Per-Hop Internet Measurement Protocols

A thesis

submitted in partial fulfilment

of the requirements for the degree

of

Doctor of Philosophy

at

The University of Waikato

by

Matthew Luckie

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2006

Abstract

Accurately measuring per-hop packet dynamics on an Internet path is difficult. Currently available techniques have many well-known limitations that can make it difficult to accurately measure per-hop packet dynamics. Much of the difficulty of per-hop measurement is due to the lack of protocol support available to measure an Internet path on a per-hop basis. This thesis classifies common weaknesses and describes a protocol for per-hop measurement of Internet packet dynamics, known as the IP Measurement Protocol, or IPMP. With IPMP, a specially formed probe packet collects information from intermediate routers on the packet's dynamics as the packet is forwarded. This information includes an IP address from the interface that received the packet, a timestamp that records when the packet was received, and a counter that records the arrival order of echo packets belonging to the same flow. Probing a path with IPMP allows the topology of the path to be directly determined, and for direct measurement of per-hop behaviours such as queueing delay, jitter, reordering, and loss. This is useful in many operational situations, as well as for researchers in characterising Internet behaviour.

IPMP's design goals of being tightly constrained and easy to implement are tested by building implementations in hardware and software. Implementations of IPMP presented in this thesis show that an IPMP measurement probe can be processed in hardware without delaying the packet, and processed in software with little overhead. This thesis presents IPMP-based measurement techniques for measuring per-hop packet delay, jitter, loss, reordering, and capacity that are more robust, require less probes to be sent, and are potentially more accurate and convenient than corresponding measurement techniques that do not use IPMP.

Acknowledgements

I am fortunate to have been advised by Tony McGregor. He provided continuous encouragement, support, and patience throughout the course of this work, as well as a continuous stream of feedback that was always helpful in completing this work. Thanks also go to Murray Pearson and John Cleary, members of my advisory board, who provided feedback in the preparation of this thesis.

Numerous members of the WAND network research group at Waikato helped me throughout this work. James Spooner and Alan Holt carefully read drafts of this thesis and supplied perspectives that come with real-world experience; these perspectives greatly improved this work. In addition, James provided a hardware implementation of an early version of IPMP and provided guidance on how it could become more hardware-friendly. James also supplied a crash course in writing VHDL that enabled me to complete the hardware implementation, simulate it, and then tidy up the code so that it would synthesise on WAND's non-blocking cross-bar switch. I also pestered Matt Brown, Jamie Curtis, Matt Jervis, and Brendon Jones for assistance with various WAND resources, and they were always helpful.

Other researchers provided feedback on my work at conferences, and as part of the peer-review process. Mark Foster (NASA Ames Research Center) provided public encouragement and perspective, as well as a review of a draft of this thesis.

Finally, many other people supported me prior to, and over the course of this work. My thanks and love go to Jo Ball, my parents Wendy and Kevin, as well as Jenny Ball, Mike Wright, Nick Grieve, Minty, Monty, and Cassie.

This work was funded, in part, by a University of Waikato doctoral scholarship, a University of Waikato computer science scholarship, NLANR/MNA under cooperative agreement no. ANI-0129677, and a scholarship from the WIDE group.

Contents

List of Figures

List of Tables

List of Publications

M. Luckie, A. McGregor, and H.-W. Braun, “Towards Improving Packet Probing Techniques,” in *Proceedings of ACM/SIGCOMM Internet Measurement Workshop 2001*, San Francisco, CA, Nov. 2001, pp. 145-151.

M. Luckie and A. McGregor, “IPMP: IP Measurement Protocol,” in *Proceedings of Passive and Active Measurement Workshop 2002*, Fort Collins, CO, Apr. 2002, pp. 168-176.

M. Luckie and A. McGregor, “Segmentation of Internet Paths for Capacity Estimation,” abstract presented at the *IMRG/CAIDA ISMA Bandwidth Estimation Workshop 2003*, San Diego, CA, Dec. 2003.

M. Luckie and A. McGregor, “Path Diagnosis with IPMP,” in *Proceedings of ACM/SIGCOMM Network Troubleshooting Workshop 2004*, Portland, OR, Aug. 2004, pp. 259-264.

K. Cho, M. Luckie, and B. Huffaker, “Identifying IPv6 Network Problems in the Dual-Stack World,” in *Proceedings of ACM/SIGCOMM Network Troubleshooting Workshop 2004*, Portland, OR, Aug. 2004, pp. 283-288.

M. Luckie, K. Cho, and B. Owens, “Inferring and Debugging Path MTU Discovery Failures,” in *Proceedings of Internet Measurement Conference 2005*, San Francisco, CA, Oct. 2005, pp. 193-198.

List of Acronyms Used

AS	Autonomous System
ASes	Autonomous Systems
ASIC	Application-Specific Integrated Circuit
ASN	Autonomous System Number
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
BER	Basic Encoding Rules
BGP	Border Gateway Protocol
BPF	Berkeley Packet Filter
BSD	Berkeley Software Distribution
CDMA	Code Division Multiple Access
CE	Congestion Experienced
CM	Capacity Mode
CPU	Central Processing Unit
CRCnet	Connecting Rural Communities Network
DF	Don’t Fragment
DoS	Denial of Service

DSP	Digital Signal Processor
ECN	Explicit Congestion Notification
FCS	Frame Check Sequence
FIFO	First-in First-out
FIN	TCP flag that signals the sender has no more data to send
FPGA	Field-Programmable Gate Array
GPS	Global Positioning System
HLIM	Hop Limit
HMAC	Keyed-Hashing for Message Authentication
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ID	Identification
IETF	Internet Engineering Task Force
ILP32	32 bit Integers, Longs, and Pointers
IMAP	Internet Message Access Protocol
I/O	Input/Output
IP	Internet Protocol
IP-ID	IP header’s ID field
IPv4	IP version 4
IPv6	IP version 6
IPMP	IP Measurement Protocol

ISP	Internet Service Provider
LP64	64 bit Longs and Pointers
LRU	Least Recently Used
MAC	Media Access Control
MIB	Management Information Base
MF	More Fragments
MRU	Most Recently Used
MTU	Maximum Transmission Unit
NTP	Network Time Protocol
OWAMP	One-way Active Measurement Protocol
PC	Personal Computer
PCI	Peripheral Component Interconnect
PDU	Protocol Data Unit
PMTUD	Path MTU Discovery
POS	Packet Over SONET
POSIX	Portable Operating System for Unix
PPSKit	Pulse Per Second Kit
PSAMP	Packet Sampling working group
QoS	Quality of Service
RFC	Request For Comments
RTRP	Real-Time Reference Point
RTT	Round Trip Time

SAR	Segmentation and Reassembly
SCDR	Sub-Capacity Dispersion Range
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SMP	Symmetric MultiProcessor
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical Network
SYN	TCP/IP Synchronise Control Flag
TCP	Transmission Control Protocol
TSC	Time Stamp Counter
TTL	Time To Live
UDP	User Datagram Protocol
UTC	Universal Time Coordinated
VoIP	Voice over IP
VPS	Variable Packet Size
VHDL	VHSIC Hardware Description Language
WAND	Waikato Applied Network Dynamics

Chapter 1
Introduction

1.1 The Problem

The limitations of existing measurement techniques limit the accurate measurement, analysis, and modelling of Internet behaviour [3]. The goal of Internet measurement is to understand why the Internet, an Internet path, or an Internet hop behaves the way it does. This thesis is an investigation into the per-hop measurement of Internet packet dynamics.

Network operators, engineers, users, and researchers have devised many tools and techniques to gain insight into the behaviour of an Internet path through measurement. The development of these tools and techniques is in large part motivated for two reasons. First, there exists an operational need to monitor network behaviour and to diagnose an Internet path when the performance of the path or a network application is poor. Second, there exists a need to understand the structure of the Internet and the characteristics of Internet paths in order to build accurate models for simulation of new Internet protocols and applications. The questions asked about the Internet in the operational community and by researchers have tended to reduce to basic statistical questions such as:

1. What is the packet delay between a source and a destination?

2. How variable is the packet delay on this path?

3. What is the path between a source and a destination?

4. Where in the path are packets being lost, reordered, or encountering significant delay?

5. What is the capacity of this path?

6. Where in this path is the capacity limited?

7. Which routers on this path are congested?

Significant attention and effort has gone into developing tools and techniques that are capable of answering questions like these. The most popular tools and techniques have tended to be simple. For example, two of the oldest and simplest tools, `ping` [4] and `traceroute` [5], are ubiquitous and are often the first tools used to debug an Internet path. The popularity of these simple tools is despite their limited ability to convey real insight to their users, and their limited ability to answer beyond the first three basic questions listed.

The support for measurement included in the Internet protocols has not evolved with the need to answer increasingly complex questions of the Internet. Current measurement approaches often estimate the actual properties of an Internet path. Measurement techniques should be accurate, unobtrusive, robust, and scalable [6]. Current tools and techniques designed to help answer the last four questions in the list do not meet all of these properties, are comparatively complex – and as a result, are not widely deployed. These limitations often arise due to the inability of current packet probing techniques to robustly and accurately isolate one-way and per-hop behaviours.

The ability to isolate the per-hop behaviour of each hop in an Internet path is an important and necessary step towards inferring *where* some interesting behaviour occurs. Currently available techniques for inferring the per-hop behaviour of each hop in an Internet path have to probe each hop separately to the others and then infer per-hop behaviour by separating the behaviour of one hop from the behaviour of previous hops. This is accomplished either by using TTL-limited packets similar to the technique popularised by `traceroute`, or by targeting each hop individually. This technique is unable to definitively separate the behaviour of one hop from the behaviours of other hops in a path. In addition, this technique is unable to infer per-hop behaviour for hops on the reverse path without explicit cooperation from the end host. A reverse-path `traceroute`, while a simple and operationally important concept, is absent from operational toolkits.

The relatively complex tools and techniques that can infer per-hop behaviours have met with limited success outside of the research community, due in part to caveats and limitations these tools have that affect their reliability and robustness. As a result, some performance faults in the Internet are not diagnosed due to the lack of robust tools and techniques to infer, identify, and characterise these faults. This thesis argues that this problem exists because of a lack of basic Internet protocol support to robustly, reliably, and accurately measure per-hop behaviours on an Internet path.

In this thesis, an Internet protocol designed for per-hop measurement of Internet packet dynamics is presented. The protocol proposed to address the problem, the IP Measurement Protocol (IPMP), enables per-hop measurement of Internet packet dynamics by providing the means for routers to embed simple information into the packet useful for directly measuring the dynamics of the packet on a per-hop basis. As this operation requires a small modification to the forwarding path in routers, a significant part of this particular problem is establishing the practical feasibility and usefulness of IPMP, as router vendors are hesitant to increase the complexity of the forwarding path without good reason.

1.2 Overview of Thesis

This chapter presents the thesis problem and the contributions of this thesis to the field of Internet measurement. In the next chapter, a taxonomy of four metrics that can be measured through packet probing is provided. The tools and techniques available to measure or infer each metric on an end-to-end, one-way, and per-hop basis are described, as well as the underlying motivation for doing so. For each technique described, limitations which reduce the effectiveness of the tool are identified and discussed. In chapter 3, the fundamental limitations of the current measurement support found in the Internet are identified and discussed. Chapter 3 also presents a series of desirable features that a measurement protocol would require in order to overcome or reduce these limitations. One of the fundamental features identified by Van Jacobson [7] as important for improving per-hop measurement of Internet packet dynamics is the ability to segment an IP path into individual IP hops, using a method which is not flawed as the TTL method is. Having such a method would improve the per-hop measurement of Internet path characteristics, as the behaviour of each individual hop would be separated from the behaviour of other hops in the path.

The Internet protocol proposed to address the problem, IPMP, is introduced in chapter 4. IPMP is designed for per-hop measurement of Internet packet dynamics. IPMP enables per-hop measurement through the introduction of a *path record* structure. A path record contains an IP address, the TTL of the packet when seen, a timestamp recording when the probe was received, and a flow-counter that records the position of a particular probe amongst a series of IPMP probes belonging to the same flow. As IPMP requires a modification to the forwarding path in routers, IPMP was guided by the philosophy that it should be kept as simple as possible to increase the likelihood of deployment without providing a vector for Denial of Service (DoS) attacks.

Chapter 5 discusses implementation experience of IPMP both in software and in hardware. First, a model of interactions between Internet protocol stacks and IPMP is presented. Then, the specific interactions between two software protocol stacks are detailed, for which ready access to the source code is available. The two protocol stacks in question, FreeBSD and Linux, are substantially different in their design and implementation, and provide insight as to the implementability of IPMP. Then, a third implementation of IPMP, in hardware, is described. That implementation modifies the forwarding path of a non-blocking cross-bar Ethernet switch so that IPMP packets are modified in-line as each byte is deserialised. The forwarding performance of the FreeBSD software implementation and the Ethernet hardware implementation is also examined by passively measuring the forwarding performance of each system with and without IPMP.

The final chapters of the thesis discuss the usefulness of IPMP. First, chapter 6 presents a series of measurement techniques which make use of IPMP. For each technique, a comparison is made between the measurement techniques currently available and the technique proposed which uses IPMP. Then, chapter 7 discusses application experience gained by measurement of a rural wireless network where IPMP is available on all IP routers. Finally, chapter 8 reviews related work, and chapter 9 concludes the thesis and looks ahead at future work.

1.3 Contribution of this Work

The thesis of this work is that a protocol for per-hop measurement of Internet packet dynamics is both feasible and useful. The core contributions of this thesis are:

1. the refinement of a measurement protocol (IPMP) that provides the ability to measure Internet packet dynamics on a per-hop basis;

2. the implementation of IPMP and an analysis of the exhibited performance;

3. refinement of existing packet probing techniques to utilise the features of IPMP;

4. an explanation of deployment experience gained by the deployment of IPMP in a wireless network in all routers.

The IPMP protocol was first proposed by Tony McGregor in 1998 [8]. This thesis describes a substantially refined version of IPMP. My contributions to the IPMP protocol include the rearrangement of echo header fields, removal of unnecessary fields (the returned TTL field, length field, and data field), the addition of faux port numbers, the addition of identification and sequence number fields, as well as the ability to query for specific reported times of interest in the information protocol. In the course of this work, James Spooner provided feedback on how to improve the format of IPMP so that it is easier to process in hardware, and provided an implementation of an earlier version of the protocol. Additionally, Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson suggested IPMP include a flow counter field in “User-level Internet Path Diagnosis” [1].

IPMP addresses many limitations of current methodologies for per-hop measurement of Internet packet dynamics by enabling packet probing techniques to simultaneously measure both the packet dynamics and the path taken by a packet in a single packet exchange. The protocol is tightly constrained, efficient, and easy to implement. These characteristics are important, and are intended to make IPMP suitable for implementation by router manufacturers.

