that several copies of the program may run different parts of the analysis at the same time. The computing resources required for this are modest so many simultaneous runs are feasible.

Table 14.1 shows the factor levels of the basic analysis structure used for the non packet event based simulations. Table 14.2, Table 14.3 and Table 14.4 show explanations of the abbreviated factor level names. Direction indicates the sources state, few or many, and the number of vantage points used to collect the input data. All data reported in the graphs utilises all available vantage points. There were two data sets used, and reduced vantage point simulations were carried out in each case, but the results added little to the picture created by the full VP analysis. The D_MANY_R (real) cases are when the many sources occur more than twice, which reduces the amount of data used from the MDA set extraction to Traceroute style data. This is called real because repeated sources are likely to be a factor that makes Doubletree perform better and thus a more realistic scenario for the configuration of Doubletree is provided. The sources are duplicated in small numbers because data collection in the few to many direction repeated some destinations. Selecting these destinations as data results in a reduced data set for many to few analysis. Timing indicates the number of stages, including a final stage

where all remaining cases are evaluated simultaneously. T_STAGGERED is when no simultaneous analysis occurs. T_WIN500 analyses traces simultaneously within groups of 500 traces per vantage point and then processes control information before analysing the next group of 500. T_SSWIN cases are sources windows of various sizes, which gather information simultaneously across vantage points and then share control information with the next group of vantage points. The reference or control experiment for T_SSWIN real is T_NOSAVINGS, which performs the reduced set of many source analyses simultaneously. Control indicates when control information is shared. C_AFTERSTAGES indicates that after each stage control information is shared. Where simultaneous analyses are employed this means that there is no benefit from control information between members of this group of traces. C_10000PACKETS and C_100000PACKETS wait for the given number of packets per vantage point then share control information. C_100PERCYCLE is the case where there are 100 evenly spaced occasions in a cycle of data collection where control data is shared.

Mode No	Direction	Timing	Control
3	D_FEW20S	T_1STAGE	C_AFTERSTAGES
6	D_FEW20S	T_2STAGE	C_AFTERSTAGES
9	D_FEW20S	T_10STAGE	C_AFTERSTAGES
12	D_FEW20S	T_100STAGE	C_AFTERSTAGES
15	D_FEW20S	T_1000STAGE	C_AFTERSTAGES
18	D_FEW20S	T_10000STAGE	C_AFTERSTAGES
21	D_FEW20S	T_STAGGERED	C_AFTERSTAGES
24	D_FEW20S	T_STAGGERED	C_10000PACKETS
27	D_FEW20S	T_STAGGERED	C_100000PACKETS
30	D_FEW20S	T_STAGGERED	C_100PERCYCLE
33	D_FEW20S	T_WIN500	C_AFTERSTAGES
36	D_FEW20S	T_WIN500	C_10000PACKETS
39	D_FEW20S	T_WIN500	C_100000PACKETS
42	D_FEW20S	T_WIN500	C_100PERCYCLE
45	D_MANY20D	T_1STAGE	C_AFTERSTAGES
48	D_MANY20D	T_2STAGE	C_AFTERSTAGES
51	D_MANY20D	T_10STAGE	C_AFTERSTAGES
54	D_MANY20D	T_SSWIN500	C_AFTERSTAGES
57	D_MANY20D	T_SSWIN3000	C_AFTERSTAGES
60	D_MANY20D	T_SSWIN18000	C_AFTERSTAGES
63	D_MANY20D	T_SSWIN100000	C_AFTERSTAGES
66	D_MANY20R	T_SSWIN500	C_AFTERSTAGES
69	D_MANY20R	T_SSWIN3000	C_AFTERSTAGES
72	D_MANY20R	T_SSWIN18000	C_AFTERSTAGES
75	D_MANY20R	T_SSWIN100000	C_AFTERSTAGES
78	D_MANY20R	T_NOSAVINGS	C_AFTERSTAGES

Table 14.1: Table of possible factor levels for trace based simulators.

Direction factor level	Explanation
D_FEW5S*	Few (5) sources to many destinations
D_FEW10S*	Few (10) sources to many destinations
D_FEW20S	Few (20) sources to many destinations
D_MANY5D*	Many sources to few (5) destinations
D_MANY10D*	Many sources to few (10) destinations
D_MANY20D	Many sources to few (20) destinations
D_MANY5R*	Many sources to few (5) destinations each source with more than two occurrences
D_MANY10R*	Many sources to few (10) destinations each source with more than two occurrences
D_MANY20R	Many sources to few (20) destinations each source with more than two occurrences

Table 14.2: Table of explanations of direction factor levels.
* - not in Table 14.1

Timing factor level	Explanation
T_1STAGE	All traces are carried out simultaneously This is a no savings case
T_2STAGE	One trace from each source is carried out then the rest simultaneously
T_10STAGE	Nine traces from each source are carried out sequentially, then the rest simultaneously
T_100STAGE	99 traces from each source are carried out sequentially, then the rest simultaneously
T_1000STAGE	999 traces from each source are carried out sequentially, then the rest simultaneously
T_10000STAGE	9999 traces from each source are carried out sequentially, then the rest simultaneously
T_STAGGERED	All traces are carried out sequentially
T_WIN500	All traces are carried out sequentially in groups of 500 simultaneously in each VP
T_SSWIN500	A sources window, size 500 is used
T_SSWIN3000	A sources window, size 3000 is used
T_SSWIN18000	A sources window, size 18000 is used
T_NOSAVINGS	This is the reference for SSWIN levels. No window as all traces carried out simultaneously

Table 14.3: Table of explanations of timing factor levels.

Control factor level	Explanation
C_AFTERSTAGES	Control packets are sent after stages
C_10000PACKETS	Control packets are sent after 10000 packets in a VP
C_100000PACKETS	Control packets are sent after 100000 packets in a VP
C_100PERCYCLE	Control packets are sent on 100 occasions in a data collection cycle, in a VP

Table 14.4: Table of explanations of control factor levels.

14.2 Experimental design and data collection

14.2.1 Doubletree using extracted MDA data

Run 1 MDA Traceroute data from 21 vantage points was sampled to yield data in the form of standard Traceroute. This data was used because the data set contained traces to the same destination from more than one vantage point. This gives Doubletree the opportunity to demonstrate improvement over regular Traceroute and gives a better comparison with the likely configuration of Atlas and similar systems. The full data set is used for some analyses and a reduced data set is used where data is included when there are more than two destinations existing in the set.

14.2.2 Doubletree using regular Traceroute data

Traceroute data from 23 vantage points collected by CAIDA on 17/6/2013 were used to simulate Doubletree. None of the destinations were repeated in this data set, placing a limitation on the usefulness of the global stop set when the analysis was performed in the forward direction. Similarly, the local stop set was not particularly useful when the analysis was carried out in the reverse direction.

14.2.3 Doubletree validation for both data sets

Both programs (the one using extracted MDA data and the one using regular Traceroute data) were able to dump information about which traces were being analysed. Each line contained the destination address, the vantage point ID and the direction factor level. The data was than analysed using an external Perl program to count instances of different direction factor levels (see Table 14.2) for different vantage points. A hash table was used to store this information and any recurrences of the same combinations were counted as errors *i.e.* these would be unnecessary repeat analyses. In the MDA case the expected multiples of 70000 Traces/-Vantage Point were found for each level of the direction factor, which includes vantage points used, as shown in Table 14.5. For classic Traceroute the numbers of Traces per Vantage Point varied; see Table 14.6 for totals. The trace count levels match the data files and also indicate no missed traces or repeated trace, as there were no trace analysis errors found using either data set.

direction mode	count of all traces	divide by 70000
d_few5	350000	5
d_many5	350000	5
d_many5r	350000	5
d_few10	700000	10
d_many10	700000	10
d_many10r	700000	10
d_few20	1400000	20
d_many20	1400000	20
d_many20r	1400000	20
errors	0	

Table 14.5: Processing of traces by the MDA data based Doubletree simulator in different direction modes.

direction mode	count of all traces
dfew5	592166
dmany5	592166
dfew10	1047236
dmany10	1047236
dfew20	2226652
dmany20	2226652
errors	0

Table 14.6: Processing of traces by the classic Doubletree simulator in different direction modes.

14.2.4 Doubletree algorithm for both data sets

Once the correct sequence of trace analyses was able to be performed by the simulator, a Doubletree algorithm was created that stored pairs of observed addresses in the local and global stop sets. The pairs stored the source address and related visited node or destination address and related visited node respectively. The algorithm was tested and validated by dumping information from several traces and analysing the results by hand to confirm that the correct pairs were added to the local and global stop sets. For the MDA data a simple algorithm was written to extract classic Traceroute like data, as this is what is required for simulating Doubletree and this data contains repeated destinations which is desirable for making best use of Doubletree.

14.3 Results and discussion

14.3.1 Extracted MDA data analysis

Fig. 14.1 shows the amount of control traffic observed when the number of stages over which control traffic is used is varied, in the few to many cases. As the number of stages increases the amount of control traffic increases. Furthermore, the T_WIN500 case has less stages than T_STAGGERED and shows reduced control traffic. Fig. 14.2 shows vari-

ous numbers of stages: global only performance shows a slight improvement of 20,000 probe and control packets for 10,000 stages over 1 stage, however staggered is less economical by 100,000 packets, (also see Table 14.7 for details). The same table also shows that 10,000 stages is also the best performer for global and local stop sets used together. Local shows a 68% saving over global in each case. This simulator is unable to run the local data simultaneously, as it runs one trace at a time. However, because local stop set data becomes available immediately within a VP (as distinct from the global stop set which must be communicated across the Internet) the sequential process is likely to produce a similar result to a more advanced simulator as only the order of node discovery changes. Regardless of when a node is first seen and in which trace, the other occurrences will be found resulting in similar behaviour for small changes in order of discovery.

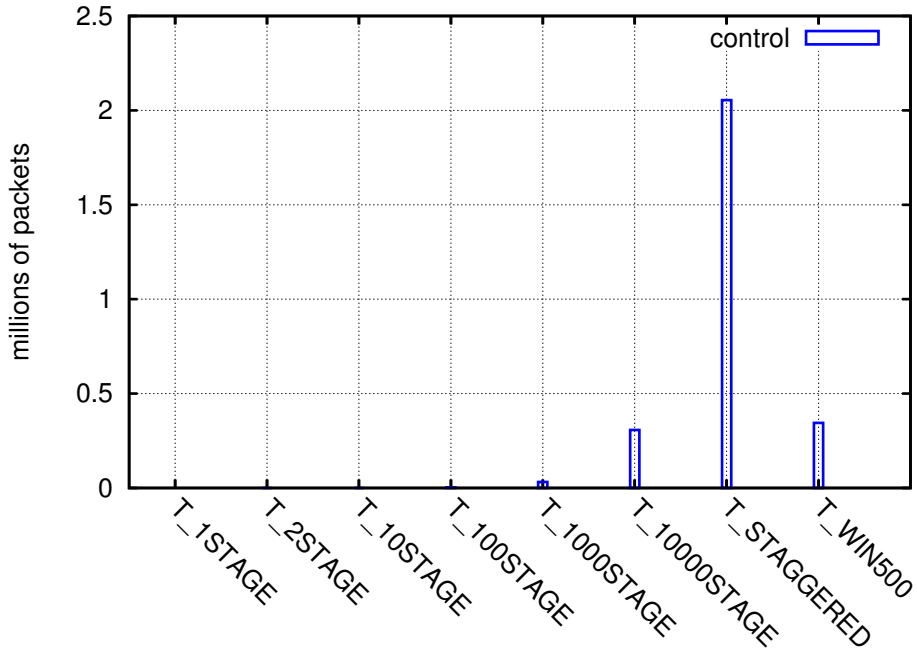

Figure 14.1: MDA data Doubletree control packets sent, control applied after stages, sources few (20) to many destinations 70000.

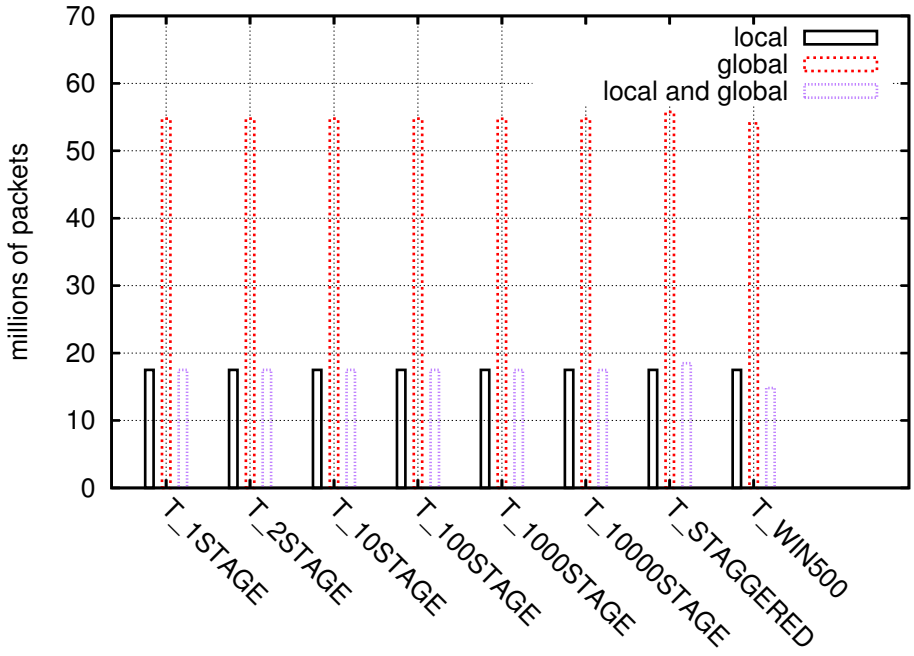

Figure 14.2: MDA data Doubletree probe and control packets sent, control applied after stages, sources few (20) to many destinations 70000.