The first half of this thesis focuses on the feasibility of IPMP. First, implementation models of IPMP are presented to show that the protocol has been designed with processing efficiency as a primary design criterion. Second, actual software and hardware implementations are presented and discussed, as well as measured to show that IPMP can be quickly processed in the forwarding path of a router.

The second half of this thesis focuses on the usefulness of IPMP. This is shown in two parts. First, IPMP-based measurement techniques are presented that are more accurate, less obtrusive, more robust, and more scalable than their counterparts. Second, the utility of IPMP in a real-world situation is presented. Since 2002, the Waikato Applied Network Dynamics (WAND) network research group has operated a predominately 802.11b wireless infrastructure network that connects schools in rural settings to the Internet. Each router in this network is software-based and lends itself to customisation and modification. The utility of IPMP in understanding and improving this network through measurement is demonstrated.

Chapter 2
Background

2.1 Introduction

This chapter reviews four basic Internet performance metrics, discusses the motivation to measure them, and reviews the current tools and techniques available to measure them. The metrics – delay, loss, reordering, and capacity – indicate the performance that an Internet path is able to provide between two systems. The performance of all Internet protocols and applications is directly dependent on these four metrics.

As TCP is the most widely used transport protocol in the Internet [9] and is likely to remain that way in the future, special attention is paid to the dependencies that TCP has on delay, loss, reordering, and capacity. Beyond TCP, many real-time, interactive, multimedia services delivered using other transport protocols have strict Quality of Service (QoS) requirements, which are comprised of these metrics.

This chapter discusses measurement of these metrics at three levels. Measurement of these metrics at different levels can give different insights as to the behaviour of a path. The first level discussed is the measurement of these metrics in an end-to-end scenario. That is, the measurement of a metric on the complete path between a source and a destination. Internet applications and services are delivered on an end-to-end basis between two hosts, and so the end-to-end behaviour of an Internet path is important to the overall performance of the service.

The second level is the one-way scenario – the measurement of a metric either on the forward path or the reverse path. Many protocols and services are more dependent on the behaviour of a path in one direction than compared to the other, and so the ability to measure the behaviour of the forward and reverse paths independently is important.

The third level is the hop-by-hop scenario – the measurement of a metric for each IP link on an end-to-end path. The ability to measure these metrics on a hop-by-hop basis allows a user or operator to determine *where* in a path significant delay, jitter, loss, and reordering occurs, or where the capacity-limiting link is, and is useful for Internet path diagnosis [1]. For each metric – delay, loss, reordering, and capacity – this chapter:

- provides a definition of what is being measured;
- establishes the operational motivations to be able to accurately measure the metric in an end-to-end scenario, a one-way scenario, and a hop-by-hop scenario;
- discusses the current tools and techniques available to measure the metric in an end-to-end scenario, a one-way scenario, and a hop-by-hop scenario;
- identifies the limitations of these tools and techniques.

The rest of this chapter begins with a short review of Internet topology measurement in order to motivate per-hop measurement of Internet packet dynamics. Then, the four metrics are defined, their measurement is motivated, and the tools and techniques available to measure them are discussed. Many current tools and techniques share a series of common limitations that inhibit their operational use. The three major limitations are that the techniques separate measurement of an IP topology from the measurement of the metric in question such that the number of probes required grows in proportion to the length of the path, that the measurement of hops later in a path is less accurate, and that the tools often rely on the timely delivery of ICMP responses. As the limitations of the current tools and techniques are explored, the motivation for a new Internet protocol designed for per-hop measurement of Internet packet dynamics is established.

2.2 Internet Topology

2.2.1 Definition

Topology is defined as the arrangement of nodes and links (the path) that a packet follows from a source to a destination. Internet topology condenses this definition into the arrangement of IP routers that a packet visits. Internet topology can be reported at multiple layers, including the IP path, the Autonomous System (AS) path, and the geographical path. An IP path is a sequence of IP addresses between a source and a destination, each of which represents a single interface of each router in the path. However, Internet inter-domain routing occurs with Border Gateway Protocol (BGP) [10] between Autonomous Systems (ASes), with each AS having a unique number. An AS path therefore represents the sequence of AS numbers that a packet follows between a source and destination; an AS number represents a unit of routing policy [11] and typically maps to an Internet Service Provider (ISP) or organisation [12]. A geographical path is a sequence of cities or countries that a packet visits as it is forwarded to a destination. Topology is typically measured as an IP path and then, if desired, condensed into an AS path or a geographical path.

2.2.2 Motivation for Internet Topology Measurement

A primary motivation of Internet topology measurement in the context of per-hop measurement of Internet packet dynamics is to discover *who* is involved in forwarding packets between a source and a destination. Knowledge of the IP path between a source and a destination can be useful in isolating a network fault or anomaly, as knowledge of the path provides a mechanism to narrow the problem search space. Translating an IP path into an AS path is useful for determining the sequence of ASes visited on a path, and for inferring AS relationships. Translating an IP path into a geographical path is useful for geographical analysis of Internet paths [13] and for forming a basis for measuring IP path symmetry [14].

A number of large-scale Internet mapping projects continuously probe the Internet with a goal to infer a complete and unbiased view of the Internet topology. These projects distribute Internet topology measurement systems across the Internet. Examples of such projects include the Internet Mapping Project [15], Skitter [12], and DIMES [16]. Primary motivations for large-scale Internet topology mapping include the ability to provide realistic models for designing and measuring new routing protocols [17], providing data to understand the structure of the Internet [18], defending against attacks on core Internet infrastructure [19], and determining optimal mirror placement [20, 21]. In addition, Internet-scale measurement of link behaviours and characteristics [22] is guided by knowledge of the underlying topology itself, as "annotating a map with measured properties becomes practical when the basic map of the topology helps decide how to take detailed measurements" [23].

As this thesis is concerned with measurement of Internet packet dynamics on a per-hop basis, this thesis focuses on measurement of the IP topology, However, it is worth noting that IP topologies which are translated into AS topology are more dense than AS topology inferred from BGP tables [18].

2.2.3 Internet Topology Measurement Techniques

2.2.3.1 Traceroute

The forward IP path to a destination can be inferred using `traceroute` [5]. `traceroute` works by sending packets with the Time To Live (TTL) field in the IP header limited, so that successive probes will expire at intermediate hops until the destination is reached. The router where a probe expires should send an ICMP time exceeded message to the source of the probe, signalling that the TTL of the probe expired at the router. The forward path is inferred by systematically sending a sequence of TTL-limited IP packets with an increasing TTL value that is initially set to one, and then extracting the source IP address from each ICMP time exceeded message that is received as successive routers discard each probe due to their TTL expiring. `traceroute` probes are usually UDP probes to unused ports, so that when a probe reaches the destination, it will elicit an ICMP port unreachable message and the source will know not to probe any further.

There are numerous limitations to the TTL-limited probing technique used by `traceroute`. First, the technique requires that each hop be probed separately to previous hops. If the path changes while it is being probed, or is load balanced such that consecutive probes follow different IP paths, it is possible that a path that does not actually exist will be reported. Second, it relies on each router to generate an ICMP time exceeded message. This is not the case in scenarios where some operators disable ICMP message generation due to DoS concerns. Third, in order to infer the entire forward path, it relies on the probes being allowed to reach their destination. This is not the case in scenarios where a firewall drops unrecognised packets by default. Fourth, the technique is restricted to inferring the forward path, because an IP packet has a single destination; any reply packet is created from scratch with the IP TTL value set to a default value assigned by the system. Therefore, in order to obtain the IP topology of the reverse path, the destination host has to provide a method to cooperate with the source host. This cooperation with a destination host has to come in the form of a protocol or other known mechanism.

One approach to inferring the reverse path is to use a public `traceroute` service. A public `traceroute` service is typically provided by a web-form that allows a user to specify the target IP address and the `traceroute` options to use. The service will then conduct a `traceroute` to the target and display the output in a web page. A list of public `traceroute` services and the AS where they are hosted can be found at `http://www.traceroute.org/`. The main limitations of this approach to inferring the reverse path is that few hosts support this facility, and finding a `traceroute` service near to the destination of interest can be inconvenient.

2.2.3.2 IPv4 Record Route

Another approach to determining the IP path between a source and a destination is to use the IPv4 record-route option [24]. The record-route option is a feature built into the Internet protocol that allows a source machine to request that routers insert their IP address into space reserved in the IP header as they forward the packet. When the packet is returned, the source may determine the path taken by the packet. The

0–1: bit 0	bit 1	bits 2–7	bits 8–15	bits 16–31
F	C	Number	Length	ID Number
Outbound Hop Count				Return Hop Count
Originator IP Address				

Figure 2.1: Format of the traceroute IP option defined in RFC 1393

bits 0–7	bits 8–15	bits 16–31
Type	Code	Checksum
ID Number		unused
Outbound Hop Count		Return Hop Count
Output Link Speed		
Output Link MTU		

Figure 2.2: Format of the ICMP traceroute message defined in RFC 1393

advantage of using the record-route option is that it allows the forward and reverse paths to be captured in a single packet exchange.

There are numerous limitations to using the IP record-route option, however. First, due to the small maximum size of the IP header, a packet may hold up to 9 addresses until it is full. This limit will not allow the complete forward and reverse IP paths to be captured for most Internet paths. Second, because IP options are difficult to process efficiently [25], IP options are often ignored, and packets containing IP options may be discarded or processed at low-priority. Third, it is not possible to determine where in the path each router is in relation to other routers without probing the path with a tool like `traceroute`. Finally, this option is not available in IPv6.

2.2.3.3 Traceroute Using an IP Option

An experimental protocol for `traceroute` described in RFC 1393 [26] – “Traceroute Using an IP Option” – introduced a method to obtain the IPv4 topology of both the forward and reverse paths between a source and destination. A source host sends an ICMP echo request (or some other request packet) to a target, embedding the traceroute IP option in the packet. The format of the option is shown in figure 2.1. Each router that understands this option sends an ICMP traceroute message to the originator IP address specified in the IP option, before forwarding the packet to the next hop. The format of the ICMP traceroute message is shown in figure 2.2. When the destination receives a request packet with the traceroute IP option, it makes a few modifications to the option and then includes the traceroute IP option in a reply packet addressed to the source.

In the 12 years since being published in RFC 1393, the traceroute IP option has not been widely implemented or deployed in the Internet. There are a number of limitations to this approach to `traceroute`, which may have worked against the protocol evolving beyond experimental status and being widely deployed. The option lends itself to amplification of a simple flooding DoS attack, where a malicious sender could spoof the originator IP address in the traceroute IP option and cause routers on the path to amplify the attack by sending unsolicited ICMP traceroute messages to the target. In addition, IP options are difficult to process efficiently [25], and each intermediate router has to examine the options present and then take appropriate action.

2.2.4 Resolving Router Aliases

The techniques described in section 2.2.3 infer the IP path that a packet follows between a source and a destination. When combining multiple IP paths, such as the forward and reverse IP paths between a pair of addresses, or IP paths collected from a distributed set of machines, it can be useful for the purposes of further diagnosis and understanding to convert the IP topology into router topology. The router topology is concerned with the arrangement of routers in an Internet path.

The distinction between IP topology and router topology arises because a router has multiple interfaces, and each interface has at least one address. When a router generates an ICMP message, it often sets the source address to be one from the outgoing interface used to transmit the packet. As the outgoing interface used depends on a packet's destination, a pair of hosts that both use `traceroute` to discover the forward path to the other host may yield a different set of IP addresses, even if the same routers are visited.

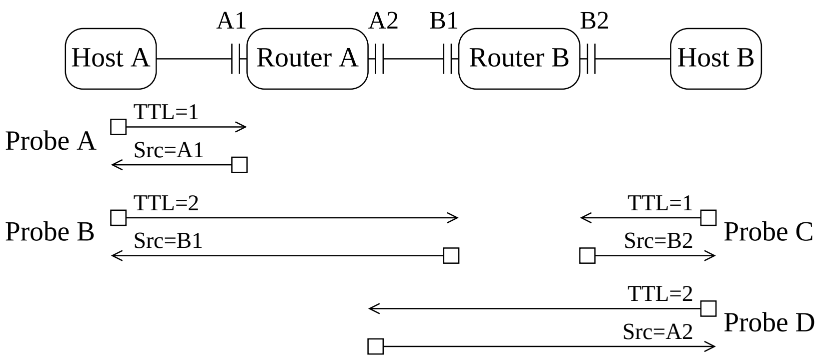


Figure 2.3: Illustration of two hosts using `traceroute`

Figure 2.3 shows an example of this. Despite the path being symmetrical between the two hosts, `traceroute` returns router addresses A1 and B1 when run from host A, while `traceroute` returns B2 and A2 when run from host B.

2.2.4.1 Iffinder

`iffinder` [12] resolves router aliases by exploiting the same ICMP implementation strategy that necessitates the technique. `iffinder` solicits ICMP port unreachable messages by sending UDP probes to unused ports for each interface in the IP topology. As the tool is run from a single host, each router that selects a source address based on the outgoing interface will send response packets with the same source IP address, allowing a router's aliases to be resolved by matching responses with common source IP addresses. This technique can quickly resolve router aliases because each address can be resolved with a single probe.

There are a number of limitations to this technique, however. First, this method cannot resolve router aliases if a router does not reply to these probes with the same source address for each probe. For example, a router might send ICMP messages with a source address set to the destination address probed with the UDP packet. Second, some interfaces are configured to not send any ICMP destination unreachable messages, but show in the IP topology because ICMP time exceeded messages are not suppressed [27]. Third, resolving aliases by sending probes targeted to routers may appear as malicious probing in some scenarios and result in abuse complaints.

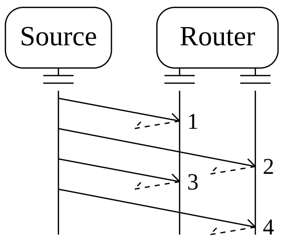


Figure 2.4: Resolving router aliases with `ally` using IP-ID

2.2.4.2 Ally

Another approach to resolving router aliases is to exploit a common implementation strategy of assigning unique IP-ID values from a counter held centrally by the router. The IP-ID field is used to uniquely identify a packet so if the packet is subsequently fragmented by the network, packet fragments arriving at a destination can be reassembled into their respective packets [24]. `ally`, the alias resolver of Rocketfuel [28], tests if two specified IP addresses belong to the same router by sending a series of packets alternately to pairs of addresses it suspects of being aliases, as figure 2.4 shows. If the IP-ID fields from the reply packets are progressive (allowing for a small amount of reordering on the forward path) and the range between the IP-ID values is small, then it is likely that the addresses probed are aliases for the same router.

The advantage of the IP-ID approach to resolving router aliases is that it is possible to use any type of probe which solicits a response from a router with a sequential IP-ID value. Indeed, the IP-ID technique has been reported as being much more effective than the UDP technique used by `iffinder` [19]. There are a number of limitations to the IP-ID technique, however. The Internet protocol does not define semantics on how to assign IP-ID values [24]. Some systems send ICMP responses with the IP-ID field set to zero [1], and some systems send packets with a randomised IP-ID field in order to avoid inadvertent information leakage [29]. Second, the IP-ID technique does not work with IPv6 topologies because the IPv6 header does not have an IP-ID field. Third, the IP-ID technique requires many more probes than the UDP technique to resolve router aliases.

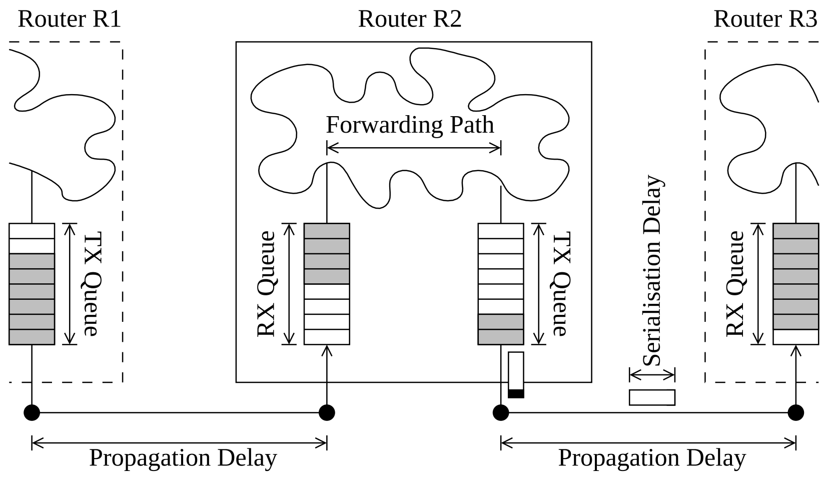


Figure 2.5: Contributors to fixed and variable delay factors

2.3 Delay

2.3.1 Definition

Packet delay is the time it takes a single packet to get from one point in the network to another. Packet delay consists of two groups of factors: fixed components and variable components. Figure 2.5 shows the main fixed and variable delay components for many common link types.

The main fixed components of packet delay are the serialisation delay from a node and the propagation delay along a link. The serialisation of a packet involves sending the packet with any necessary layer 2 headers one bit at a time onto the medium. Likewise, the propagation delay along a link is fixed, as the signal used to send each bit takes the same amount of time to reach the other end of the media. The fixed components therefore place a lower-bound on the delay that a packet will incur in transit to a destination.

The main variable components of packet delay are queueing and forwarding delays at the node. The total queueing delay a packet experiences is variable and depends on how full the various queues are when the packet is processed; the length of each queue depends on cross traffic from other sources. The forwarding delay a packet experiences is also variable, as it depends on how quickly the router can decide the next-hop the packet should be forwarded to; the decision may be quicker if the route is cached at the time the decision is made. In some networks, a network interface will also have to hold a packet for some time until the medium is clear for transmission. The variable components are responsible for *jitter*, or the difference in delay experienced by a series of packets in relation to each other.

2.3.2 Motivation for Delay Measurement

2.3.2.1 End-to-End Motivation

End-to-end packet delay is important to interactive applications and services that provide real-time feedback to a user, such as telnet and ssh, as "interactive response is perceived as 'bad' when low-level feedback (character echo) takes longer than 100 to 200ms" [30]. Similarly, jitter is important to real-time and multimedia applications such as real-time Internet gaming and Voice over IP (VoIP), where the predictable and constant arrival of a series of packets is required for the application to be usable.

Beyond interactive applications and services, the speed of a TCP bulk transfer depends on end-to-end packet delay and jitter. As the TCP protocol itself uses packet delay and jitter measurements to determine a suitable retransmission timer, being able to measure end-to-end packet delay and jitter is a useful prediction of TCP performance. Fluctuations in packet delay can have a significant negative impact on a TCP connection. This is because TCP may react to fluctuations in packet delay by waiting longer before retransmitting a packet which is lost to avoid retransmitting a packet that is simply delayed longer than usual, or may retransmit data too soon, wasting capacity.

2.3.2.2 One-way Motivation

One-way packet delay and jitter is important to applications and services which depend on the delay of a packet over one direction of a path more than the delay in the other direction. Knowledge of the one-way delay of a packet allows path symmetry to be inferred, which is useful in determining the direction that a packet incurs the most delay. In order to account for propagation delay, some timing sensitive protocols assume that the delay between a client and server is symmetrical. An example of a timing sensitive application is Network Time Protocol (NTP) [31]; if the propagation delay between an NTP client and server is asymmetrical, the client's clock will be offset from real-time.

2.3.2.3 Per-hop Motivation

Measurement of packet delay and jitter at a per-hop level provides the ability to determine the hops which contribute significant delay and jitter. Being able to infer which hops contribute significant delay is an important step towards determining who is responsible for contributing to poor end-to-end performance and then improving the network through targeted engineering.

2.3.3 Delay Measurement Techniques

2.3.3.1 Ping

The most widely used tool for measuring end-to-end packet delay is `ping` [4], which measures Round Trip Time (RTT) with an ICMP echo request and reply sequence. While the measurement of delay with `ping` is satisfactory in many situations, it has a number of limitations.

First, there has been significant debate as to the reliability of measuring delay to a router by sending packets addressed to it, as routers are designed to forward packets as quickly as possible. There is evidence to suggest that ICMP packet generation times, in general, are not a significant source of measurement error in the modern Internet [32]. However, in order to determine if a particular router introduces significant measurement error, the router's ICMP echo reply generation time has to be profiled first. The implication of this is that it is not reliable to measure packet delay to a router using `ping`.

Second, ICMP packets may be forwarded and queued with a different priority compared to other packets in some networks. When this occurs, the delay measured with `ping` will be different to the delay encountered by other protocols and experienced by other applications. The difference in network delay measured by `ping` in this instance is a measurement artifact.

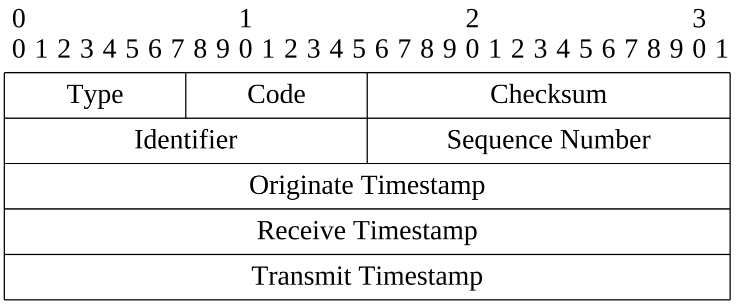


Figure 2.6: Format of the ICMP timestamp message defined in RFC 792

Third, there is the question of where in the source host a packet should be timestamped: in user-space, in kernel-space, or by the network interface itself. This concern is not limited to `ping` [33]. The host timestamp issue is discussed in further detail in section 2.3.4.

2.3.3.2 ICMP Timestamp

A number of techniques exist for measuring one-way packet delay and jitter. All require some level of cooperation from the destination. The first method is to construct an ICMP timestamp request message, as outlined in RFC 792 [34]. The ICMP timestamp message, shown in figure 2.6, contains three timestamps – the originate timestamp, the receive timestamp, and the transmit timestamp. The originate timestamp is included by the source, while the destination inserts the time it received the packet in the receive timestamp, and the time it transmits the reply in the transmit timestamp.

If both hosts involved in the timestamp exchange have synchronised clocks, the forward path delay can be calculated by subtracting the originate timestamp from the receive timestamp, and the reverse path delay can be calculated by subtracting the transmit timestamp contained in the ICMP timestamp reply packet from the time the response packet arrives back at the source host. However, without prior knowledge of the clock-state of the target host, it is impossible to measure absolute one-way delay of the probe packets [35].

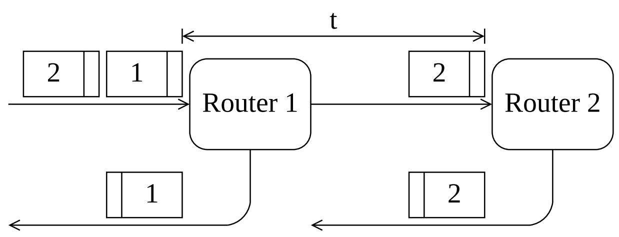


Figure 2.7: Measurement of per-hop queueing delay with `cing`

2.3.3.3 One-way Active Measurement Protocol

In order to determine the clock-state of another system, some level of cooperation is required with the system by way of a protocol. One-way Active Measurement Protocol (OWAMP) is a protocol designed with measurement of one-way packet dynamics in mind [36, 37]. A source host obtains the cooperation of a destination by establishing a TCP connection, which is used to control the measurement process. The source may then negotiate a process of UDP probes with the destination, with probe packets flowing in a single direction; that is, either from the source to the destination, or from the destination to the source. Each probe includes a timestamp inserted by the sender, a sequence number, and the estimated error of the timestamp if available.

The OWAMP protocol has a number of advantages over using the ICMP timestamp request protocol for measuring one-way packet delay. It provides simple mechanisms to establish the synchronisation state of a sender's clock, to negotiate the size and UDP ports of measurement probes, to authenticate and encrypt measurement probes for protection against manipulation by a third party, and to recover measurement results from a destination when measuring the delay characteristics of the forward path. The main limitation of the OWAMP protocol is that it is not likely to be offered as a service by routers because operators may be hesitant to allow TCP connections from arbitrary hosts to their routers.

2.3.3.4 Cing and Tulip

There are no techniques currently available to measure per-hop packet delay, due to the lack of a protocol or technique to determine the offset of an arbitrary clock from real time. However, two techniques exist to measure per-hop jitter using the ICMP timestamp protocol.

The first technique, implemented in `cing` [38] and illustrated in figure 2.7, infers queueing delays on a particular hop by sending sequences of two back-to-back ICMP timestamp request packets to adjacent routers. The basic idea is that queueing delay between adjacent routers can be inferred by comparing timestamps in ICMP timestamp reply packets received from the routers. The technique requires that the path to the first router is a prefix of the path to the second router, so that both packets in the pair arrive at the first router at approximately the same time. The first packet is addressed to the first router, while the second packet continues one hop further. Both packets solicit ICMP timestamp replies. Assuming that the clocks do not drift relative to each other and that at least one of the pairs does not incur any queueing delay between the two routers, queueing delay can be measured by changes in the timestamp provided by the second router relative to the timestamp provided by the first.

The second technique – implemented in `tulip` [1] – infers queueing delays towards a particular hop by sending a series of ICMP timestamp request packets to the hop, and inferring jitter by comparing the timestamp received in the ICMP timestamp response with the transmit time of the probe. The main advantage of this approach over `cing`'s approach is that it does not require adjacent routers to support the ICMP timestamp request option in order to determine queueing delay towards a particular router.

ICMP timestamp techniques, like those used in `cing` and `tulip`, share a number of limitations. First, they are forward path bound, as it is not possible to identify or measure individual routers on the reverse path without the cooperation of a destination host. Second, they also depend on the forward path to an intermediate router being a prefix of the forward path to the actual destination, otherwise the measurement is of behaviours that include hops other than the particular hop of interest. Third, the resolution of the ICMP timestamp fields is limited to one millisecond. This resolution is sufficient for inferring the location of significant or pathological queueing delays, but not for profiling the queueing behaviour of many modern Internet paths. Fourth, the ICMP timestamp request packet may queue internally in a target router for a significant amount of time while other tasks with a higher priority complete, leading to a source of measurement error if other packets continue to flow through the router. In [1], some routers were seen to take 100ms to 300ms longer than normal to generate an ICMP timestamp reply. As this behaviour occurred approximately every 60 seconds, it was reasoned to coincide with when the router pushed forwarding tables to its line cards.

2.3.4 Accounting for Host Delays

Historically, the transmit and receive timestamps used to calculate RTT were generated in user-space. In the case of `ping`, the transmit timestamp was taken before the echo request packet was sent, and the receive timestamp was taken after the echo reply packet was received in user-space.

Measurement of packet delay with timestamps that are generated in user-space has a number of limitations, as the measurement of delay may also include host delays in processing the packet and the effect of how busy the host is. In the transmit path, a timestamp taken in user-space immediately before making a system call to send a packet will include the time it takes the system to determine the outgoing route, and the time the packet spends in queues waiting to be passed to the network interface. In the receive path, a timestamp taken in user-space immediately after receiving a packet will include the time it takes for the packet to make its way through the IP stack, and the time the packet spends in a queue before the application is scheduled and can read it.

A diagram of the transmit and receive paths in a Unix-like operating system is shown in figure 2.8. A host can often obtain a more accurate packet timestamp than the timestamp provided in user-space, at the expense of additional application complexity. For example, some systems provide a socket option or mechanism to conveniently timestamp a received packet in the kernel before the packet is passed to a user-space application.

An example of such a mechanism is the SO_TIMESTAMP socket option, which is common amongst systems derived from Berkeley Software Distribution (BSD),

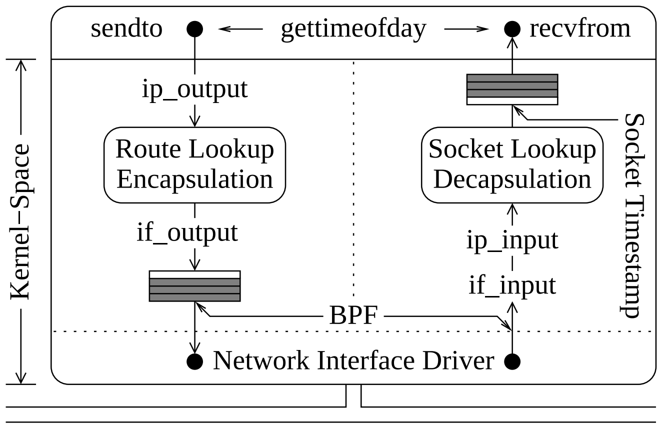


Figure 2.8: Packet timestamping inside a host

and applies to connectionless sockets, such as those used for UDP and ICMP. In the SO_TIMESTAMP case, a timestamp is generated immediately before the packet is put in a socket queue for the application to then read it from. This is useful in situations where a measurement application may be contending for the Central Processing Unit (CPU) with other applications running on the same system, which may result in the packet spending additional time in a socket buffer waiting to be read.

However, there is not a similar socket option available to timestamp packets in the transmit path. There is, however, the Berkeley Packet Filter (BPF) [39], which provides a mechanism to timestamp packets closer to the network interface. In the receive path, BPF allows an application to obtain a timestamp before the packet is passed to the network protocol handler. In the transmit path, BPF allows an application to obtain a timestamp either immediately before, or immediately after the packet was passed to the network interface; the sequence depends on the implementation of the network interface driver. In this scenario, an application may send a packet using a regular socket, and then obtain the time the packet was passed to the network interface using a BPF socket.