Timing mode	global only packet counts	global and local
1STAGE	54,765,453	17,499,135
2STAGE	54,765,497	17,499,179
10STAGE	54,765,651	17,499,333
100STAGE	54,766,595	17,500,277
1000STAGE	54,759,518	17,493,200
10000STAGE	54,745,680	17,479,362
STAGGERED	55,753,608	18,487,290

Table 14.7: Doubletree few sources to many destinations where packet counts are global only and global plus local, probe plus control traffic.

Fig. 14.3 illustrates the control traffic used by various sources window configurations along with the reference single stage result, when all of the MDA data is used (as opposed to a restricted data set) to evaluate many to few performance. T_1STAGE and T_SSWIN100000 are lower than the other cases because no control traffic is transmitted at the end of the final stage in the simulation. The three other sources windows

cases show similar results. This is because smaller windows update less VPs but they do this more often, thus cancelling out most overall change. Fig. 14.4 illustrates improvement in traffic used (the control is T_1STAGE) due to the global stop set, with sources windows applied. SSWIN500 shows a 10% improvement compared to the 1 stage control. In this case, the local stop set has limited effect and is also prohibitively expensive to evaluate computationally. This is because the local stop sets belonging to the many sources were stored in an array and each time a new source was added, many ordered array elements had to be rearranged. As the numbers grow large this process can become extremely slow, as it did during these experiments. When all of the data is used for analysis, most sources are only used once and therefore the local stop sets can not help to reduce probe counts.

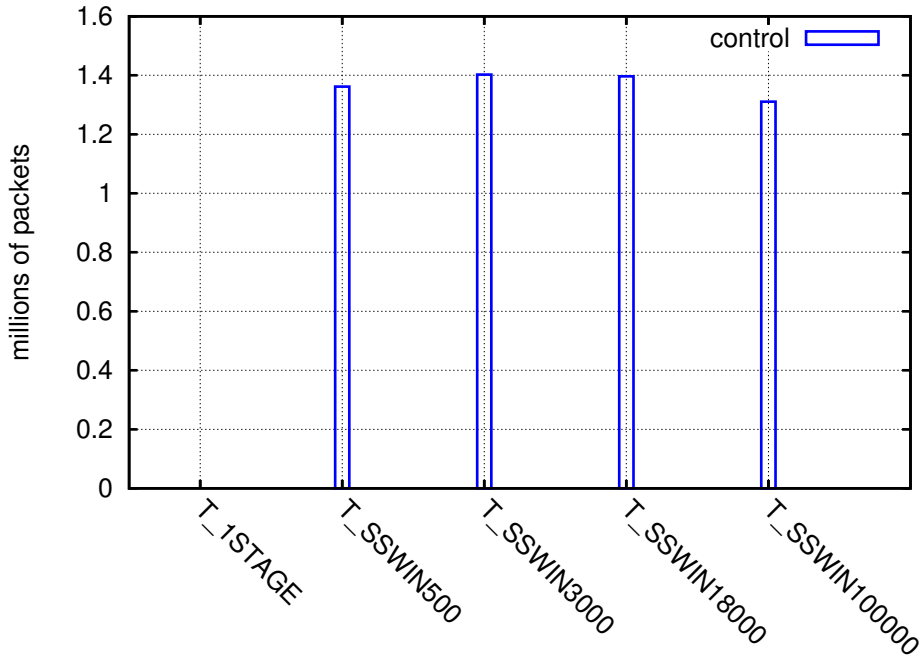

Figure 14.3: MDA data Doubletree control packets sent after sources windows, sources many to few destinations 20.

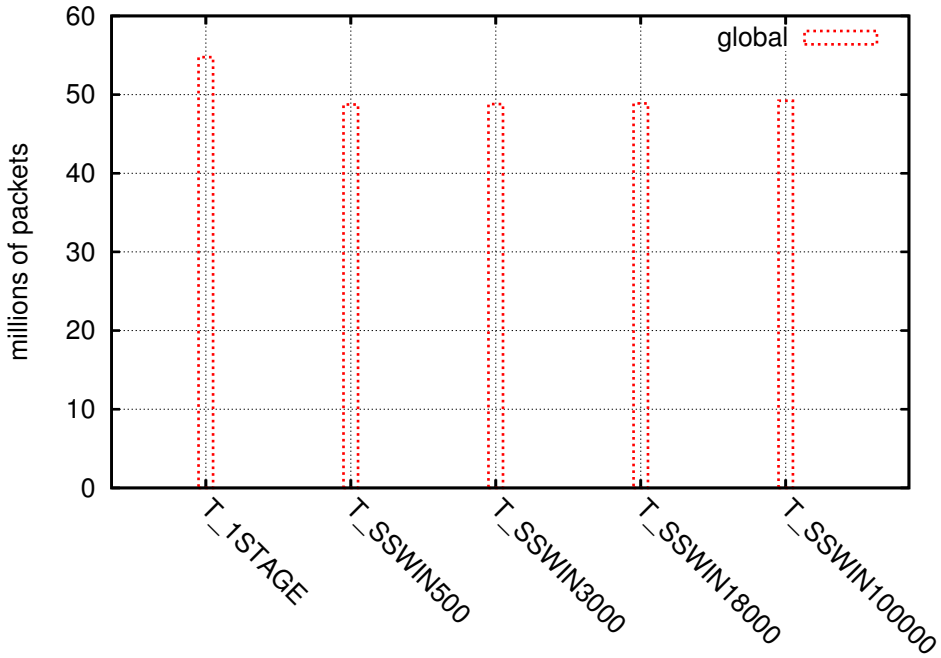

Figure 14.4: Graph of MDA data Doubletree packets sent, control applied after sources windows, sources many to few destinations 20.

Fig. 14.5 and Fig. 14.6 show the results for many to few where the data is restricted to cases where the source occurs more than twice *i.e* there are more than two traces with that source address. The NOSAVINGS case is zero for control traffic, because this is the simultaneous analysis control case and thus it has no control traffic (as there is no opportunity to make use of it). This option resembles a pure Traceroute control. The data restriction gives a reasonable rate of repeated sources (in future work the data collection method could be modified to achieve this), which in turn creates an opportunity to see the benefits of using the local stop set. Fig. 14.7 shows the distribution of occurrence rates for sources in the data set. While most sources only appeared once, there were 139,757 sources (17%) that appeared more than twice that we could use to create the restricted data set. All of the sources windows gave a useful improvement in performance, however the smaller windows were slightly better. In all cases apart from the no savings control, the global and local stop sets led to a reduction in probe traffic. Combined use of both local and global stop sets resulted in a further reduction compared with

using one stop set alone. The total reduction for both stop sets was approximately 20%. This suggests that if Doubletree is run with sources windows from large numbers of vantage points there are likely to be useful savings if the same vantage point is used more than twice and there are destinations that are probed by different vantage points.

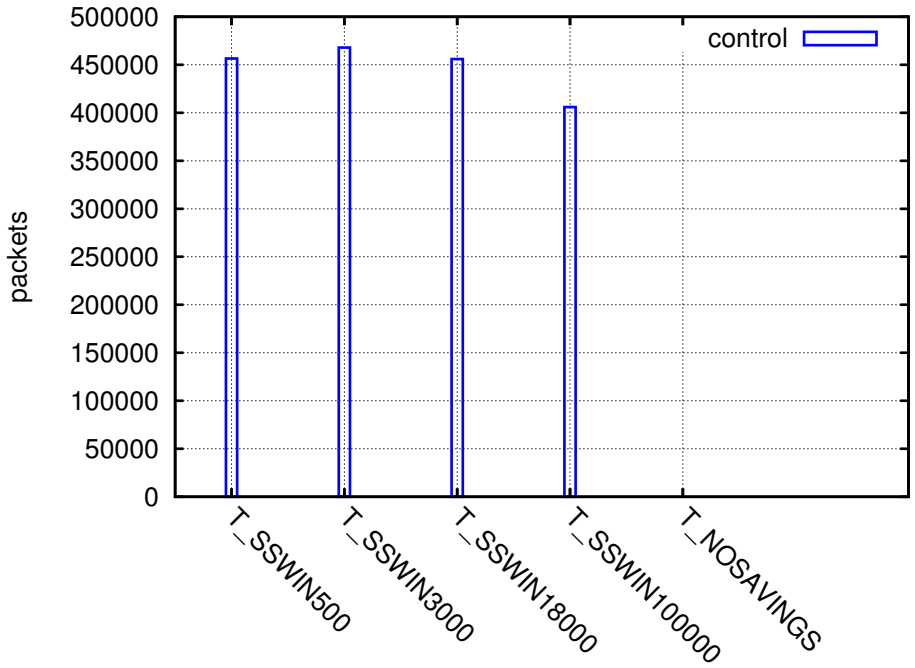

Figure 14.5: MDA data Doubletree control packets sent after sources windows, sources many to few destinations 20. The data includes only the restricted (or real) MDA set.

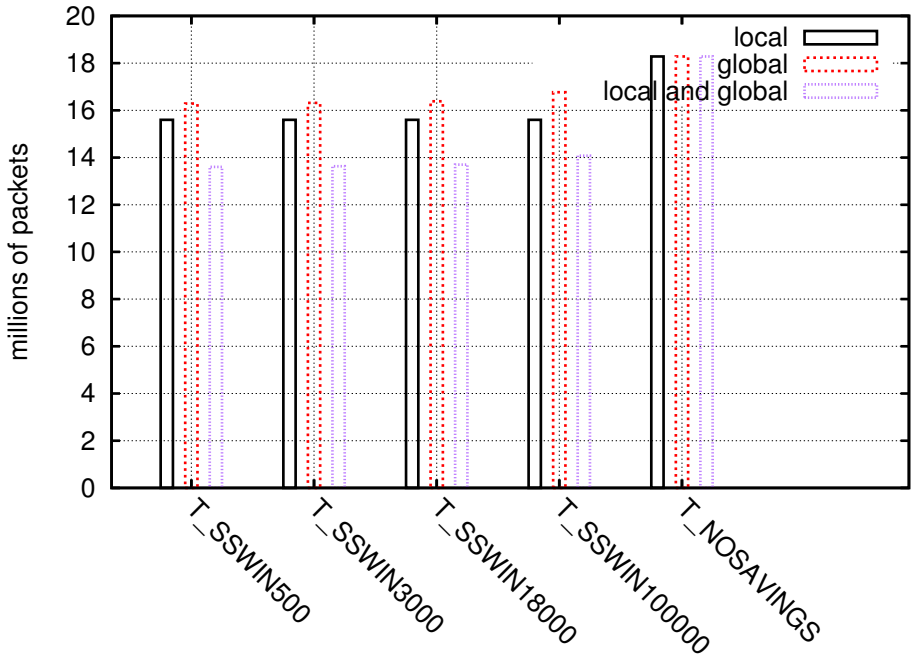

Figure 14.6: MDA data Doubletree packets sent, control applied after sources windows, sources many to few destinations 20. The data includes only the restricted (or real) MDA set.

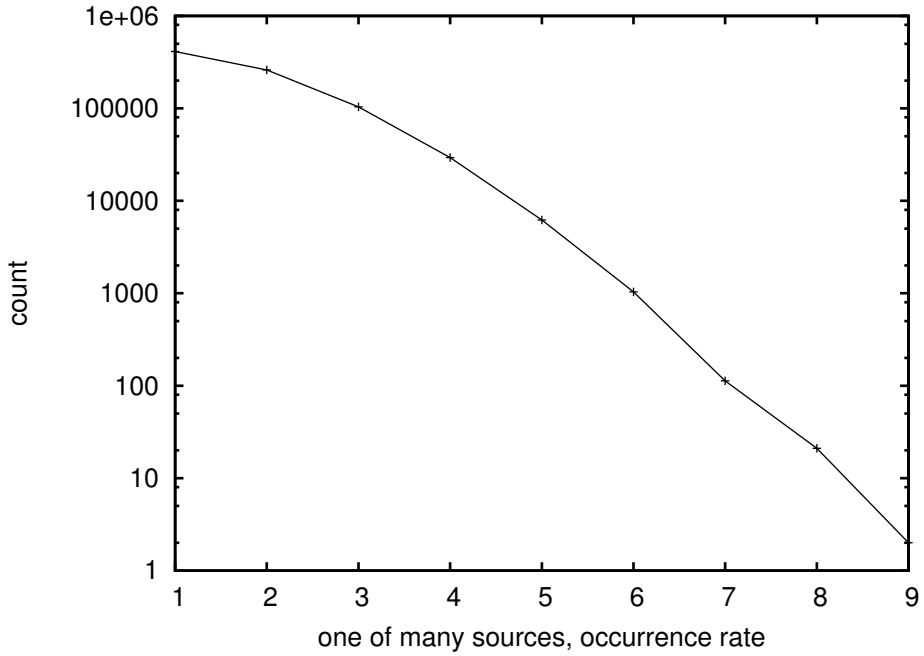

Figure 14.7: Source occurrence rate frequency in the MDA data set, in the many to few direction. Occurrence rate is the number traces performed from a given source.

14.3.2 Regular Traceroute data analysis

Fig. 14.8 shows the 73% saving of using local rather than global stop set data for the few to many case where the timing factor is set to T_STAGGERED and the control factor is varied. For this data set there is no advantage in using global stop set information as none of the destinations are repeated. However, compared to C_AFTERSTAGES, the savings of the 10000 and 100000 packet modes along with the 100 per cycle mode can be seen when global stop set data is still collected in the simulation.