In order to quantify the difference in packet timestamps for the various locations inside a host identified in figure 2.8, the difference in packet timestamps through a host was measured. The host used in this experiment is a Pentium Pro 180 running FreeBSD 4.11, which was otherwise idle during the experiment. While the CPU of the system used in this experiment is slow compared to what is available for

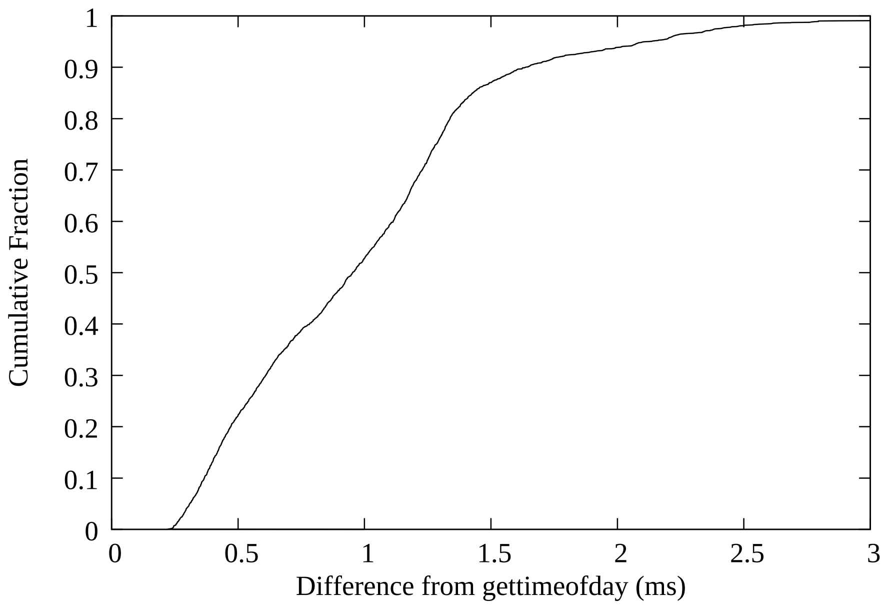


Figure 2.9: Cumulative distribution of timestamp differences in the transmit path

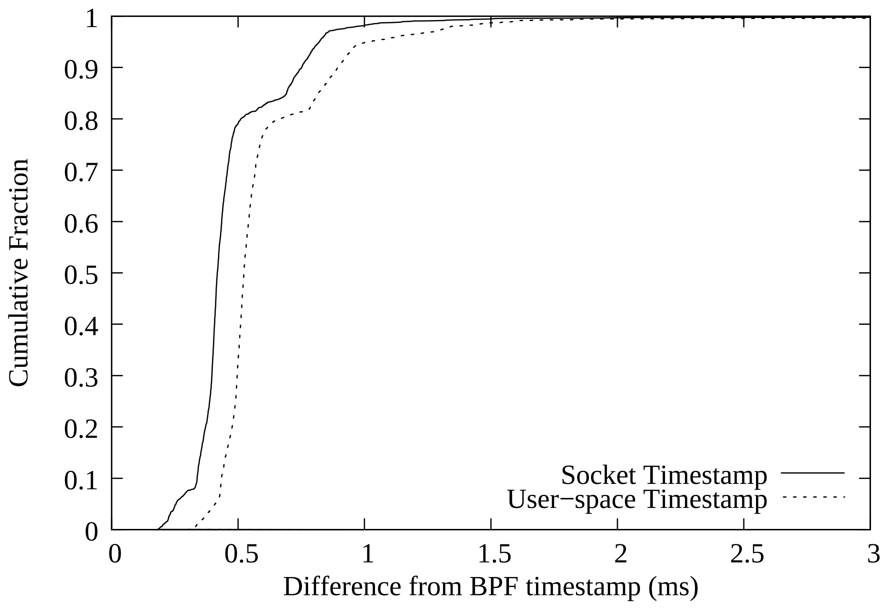


Figure 2.10: Cumulative distribution of timestamp differences in the receive path

desktop computers in 2006, it is faster than some CPUs which are used in low-power situations for software routers, like those that will be discussed in chapter 7. A specially modified version of `scamper` [40] was used to collect the packet timing data. `scamper` is a measurement application which can conduct `traceroute` and other measurement techniques in parallel to fill a specified packets-per-second rate.

Figure 2.9 shows the cumulative difference between the user-space timestamp and the timestamp returned by BPF for a series of 44-byte packets in the transmit path. This graph shows a processing delay of up to 1.3ms for 80% of packets sent, indicating that the time taken to determine the outgoing route and encapsulate the packet for transmission is a source of significant and unpredictable measurement error.

Figure 2.10 shows the cumulative difference of the BPF timestamp to the socket and user-space timestamps for packets received in response to the probes sent. This graph shows a processing delay of up to 0.75ms for 80% of packets received, which suggests the kernel to user-space delay is shorter in the receive path than the transmit path. These results indicate the use of a socket timestamp option does not reduce measurement error due to host-processing by a significant amount on an otherwise idle host.

2.4 Loss

2.4.1 Definition

Packet loss occurs when a packet is either dropped or damaged in transit. Packet loss is measured by the absence of an expected packet. Determining when loss occurs is not entirely straight forward, as a decision has to be made when to declare a packet lost rather than simply delayed for a long time. In many situations, a packet might as well have been lost if it is delayed so long that the eventual arrival of the packet is of no use.

There are many possible causes of packet loss. For example, a packet may be dropped due to congestion on the path. Congestion-induced loss occurs when a queue fills to capacity, leaving no space for additional packets. A packet may also be damaged or corrupted in transit, which may then cause it to be discarded in the network, or be delivered to the destination where it is subsequently discarded. A packet may also be discarded by a router that drops packets under load.

Other types of packet loss exist. For example, a packet may be dropped due to equipment failure in the network, or may be larger than the media's Maximum Transmission Unit (MTU) size and unable to be fragmented, or is fragmented in the network but not all fragments subsequently delivered.

2.4.2 Motivation for Loss Measurement

2.4.2.1 End-to-End Motivation

The measurement of end-to-end packet loss is important to interactive applications and services, as a retransmitted packet adds at least an extra RTT to the time it takes to successfully receive a packet and update the user interface. In addition, interactive applications such as VoIP and video conferencing will tolerate small amounts of packet loss, but when the loss rate rises above a certain level, the quality of these applications reduces very quickly. Packet loss is especially important to all TCP-based applications, because TCP assumes that packet loss is due to network congestion and will reduce the rate at which it transmits packets when it detects packet loss. Sustained packet loss will result in poor TCP performance, as TCP will exponentially reduce the rate of transmission for each packet lost.

2.4.2.2 One-way Motivation

The measurement of one-way packet loss is important to scenarios where the loss of a packet in one direction is more critical than a packet lost in the other direction. For example, in a TCP bulk transfer situation where data packets flow in one direction and acknowledgement packets flow in the other, the loss of a TCP acknowledgement packet is likely to have less of an impact on the time it takes the transfer to complete. The loss of an acknowledgement packet without data will not significantly affect the TCP connection if another acknowledgement packet arrives shortly after the first would have, as TCP acknowledgements are cumulative. In this scenario, a burst of new data segments will enter the network to replace those which were acknowledged. While a large burst of packets sent back-to-back is more likely to cause congestion than if the packets were sent gradually, recent work suggests that less than 5% of moderate bursts of up to 15 packets result in a lost data segment from that burst [41]. A lost data segment requires retransmission, and the sender will halve the rate at which it sends new segments into the network in response to the congestion.

2.4.2.3 Per-hop Motivation

Measurement of packet loss at a per-hop level provides the ability to determine where loss is actually occurring. Determining the hop where packets are dropped is an important operational diagnostic ability, as it can indicate an under-provisioned link causing congestion, or a link with some hardware or link-layer failure. Knowing where packet loss occurs is useful in determining the responsible party, and in planning network upgrades.

2.4.3 Loss Measurement Techniques

The techniques for measuring end-to-end packet loss are similar to those used for measuring end-to-end delay, except that loss is measured by the absence of an expected reply packet within a defined time limit. Therefore, a convenient method to measure end-to-end packet loss is with `ping` or some other tool that solicits ICMP responses. However, a significant problem with this is that the generation and forwarding of an ICMP packet may be rate-limited or disabled in some routers and operating systems to guard against DoS attacks, as ICMP packets are relatively expensive to generate and respond to compared with forwarding the same packet. Similarly, one-way loss can be measured with a measurement protocol such as OWAMP [37] so long as the destination host supports the protocol. If both directions are to be measured, the source host must negotiate two measurement sessions with a destination. The first session measures one-way loss on the forward path, while the second session measures one-way loss on the reverse path.

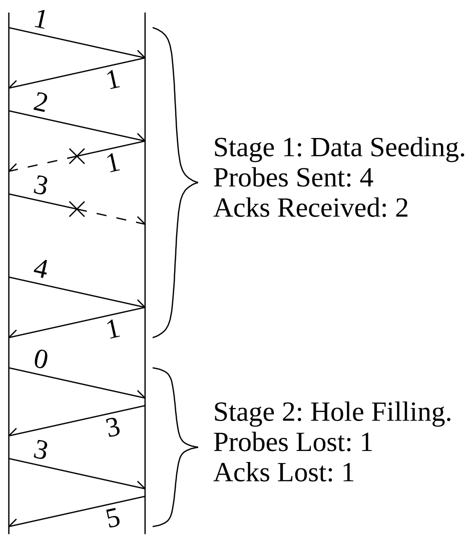


Figure 2.11: Detecting one-way loss with `sting` using TCP

2.4.3.1 Sting

One-way loss between a source and a destination host can also be inferred with `sting` [42], which is capable of measuring both forward and reverse path loss using the TCP protocol, so long as a destination host accepts TCP connections on a port known to the source. Figure 2.11 illustrates the `sting` technique. `sting`'s basic strategy is to initiate a TCP connection and then send a series of out-of-sequence probe packets. When a TCP receiver receives an out-of-sequence data packet, it immediately sends an acknowledgement for the last in-sequence data it received. In `sting`'s case, none of the TCP packets it sends to measure packet loss are sent in-sequence, so it expects to receive an acknowledgement for each data probe it sends.

This first stage – known as data seeding – determines the end-to-end loss of the path by comparing the number of data probes sent with the number of acknowledgement packets received. The second stage – known as hole filling – determines which data probes were lost on the forward path to the receiver; that is, the data holes in the receiver's buffer. `sting` begins 'filling the holes' by sending the missing first segment in order to have the receiver send an acknowledgement packet acknowledging all bytes up to the first missing data segment, if any. `sting` then re-transmits the data packet for the missing segment, and any subsequent missing segments it learns of through the hole filling process, until it has accounted for all data probe packets sent to the destination.

`sting`'s main advantage over using a measurement protocol such as OWAMP is that the ubiquitous nature of TCP means `sting` will work in a far greater number of scenarios. However, there are limitations to the technique. First, it is not possible to reliably control the series of packets sent by the destination on the reverse path to the source; while the source is able to send large probes – which are arguably more likely to be lost than smaller probes – a TCP receiver has little control over the size of any acknowledgement packet returned. For this reason, `sting` may be more able to measure forward path loss than reverse path loss.

Second, `sting` requires a number of system privileges and kernel support to work optimally. In order to send out-of-sequence TCP packets and to receive raw TCP packets, the kernel must allow the ability for an application to send these probes with some form of data link access, which requires root-level access to the system. Similarly, should the source host's kernel respond automatically to unexpected TCP acknowledgement packets arriving in response to `sting` probes, `sting` will require the ability to prevent the kernel from interfering in the measurement. This requirement can be met with a temporary firewall rule, although there is no standardised method to do this across operating systems, and the kernel may not have a firewall enabled.

Third, some devices such as application-level firewalls and load-balancers require the sender to send a valid request for the port being accessed, or they will reset the connection as soon as an invalid request is detected. `sting` comes with a valid HTTP request built in, but in order for `sting` to use another service such as SMTP or IMAP requires `sting` to be modified.

2.4.3.2 Tulip

Per-hop packet loss on the forward path towards a destination host may be inferred using the IP-ID technique used in `tulip` [1]. The IP-ID technique exploits the common implementation strategy of assigning IP-ID values from a counter held centrally in the line-card or router when generating packets [28]. `tulip` infers

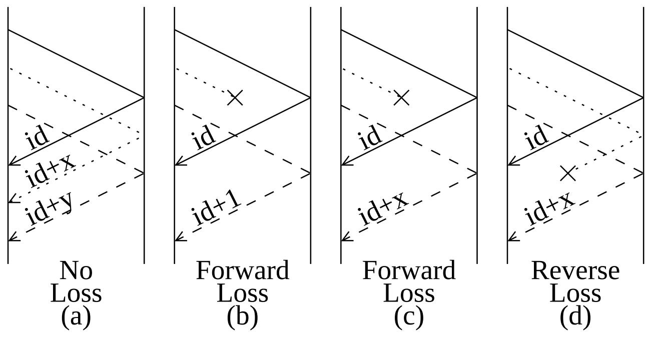


Figure 2.12: Detecting loss with `tulip` using the IP-ID field (source: User-level Internet path diagnosis [1])

per-hop packet loss by sending three TTL-limited probe packets, with the intent of inducing loss on the middle packet. The two outer packets are small *control* packets while the inner packet is a large *data* packet. The large data packet is more likely to be dropped than the two small control packets if the packets encounter a congested link, as it requires more queue resources. Loss on the forward path can be inferred if the middle packet is dropped on the forward path and the two control packets arrive back-to-back at the router where their TTL expires. As shown in figure 2.12b, the control packets will be assigned incremental IP-ID values, and it can therefore be inferred that the loss occurred on the forward path.

However, `tulip` is unable to infer reverse path loss with this technique. This is because the IP-ID values in the control packets are the same in this circumstance (figure 2.12d) as when the data packet is lost on the forward path and a packet is generated for a third party in between the two control packets (figure 2.12c). The technique also depends on each router assigning incremental or some other predictable series of IP-ID values so that `tulip` can establish the arrival order of the three packets. If `tulip` cannot determine the strategy used to assign IP-ID values to responses, then it cannot determine the direction of packet loss. In addition, `tulip` cannot definitively determine that any measured loss occurred at the hop being targeted, because a probe may be lost anywhere prior to the targeted hop.

2.5 Reordering

2.5.1 Definition

A stream of packets is defined as an ordered series if packets are delivered to the destination in the same order as they were sent. Thus, reordering occurs if any packet in a series of packets arrives at a destination in a different position than it was sent in. Historically, packet reordering was believed to be an unusual event which signalled some significant underlying event in the IP path, such as a route change, although it is now accepted that this is not the case [43].