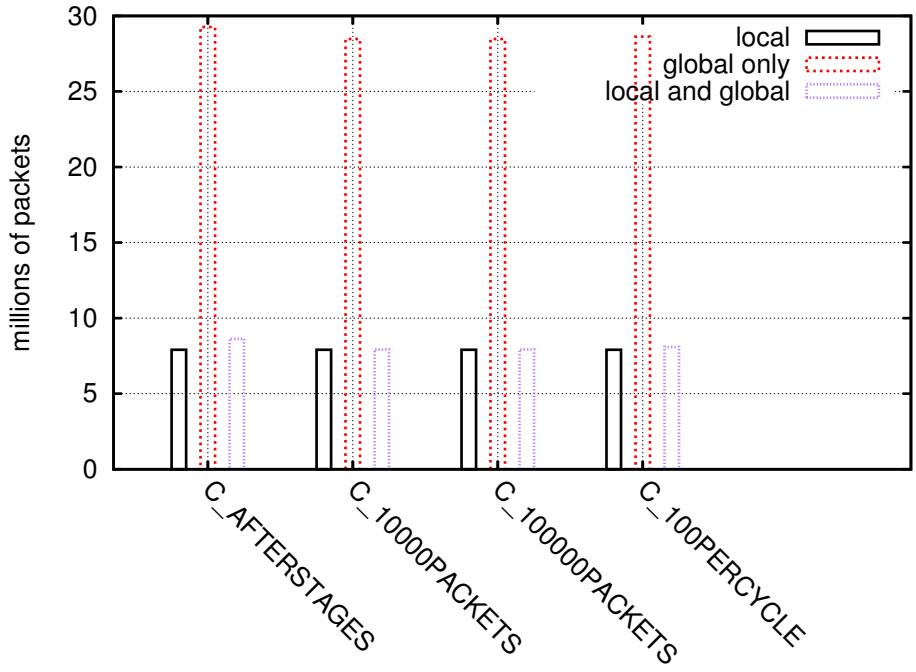

Figure 14.8: CAIDA data Doubletree packets sent, sources few (20) to many destinations.

Fig. 14.9 shows approximately a 5% saving for sources windows (many to few, CAIDA data) in terms of total packets sent compared with control 1 stage. Control traffic for the sources windows was approximately 13,500 packets. Control traffic for 1 stage was zero as there were no further traces to process after the first stage was completed.

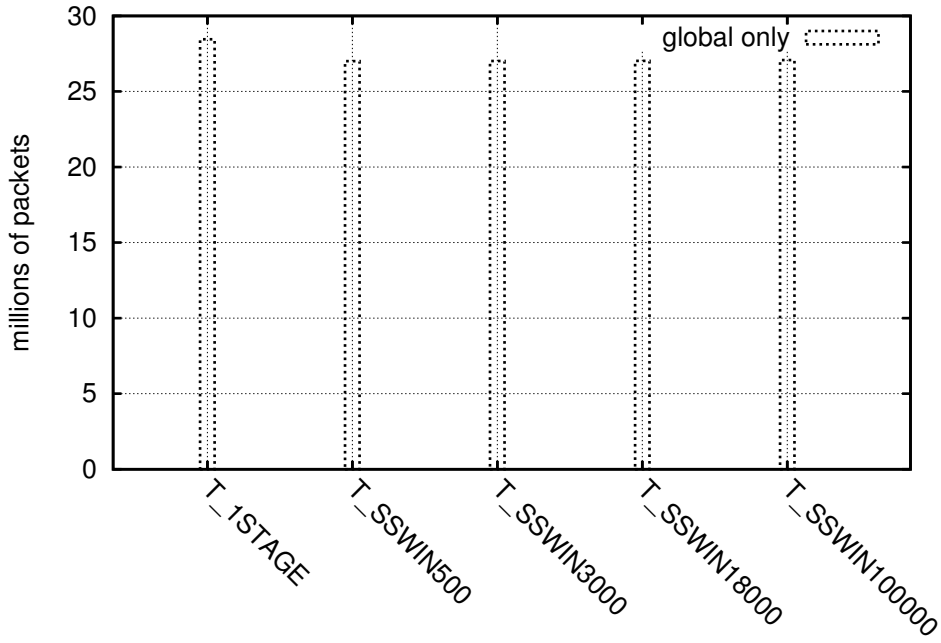

Figure 14.9: CAIDA data Doubletree control and probe packets sent, sources many to few destinations (20).

Using the full MDA set in the few to many direction traffic for local only showed a 68% saving over global only. T_WIN500 showed a slight advantage over the other timing modes. For the full MDA set in the many to few directions using sources windows a 10% improvement was seen over the control for global stop set based analysis. The smaller sources windows sizes are the best options for this situation.

For the restricted MDA set also analysing sources windows combined use of global and local stop sets resulted in 20% savings over control. Once again the smaller sources windows settings are best. This shows the benefit of designing the data set so that both local and global stop set information may be used to advantage. Ideally in a data set on a distributed system like Atlas, the destinations would not be repeated as often, and the sources would be repeated more often than in our data set. This would reduce the effect of the global stop set and increase the effect of the local stop set to some extent.

For the CAIDA data set in the few to many direction local only analysis gave a 73% saving over global only for the timing factor staggered and

the control factor varied. There was no advantage of using control information as there were no repeated destinations. For the CAIDA data set in the many to few direction sources windows gave a 5% saving over control. There were many repeated destinations that allow the global stop set to have an effect in this analysis. There were however no repeated sources, so there was no opportunity to benefit from local stop sets.

14.3.3 Many to many

This research has primarily focussed on few to many and inferred from many to few situations. The many to many situation is also relevant because of the trend in Internet research to develop facilities with large numbers of vantage points [24]. Such facilities can be used for mapping Internet topology.

In order to propose an efficient method for using highly distributed facilities to map Internet topology, a number of constraints must be considered. Firstly, the rate of change of Internet virtual paths should be considered, as Doubletree assumes constant Internet topology for the duration of data collection. At first glance, the work of Cunha [20] suggests that even very short data collection cycles, *e.g.* 1 hour, are still too long as 60% of routes have a duration of an hour or less. On the other hand, over a longer time the results of this thesis see a slower rate of permanent change: for non load balancer route changes a rate of 0.3% paths per hour was seen, and for load balancer internal changes of 0.1% paths per hour was seen. Taken together, this information suggests that a data collection cycle of 6-12 hours may be feasible.

The key variables in prescribing a possible strategy for mapping Internet topology are:

- What is the trace rate at a given vantage point?
- How many vantage points in a group send to the same set of destinations?
- How many of these groups of vantage points, thus what is the total number of vantage points?

- How long does one cycle of data collection run for?

When a highly distributed system is assembled, typically the power of the vantage points is low. Assuming that one doubletree (non MDA) traceroute can be performed per vantage point per minute, a vantage point can probe 360 destinations (6 hr * 60 min/hr * 1 trace/min) in a cycle.

The number of vantage points that are sending probes to a group of common destinations determines the number of times the destinations are probed in a data cycle. Doubletree will prevent the same destination from being probed repeatedly so it is possible to be a little generous with this setting. We suggest that true end host addresses may be used because these machines and nearby routers are ever more powerful in today's Internet and very few probes will actually reach them. On the other hand, the question also arises as to whether a smaller setting here helps to maximise the discovery of cross links between paths in the topology map. Doubletree limits probing to only new territory, so there is some leeway. One feasible possibility is 100 vantage points in a group probing the same set of 360 destinations and 1000 groups for a total of 100,000 vantage points. In one cycle there are 36,000,000 Doubletree Traceroutes (1000 groups of 100 VPs to 360 common destinations).

For comparison CAIDA collects scamper Traceroute data, but does not use Doubletree: "Destinations are selected randomly from each routed IPv4 /24 prefix on the Internet such that a random address in each prefix is probed approximately every 48 hours (one probing cycle)." This approach does not use repeated groups of destinations. A weakness of the Doubletree approach is that if topology changes occur beyond the node that has been previously seen, then the changes will not be discovered. On the other hand some changes are temporary and it is also possible to find indications of changes that are not real but rather artefacts [45]. An artefact can be caused by a load balancer being found in the reverse direction and various of its successors being identified when expired TTL ICMP packets are returned.

Doubletree is unlikely to be extended to include Paris Traceroute behaviour as traces are made up of fragments of traces collected under

differing flow IDs. This means that the advantages of Paris Traceroute where a true path is discovered can not be achieved. Paris Traceroute offers the benefit of following a continuous path through load balancers in a trace. In Doubletree destinations can be repeated from different VPs and the different source address means that it is usually not possible to trace the same path through load balancers by having the same flow ID. Probing from the same VP involving the local stop set and the first part of the trace is also affected by changing destination addresses making maintaining a constant flow ID unlikely.

Chapter 15

Megatree using a trace based simulator

15.1 Introduction

Doubletree stores information about previously seen nodes towards the source and towards the destination of traces. What if this idea of storing topology data was applied to MDA data and information about previously seen load balancers was stored? This concept was named Megatree to indicate the origin of the idea. The simulation of Megatree requires the collection of Traceroute MDA data, and would serve no purpose if classic or non MDA Paris Traceroute data were used. Megatree does not start in the middle of a path like Doubletree but rather starts by performing a conventional Traceroute analysis sending a small number of packets to each hop. The algorithm then scans the initial trace for load balancer divergence and convergence point pairs and stores these in a set, much like Doubletree's stop sets. This approach was later refined, where a single MDA analysis is performed with a few extra exploratory probes to recognise load balancers in the trace in order to reduce traffic even further. The algorithm does not associate global data with particular destinations but rather periodically distributes the locally stored data amongst the vantage points. When and to which vantage points this distribution of data occurs depends on the factor settings of the particular simulation. In Megatree, the global data comes from an accumulation of local data, which means that local and global sets overlap unlike Doubletree. For

this reason the sets are called unshared and shared load balancer data. It should be noted that shared data is obtained by transmitting unshared data to a controller at specified times and then transmitting the collected data to some or all vantage points. In simulations where global data is updated after every set of traces across the vantage points, this will mean that there is no difference between the results for shared only information and shared plus unshared information used.

The main contribution of this work is a new algorithm Megatree, to reduce required probing and cost of collecting topology information that includes load balancers, as is typically collected using Traceroute MDA. Megatree is simulated using warts analysis as a trace based simulator, in the many sources to few destinations direction, including sources windows, and also the few sources to many destinations direction scenarios.

15.2 Experimental design and data collection

The per-flow MDA data collected in run 1 on CAIDA was used for this analysis. A simulation program using warts analysis was written called Basic Internet Simulator Megatree (BISM) that applied the factor levels used in the non event based Doubletree analysis (BISD) where possible. BISM includes a routine that finds the (divergence point, convergence point) pairs in the current trace. A simple approach to finding the convergence point was used, which counted the number of nodes at each hop count. This identified diamonds and a node convergence was usually located at a node count of one (width) as shown in Fig. 15.1. This was actually the next node after convergence in most cases but this simple approach helped to keep the algorithm robust, though in cases where multiple next hops to the true convergence point existed the load balancer could be rediscovered rather than having the trace cost reduced. This would then slightly underestimate possible savings. In asymmetric cases this algorithm sometimes found a later hop after the expected result with this approach.

If a pair was not already stored then it was stored along with the number of probe packets associated with it. This probe number also included

a portion of the bringing forward probes (used to find flow IDs for hops in nested load balancers). The extra probe count derived from bringing forward was based on the number of nodes in the load balancer compared to other load balancers in the same trace.

Analyses with varying factor levels were run in groups, and usually several groups were run simultaneously on the simulation host. The levels of the timing and control factors determined when local (unshared) information was shared with other VPs to add to the global set (shared).

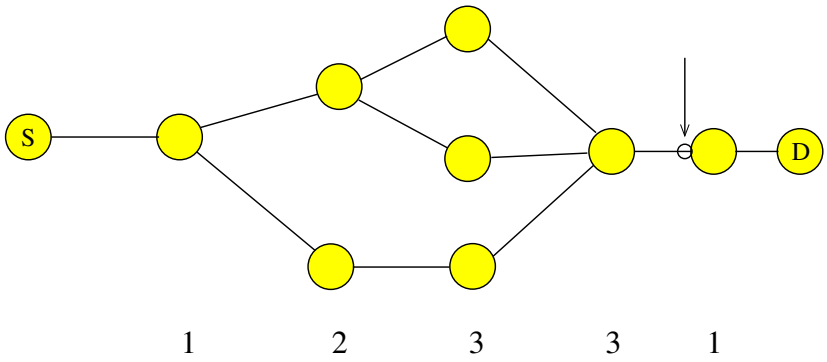

Figure 15.1: Diagram of the convergence point often found one hop later than true convergence, showing hop width counts.

15.2.1 External Validation

A simulator dump of destination addresses, vantage point identities and direction factor levels was checked using an external Perl program. The Perl analysis collected hash table data with direction factor level, vantage points and destination address as keys in that order. If a key already existed it was counted as an error. Zero key errors were found. The total counts of keys across VPs for one direction was divided by 70000 (the number of destinations per VP), to give the number of vantage points that contribute. This was shown to equal the number of VPs specified in the direction factor level, as shown in Table 15.1.

direction mode	count of all traces	divide by 70000
d_few5	350000	5
d_many5	350000	5
d_many5r	350000	5
d_few10	700000	10
d_many10	700000	10
d_many10r	700000	10
d_few20	1400000	20
d_many20	1400000	20
d_many20r	1400000	20
errors	0	

Table 15.1: Processing of traces by the Megatree simulator in different direction modes.

To validate the algorithm several trace results were dumped and analysed by hand to confirm that the correct (divergence, convergence) pairs were added to the unshared and shared sets. In practice there were a few divergence points that paired with a larger number of convergence points and this is a minor flaw in the simple algorithm used. These may be highly complex load balancer diamonds, which occur in small numbers as we reported earlier. There may also be cases where a convergence point is also a divergence point or there may be highly asymmetric load balancer diamonds followed by a load balancer. The use of alias resolution could improve this situation by helping to identify convergence.

In the simulator the unshared and shared sets contained similar data eventually; only the transfer to the shared set was delayed in time sequence to reflect the frequency that distributed updates would occur for the given factor set. As expected, the unshared sets do not ever obtain data from other vantage points.

15.3 Results and discussion

Fig 15.2 shows number of convergence points found for a given divergence point during the simulation. This shows that the cases of many

convergence points found for one divergence point are few and that the simple load balancer finding algorithm performs adequately.

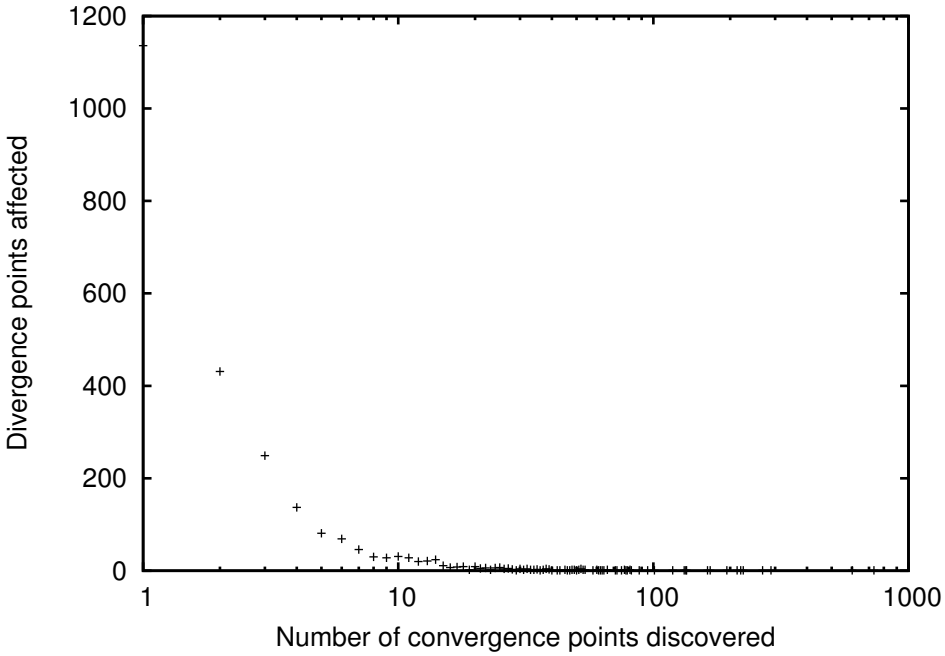

Figure 15.2: Number of divergence points where BISM discovered multiple convergence points.

Fig 15.3 and Fig 15.4 show control traffic and combined control plus probe traffic for the few to many scenario where number of stages is varied or WIN500 is applied. All cases had a control factor setting of AFTERSTAGES. The WIN500 scenario meant that traces were grouped into sets (windows) of 500 simultaneous analyses at each VP and in addition control factor AFTERSTAGES specifies a window as a stage. Control factor levels are explained in Section 14.4 and timing factors are explained in Section 14.3. The control experiment or reference case is 1STAGE 'shared' as the shared information will have no effect in this case. The STAGGERED 'shared' case, where LB information is frequently shared between VPs, shows a 30% improvement over control 1STAGE 'shared' information usage only. All of the 'unshared' and 'unshared and shared' cases show a similar improvement over 1STAGE 'shared'.

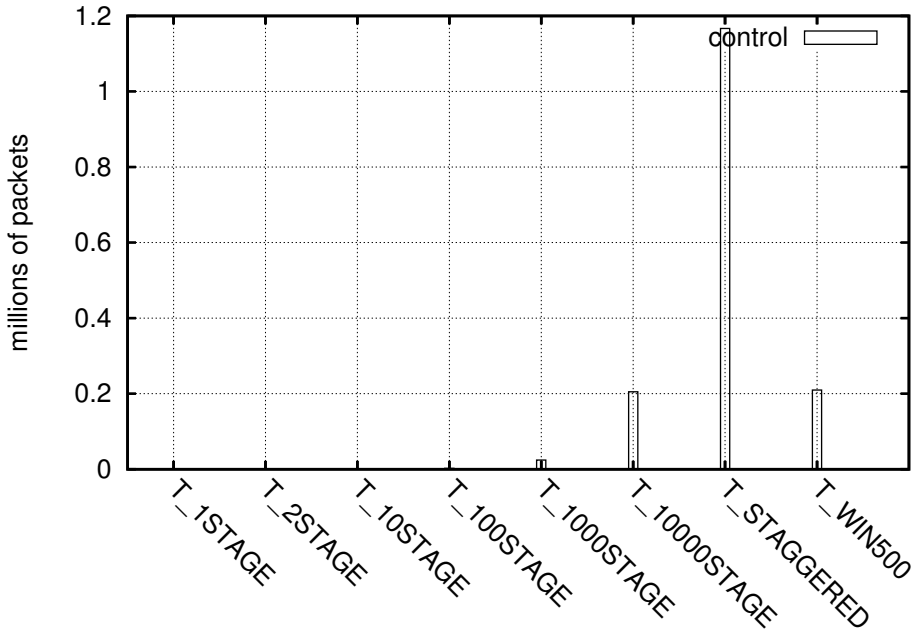

Figure 15.3: Megatree control packets sent, control applied after stages in stage count, few to many. 20 sources.

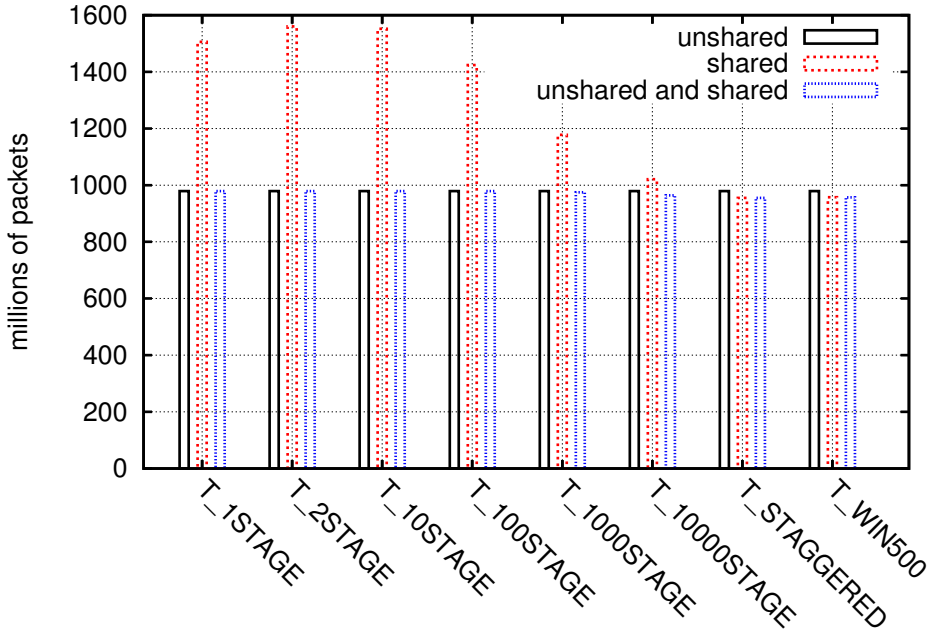

Figure 15.4: Megatree control and probe packets sent, control applied after stages in stage count, few to many. 20 sources.

Fig 15.5 and Fig 15.6 show control traffic and combined control plus probe traffic for the few sources to many destinations scenario where the control factor is varied for STAGGERED and WIN500 levels of the timing factor. These results show that it doesn't really matter what control factor is used, as the the reductions in control traffic are tiny compared to the overall amount of probe traffic required.

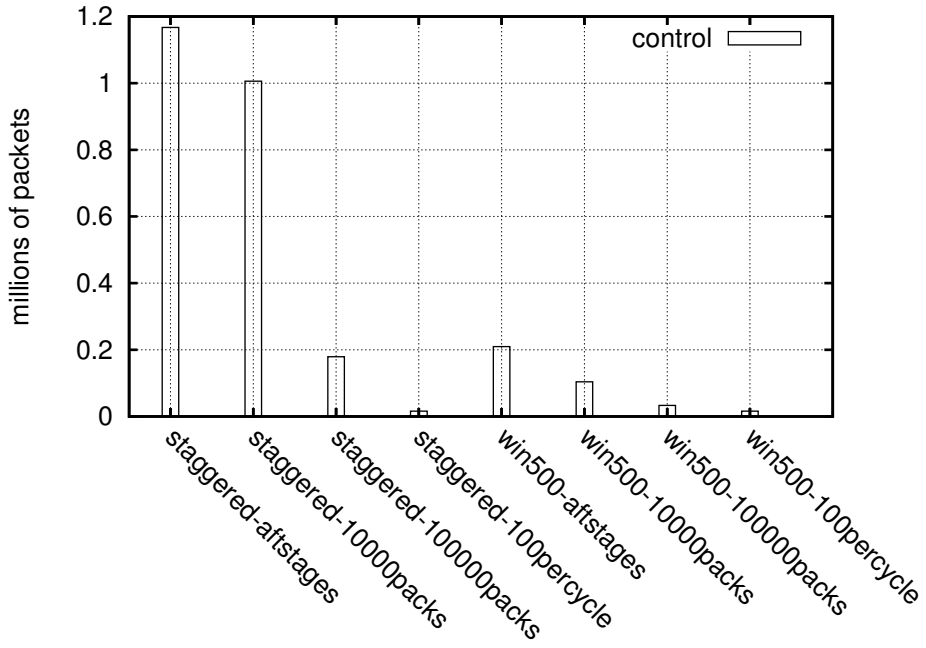

Figure 15.5: Megatree control packets sent for few sources to many destinations where staggered and window 500 cases are studied with different control traffic scenarios.

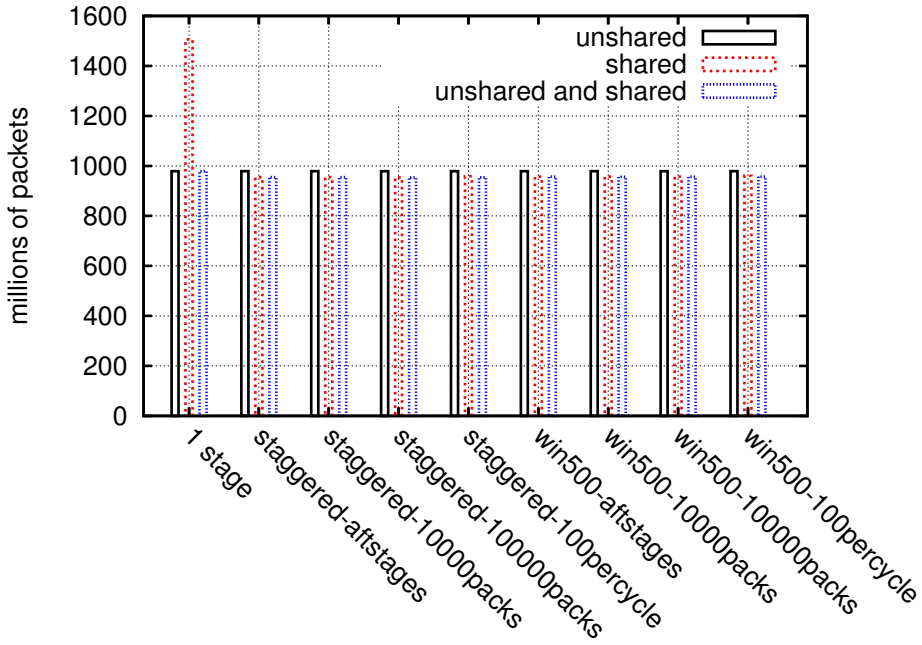

Figure 15.6: Megatree control and probe packets sent, for few sources to many destinations where staggered and window 500 cases are studied with different control traffic scenarios.

Another experiment was performed using a direction of many sources to few destinations with the aim of studying the effects of reduced data set size and sources windows. The keys in the next three figures show the number of traces for each of the few destinations in the data set. The X-axis shows number of sources windows and 1STAGE is baseline for comparison. After each group of sources (VPs) ran its traces, accumulated control information was sent to the next group of sources. In the 1STAGE case there was no control information because the analysis ended before it was sent, as there was no next stage requiring control information.

Fig 15.7 shows probe traffic when unshared information only was used. Some sources were repeated in this analysis, which allows the unshared information to be useful. However this did not have a major effect as unshared only performance was similar to the control. Even when the data set was reduced to those sources with more than two repeats, unshared savings did not contribute greatly. It seems likely that more repetition of

sources would be needed to achieve a greater saving.

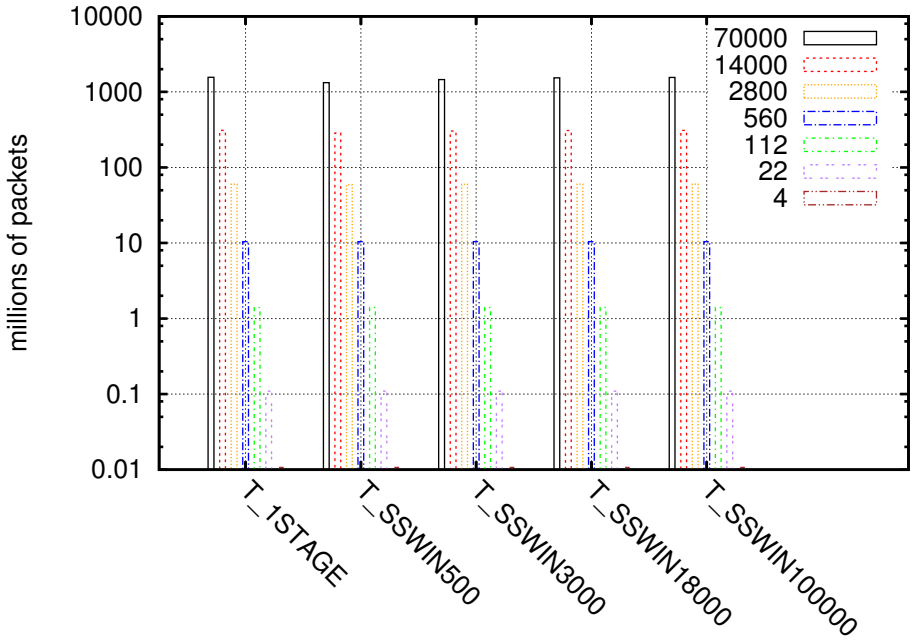

Figure 15.7: Megatree probe packets sent, unshared VP information used only, many sources to few destinations (20), where the number of traces per destination is varied.