Reordering can occur in several scenarios. If load-balanced paths with different delays exist, then reordering can occur if the first packet in a series takes a longer path than one of the packets which follow. Reordering can also occur when a router forwards packets in a different order than their arrival order. One strategy to implement a fast forwarding path is to combine parallel paths made with slower parts [43]. When two back-to-back packets enter a forwarding path of such a router, it is possible that the first packet may exit the forwarding path behind the second packet if the processing delay for the second packet is shorter at that particular time. Packets can also be reordered by a router if packets of different sizes are placed into different input or output queues that optimise the treatment of a packet based on its size.

2.5.2 Motivation for Reordering Measurement

2.5.2.1 End-to-End Motivation

End-to-end packet reordering is not a particularly interesting metric, as few services simply echo reply packets back to the sender. However, being able to establish if any reordering is taking place end-to-end is an important first step towards understanding reordering behaviour exhibited by an Internet path.

2.5.2.2 One-way Motivation

The ability to detect if packet reordering is occurring in a particular direction is important for many classes of applications, but especially to TCP. Packet reordering can have several negative impacts on the ability of TCP to use the network in a fair manner [44]. As TCP acknowledges the last in-sequence segment received, a TCP sender with fast retransmit will retransmit the next unacknowledged data in the sender's window when a third duplicate acknowledgement is received, inferring that the next segment was lost. If the segment was reordered and not lost, then retransmitting it will result in the network doing unproductive work, and the sender will needlessly reduce the rate at which it sends new segments into the network believing that it encountered congestion. An acknowledgement that arrives out of order acknowledging multiple segments will cause a burst of new segments to be sent as in the loss case described in 2.4.2, while a late acknowledgement that acknowledges old segments will be discarded by the receiver, and performs no useful function.

Reordering is also important to real-time applications that trade off the quality of data presented with the prompt processing and displaying of incoming data. Such an application may be confused by a reordered packet, present the data available, and discard any packet that subsequently arrives late.

2.5.2.3 Per-hop Motivation

Measurement of reordering at a per-hop level provides the ability to determine where reordering is occurring, and may offer insight into the possible cause. For example, a technique may be able to infer the IP topology of alternating paths of load-balanced IP paths where alternating packets take different IP paths.

2.5.3 Reordering Measurement Techniques

End-to-end reordering can be measured, as with delay and loss, by sending a series of probe packets towards a destination, and then observing the order of the reply packets. If the reply packets arrive in a different order than the order their cor-

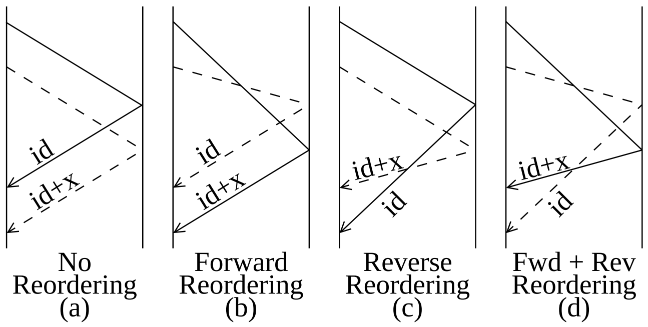


Figure 2.13: Detecting reordering with `tulip` using the IP-ID field (source: User-level Internet path diagnosis [1])

responding request packets were sent in, then reordering has occurred somewhere in the path. However, such a measurement might fail to identify reordering if it occurred in both the forward and reverse directions.

2.5.3.1 Tulip

`tulip` [1] measures one-way reordering using a variation of the IP-ID technique it uses to measure one-way loss. `tulip` sends two packets to the target that solicit an ICMP message with incremental IP-ID values; it can do so with UDP probes to high-numbered ports, TCP SYN probes, ICMP echo request probes, and ICMP timestamp request probes. The IP-ID field records the arrival order of the probe packets at the target.

The four reordering cases measurable are shown in figure 2.13. If the responses arrive in order from the destination with sequential IP-ID values, they were not reordered (unless they were reordered an even number of times on either the forward or reverse paths). If the responses arrive out of order, but with sequential IP-ID values, then reordering occurred on the forward path. If the responses arrive out of order without sequential IP-ID values, then reordering occurred on the reverse path. Finally, if the responses arrive in order but without sequential IP-ID values, there was reordering on both the forward and reverse paths.

It is also possible use a similar technique to infer the rate of per-hop reordering on the forward path, by sending TTL-limited probes which solicit ICMP responses with incremental IP-ID values from intermediate hops. While reverse path reordering can be determined from the reply packets, it is not reliable to use this information to determine where on the reverse path packet reordering occurs unless the path (or segments where reordering occurs) is known to be symmetrical.

2.6 Capacity

2.6.1 Definition

In this thesis, capacity is defined as the maximum possible volume of data that can be transferred by the network between a source and a destination over a defined time period. Capacity is defined this way in line with common practice [2] in order to avoid confusion that arises through use of the term ‘bandwidth’, which is defined in the communications field as the difference between the highest and lowest frequencies of a transmission channel.

If the source and destination are adjacent and directly connected network interfaces, then the capacity is the serialisation rate of the link between them, less any packetisation overhead below the IP layer. If the source and destination are adjacent systems with layer 2 devices such as Ethernet switches between them, then the capacity between the systems is the smallest capacity of the links between them. Some IP hops consist of multiple physical paths used in parallel; in these cases, the capacity of the hop will be the sum of the capacity of each path. Finally, if the source and destination are separated by a series of IP hops, then the capacity of the path corresponds to the hop with the smallest capacity in the path.

As a packet traverses a network path from a source to a destination, it is likely to encounter links with different capacities. The maximum throughput that can be obtained between a source and a destination will be the maximum capacity available from the slowest link of the path, defined as the *narrow link* by Dovrolis et al. in [2].

2.6.2 Motivation for Capacity Measurement

The capacity of a path places a limit on how quickly some volume of data is able to be transferred between a source and a destination. Round-trip capacity is not particularly interesting to measure, as few applications are dependent on the ability

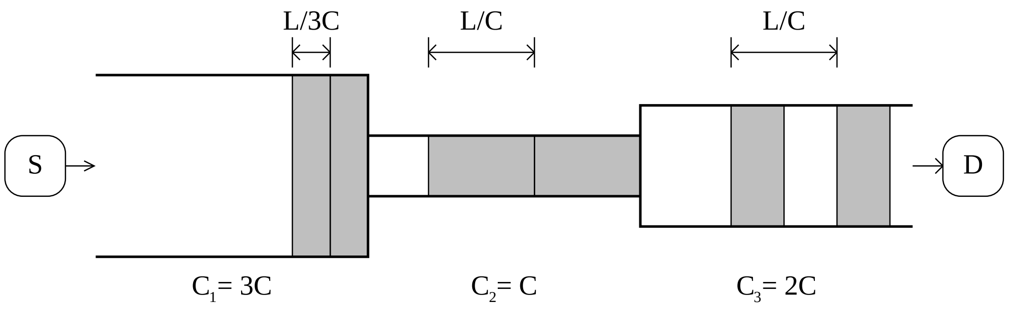


Figure 2.14: Illustration of the packet-pair technique (source: Packet-dispersion techniques and capacity estimation methodology [2])

of an end-to-end path to carry the same packet both to and from a destination. The performance of a capacity-bound TCP bulk transfer is almost certainly limited by the capacity of the path that the data is carried over, assuming that the path in the reverse direction is capable of delivering acknowledgement packets in a timely and reliable fashion.

Therefore, capacity estimation is primarily concerned with the one-way measurement of capacity from a source to a destination. The capacity of a path is limited by the slowest link in the path. The ability to identify the capacity of the narrow link and the position in the path where it occurs is helpful for determining where capacity is limited.

2.6.3 Capacity Measurement Techniques

Capacity estimation is a complex and difficult problem to solve with currently available Internet protocols. Lai and Baker suggest that the ideal properties of a bandwidth estimation technique are that it is accurate, quick, robust, and unobtrusive [6]. Current capacity estimation techniques measure network links with varying levels of accuracy, speed, robustness, and obtrusiveness.

2.6.3.1 Packet Pairs

The capacity of the narrow link C can be estimated using packet-pairs, as first discussed by Keshav in [45]. Figure 2.14 illustrates the packet-pair technique. A source sends two packets of the same size back-to-back to a destination. Each packet encounters a serialisation delay, the length of which is determined by the size of the packet. If the two packets enter a link back-to-back, then the capacity of that link can be inferred by the dispersion of the two packets as they exit the link; that is, the time elapsed between the last bit of the first packet from the last bit of the second packet. Assuming that there is no cross traffic, the dispersion of the packet-pair will not change after the packets exit the narrow link, and the capacity of that link can be calculated by the by dividing the packet size by the spacing of the two packets as they arrive at the destination.

There are a number of challenges to the packet-pair technique, and to capacity estimation in general. One of the most significant challenges is accounting for the effects of cross traffic on the probes [6]. First, in order to correctly estimate the capacity of the path, it must be possible for at least one packet-pair to be sent over the narrow link back-to-back; the likelihood of this happening depends on the effect that cross traffic has on the packet-pair before it is sent over the narrow link. Second, if a packet-pair exits the narrow link back-to-back, the dispersion of the two packets may change according to the cross traffic they subsequently encounter. The packet-pair may be separated further by cross traffic injected between the two packets. Similarly, if the first packet in the pair queues behind cross traffic and the second packet is allowed to catch up to the first, then the dispersion between the packet-pair will be compressed. Third, if the packet-pair dispersion is communicated by acknowledgement packets that are forwarded along the reverse path, these packets may also be distorted as they are forwarded.

One approach to reducing the scope for reverse-path distortion of packet-pair dispersion is to record the arrival times of each packet-pair at the destination. Doing so can significantly increase the accuracy of the measurement, although it requires explicit support at the destination host. However, the capacity estimation technique still has to account for distortion of packet-pair distortion from cross traffic on the forward path. Early work, such as `bprobe` from Carter and Crovella [46] and preliminary work on `nettimer` by Lai and Baker [6] focused on statistically determining which dispersion measurement relates to the narrow link capacity, typically by determining the mode of the distribution of capacity estimates [2]. However, Paxson noted in [47] that the distribution of packet-pair dispersion measurements is multi-modal, and that the capacity estimate is not necessarily the mode of the distribution. Dovrolis et al. [2] presented a seminal review of the impact of cross traffic on capacity estimation tools, and provided a capacity estimation methodology [48].

There are currently two known methods to infer which packet dispersion mode corresponds to the underlying capacity of the narrow link. The first method, introduced by Dovrolis et al. [2] and implemented in `pathrate`, infers which mode corresponds to the underlying capacity by inferring which modes would result in under-estimating the capacity of the path. These modes are referred to as belonging to the Sub-Capacity Dispersion Range (SCDR). By definition, the next mode in the series corresponds to the capacity of the path. This mode is referred to as the Capacity Mode (CM).

`pathrate` infers the SCDR, and by inference the CM, by sending packet-trains of increasing length until the distribution has a single mode. As long packet-trains are more likely to become dispersed by cross traffic, measuring the dispersion of these packet-trains will under-estimate the path's capacity. The width of the single mode corresponds to the SCDR.

The second method for inferring which dispersion value corresponds to the capacity, introduced by Kapoor et al. [49] and implemented in CapProbe, makes the observation that the packet-pair from a large number of samples which incurred the minimum delay (the sum of the delay of both packets) is likely to have encountered no cross traffic at any link, including the capacity-limiting link. The dispersion of this packet-pair therefore reflects the capacity of the narrow link.

2.6.3.2 Pathchar

The capacity of each hop on the forward path can be inferred using a number of different techniques. Jacobson was the first to present a tool capable of estimating the capacity of each hop on the forward path. Jacobson's tool, `pathchar` [7], uses a Variable Packet Size (VPS) technique where the extra delay incurred by a series of progressively larger packets is measured [50]. As the length of time it takes to serialise a packet grows with packet size, the VPS technique measures an *RTT line*.

`pathchar` uses TTL-limited packets to probe each hop on the forward path towards the destination. The assumption of the VPS technique is that a larger packet will incur additional delay compared to a smaller packet proportional to the serialisation rate of the link measured. As discussed in section 2.3.1, a packet encounters both fixed and variable delay factors; the main fixed delay factors consist of the time it takes to serialise a packet of a given size, while the main variable factors are the queueing and forwarding delays. `pathchar` sends a large number of packets of varying sizes to each hop, with the assumption that at least one packet for each packet size sent will encounter minimum queueing and forwarding delays for all hops in the path. The packets with the shortest observed round-trip-times therefore measure the serialisation and propagation delays for a particular packet size. As the serialisation delay for a packet is a function of the packet's size, `pathchar` fits the RTT line using the linear least squares fit method.

There are a number of limitations to the technique used by `pathchar`. First, as was noted by Downey in [50], the measurement of the minimum delay on a per-hop basis, and thus the slope of the RTT lines used in estimation of the capacity, depends on all queues in the path being empty at least once while a particular hop is being measured. For this reason, `pathchar` may not accurately estimate the capacity of links after a persistently congested link, because at least one probe for each packet size is required to be forwarded through all routers without experiencing congestion. Second, store-and-forward devices such as Ethernet switches cause the technique to significantly underestimate the capacity of the hop, because there are multiple serialisations in the path [51]. The `pathchar` technique assumes a single serialisation. Third, the clock available on a host which runs `pathchar` is often not sufficient to measure the extra time to serialise a 1500 byte IP packet on high-speed paths such as OC48c links [33, 51], because the resolution of most clocks for most operating systems is a microsecond. Fourth, the technique is not able to infer the correct capacity of a multi-channel link, because each probe follows a single channel through the link. Finally, the technique is unable to infer the capacity of hops on the reverse path.

2.6.3.3 Nettimer

Another approach to measuring capacity on a per-hop basis is found in `nettimer` by Lai and Baker [52]. The `nettimer` technique works by first estimating the RTT line using the VPS technique described for `pathchar`, but for the entire forward-path, rather than per-hop. In this case, the slope of the RTT line represents the sum of per-hop RTT lines for all hops on the forward path. Then, it uses a technique known as *packet tailgating*, where pairs of packets are designed to queue together to the hop being measured. The first packet – the *tailgater* packet – is a TTL-limited MTU-sized probe which is set to expire at the end of the particular hop being measured. The tailgater packet is typically 1500 bytes in size, which corresponds to the vast majority of path-MTU values in the Internet due to the wide-spread nature of Ethernet. The second packet – the *tailgated* packet – is a minimum-sized packet which continues onto the destination. The tailgated packet is 40 bytes in size, which corresponds to a TCP FIN packet encapsulated in an IPv4 header with no options. The tailgated packet will queue behind the tailgater until the tailgater is discarded, and the tailgated packet will then be forwarded with minimum delay to the destination assuming no cross traffic. `nettimer` conducts the packet tailgating phase for each hop in the forward path, beginning with the first hop. As the tailgated packet is delayed by the tailgater packet until the tailgater is dropped, the receiver can determine how long the tailgated packet was delayed by the tailgater compared to the previous hop, and thus the time to serialise the tailgater at the hop where it was dropped.

The advantages of this technique compared to the `pathchar` technique are that it sends less probes, can detect multi-channel links by sending additional small probes with the tailgater probe, and does not rely on the ICMP time exceeded message for the tailgater probe being delivered promptly, because these messages are not used in the delay calculation. There are a number of limitations of this technique, as discussed by the authors of the technique [52]. First, the host system must be able to supply the appropriate network interface with packets fast enough for them to be sent back-to-back. Second, the technique relies on the packets being sent back-to-back on the link being measured, which cannot be guaranteed on

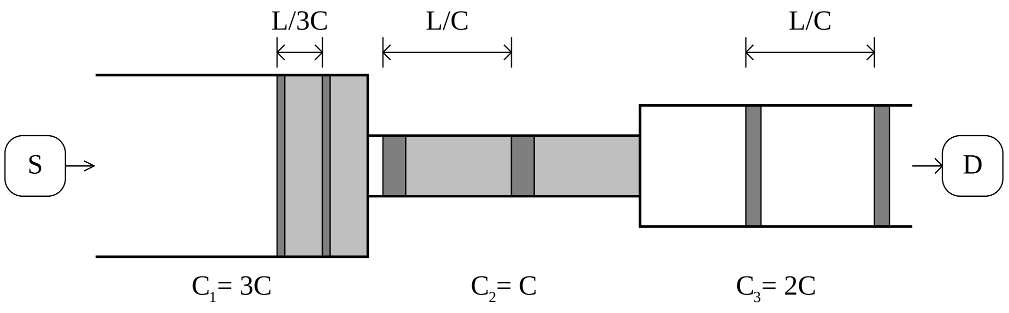


Figure 2.15: Illustration of cartouche probing

a link which is much faster than the previous link; assuming a 1500 byte tailgater packet and a 40 byte tailgated packet, this technique will fail when the ratio of output to input bandwidths of a router exceeds 37.5. Third, a persistently congested link anywhere in the path reduces the ability to measure the capacity of any hop in the path, because the small packet is always routed to the destination. Fourth, any measurement error for any of the hops accumulates and affects the accuracy of capacity estimates for later hops. Finally, this technique also suffers from the multiple-serialisation problem that causes `pathchar` to underestimate capacity [51].

2.6.3.4 Cartouche Probes

A third approach to measuring capacity on a per-hop basis is to use cartouche probes, introduced by Harfoush, Bestavros and Byers [53]. Cartouche probing is able to estimate the capacity of a contiguous set of links, although this thesis focuses on using cartouche probing to measure capacity on a per-hop basis. The approach, illustrated in figure 2.15, is to send a series of packets that alternate between large tailgater packets and smaller tailgated packets. The tailgater packets are TTL-limited and set to expire at a particular hop, while the tailgated packets continue to the destination and are designed to record the dispersion of the tailgater packets as each tailgater is discarded. The tailgater packets are known as *magnifier* packets while the tailgated packets are known as *marker* packets. As the marker packets are small, their dispersion is less likely to be affected by cross traffic compared with the magnifier packets. In figure 2.15, the marker probes record the dispersion of the large tailgater packets when they were discarded at the egress of hop 2.

The cartouche approach to estimating the capacity of each hop consists of a number of steps. To begin with, the capacity of the forward path is estimated using a packet-pair technique, such as the packet-pair techniques of `pathrate` and CapProbe. The purpose of this first step is to determine how many cartouche probes are required to be sent between marker probes so that the marker probes remain separated until they reach the destination. This condition will hold with two cartouches consisting of a 1500 byte tailgater and a 40 byte marker provided at no point in the path the difference between the capacity of a hop and the capacity of the path is greater than $(1500 + 40) / 40 = 38.5$, because that would provide the ability for the marker packets to serialise back-to-back at the capacity limiting link.

The next step attempts to determine the capacity of a particular hop using the cartouche probe formation shown in figure 2.15. If the capacity estimate for the hop is less than the capacity estimate for the path prefix up to the hop, then the requirement for the cartouche probes to be sent back-to-back across the target hop was fulfilled. Otherwise, a different probing structure known as a cartouche train is used to estimate the capacity of the hop. The key observation of a cartouche train is that though it is not possible to have a magnifier probe queue with two marker packets immediately preceding and following, it might be possible to cause the marker packets to become more dispersed over the hop by injecting additional magnifier packets, which provides the basis for estimating the capacity of the hop.

As with other capacity estimation techniques which measure capacity using packet dispersion measurements, the accuracy of the cartouche approach is limited by distortion effects of cross traffic on the spacing of the marker packets. The approach described by Harfoush et al. is to select the last local mode of estimated capacities based on the reasoning that marker packets are more likely to be dispersed after the hop being targeted. As with other per-hop capacity estimation techniques, cartouche probing requires each hop to be probed separately to previous hops. As many samples of a path are required in order to build a robust estimate, this can result in significant measurement load on the path.

2.7 Summary

The tools and techniques reviewed in this chapter have a number of characteristics in common, as most are designed to use functionality found in existing hosts and routers. Using existing functionality allows tools to be designed and implemented without requiring new protocols to be deployed. The challenges facing active measurement therefore consist of increasing the accuracy of the technique, reducing the impact of measurement on the network, and increasing the robustness of the technique. To some degree, these challenges work against each other. For example, it can be tempting to increase accuracy of a tool or technique by probing the path more, thereby increasing the impact of measurement on the network.

Some tools and techniques overcome the challenge of increasing accuracy and robustness of a technique without increasing the probe rate by obtaining explicit cooperation of an end host. However, many measurement tools which can be useful for inferring per-hop behaviours have little scope for seeking the specialised cooperation of an intermediate router due to denial of service concerns.

The next chapter examines the limitations of existing measurement techniques in further detail. In doing so, the motivation for a protocol designed to support per-hop measurement of Internet packet dynamics is further outlined, and the desirable features of such are protocol are formally identified.

Chapter 3

Overcoming Common Limitations

3.1 Introduction

The previous chapter examined the motivation to measure four metrics – delay, loss, reordering, and capacity – on an end-to-end, one-way, and per-hop basis. Currently available tools and techniques to measure each metric on these bases were reviewed, and the limitations of each approach were discussed. In doing so, a number of limitations that are common across measurement techniques were highlighted – particularly with per-hop measurement techniques – which are difficult to overcome with the current absence of an Internet protocol with support for per-hop measurement of Internet packet dynamics.

This chapter looks towards a protocol designed for per-hop measurement of Internet packet dynamics by discussing how common limitations to measurement might be overcome. This chapter begins by reviewing common limitations of current measurement tools and techniques. Then, a series of desirable features and protocol requirements for such a measurement protocol are detailed. Some desirable features follow directly from addressing a common limitation, while others are more general protocol requirements. Possible ways of implementing each desirable feature are examined. This chapter concludes by looking ahead at chapter 4, where a protocol designed for per-hop measurement of Internet packet dynamics is presented.

3.2 Common Limitations

3.2.1 Forward Path Bound

Many measurement techniques are limited to measuring the forward path, even though the reverse path is often also of interest. This limitation arises because an IP packet is unidirectional, and the TTL field for any reply packet sent from a destination is set to a default pre-determined value. Unless the destination and all routers on the forward and reverse paths permit source-routed probes, or the destination provides a method to cooperate with the source, a source is limited to measuring the forward path towards a destination due to the lack of cooperation from the destination.

In "Heuristics for Internet Map Discovery" [54], Govindan and Tangmunarunkit reported that 8% of routers measured permitted source-routed probes. However, they also received many abuse complaints from operators who noticed the source-routed probes. While source-routed probes have utility for topology discovery, they have little utility for measurement of packet dynamics, as IP options require extra processing and therefore have different dynamics to packets that do not have IP options [25].

3.2.2 Static Path Behaviour Assumptions

In addition to being forward path bound, measuring a path on a per-hop basis with TTL-limited probes is not a robust, reliable, or accurate method. It is not possible to definitively separate the behaviour of a particular hop from the behaviour of previous hops when using TTL-limited probes. Many per-hop measurement techniques assume that each extra hop travelled towards a destination has an additive effect on delay, jitter, loss, and reordering, and that it is possible to estimate the behaviour of a single hop in the path by subtracting the behaviour of the path prefix leading to the hop. This assumption in turn depends on both the path and its behaviour being constant over the course of the measurement.

While most Internet routes are persistent over time scales of days, a few change more regularly than this [55]; in particular, 9% of routes measured with `trace-`

`route` in [55] changed on a time scale of tens of minutes. Similarly, while the behaviour of most Internet paths appears to be stable over timescales 10-30 minutes [56], there is evidence to suggest that at much shorter timescales, such as the lifetime of a TCP connection, significant variation in RTT can be observed [57].

If the IP topology of the forward path is not constant during measurement, due to IP-layer load balancing or a change of route, then it is no longer safe to use the subtraction method to infer per-hop Internet packet dynamics. This is because successive TTL-limited probes may follow different paths. If some ICMP messages come from a different router to other messages in response for the same TTL, then the path change can be detected. In this case, a tool can detect the requirement that each probe follows the same path is not met and can stop probing. If a prefix of the forward path changes while later hops are being probed but the ICMP messages come from the same hop, the path change will not be detected, and the measurement will be flawed. If the IP topology of the forward path remains constant, but the load of the path changes, then the assumption it is possible to measure the additive effect of subsequent hops on packet dynamics will not hold.

3.2.3 ICMP Rate-Limiting

Some measurement tools and techniques rely on the timely delivery of an ICMP message in response to a probe, as ICMP is an integral part of the IP protocol [58] and is thus ubiquitous. Generating an ICMP response requires significant additional resources from a router compared to forwarding a packet. In addition, some ICMP messages are not absolutely critical to the operation of the Internet. Implementations of ICMP in both end-hosts and routers therefore often limit the rate at which ICMP packets are generated. ICMP rate-limiting is problematic for techniques that rely on these messages, as rate-limiting constrains the speed at which a host can measure the network. In addition, any probe which is not replied to due to rate-limiting causes the network to do unproductive work. ICMP rate-limiting is particularly problematic for techniques that depend on ICMP messages when measuring packet loss, as any absence of an expected ICMP message could be due to rate-limiting and not packet loss.

The practice of rate-limiting ICMP packet generation was first formalised in RFC 1716 – Towards Requirements for IP Routers [59]. RFC 1716 highlights two significant problems associated with sending ICMP messages. First, generating an ICMP message consumes resources on the reverse path, which may contribute to congestion. Second, generating an ICMP message consumes router resources such as CPU time and memory, which may interfere with the router's ability to run routing services such as BGP.

RFC 1716 outlines a number of different ways in which ICMP rate-limiting may be implemented. All three scenarios reduce the robustness of measurement techniques that rely on ICMP response messages. The first scenario described is a count-based rate-limiting method that works by sending an ICMP packet for every N packets per source host. The second scenario described is a timer-based rate-limiting method that works by sending an ICMP packet once for some period of time per source host. The third scenario described is a bandwidth-based rate-limiting method that limits the percentage of an interface's capacity that may be consumed by ICMP packets that are originated at the interface.

3.2.4 ICMP Disabling

In addition to rate-limiting ICMP, some routers and end-hosts are configured to not send ICMP messages; some are configured to not send any messages at all, while others disable particular ICMP types. For example, a router may send ICMP time exceeded messages, but be configured to not send other ICMP message types, such as timestamp replies, echo replies, or destination unreachable messages. If a measurement technique depends on ICMP support which is disabled, then the technique will either not work, or operate with reduced functionality.

Recent research as part of work done with `cing` [38], discussed in section 2.3.3, found that of 20206 random targets measured, the ICMP functionality required for `cing` to measure per-hop queueing delay – ICMP echo, timestamp, and time exceeded – was available in 92.92% of routers measured. However, 37.22% of paths measured had all routers with the required ICMP functionality available. Follow-up work by Mahajan et al. [1] found similar results, and added observations that some

ICMP disabling is systematic. For example, they found that none of AT&T's routers measured had ICMP timestamps enabled.

3.2.5 Firewalls and Filters

An increasing number of networks stop some packets from reaching their destination by permitting selected packet types and protocols to pass, while discarding the rest. There are many possible reasons for a deny-by-default firewall configuration. Network security concerns are often cited as one reason for restricting what may pass [60]. The effect this has on measurement depends on the configuration of the firewall and the minimum protocol functionality required by a measurement technique. UDP and TCP packets to unused ports are likely to be filtered, as filtering these packets does not impact on supported services. However, filtering these packets reduces the ease at which the forward topology may be discovered using `traceroute`. Similarly, ICMP packets are often filtered if they are not considered crucial to the operation of the network. This behaviour prevents informational ICMP probes such as echo request and timestamp request from being delivered to the destination, and prevents Path MTU Discovery (PMTUD) from working [60].

3.2.6 Hidden Queues

A router is designed to forward packets as fast as it can. Many routers optimise for the common case by implementing what is known as a *fast path*. The fast path is the path taken through a router by a packet that does not require special processing. A packet that requires special processing is placed in a queue outside of the fast path for processing by a separate module. Precisely which packets require special processing depends on the router's implementation. However, likely candidates for special processing include packets with IP options such as source-routing and record-route, packets which require fragmentation, and packets which cause an ICMP message to be returned to the source.

When a packet that requires special processing follows a different path through a router compared to other packets, a 'hidden queue' is created, which has properties of its own. A hidden queue has a number of negative effects on the accurate measurement of network properties, as packets in such a queue may be lost or delayed in a manner which does not reflect the way other traffic queues. A hidden queue may therefore create artifacts that are only seen by measurement probes and not by regular traffic. Mahajan et al. provide an example of a hidden queue in [1], where ICMP timestamp request packets queue internally while the router is busy with higher-priority tasks.

3.2.7 Specialised Cooperation Required

In order to improve the accuracy of particular techniques, some tools seek the cooperation of the destination host. These tools typically operate in a client-server mode. The server is often a user-space process that runs on a pre-negotiated or predetermined port, or is a service accessed with a web browser. The client negotiates a measurement with a server, and then obtains results out-of-band. The specialised cooperation of an end-host provides the ability to measure paths that a client is unable to measure by itself. Depending on the measurement, it can also lead to an increased result quality.