Fig 15.8 shows control traffic. Where the sources window was large compared to the data set size, reduced control traffic was seen, as the final stage for which no control traffic was sent was large. Otherwise control traffic increased with sources windows size.

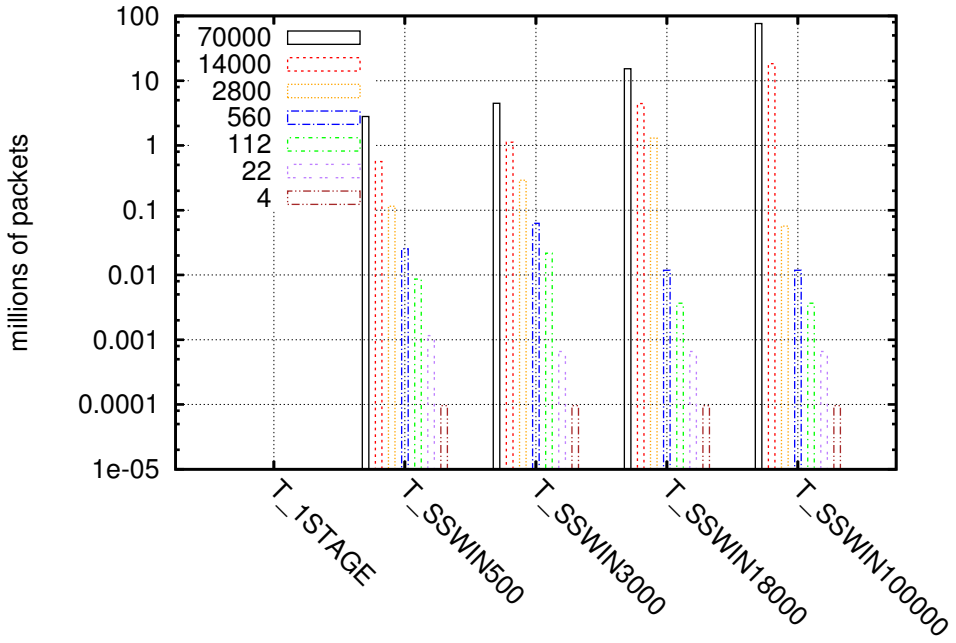

Figure 15.8: Megatree control packets sent, many sources to few destinations (20), where the number of traces per destination is varied.

Fig 15.9 shows probe plus control traffic when 'unshared and shared' information is used. We see a 40% saving in traffic when sources windows are used. Similar savings are seen for reduced scale. This saving is more easily seen on a linear scale, such as in Fig 15.10.

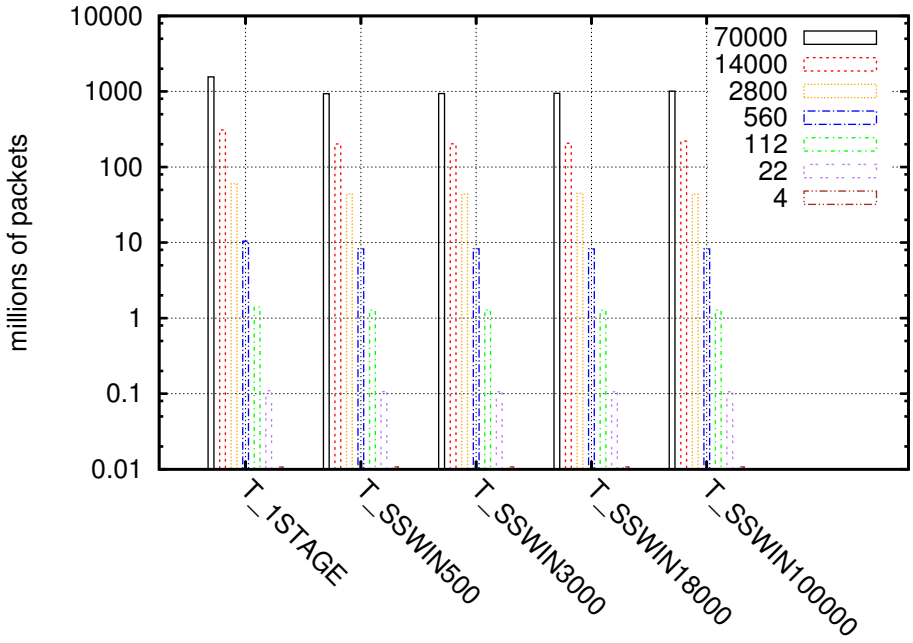

Figure 15.9: Megatree probe and control packets sent, unshared and shared information used, many sources to few destinations (20), where the number of traces per destination is varied.

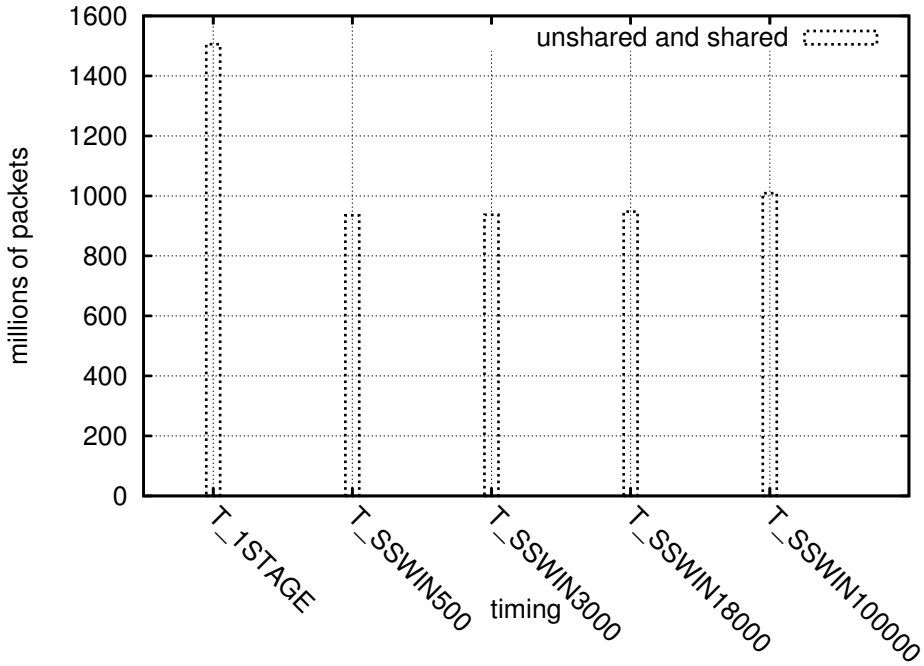

Figure 15.10: Megatree probe and control packets sent, unshared and shared information used, many sources to few destinations. Graph shows sources windows for 20 destinations.

Megatree shows a 30% saving in the few to many case and a 40% saving in the many to few situation when sources windows are applied. In the few to many situation Megatree does not rely on a high degree of repetition of destinations as the same load balancer may be encountered on the path to different destinations. Sources windows usage helps to minimise control traffic and maximise savings in total traffic.

There was very little difference between savings for the two implementations, which were initial Traceroute and exploratory probing, suggesting that the simpler initial Traceroute method could be a useful initial implementation of Megatree.

Megatree offers a useful amount of savings and adapts well to a variety of data collection scenarios. In particular, highly repeated destinations are not vital to its usefulness which means that a greater variety of trace set designs are likely to be viable, including those with reduced repetition of destinations. It is however still desirable to share control information.

15.3.1 Many to many

A similar approach to the proposed Doubletree strategy, in Section 14.3.3, could be used for Megatree, however Megatree does not rely on repeated destinations in the same way, so the same requirement for these would not exist. Under this relaxed structure Megatree could still be extended to include local stop set Doubletree without the global stop set component of the algorithm added in.

When the Doubletree and Megatree results are compared for the MDA data set and sources windows analysis, Megatree uses 20 times as much traffic as Doubletree including savings. The analysis of topology including load balancers in the Internet clearly requires greater volumes of traffic. However, as technology improves this type of approach may become more feasible for regular implementation, especially if the cycle times come to compare more favourably with the time frames over which permanent change is expected to occur in the Internet.

Another possibility for Megatree is incorporation into an approach similar to Ingress Point Spreading [11], to increase load balancer topology discovered while maintaining a reasonable level of economy.

Part IV

Summary and conclusions

Chapter 16

Summary

16.1 Preliminary tasks

16.1.1 Stopping values

In order to configure the scamper MDA algorithm with stopping values based on joint confidence, stopping values were generated using the Veitch algorithm and extrapolation. Using these values, data was collected from the Internet using very high confidence to find as many successors as possible to confirm the utility of these stopping values. This data was used to estimate actual 99% stopping values required by the Internet and it was found that values higher than what we extrapolated, seem to be required in practice. Using the original assumption about successor likelihood to be discovered the extrapolated values that we used were intended to err on the high side, so were likely to waste some extra probing traffic, but should have achieved the level of confidence required if the assumptions were correct.

The Veitch algorithm and subsequently the extrapolated values were based on the assumption that there is an equal chance of finding each subsequent successor. There is a disparity between the stopping values that we predicted and the experimental results, so this suggests that the assumption may not hold in practice.

As computers become more powerful it should become possible to generate more stopping values without the use of extrapolation. This may be based on a modified assumption about the likelihood of finding succes-

sors.

16.1.2 Data collection

Traceroute MDA data was collected from CAIDA and PlanetLab for the analysis of load balancers using UDP, TCP and ICMP probing. Economical and some full analysis of per-destination load balancers was performed.

It would have been desirable to use larger numbers of vantage points for these experiments however access to CAIDA was restricted, though much appreciated. We then decided to use a comparable number of PlanetLab vantage points for balance. With more time it should have been possible to solve the problem of being unable to run all of the scamper modes on PlanetLab. The main other difficulty in using PlanetLab was the high frequency at which vantage points which we were using became unavailable. Because PlanetLab nodes are highly shared, one has to be very careful not to use too much computing power on any one vantage point. The use of the PlanetLab system was however greatly appreciated. Ideally it would be desirable to use a facility like Atlas in the future as this distributed approach is likely to still allow sufficient comparison between data collection runs, along with providing wide coverage of the Internet and its edges, and in particular true forward and reverse path analysis.

One of the problems with the Traceroute data made available by CAIDA is that none of the destinations are repeated from different vantage points. This is aimed at being able to economise on previously seen network structure to avoid repeated mapping analysis. It would be recommended for further analysis of Traceroute data by Doubletree simulators to perform Traceroute data collections with destination repeats built in.

16.2 Direct analysis of load balancer data

16.2.1 Load balancer prevalence

We investigated the population frequencies of load balancer diamonds of three types for three packet probe types to help understand the evolution of the Internet. We observed a small rise for UDP and ICMP prob-

ing per-flow load balancers. A small decline was seen in UDP probing per-destination load balancers. Per packet load balancers, regardless of protocol, demonstrated a small rise in popularity.

It appears that the percentage of paths containing a load balancer is a problematic statistic in terms of repeated analysis of nodes near vantage points. For instance our analysis was susceptible to differences in the type of network structure near to CAIDA as opposed to PlanetLab vantage points. The percentage of nodes containing a load balancer shows similar patterns in the results but is likely to be less susceptible to these interferences. In the future it will also be beneficial for this type of analysis to process larger data sets from more vantage points. It may also become less problematic to probe end-hosts as the amount of probe traffic reaching them becomes less significant due to increasing traffic capacity.

16.2.2 Efficient analysis of per-destination load balancer divergence points

Reduced destination ID analysis of per-destination load balancer populations was performed, because of the possibility of finding relevant information about populations of load balancer diamonds at reduced probe traffic cost. Similar results about diamond frequencies to full destination ID analysis were achieved with greatly reduced probe traffic requirements.

It is important to understand the different goal of the reduced destination ID per-destination analysis, where identification of the diamond divergence point was a priority as was greatly reducing traffic required for analysis. The usual more expensive goal is to identify all load balancer successors.

Further steps are possible in reducing the amount of probe traffic consumed by the reduced per-destination load balancer analysis. This is because the stopping values for this approach can be reduced as well as the number of available destination IDs. However it may be desirable to slightly increase the number of available destination IDs without greatly increasing the amount of traffic to give a higher confidence of finding load balancers. This approach could lend itself to per-flow load

balancers as well, if reduced information provided at this lower traffic level is found to be worthwhile, namely the identity of the main divergence points of load balancer diamonds.

16.2.3 Load balancer turnover

Understanding load balancer turnover, in particular that of the actual load balancing nodes, is helpful in understanding how the building blocks of the Internet are changing. Change to the load balancer population per week ranging from 0.85% to 1.92% were recorded across various measures of load balancer type: unique and distinct, and per-flow and per-destination. There were also measurements for populations of matching successors and populations of identical load balancer internal structure. These gave results that averaged 0.90% whereas the analysis of load balancing nodes averaged 1.72% per week. It seems like internal change within an existing load balancer is less frequent than change in load balancer node populations (which are likely to also affect internal structure).

There is a high rate of rediscovery of changed load balancers in paths to different destinations or from different sources. Therefore if a load balancer was still there after a route change that took the load balancer out of one path, it is likely to be seen in another path.

We note that the rate of permanent change appears to be slower than the rates of change seen with high frequency sampling, as observed by other researchers.

16.2.4 Diamond structure

Diamond structure has been quantified in load balancer populations in the past [10], however we introduced literal width to add to this information as well as to help detect change in the Internet since the time the previous analysis occurred. Some wider diamonds were seen (double for maximum and minimum width), than were seen in previously published work. Longer diamonds were seen in the previous work: 20 maximum compared to 17 maximum for us. Literal width measurements demonstrated that there is a small population of very complex and highly nested

load balancers that would not have been apparent using previous metrics.