However, specialised cooperation does not lend itself to being widely deployed, for a number of reasons. First, routers are poor choices to run specialised measurement software on. Router vendors and operators may perceive that including specialised measurement software needlessly increases implementation complexity and detracts from the desire to build a robust system designed to forward packets as fast as possible. Not being able to use a router as a measurement target reduces the ability to measure a path on a per-hop basis. Second, offering such a service may provide a vector for denial of service or abuse, as a service may have to allocate and use resources in support of the measurement. Third, while accurate measurement is of operational benefit, it is more difficult to argue for the widespread deployment of many specialised measurement services, compared to offering a general purpose protocol that is shared amongst measurement techniques.

Link type	Bit rate (Mb/s)	Bytes between clock ticks 1ms	1μs
10baseT	10	12.21kB	1.25B
100baseT	100	122.07kB	12.50B
OC3c POS	155.52	189.84kB	19.44B
OC12c POS	622.08	759.38kB	77.76B
1000baseT	1000	1220.70kB	125.00B
OC48c POS	2488.32	3037.50kB	311.04B
OC192c POS	9953.28	12150.00kB	1244.16B

Table 3.1: Bytes between clock ticks for various link types

3.2.8 Imprecise Timekeeping

In many situations, a satisfactory measurement can be made using an imprecise clock or using low resolution timestamps. For example, the user may simply require an approximate delay estimate to quickly infer round trip delay and jitter, in order to infer the nature of some operational problem. However, many applications and techniques require more precise timekeeping. The support provided for precise timekeeping in Internet measurement is often inadequate.

Fundamental to precise delay measurement is a precise clock and timestamps of an appropriate resolution. Many host-based measurement tools use user-space timestamp facilities, such as gettimeofday. The resolution and accuracy of the timestamp facilities provided by various operating systems for the gettimeofday function is not sufficient for some delay measurements. The timestamp returned by the gettimeofday function has a maximum resolution of one microsecond, and the current time is reported poorly on some systems. Similarly, the timestamp included in an ICMP timestamp reply has a maximum resolution of one millisecond. The resolution of these timestamps is fixed, and limits the ability to measure per-hop delay and jitter on high speed networks.

Table 3.1 shows the number of bytes that can be serialised in 1 millisecond (the maximum resolution of an ICMP timestamp) and in 1 microsecond (the maximum resolution of gettimeofday). A timestamp resolution of 1ms is inadequate to measure jitter on a fine-grained timescale; at OC192c, almost 12MB of data may pass before the timestamp advances.

3.2.9 Free-Running Clocks

The ability to measure one-way or per-hop delay requires synchronised clocks, or the ability to determine the offset from real-time of each clock involved. Currently, it is difficult to determine basic clock statistics from a third-party host, such as the clock's offset from real-time and its resolution, unless it offers an NTP service. Because of this, there are no known techniques for measuring the one-way delay between two arbitrary points on a path between a source and a destination.

3.2.10 Unable to Reliably Resolve Router Aliases

It can be useful to resolve router aliases when inferring Internet topology or measuring a path from multiple points. Currently available techniques to resolve router aliases are complex and rely on router implementation quirks; that is, router implementation strategies that are not specified in any Internet protocol standard. Hal Burch reviews three different approaches to resolving router aliases in his thesis [19]: probing unused UDP ports in order to solicit ICMP messages with the same source address, inferring aliases by soliciting responses with progressive IP-ID values, and inferring aliases by causing routers to rate-limit ICMP generation. The first two of these techniques are discussed in section 2.2.4.

Burch [19] found that of the three techniques, the IP-ID technique was the most successful at resolving router aliases, and could resolve nearly twice as many router aliases than the UDP technique. However, the IP-ID technique required an order of magnitude more probes to be sent. The last technique – resolving router aliases by correlating rate-limiting episodes – requires more probes to be sent than the UDP technique and resolves less aliases. As discussed in section 2.2.4, the IP-ID technique is not useful for resolving IPv6 router aliases, because the IPv6 header does not have an ID field.

3.3 Desirable Features

In this section, a series of features that would be useful in a protocol designed for per-hop measurement of Internet packet dynamics are defined.

3.3.1 Integrated Path and Packet Dynamics Measurement

As discussed in section 3.2.2, it is not safe to assume static path behaviour. As a result of experience gained in developing `pathchar`, Van Jacobson identified a need for a more robust per-hop measurement technique than the TTL method as important for future research [7].

In order to measure per-hop dynamics of a packet as is routed to its destination, each router which forwards the packet is required to signal, to the sender, when the packet arrived. The required primitive that records when a packet arrived is a timestamp. Therefore, it is possible to segment a path into individual hops if it is possible to obtain a timestamp that represents the time the packet was received for each router which forwards the packet. The path becomes segmented because the delay properties of each hop in the path can be determined using timestamps reported by adjacent hops. A number of possible implementation strategies exist for obtaining a timestamp from each router in a path.

One possible implementation strategy is for each router in a path to send a reply to a specially marked probe before forwarding the probe to the next-hop, similar to the methods used with the traceroute IP option [26] and `mtrace` [61]. With this approach, the sender has scope to control the size of and contents of each probe packet sent, as packet dynamics are communicated separately to the probe. However, there are a number of limitations to this approach. First, a single probe could cause an order of magnitude more packets to be generated in response to the probe, which provides a vector for denial of service attacks. Second, in addition to forwarding the packet, a router would be required to generate a separate reply packet from scratch, which is likely to require additional resources and thus be rate-limited.

Another possible implementation strategy is for each router in a path to embed information useful for measurement in the specially marked probe packet, similar to the method used with the IP record route option [24]. The main limitations of this approach are that: the probe packet must be large enough for each router to embed useful information in the packet; if the probe packet is lost, all embedded information is lost with it; and the source host has little control over the contents of the packet as it is altered through the network. One advantage of this approach is that if the protocol is designed for efficient processing in the forwarding path, it does not create a denial of service vector.

3.3.2 Able to Measure the Reverse Path

As discussed in section 3.2.1, many measurement techniques are only capable of measuring the forward path, even though the reverse path is also often of interest. For example, a protocol that enables a source host to determine the sequence of routers that a packet traverses on the reverse path is immediately useful to both operators and end-users. Existing techniques can only provide detailed per-hop information for the forward path without specific cooperation from the destination host. A number of possible implementation strategies exist for a protocol capable of measuring the reverse path.

First, a measurement control protocol could be defined which negotiates a measurement for the reverse path with a target host, similar to the method used by OWAMP. The advantage of this method is that it provides the ability to control the measurement process, which is important to measurements that require precise probing control. As useful as a measurement control protocol is, experience with existing measurement protocols indicates that they are unlikely to receive widespread adoption due to the complexity involved in implementing, deploying, and controlling access to deployed systems [1].

Second, an echo protocol could be defined where a source host sends a request packet and expects the destination host to send a reply packet. The reverse path is measured from the point where the echo reply packet is sent from the destination host. The advantage of this method over a measurement control protocol is that an echo protocol is simple enough to be implemented in a kernel or in a line card, as an echo protocol does not require state to be kept at the end host.

3.3.3 Minimal Overhead in Routers

As discussed in sections 3.2.3 and 3.2.4, existing Internet protocol features such as ICMP, which some techniques depend on to infer packet dynamics, are often rate limited or disabled, due to vectors they can provide for overwhelming the ability of a system to operate normally. Similarly, the path taken through a router by a probe used for measurement of packet dynamics may be different to the path taken by other traffic, introducing hidden queues, as discussed in section 3.2.6. However, a measurement protocol capable of measuring packet dynamics on a per-hop basis will require support from routers. In order to make a protocol useful for measurement, the protocol must introduce minimal overhead so that measurement probes can follow the same forwarding path as any other packet through the router.

3.3.4 Precise Timestamping

As discussed in section 3.2.8, current protocol support for precise timestamps is limited. As networks get faster, the ability to accurately measure and characterise per-hop behaviours on very high speed networks is challenged. In his Ph.D thesis on high-resolution passive network measurement, Donnelly discussed how the required timestamp resolution and clock frequency might be defined for passive measurement of various link types [62]. For example, if it is sufficient to be able to distinguish the arrival order of two packets on a link, then the minimum timestamp resolution required is "less than the transmission time of a minimum sized packet on the media." For Ethernet links, the minimum packet size is 64 bytes, which corresponds to 46 IP packet bytes [63]. For Packet Over SONET (POS) links, the minimum IP packet size is 28 bytes.

However, this resolution is not sufficient to measure jitter. As the minimum transmission unit on most networks is a byte, an ideal clock resolution for measuring jitter is an eighth of the bit frequency of the link being measured [62]. Therefore, the ideal resolution of a timestamp used for measuring delay jitter on a particular link type is a function of the time it takes to serialise a single byte on that link. The

Link Type	Bit Rate (Mb/s)	Clock Wrap Time 32 bits	48 bits	64 bits
10baseT	10	00:57:15	$>$ 7 years	$>$ 467953 years
100baseT	100	00:05:43	$>$ 37 weeks	$>$ 46795 years
OC3c POS	155.52	00:03:40	$>$ 23 weeks	$>$ 30089 years
OC12c POS	622.08	00:00:55	$>$ 5 weeks	$>$ 7522 years
1000baseT	1000	00:00:34	$>$ 3 weeks	$>$ 4679 years
OC48c POS	2488.32	00:00:13	$>$ 1 week	$>$ 1880 years
OC192c POS	9953.28	3.452 sec	$>$ 2 days	$>$ 470 years

Table 3.2: Byte clock wrap time for various link types

timestamps used in the measurement process should be able to be used to detect jitter on future networks.

For these reasons, this thesis argues that a measurement protocol should provide support for retrieving the timekeeping information necessary in order to improve delay measurement. The protocol should allow a host or router to use a clock suitable for measuring delay at a suitable scale for the situation. For a Personal Computer (PC)-based host, this may mean using a software clock based on the Time Stamp Counter (TSC) register that counts CPU cycles as suggested by Pásztor and Veitch [64]. For a router, this may mean using a clock which runs at an eighth of the interface's bit frequency so that per-byte delay into the router can be determined.

Table 3.2 shows the length of time it will take the value from a clock that counts bytes to wrap around for various link types, depending on the width in bits of the clock value. Table 3.2 shows that a 32-bit value is unlikely to be sufficient to measure the generation of high-speed links beyond OC192c. For example, an OC768c POS link will wrap a 32-bit byte counter in less than one second.

3.3.5 Obtain Clock Synchronisation State

As discussed in section 3.2.9, in order to measure one-way delay between two points in the network, the clocks must either be synchronised, or there must be a method to determine the absolute offset of one clock from the other.

3.3.6 Denial of Service Resistant

A protocol for per-hop measurement of Internet packet dynamics must not add any DoS vector. There are two classes of DoS attacks: logic attacks, in which known protocol implementation flaws are exploited; and flooding attacks, in which the target is overwhelmed by the volume of incoming requests [65]. The scope for a logic attack can be reduced by ensuring the protocol is well defined and simple to implement. The scope for a flooding attack can be reduced by ensuring the protocol does not provide the ability to amplify an attack, and ensuring the protocol does not require state to be kept and is able to be efficiently processed so that a victim's CPU cannot be overwhelmed.

3.3.7 Measure Different Traffic Types

Many tools probe the network with UDP or ICMP probes and solicit ICMP responses. There are a number of reasons for doing so. The functionality required by the technique often motivates their use. For example, an ICMP timestamp request is a convenient method of obtaining a timestamp from a router. Similarly, in order for a technique to infer packet dynamics using the IP-ID field, the sender must cause the destination to generate a new packet; sending a UDP probe to an unused port is a fairly unintrusive method of doing this. In addition, UDP and ICMP probes are more simple to construct, to send, and to receive compared to, for example, probes encapsulated in TCP.

However, these probes may not reflect the actual performance seen with other traffic types due to different queueing priorities intermediate systems may enforce. For example, some networks may prioritise the forwarding of TCP packets; others may rate limit the forwarding of UDP and ICMP packets. A protocol for per-hop measurement of Internet packet dynamics should be able to signal to intermediate routers how the packet should be queued if the router forwards packets on a basis other than First-in First-out (FIFO).

3.3.8 Resolve Router Aliases

As discussed in section 3.2.10, existing measurement techniques to resolve router aliases are not reliable because they rely on implementation quirks. A protocol for per-hop measurement of Internet packet dynamics is likely to encounter router aliases. Therefore, such a protocol should provide a mechanism to determine which addresses are aliases for the same router.

3.3.9 Partial Deployment Utility

A protocol for per-hop measurement of Internet packet dynamics will require support to be built into end-hosts and routers. As support for such a protocol will be deployed incrementally, a protocol must be useful even without complete deployment.

3.4 Summary

The thesis of this work is that a protocol for per-hop measurement of Internet packet dynamics is feasible and useful. In this chapter, the motivation of why a measurement protocol designed for per-hop measurement of Internet packet dynamics is *useful* was presented. The limitations of some techniques are shared with other techniques that measure different properties. In particular, tools that manipulate the TTL of outgoing probes share similar limitations; they are forward path bound, and they cannot definitively separate the behaviour of prior hops. With the distilled set of limitations enumerated in section 3.2, a series of desirable features were defined that a protocol designed for per-hop measurement of Internet packet dynamics might have. In particular, the ability to combine path and dynamics measurement into a single packet exchange is identified as an important protocol feature. In the next chapter, a protocol designed with measurement in mind which addresses the limitations identified in this chapter is described.

Chapter 4

IP Measurement Protocol

4.1 Introduction

The Internet relies on cooperation and interoperability. To date, Internet measurement and diagnosis tools have mostly relied on features found in existing general-purpose Internet protocols, despite the many limitations this reliance has. In the previous chapters, the motivation for a measurement protocol was established. Chapter 2 discussed various measurement techniques currently available, what they rely on, and their relative advantages and limitations, for measuring delay, loss, reordering, and capacity;

Chapter 3 looked at overcoming the common limitations which are found in many measurement techniques, and discussed a series of desirable features that a protocol designed for per-hop measurement of Internet packet dynamics might have.

This chapter presents the IP Measurement Protocol (IPMP). IPMP is an Internet protocol designed explicitly for the purpose of per-hop measurement of Internet packet dynamics. IPMP is designed to solicit the cooperation of routers. A router that supports IPMP inserts information into a specially formed packet as it is processed. Although only simple information is added, this enables per-hop direct measurement of packet dynamics such as propagation delay, delay jitter, packet reordering, and loss. A concise specification of IPMP in Internet-Draft format is included in appendix A. IPMP was first proposed by McGregor in 1998 [8].