There have been changes in the Internet since 2007 and 2009 [10] such as the rise of Content Delivery Networks (CDNs) [14], so we can expect that the Internet has grown and changed. It may be desirable in routine analyses of the Internet to examine highly nested and complex load balancers using the literal width statistic. In the future it may be feasible to use end hosts as destinations as opposed to MIDAR router destinations, and this should allow a greater rate of discovery of complete diamonds.

16.2.5 Five tuple

A collection of non five-tuple fields were analysed using modified Traceroute on the same Internet paths (source, destination pairs). It was of interest to know if the basis of packet streaming and load balancer successor selection in the Internet was consistent across different routers as there may be small populations of load balancers that respond to variation in fields outside the classic five-tuple.

The results suggest that small populations of load balancers that forward traffic based on alternative fields are possible, however the even result across nearly all of the fields tested suggests that these hits may still be due to random noise. On the other hand many of these positives were the same load balancers for different tests. Analysis was further refined to focus on cases where there were few misses, where 89 cases with zero misses were found.

One avenue of future work would be to collect a set of responding load balancers and increase the amount of repetition in the testing, including varying the field being tested to discover with greater certainty that the load balancers do respond to non five-tuple fields.

16.2.6 Black holes in load balancers

A study of the occurrence rate and life span of black holes internal to load balancers was undertaken, as it was desirable to find out if this

scenario was likely to cause connectivity issues of similar importance to other black holes.

A quantity of approximately 10% of the rate of black holes found by Hubble was found for black holes in load balancers. 75% of these were long lived, longer than six hours.

Though we expected discontinuities in portions of networks associated with particular load balancer successors to be automatically avoided by traffic being rerouted to other successors and their subnetworks, we investigated what happens in practise. It appears that there are a small number of cases where black holes in load balancers can block a flow of traffic *e.g.* a TCP connection. This could block a particular users access to a service at a particular destination until a new connection and flow ID is created. It is possible that black holes internal to load balancers may be hidden from systems that find black holes and thus no action would be taken to repair them.

16.2.7 Finding load balancer successors

In this research analysis of the efficiency of finding load balancer successors was performed, using three different modes of choosing flow ID. If new techniques could be found that improve on regular Traceroute MDA and the amount of traffic that it uses, this could be a useful outcome for some facilities that probe the Internet.

Native flow ID incrementing probing in scamper gave the best performance for probes sent and random flow ID selection found more load balancers, but the bit flipping case, which might have been predicted to make better use of modern hash functions lagged behind. These results are not contradictory as different performance characteristics are measured. In the light of the stopping values experiment results, it would be worth using higher joint confidence levels in collecting the data for this experiment, to see if the CDF tail lengths differ between the different modes.

In the future we hope to isolate groups of routers that more efficiently find successors, and investigate hashing functions that are used by popular router devices. This could lead to the design of some more intri-

cate flow ID selection algorithms that better reflect real-world router behaviour.

16.3 Simulation

Simulations of Doubletree and Megatree were performed that were designed to determine if these algorithms could be applied in such a way as to reduce traffic cost and increase efficiency.

Using the IS0 simulator, in Section 13.4 we see that sources windows bring the cost of Doubletree into a useful range (44% saving over Traceroute) when analysing the many to few scenario.

In BISD (Doubletree) the few to many full MDA data scenario, global only analysis gave a slight improvement for 10,000 stages. For the full MDA data set, SSWIN500 showed a 10% improvement over 1STAGE. For the restricted MDA data set, SSWIN500, SSWIN3000 and SSWIN18000 gave 20% reductions for the combined use of local and global stop sets. For the CAIDA data set, sources windows gave a 5% saving for global only analysis and no benefit from local analysis was expected.

In Section 14.3.1 we saw the combined effects of local and global stop sets for Doubletree with many sources to few destinations and a start TTL favouring the use of global stop set information. In particular sources were repeated from three to nine times. This resulted in a useful local saving so repeating a source further (*e.g.* 100 times) in a batch could give a larger local saving. If Doubletree is used on a many sources to many destinations system this would be a feasible recommendation. The number of common destinations in our analysis required to achieve the useful global saving that we observed was very high and this level of repetition may not be feasible in an implementation of many to many analysis. The global saving is unlikely to be as high in such a system. The amount of repetition of destinations is likely to determine if this also applies to the many sources to many destinations situation.

In BISM (Megatree) for few to many, stages analysis showed a 30% saving of traffic due to the use of local data. A similar result was seen when control data utilisation was varied. In the many sources to few des-

tinations scenario 40% savings in traffic were seen using the full data set for sources windows, when shared and unshared data sets were applied.

In considering the requirements to design a regime for many sources to many destinations probing, Megatree has a lesser requirement for repeated destinations than Doubletree, though the greater amount of traffic required by Megatree than Doubletree, even after savings, could still be a stumbling block to regular large scale usage.

Chapter 17

Conclusions

17.1 Questions addressed

Some steps have been taken in our research to better understand topology and improve topology discovery with emphasis on load balancers. Small changes have been observed for load balancer prevalence. A low traffic approach to studying populations of per-destination load balancer diamond divergence points was developed and tested successfully. Load balancer turnover of divergence points (1.72% per week) along with internal structure (0.90% per week) was performed. These measures of sustained change were lower than those reported by others for short term change. For diamond structure wider and slightly shorter diamonds were seen than previously reported by others and a statistic sensitive to complex nesting was introduced. In attempting to determine if per-flow load balancing behaviour consistently uses the fields of the classic five-tuple small populations of non-conforming load balancers may have been found. When attempting to improve efficiency of finding load balancer successors native source port incrementing gave the best performance in terms of probes sent, whereas random source port selection was better at finding load balancers. The new sources windows approach was successful at implementing cost savings in the Doubletree and Megatree simulators, in the many sources to few destinations direction of analysis.

An attempt was made to detect load balancer successors at very high confidence so that Internet sourced stopping values could be predicted by knowing all successors. The results suggested that there may not have

been equal chance of finding all successors of the load balancers tested in some cases.

Black holes in load balancers were detected by running then targeting Paris Traceroute analysis after an initial MDA Traceroute analysis. The number of black holes found was 10% of the rate of regular black holes based on numbers of paths probed and how often.

Large numbers of vantage points were approximated in the simulators by using the many sources to few destinations direction of analysis. Based on the results of this analysis, data collection designs were suggested for Doubletree and Megatree.

Some of the algorithms used in our research used simple designs. In some cases when load balancer diamonds were found this was done using a simple width based approach. When per-destination load balancers were found, this was done in most case using a simple low traffic approach. Also this research was carried out without the use of alias resolution. It may be desirable to improve on these scenarios in the future or it may be desirable to maintain the lower levels of resources associated with this approach given sufficiently satisfactory results.

17.2 Contributions of Thesis

The contributions of this work include:

- Usable stopping values for joint confidence and higher confidence levels than were previously available, and estimated stopping values from Traceroute MDA data, which reflect non idealities of the real world Internet.

- New estimates of load balancer prevalence. Included was an improved statistic that is not affected by proximity of load balancers to vantage points.

- A low traffic means to identify load balancer diamond initial divergence points.

- Quantification of load balancer turnover by analysing IP addresses of interfaces of load balancing routers over time, along with further

analysis of internal diamond changes including study of successor set matches.

- New estimates of statistics that describe load balancer diamonds. The literal width statistic was introduced.

- Investigation of the effect of non classic 5-tuple fields on per-flow load balancers was performed.

- Investigation of the occurrence and life span of black holes in load balancers.

- The efficiency of several ways of assigning port value flow IDs when using Traceroute MDA.

- Simulation where Megatree and sources windows were introduced, and cost analysis of Doubletree and Megatree was performed. An attempt was made to appraise the usefulness of these algorithms for distributed systems like Atlas.

17.3 Significance

The significance of this thesis includes the results of analyses of load balancers both in terms of generation of new statistics and updates of previous analyses. This helps with understanding of how the Internet is evolving in terms of load balancers. Some irregular per-flow load balancers were examined along with factors affecting the efficiency of finding load balancers. Progress in these areas could help improve Internet reliability and performance. A small though important population of black holes was observed in load balancers. Making these black holes in load balancers known along with others that are regularly reported could help improve Internet reliability. The results of simulation of Doubletree and Megatree suggest that systems with many vantage points can operate efficiently using these algorithms to collect topology data. Or it may be that Doubletree and Megatree can be combined with other advances in topology discovery. Efficient discovery of Internet topology is an important tool in monitoring the evolution of the Internet. These are all areas

that are of interest to the designers of modern distributed systems for analysing Internet topology.

17.4 Future work

It may become necessary to extract natural stopping values from larger topology data collection runs on the Internet to help determine if the equal successor probability assumption for load balancing is frequently violated. The small data sets used in our research and stopping value results were susceptible to being affected by a small population of non ideal load balancers. This was because the number of load balancer cases in the cut off zones of the CDFs was often very small and thus highly influential on the choice of natural stopping values. In determining if most of the populations of load balancers seen with varying numbers of successors is ideal, it may be possible to compare ideal CDF distributions with those measured in the analysis to get an idea of the extent of non ideal load balancers (in terms of equal probability of finding successors).

It may be desirable to monitor load balancer prevalence from time to time and in particular to build up a profile of change over time using the new statistic that counts each load balancer interface once. On the other hand a version that makes use of alias resolution may be preferable. This would mean that load balancers are counted once rather than load balancer interfaces.

It may be feasible to perform rudimentary load balancer analysis of the Internet using distributed systems like Atlas with the low traffic approach, as diamond primary divergence points can be located economically. This is also because the degree of address scanning in the case of low traffic per-destination analysis is limited and should be acceptable. This is dependent on whether locating and identifying primary diamond divergence points is considered to provide sufficient information to describe changing populations of load balancers.

Further research on load balancer turnover may be necessary to determine feasible time frames for data collection cycles depending on how often virtual path changes are reversed. If in spite of many short term

changes, many virtual paths are long lived in comparison to this rate of change, it may be feasible to carry out data collection cycles over periods of several days or even more than a week.

It may be helpful to perform large scale load balancer diamond analysis from time to time and use the literal width statistic to locate and study areas of high load balancer complexity in the Internet.

Further work on non classic five-tuple fields affecting load balancing could involve finding populations of such load balancers and performing further tests to confirm the consistency of the behaviour and to find the number of fields that exhibit this behaviour in the isolated load balancers. Determining what types of load balancers behave incorrectly could eventually help improve load balancer performance, as these aberrant load balancers are not likely to stream correctly. This is because changes in fields outside the classic five-tuple are not carefully controlled, as the five-tuple fields are when streaming data in many cases.

Further research on the efficiency of finding load balancer successors could make use of higher confidence levels and other ways of selecting flow IDs. Further use could be made of how particular types of load balancers choose successors based on flow IDs. It may even be possible to identify the type or brand of some load balancers encountered and to group data based on this information.

It may become worthwhile and feasible to look for black holes in load balancers routinely, as there may be a high enough frequency of these black holes with a sufficiently long lifespan to warrant this. The problem is likely to appear in the form of certain TCP connections being blocked.

Sources windows help give Doubletree and Megatree efficiencies, however other advances such as ingress point spreading could be used in conjunction with these to provide even greater improvement. If Megatree is seen as a useful option it may be possible to build some of the behaviour of Doubletree into it as well for non load balancing parts of traces.

Part V

Appendices and bibliography

Appendix A

C program based on the Veitch algorithm for predicting stopping values

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <errno.h>
#include <limits.h>

static long conf = 0;

#define OPT_CONF      0x0001

int string_tolong(const char *str, long *l)
{
  char *endptr;

  *l = strtol(str, &endptr, 0);
  if(*l == 0)
    {
      if(errno == EINVAL) return -1;
    }
  else if(*l == LONG_MIN || *l == LONG_MAX)
```

```
    {
      if(errno == ERANGE) return -1;
    }

  return 0;
}

static void usage(const char *argv0, unsigned int opt_mask)
{
  fprintf(stderr,
          "usage: udptcp [-?c]\n");
  return;
}

static int check_options(int argc, char *argv[])
{
  char ch;
  long lo;
  char *opts = "c:";
  char *opt_conf = NULL;

  while((ch = getopt(argc, argv, opts)) != -1)
    {
      switch(ch)
        {
        case 'c':
          opt_conf = optarg;
          break;
        }
    }

  if(opt_conf != NULL)
    {
      if(string_tolong(opt_conf, &lo) != 0 || lo < 1)
        {
          usage(argv[0], OPT_CONF);
```

```
          return -1;
        }
      conf = lo;
    }

  return 0;
}

double p(int k, int K)
{
  if (k > K + 1) return 1.0;
  return ((double)k)/((double)K + 1);
}

double pk1(int k, int K)
{
  if (k + 1 > K + 1) return 1.0;
  return ((double)k+1)/((double)K + 1);
}

double q(int k, int K)
{
  if (k > K + 1) return 1.0;
  return ((double)K + 1 - k)/((double)K + 1);
}

double qk1(int k, int K)
{
  if (k + 1 > K + 1) return 1.0;
  return ((double)K + 1 - (k+1))/((double)K + 1);
}

int sum = 0;
int countrec(int n)
{
  int i;
```

```
  for (i=0; i<10; i++)
    {
      if (n>1) countrec(n-1);
      sum++;
    }
  return sum;
}


double sumd = 0.0;
static int n[] = { 0, 9, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14,
   i15, i16, i17, i18, i19, i20, i21;
int *in[] = { NULL, &i1, &i2, &i3, &i4, &i5, &i6, &i7, &i8, &i9, &
   i10,
                    &i11, &i12, &i13, &i14, &i15, &i16, &i17, &i18
                        , &i19, &i20, &i21 };
double tkk1 = 0, tk1k1 = 0;

int arraymax(int depth)
{
  switch(depth)
    {
      case 1:
        return n[1]-2;    break;  //7
      case 2:
        return n[2]-2-i1; break;  //15,14...7
      case 3:
        return n[3]-2-i1-i2;     break;
      case 4:
        return n[4]-2-i1-i2-i3;  break;
      case 5:
        return n[5]-2-i1-i2-i3-i4;  break;
      case 6:
        return n[6]-2-i1-i2-i3-i4-i5;  break;
```

```
case 7:
  return n[7]-2-i1-i2-i3-i4-i5-i6;  break;
case 8:
  return n[8]-2-i1-i2-i3-i4-i5-i6-i7;  break;
case 9:
  return n[9]-2-i1-i2-i3-i4-i5-i6-i7-i8;  break;
case 10:
  return n[10]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9;  break;
case 11:
  return n[11]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10;  break;
case 12:
  return n[12]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11;  break;
case 13:
  return n[13]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12;
     break;
case 14:
  return n[14]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13;
      break;
case 15:
  return n[15]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
     i14;  break;
case 16:
  return n[16]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
     i14-i15;  break;
case 17:
  return n[17]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
     i14-i15-i16;  break;
case 18:
  return n[18]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
     i14-i15-i16-i17;  break;
case 19:
  return n[19]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
     i14-i15-i16-i17-i18;  break;
case 20:
  return n[20]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
     i14-i15-i16-i17-i18-i19;  break;
```

```
      case 21:
        return n[21]-2-i1-i2-i3-i4-i5-i6-i7-i8-i9-i10-i11-i12-i13-
           i14-i15-i16-i17-i18-i19-i20;  break;
      default:
        return 0;                break;
    }
}

double factor(int level, int k)
{
  return pow(p(level,k), *in[level]) * q(level,k);
}

double factork1(int level, int k)
{
  return pow(pk1(level,k), *in[level]) * qk1(level,k);
}

double endfactor(int k, int l)
{
  int power, i;
  power = l-k;
  for(i=1; i<k; i++)
    {
      power -= *in[i];
    }
  return pow(p(k,k), power);
}

double endfactork1(int k, int l)
{
  int power, i;
  power = l-k;
  for(i=1; i<k; i++)
    {
      power -= *in[i];
```

```
    }
  return pow(pk1(k,k), power);
}

double alpharecdo(int depth, int k, int l, int nkminus1, int arrmax
   )
{
  for ((*in[depth])=0; (*in[depth])<=arrmax; (*in[depth])++)
    {
      if (depth<k-1) alpharecdo(depth + 1, k, l, nkminus1,
         arraymax(depth+1));
      else
        {
          double tkk1inc = 1.0;
          double tk1k1inc = 1.0;
          int i;
          for (i=1; i<k; i++)
            {
              tkk1inc *= factor(i, k);
              tk1k1inc *= factork1(i, k);
            }
          tkk1inc *=  endfactor(k, l);
          tk1k1inc *=  endfactork1(k, l);
          tkk1 += tkk1inc;
          tk1k1 += tk1k1inc;
        }
    }
  return tkk1 / (tkk1+tk1k1);
}

double alpharec(int k, int l, int nkminus1)
{
  if(k==1)
    {
      tkk1 = pow(p(1,k), l-k);
      tk1k1 = pow(pk1(1,k), l-k);
```

```
      return tkk1 / (tkk1+tk1k1);
    }

  tkk1 = 0.0;
  tk1k1 = 0.0;
  return alpharecdo(1,k,l,nkminus1,arraymax(1));
}


int
main(int argc, char *argv[])
{
  int i, j, k, l;
  int cyclemax = 300;
  double alphakresult;

  static double alpha95[] = { 1, 0.0050, 0.0045, 0.0041, 0.00365,
                              0.00328, 0.00295, 0.00266, 0.00239,
                              0.00215, 0.00194, 0.00174, 0.00157,
                              0.00141, 0.00127, 0.00114, 0.00103,
                              0.0 };
  static double alpha99[] = { 1, 0.0010, 0.00090, 0.00081,
         0.000729, 0.000656, 0.000590, 0.000531, 0.000478,
         0.000430, 0.000387, 0.000349, 0.000314, 0.000282,
         0.000254, 0.000229, 0.000206, 0.0 };
  static double alpha999[] = { 1, 0.00010, 0.000090, 0.000081,
         0.0000729, 0.0000656, 0.000059, 0.0000531, 0.0000478,
         0.000043, 0.0000387, 0.0000349, 0.0000314, 0.0000282,
         0.0000254, 0.0000229, 0.0000206, 0.0 };
  static double alpha9999[] = { 1, 0.000010, 0.000009, 0.0000081,
         0.00000729, 0.00000656, 0.0000059, 0.00000531, 0.00000478,
         0.0000043, 0.00000387, 0.00000349, 0.00000314, 0.00000282,
         0.00000254, 0.00000229, 0.00000206, 0.0 };
  static double alpha99999[] = { 1, 0.0000010, 0.0000009,
        0.00000081, 0.000000729, 0.000000656, 0.00000059,
```

```
        0.000000531, 0.000000478, 0.00000043, 0.000000387,
        0.000000349, 0.000000314, 0.000000282, 0.000000254,
        0.000000229, 0.000000206, 0.0 };  static double *
           alphaarray[] = { alpha95, alpha99, alpha999, alpha9999,
            alpha99999 };
  int mindiff[] = { 7, 8, 10, 13, 14 };

  if(check_options(argc, argv) != 0)
    return -1;

  if (conf > 4 || conf < 0) conf = 0;

  static double *alpha = NULL;
  alpha = alphaarray[conf];


  for (k=1; k<=16; k++)
    {
      for (l=n[k-1] + mindiff[conf]; l<cyclemax; l++)
        {
          alphakresult = alpharec(k,l,n[k-1]);
          if (alphakresult <= alpha[k])
            {
               printf("k = %d n = %d %.9f found a[k] %.9f\n", k, l,
                    alphakresult, alpha[k]);
               n[k] = l;
               break;
            }
        }
    }



  return 0;
}
```

Appendix B

C program designed to randomly simulate successor selection in load balancers to predict stopping values

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cstdlib>
#include <ctime>
#include <time.h>
#include <vector>
#include <set>
#include <unistd.h>
#include <errno.h>
#include <limits.h>

using namespace std;

static long conf = 0;

#define OPT_CONF      0x0001
```

```
int string_tolong(const char *str, long *l)
{
  char *endptr;

  *l = strtol(str, &endptr, 0);
  if(*l == 0)
    {
      if(errno == EINVAL) return -1;
    }
  else if(*l == LONG_MIN || *l == LONG_MAX)
    {
      if(errno == ERANGE) return -1;
    }

  return 0;
}

static void usage(const char *argv0, unsigned int opt_mask)
{
  fprintf(stderr,
          "usage: ./stoppingsim [-?c]\n");
  return;
}

static int check_options(int argc, char *argv[])
{
  char ch;
  long lo;
  char *opts = "c:";
  char *opt_conf = NULL;

  while((ch = getopt(argc, argv, opts)) != -1)
    {
      switch(ch)
        {
        case 'c':
```

```
          opt_conf = optarg;
          break;
        }
    }

  if(opt_conf != NULL)
    {
      if(string_tolong(opt_conf, &lo) != 0 || lo < 1)
        {
          usage(argv[0], OPT_CONF);
          return -1;
        }
      conf = lo;
    }

  return 0;
}

int
main(int argc, char *argv[])
{
  int i, j, k, l, m, n, ni, y, afcount, cycles = 10000;
  double wantconf = 0.95;
  double confl = 0;

  time_t rawtimestart, rawtimeend;
  struct tm * timeinfo;

  ni = 8;

  int start95[] = { 0, 9, 17, 25, 34, 43, 53, 62, 72, 80, 91, 100,
     110, 120,
                    129, 140, 150, 159, 171, 181, 190 };
  int start99[] = { 0, 10, 20, 30, 41, 51, 61, 73, 85, 96, 108,
     116, 130,
                    140, 151, 162, 176, 184, 200, 212, 221 };
```

```
int start999[] = { 0, 10, 22, 32, 47, 59, 74, 86, 98, 113, 125,
    140, 153,
                   168, 177, 190, 199, 224, 230, 251, 257 };
int start9999[] = { 0, 13, 26, 40, 56, 73, 82, 95, 103, 128, 144,
     165, 170,
                    181, 198, 217, 234, 243, 252, 265, 279 };
int start99999[] = { 0, 14, 27, 42, 56, 79, 88, 96, 117, 128,
    138, 159, 162,
                     189, 205, 224, 243, 248, 269, 280, 296 };

vector<int> myvector;
vector<int>::iterator it;

set< int > tmpset ;
srand((unsigned)time(0));

if(check_options(argc, argv) != 0)
  return -1;

if (conf > 4 || conf < 0) conf = 0;

double confidencearray[] = { 0.95, 0.99, 0.999, 0.9999, 0.99999 }
    ;
int cyclearray[] = { 10000, 50000, 500000, 5000000, 50000000 };
int *start[] = { start95, start99, start999, start9999,
    start99999 };

wantconf = confidencearray[conf];
cycles = cyclearray[conf];

time ( &rawtimestart );
timeinfo = localtime ( &rawtimestart );
printf ( "Beginning local time and date: %s\n", asctime (timeinfo
    ) );

printf("confidence is %.5f\n", wantconf);
```

```
printf("cycles: %d\n", cycles);
printf("(1-conf) * cycles: %d\n",
        (int)((((double)1) - wantconf) * ((double)cycles)));

for (k=2; k<=(20 + 1); k++)
{
  int *array = start[conf];
  int ninit = array[k-1];
  for (n=ninit; n<600; n++)
    {
      afcount = 0;
      for (i=0; i<cycles; i++)
        {
          int allfound = 1;
          for (j=0; j<30; j++)
            {
              for (m=0; m<n; m++)
                {
                  int random_integer;
                  random_integer = (rand() % k);
                  tmpset.insert( random_integer );
                }
              if (tmpset.size() < k)
                {
                  allfound = 0;
                  break;
                }
              tmpset.clear();
            }
          if (allfound == 1) afcount++;
        }

      confl = ((double)afcount) / ((double)cycles);
      printf("k = %d, n = %d, confidence is %f\n", k-1, n, confl)
          ;
      if (wantconf <= confl)
```

```
          {
            printf("Found %.5f\n", wantconf);
            myvector.push_back (n);
            break;
          }
      }
  }

  for ( it=myvector.begin() ; it < myvector.end(); it++ )
    printf("%d ", *it);

  time ( &rawtimeend );
  timeinfo = localtime ( &rawtimeend );
  printf ( "\nEnd local time and date: %s\n", asctime (timeinfo) );

  double dif = difftime (rawtimeend,rawtimestart);
  printf ("It took %.2lf hours to run.\n", dif/3600.0 );

  printf("\nEnd\n");
  return 0;
}
```

Appendix C

C program designed to gather CDF data for load balancers with differing successor counts

```
static void setlbprobenocount_65(const sc_traceset_t *set,
   sc_ascountset_t **probecountsets, sc_addrset_t *addrset)
{
  scamper_tracelb_t *lb = set->traces[0];
  scamper_tracelb_node_t *node;
  int i, j, k, m, l, index;
  sc_ascountset_t *probecountset = NULL;

  for(i=0; i<lb->nodec; i++)
    {
      node = lb->nodes[i];

      if(node->linkc <= 1)
        continue;
      k = 0;
      for(j=0; j<node->linkc; j++)
        if(node->links[j]->to != NULL &&
           scamper_addr_cmp(node->addr, node->links[j]->to->addr)
               != 0)
          {
            k++;
          }
```

```
      if (k < 2 || k > 10) continue;
      index = k-2;
      probecountset = probecountsets[index];

#ifndef SCAMPER_ANALYSIS_DO_REPEATS
              if(sc_addrset_get(addrset, node->addr) == NULL)
                {
                  sc_addrset_add(addrset,
                      scamper_addr_alloc(SCAMPER_ADDR_TYPE_IPV4,
                      node->addr->addr));
                }
              else
                continue;
#endif

      sc_probeset_t *probeset = NULL;
      probeset = sc_probeset_alloc();

      for(j=0; j<node->linkc; j++)
        {
          if(node->links[j]->to != NULL &&
              scamper_addr_cmp(node->addr, node->links[j]->to->addr
                  ) != 0)
          {
            if(node->links[j]->hopc == 1)
              {
                for(m=0; m<node->links[j]->hopc; m++) // sets
                  {
                    int fi = 0;  // upgrade
                    for(l=0; l<node->links[j]->sets[m]->probec; l
                        ++) // probes in set
                      {
                        int locfi = node->links[j]->sets[m]->
                            probes[l]->flowid;  // upgrade
                        if(fi == locfi) continue;  // upgrade
```

```
                    fi = locfi;  // upgrade

                    sc_probe_t *probe = NULL;
                    probe = sc_probe_alloc();
                    probe->successor = j;
                    probe->probe = node->links[j]->sets[m]->
                       probes[l];
                    sc_probeset_add(probeset, probe);
                  }
              }
          }
      }
  }

sc_uint16set_t *uint16set = sc_uint16set_alloc();
for(j=0; j<probeset->count; j++)
  {
        if(sc_uint16set_get(uint16set, &probeset->probes[j]->
           successor) == NULL)
          {
            uint16_t *no = sc_uint16_alloc();
            if (no == NULL)
              {
                fprintf(stderr, "could not malloc uint16\n");
                exit(1);
              }
            *no = probeset->probes[j]->successor;

            sc_uint16set_add(uint16set, no);
          }
        if(uint16set->count == k)
          {
                  sc_ascount_t *probecount = NULL;
                  uint16_t val = j+1;

                  if((probecount =
```

```
                            sc_ascountset_as_get(probecountset, &
                            val)) != NULL)
                          {
                            probecount->count++;
                          }
                        else
                          {
                            probecount = sc_ascount_alloc();
                            probecount->as = val;
                            probecount->count++;
                            sc_ascountset_add(probecountset,
                                probecount);
                          }
                        break;
                }
        }



      sc_uint16set_free(uint16set);
      sc_probeset_free(probeset);
    }
}

static void finish_65(void)
{
  int i, j;
  int udpgtotal = 0;
  int tcpgtotal = 0;

  sc_ascountset_t *probecountudp[] = { probecountudp2,
     probecountudp3, probecountudp4, probecountudp5,
                 probecountudp6, probecountudp7, probecountudp8,
                    probecountudp9, probecountudp10 };

  sc_ascountset_t *probecounttcp[] = { probecounttcp2,
```

```
    probecounttcp3, probecounttcp4, probecounttcp5,
                probecounttcp6, probecounttcp7, probecounttcp8,
                   probecounttcp9, probecounttcp10 };

if (foreach_65_icmp == 0)
{
 printf("TCP set probe counts for given LB valencies, new 99%%\n")
     ;
 for(i=0; i<9; i++)
   {
     int total = 0;
     printf("Valency: %d\n", i+2);
     for(j=0; j<probecounttcp[i]->count; j++)
       {
         printf("Probe number: %d, count %d\n", probecounttcp[i]->
             ascount[j]->as,
                  probecounttcp[i]->ascount[j]->count);
         total += probecounttcp[i]->ascount[j]->count;
       }
     printf("total %d\n",total);
     tcpgtotal += total;
   }
 printf("tcpgtotal %d\n",tcpgtotal);
}

 if (foreach_65_icmp == 0)
  printf("UDP set probe counts for given LB valencies, new 99%%\n"
      );
 else
  printf("ICMP set probe counts for given LB valencies, new 99%%\n
      ");

 for(i=0; i<9; i++)
   {
     int total = 0;
     printf("Valency: %d\n", i+2);
```

```
      for(j=0; j<probecountudp[i]->count; j++)
        {
          printf("Probe number: %d, count %d\n", probecountudp[i]->
              ascount[j]->as,
                   probecountudp[i]->ascount[j]->count);
          total += probecountudp[i]->ascount[j]->count;
        }
      printf("total %d\n",total);
      udpgtotal += total;
    }
  if (foreach_65_icmp == 0)
   printf("udpgtotal %d\n",udpgtotal);
  else
   printf("icmpgtotal %d\n",udpgtotal);

  sc_ascountset_free(probecountudp2);
  sc_ascountset_free(probecountudp3);
  sc_ascountset_free(probecountudp4);
  sc_ascountset_free(probecountudp5);
  sc_ascountset_free(probecountudp6);
  sc_ascountset_free(probecountudp7);
  sc_ascountset_free(probecountudp8);
  sc_ascountset_free(probecountudp9);
  sc_ascountset_free(probecountudp10);
  sc_ascountset_free(probecounttcp2);
  sc_ascountset_free(probecounttcp3);
  sc_ascountset_free(probecounttcp4);
  sc_ascountset_free(probecounttcp5);
  sc_ascountset_free(probecounttcp6);
  sc_ascountset_free(probecounttcp7);
  sc_ascountset_free(probecounttcp8);
  sc_ascountset_free(probecounttcp9);
  sc_ascountset_free(probecounttcp10);
}
```

Bibliography

[1] About Traceroute deficiencies. *http://www.paris-traceroute.net/about*.

[2] Archipelago Measurement Infrastructure. *http://www.caida.org/projects/ark/*.

[3] RIPE: Atlas project. *https://atlas.ripe.net/*.

[4] RIPE: Test Traffic Measurement Service. *http://www.ripe.net/data-tools/stats/ttm/test-traffic-measurement-service*.

[5] Post-Hoc Definition and Types of Post Hoc Tests. *http://www.statisticshowto.com/post-hoc/*, 2016.

[6] Understanding BGP Multipath. *http://www.juniper.net/documentation/en_US/junos15.1/topics/concept/bgp-multipath-understanding.html*, 2016.

[7] B. Augustin, T. Friedman, and R. Teixeira. Multipath tracing with Paris traceroute. In *Workshop on End-to-End Monitoring*, 2007.

[8] 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 '06, pages 153–158, New York, NY, USA, 2006. ACM.

[9] 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 '07, pages 149–160, New York, NY, USA, 2007. ACM.

[10] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring multipath routing in the internet. *IEEE/ACM Trans. Netw.*, 19:830–840, June 2011.

[11] Guillermo Baltra, Robert Beverly, and Geoffrey G. Xie. *Passive and Active Measurement: 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings*, chapter Ingress Point Spreading: A New Primitive for Adaptive Active Network Mapping, pages 56–66. Springer International Publishing, Cham, 2014.

[12] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. MultiPath TCP: From Theory to Practice. In *IFIP Networking, Valencia*, May 2011.

[13] Ethan Blanton and Mark Allman. On Making TCP More Robust to Packet Reordering. *SIGCOMM Comput. Commun. Rev.*, 32(1):20–30, January 2002.

[14] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content Delivery Networks*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[15] Zhiruo Cao, Zheng Wang, and E. Zegura. Performance of hashing-based schemes for Internet load balancing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 332 –341 vol.1, 2000.

[16] Center for Applied Internet Data Analysis (CAIDA). The IPv4 Routed /24 Topology Dataset. *http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml*, 2015.

[17] Cisco Systems. Load balancing with Cisco express forwarding. *Cisco Application Note*, January 1998.

[18] Kimberly Claffy, Young Hyun, Ken Keys, Marina Fomenkov, and Dmitri Krioukov. Internet Mapping: From Art to Science. In *Pro-*

ceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security, pages 205–211, Washington, DC, USA, 2009. IEEE Computer Society.

[19] Ítalo Cunha, Renata Teixeira, and Christophe Diot. Measuring and characterizing end-to-end route dynamics in the presence of load balancing. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pages 235–244, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] Italo Cunha, Renata Teixeira, Darryl Veitch, and Christophe Diot. Predicting and tracking internet path changes. *SIGCOMM Comput. Commun. Rev.*, 41:122–133, August 2011.

[21] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. Analysis of Country-wide Internet Outages Caused by Censorship. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 1–18, New York, NY, USA, 2011. ACM.

[22] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Deployment of an Algorithm for Large-Scale Topology Discovery. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 24:2210–2220, 2006.

[23] Benoit Donnet, Bruno Baynat, and Timur Friedman. Improving retouched Bloom filter for trading off selected false positives against false negatives . *Computer Networks*, 54(18):3373 – 3387, 2010.

[24] Benoit Donnet, Université Catholique, De Louvain, Timur Friedman, Université Pierre, Marie Curie, and CNRS. Internet Topology Discovery: a Survey. In *In IEEE Communications Survey and Tutorials*, 2007.

[25] Benoit Donnet and Timur Friedman. Topology Discovery Using an Address Prefix Based Stopping Rule. In Carlos Kloos, Andrés Marín,

and David Larrabeiti, editors, *EUNICE 2005: Networks and Applications Towards a Ubiquitously Connected World*, volume 196 of *IFIP International Federation for Information Processing*, pages 119–130. Springer Boston, 2006. 10.1007/0-387-31170-X_9.

[26] Benoit Donnet, Timur Friedman, and Mark Crovella. Improved Algorithms for Network Topology Discovery. In Constantinos Dovrolis, editor, *Passive and Active Network Measurement*, volume 3431 of *Lecture Notes in Computer Science*, pages 149–162. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-31966-5_12.

[27] Benoit Donnet, Bradley Huffaker, Timur Friedman, et al. Evaluation of a large-scale topology discovery algorithm. In *International Workshop on IP Operations and Management*, pages 193–204. Springer, 2006.

[28] Tobias Flach, Ethan Katz-Bassett, and Ramesh Govindan. Quantifying Violations of Destination-based Forwarding on the Internet. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 265–272, New York, NY, USA, 2012. ACM.

[29] A. Garrett, G. Drenan, C. Morris, and Inc Juniper Networks. *Juniper Networks Field Guide and Reference*. Addison-Wesley, 2002.

[30] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. *Passive and Active Network Measurement: 9th International Conference, PAM 2008, Cleveland, OH, USA, April 29-30, 2008. Proceedings*, chapter The Flattening Internet Topology: Natural Evolution, Unsightly Barnacles or Contrived Collapse?, pages 1–10. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[31] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). *https://tools.ietf.org/html/rfc1930*, 1996.

[32] B. Huffaker, M. Fomenkov, and k. claffy. Internet Topology Data

Comparison. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), May 2012.

[33] V Jacobson. Traceroute software. *Lawrence Berkeley Laboratories*, 1988.

[34] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. Dynamic Load Balancing Without Packet Reordering. *SIGCOMM Comput. Commun. Rev.*, 37(2):51–62, March 2007.

[35] Ethan Katz-Bassett, Harsha V. Madhyastha, John P. John, Arvind Krishnamurthy, David Wetherall, and Thomas Anderson. Studying black holes in the Internet with Hubble. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 247–262, Berkeley, CA, USA, 2008. USENIX Association.

[36] Ethan Katz-Bassett, Colin Scott, David R. Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V. Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. LIFEGUARD: practical repair of persistent route failures. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 395–406, New York, NY, USA, 2012. ACM.

[37] K. Keys, Y. Hyun, M. Luckie, and k. claffy. Internet-Scale IPv4 Alias Resolution with MIDAR. *IEEE/ACM Transactions on Networking*, 2012.

[38] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, October 2011.

[39] R.R. Kompella, J. Yates, A. Greenberg, and A.C. Snoeren. Detection and Localization of Network Black Holes. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2180–2188, 2007.

[40] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of Internet stability and backbone failures. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 278–285, June 1999.

[41] Matthew Luckie. Scamper: a scalable and extensible packet prober for active measurement of the Internet. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 239–245, New York, NY, USA, 2010. ACM.

[42] Matthew Luckie. Scamper. *https://www.caida.org/tools/measurement/scamper/*, 2014.

[43] Matthew Luckie, Amogh Dhamdhere, David Clark, Bradley Huffaker, et al. Challenges in Inferring Internet Interdomain Congestion. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 15–22. ACM, 2014.

[44] Matthew Luckie, Young Hyun, and Bradley Huffaker. Traceroute probe method and forward IP path inference. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 311–324, New York, NY, USA, 2008. ACM.

[45] Pietro Marchetta, Valerio Persico, Ethan Katz-Bassett, and Antonio Pescapé. Don't trust traceroute (completely). *ACM CoNEXT Student workshop*, 2013.

[46] Anthony J. McGregor. Towards Internet scale simulation. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011.

[47] T. McGregor. DoubleTree with many sources. In *ICIMP11: The Sixth International Conference on Internet Monitoring and Protection*, Sint Maartin Island, The Netherland Antilies, March 2011. IARIA.

[48] T. McGregor, H.W. Braun, and J. Brown. The NLANR network analysis infrastructure. *IEEE Communications Magazine*, 38, 2000.

[49] Tony McGregor, Shane Alcock, and Daniel Karrenberg. The RIPE NCC Internet Measurement Data Repository. In *Proceedings of the 11th International Conference on Passive and Active Measurement*, PAM'10, pages 111–120, Berlin, Heidelberg, 2010. Springer-Verlag.

[50] John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[51] Vern Paxson. End-to-end Internet packet dynamics. *SIGCOMM Comput. Commun. Rev.*, 27:139–152, October 1997.

[52] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *In Proceedings of the 7th USENIX Symp. on Operating Systems Design and Implementation (OSDI*, 2006.

[53] Jon Postel. RFC791 Internet Protocol, 1981. *Internet Engineering Task Force*, 2011.

[54] Lin Quan, John Heidemann, and Yuri Pradkin. Detecting internet outages with precise active probing (extended). *USC/Information Sciences Institute, Tech. Rep*, 2012.

[55] Lin Quan, John Heidemann, and Yuri Pradkin. Trinocular: Understanding Internet Reliability Through Adaptive Probing. *SIGCOMM Comput. Commun. Rev.*, 43(4):255–266, August 2013.

[56] Yuval Shavitt and Eran Shir. DIMES: let the internet measure itself. *SIGCOMM Comput. Commun. Rev.*, 35:71–74, October 2005.

[57] Rob Sherwood, Adam Bender, and Neil Spring. Discarte: a disjunctive internet cartographer. *SIGCOMM Comput. Commun. Rev.*, 38:303–314, August 2008.

[58] Rob Sherwood and Neil Spring. Touring the Internet in a TCP Sidecar. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 339–344, New York, NY, USA, 2006. ACM.

[59] Charles Simpson and George Riley. NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements. In Chadi Barakat and Ian Pratt, editors, *Passive and Active Network Measurement*, volume 3015 of *Lecture Notes in Computer Science*, pages 168–174. Springer Berlin / Heidelberg, 2004.

[60] Neil Spring, Larry Peterson, Andy Bavier, and Vivek Pai. Using PlanetLab for network research: myths, realities, and best practices. *SIGOPS Oper. Syst. Rev.*, 40:17–24, January 2006.

[61] W. Richard Stevens. *TCP/IP Illustrated (Vol. 1): The Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[62] Mark J van der Laan, Sandrine Dudoit, and Katherine S Pollard. Augmentation procedures for control of the generalized family-wise error rate and tail probabilities for the proportion of false positives. *Statistical applications in genetics and molecular biology*, 3(1), 2004.

[63] Darryl Veitch, Brice Augustin, Renata Teixeira, and Timur Friedman. Failure control in multipath route tracing. In *in Proc. IEEE INFOCOM*, 2009.

[64] Daniel G. Waddington, Fangzhe Chang, Ramesh Viswanathan, and Bin Yao. Topology discovery for public IPv6 networks. *Computer Communication Review*, 33:59–68, 2003.

[65] Eric W. Weisstein. Bonferroni Correction. *From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/BonferroniCorrection.html*, 2016.

[66] M. Zhang, C. Zhang, V.S. Pai, L. Peterson, and R. Wang. Planetseer: Internet path failure monitoring and characterisation in wide-area services. In *OSDI*, 2004.