The rest of this chapter discusses the design of IPMP. Section 4.2 begins the chapter at a relatively high-level, with a general overview of IPMP. Section 4.3 dis-

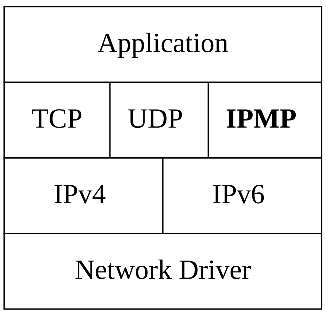


Figure 4.1: A model of an IP stack with IPMP

cusses the feasibility of the protocol in general terms to show that such a protocol is possible. Sections 4.4 and 4.5 then take a detailed look at the two protocol components of IPMP: the echo protocol, and the information protocol. As the protocol seeks the cooperation of intermediate routers, these sections detail why IPMP takes the form it does, and how the form relates to efficient processing in routers and destination hosts. Finally, in section 4.6, IPMP is compared to the desirable features discussed in chapter 3.

4.2 Protocol Overview

IPMP is encapsulated directly inside an IP packet, as shown in figure 4.1, placing it at the same level as TCP, UDP, and ICMP. IPMP has two distinct protocol components: the echo protocol, which is used to probe a network path, and the information protocol, which is used to retrieve supplementary information from each router. The echo protocol is used to directly measure per-hop path characteristics such as delay, jitter, loss, and reordering. The echo protocol is designed to be processed in the forwarding path of a router. The ability to process an IPMP echo packet in the forwarding path is an important attribute, as an IPMP echo packet will provide the most useful measurement of instantaneous network behaviour if it follows the same processing path as any other packet. The information protocol is not used to measure network behaviour, and may be processed at low priority and out of the forwarding path if necessary.

The architecture of the IPMP echo protocol is shown in figure 4.2. A source host constructs an IPMP echo request packet, addresses it to a destination host, and then

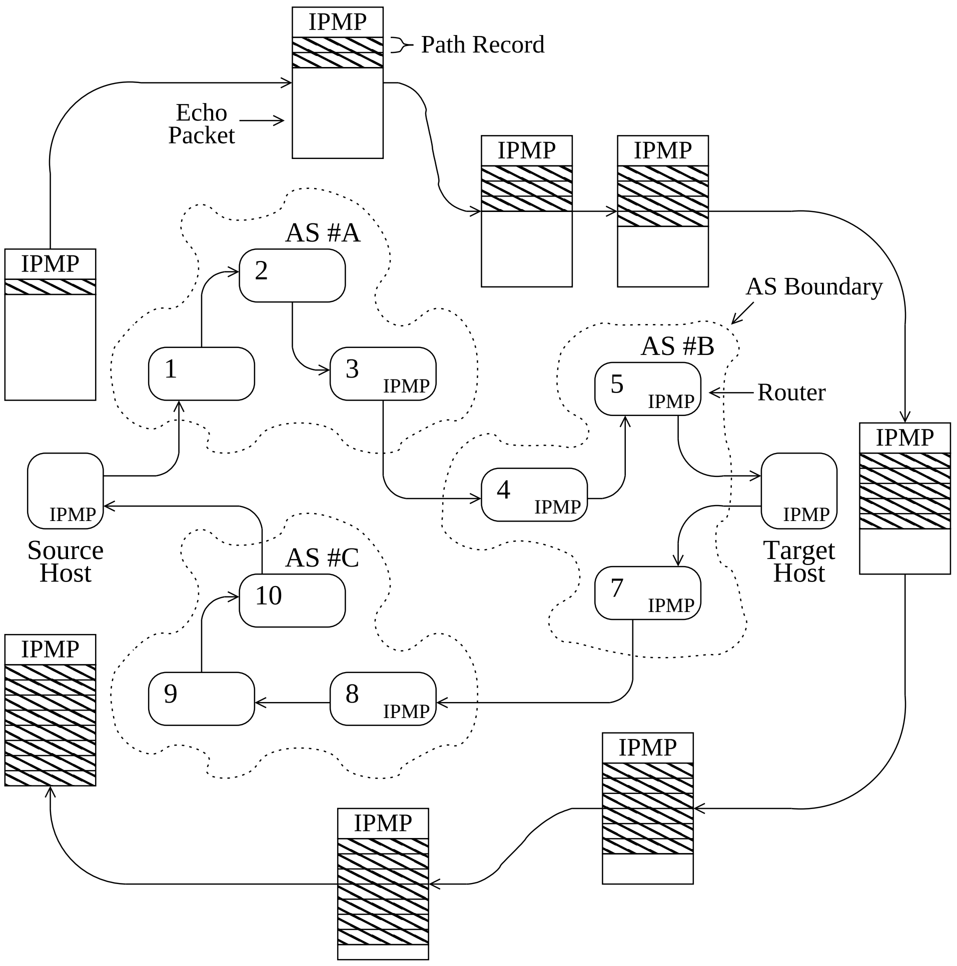


Figure 4.2: The IPMP measurement architecture

transmits the packet. In this diagram, the flow of the echo packet is shown on the outside of the network. Each router on the forward path that implements IPMP may insert information into the echo probe as it is forwarded towards the destination. In this diagram, the information that is embedded is represented as a small shaded box in the packet. The routers that implement IPMP in figure 4.2 are at hops 3, 4, 5, 7, and 8. When an echo request packet arrives at the destination, it is modified and then returned to the source as an echo reply packet. Each router on the reverse path that implements IPMP may also insert information into the echo probe as it is forwarded back to the source.

The information that is inserted into an echo probe is in the form of an IPMP *path record*. A path record may contain, among other things, an IP address of the interface that received the packet and the local time at the router when the packet was received. An IPMP echo reply packet has a (partial) record of which routers forwarded the packet, and when it was received by each of them. As a path record includes an IP address, the precise format of a path record and the method used to update an IPMP echo probe differs between IPv4 and IPv6.

If all nodes on the forward and reverse paths support IPMP, then a single IPMP echo exchange provides a forward and reverse path `traceroute` with one probe packet. A tool that sends a series of echo probes can infer per-hop dynamics such as jitter, loss, IP path changes, and load balancing. IPMP is also useful in scenarios where it is not available on all nodes in a path. Figure 4.2 shows one possible deployment scenario, where AS #A and AS #C choose to enable IPMP on border routers at the edges of their respective networks. In this case the delay, jitter, and loss characteristics of the path through a particular AS can be measured. The ability to do this would be useful in verifying that the service specified in a Service Level Agreement (SLA) is being provided, or in determining if a particular AS is contributing to a fault.

Should there be more IPMP-capable nodes in the path than an echo packet provides path record space for, a source host can restrict which nodes may insert a path record. This feature is particularly useful should the source wish to measure a path by sending small probes which reflect the packet size used by a particular application. For example, some voice and video applications send small packets regularly rather than wait for enough data to fill a path MTU-sized packet. This aspect of IPMP is discussed in further detail in section 4.4.5.

The IPMP echo protocol also provides the ability for an intermediate system to queue measurement probes as if they were a different type of packet, if that system queues packets based on their type. Such a system often queues packets based on a 5-tuple of values which consist of the source and destination IP addresses, IP protocol type, and source and destination port numbers where applicable. IPMP does this by providing three faux fields in the echo header which correspond to the IP protocol type, and source and destination port numbers. This feature is discussed in further detail in section 4.4.

As with any protocol that measures delay between two different points, there are challenges in establishing the synchronisation state of the clocks at those points. In order to measure absolute one-way or per-hop delay between two different points in a network path, each point is required to have a synchronised clock. However, not all measurements require synchronised clocks. For example, jitter can be inferred between two points simply by determining the difference between a series of packet arrival times.

IPMP's approach to timekeeping is quite different to current approaches, and consists of two distinct parts. The first part is concerned with the process of efficiently generating and storing a timestamp in a path record. This occurs in the forwarding path. As the interface which receives and timestamps an echo packet may not have a reliable or useful real-time clock, IPMP does not define the precise format of a timestamp beyond the size and location of the timestamp field in the path record. Rather, it allows the interface to generate a timestamp using the most suitable clock available on the interface. For example, a network interface may choose to generate a timestamp with a clock derived from the clock that is used to serialise incoming packets.

The second part of IPMP's approach to timekeeping is concerned with the process of translating a timestamp to real time, and determining the usefulness of the timestamp for measurement. The translation is done with information obtained in a separate packet exchange using the IPMP information protocol. The information protocol enables a source host to obtain the information necessary to be able to translate timestamps to real time, to determine the error limit for the clock, the resolution of the clock, to resolve router aliases, and obtain other optional but useful data items that the system administrator believes is of use to measurement.

4.3 Feasibility

Before IPMP is presented in detail, the feasibility of a protocol such as IPMP is examined. One important goal of IPMP is for a measurement packet to follow the same processing path as any other normal IP packet. Router vendors are concerned with forwarding path performance and are reluctant to add new features, because

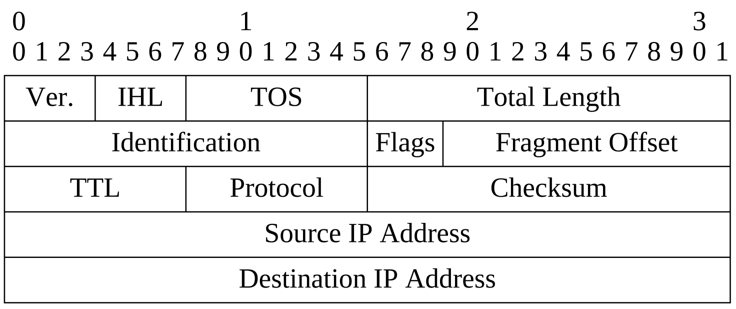


Figure 4.3: The IPv4 header without options

new features may introduce complexity and reduce forwarding path performance. The purpose of this section is to show that a protocol like IPMP is (1) technically feasible with currently available router architectures and implementations, (2) capable of having echo packets processed in the forwarding path, (3) within the bounds of what might be implementable and that the cost of doing so is not unreasonable. For software-based routers where each packet is processed in a FIFO basis by a CPU, these requirements are not difficult to meet, so long as the required operations can be done efficiently. In chapter 5, implementation experience with software-based routers is presented.

For routers with hardware optimised forwarding paths, an argument is needed to show that it is possible to process IPMP echo packets in the so-called *fast path*. The fast path is the optimised forwarding path in a router that a packet follows when the router does not have to queue the packet for special processing; that is, the packet follows the fastest possible forwarding path through the router. A packet that requires special processing is taken out of the fast path and placed in an appropriate queue for further processing, which may be done in other specialised hardware or by a general purpose CPU. The details of the fast path vary from router to router. A common implementation of a fast path is one optimised to handle the common packet forwarding case, where (1) a packet does not have any IP options that require processing, (2) the router has a route to the destination cached, and (3) the only required modification to the packet is the IP-TTL decrement operation and associated IP checksum update.

The format of the IPv4 header is shown in figure 4.3. Every IP packet that is forwarded through a router has a minimum of two operations. First, the TTL field is checked; if the TTL field is one, then the packet has expired and an ICMP time exceeded message may be sent back to the source. Otherwise, the router decrements the TTL and updates the IP checksum accordingly. Second, if the packet has not reached its destination, a route to the destination is chosen. Any IPMP-related modification to the packet must be made before the routing decision is completed if the packet is not to be delayed.

Recall from the overview in section 4.2 that a path record includes an IP address of the interface where it was received, and a timestamp that represents when the packet was received. Generating and recording a timestamp in a packet is a much easier operation than determining the next IP hop of the packet, because the timestamp in IPMP can be as simple as a counter. Similarly, most interfaces have only one IP address, so there is no decision to be made as to which IP address to use. Interfaces with more than one IP address may choose to use a single globally routable IP address for all path records it inserts which belongs to the interface and uniquely identifies it; this is discussed further in section 4.4.6.

4.4 The Echo Protocol

This section discusses the design of the IPMP echo protocol in detail. The IPMP echo protocol is the part of IPMP designed to directly measure per-hop packet dynamics on an Internet path. Section 4.4.1 discusses the overall design considerations and format of the IPMP echo protocol. Section 4.4.2 presents the format of the IPMP path record. Sections 4.4.3 and 4.4.4 discuss the details of forming a path record structure by examining the IP and timestamp fields in detail. Finally, echo processing at the source host, intermediate nodes, and the target is discussed in sections 4.4.5, 4.4.6, and 4.4.7 respectively.

4.4.1 Echo Packet Design

As discussed in section 4.2, an echo packet must follow the same processing path as any other packet for it to be a representative measurement of packet dynamics. In addition, it must not present a denial of service vector. For these reasons, processing

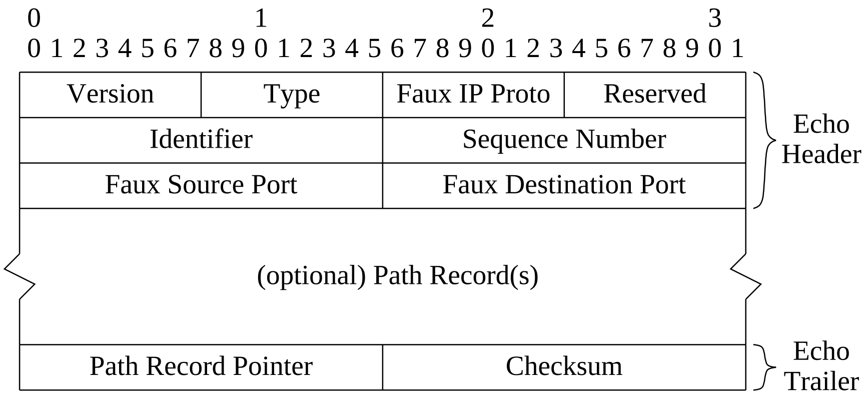


Figure 4.4: Format of an IPMP echo packet

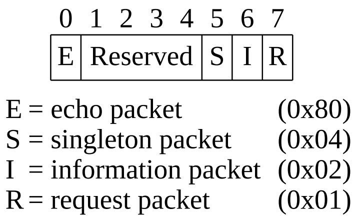

0 1 2 3 4 5 6 7
E
Reserved
S
I
R
E = echo packet (0x80)
S = singleton packet (0x04)
I = information packet (0x02)
R = request packet (0x01)

Figure 4.5: Format of the IPMP type field

efficiency is a primary design criterion for the IPMP echo protocol. The design of the IPMP echo packet has a number of purposes. First, it provides an efficient mechanism for an IPMP-capable destination host or router to determine if it should insert a path record in an echo packet. Second, it provides an efficient mechanism for an IPMP-capable host or router to modify the echo packet. Third, it provides an efficient mechanism for an intermediate system to determine how the packet should be queued, should the system queue packets other than on a FIFO basis. Finally, it provides an efficient mechanism for a measurement process executing on a source host to identify individual echo responses as they arrive and match them to echo request probes it sent.

The format of the IPMP echo packet is shown in figure 4.4. An IPMP echo packet has three components; an echo header, pre-allocated space for path records, and an echo trailer. The components are arranged in the order they are required for decision making when an echo packet is processed in the forwarding path of a router. For example, the trailer contains fields which are modified last, after processing decisions have been made. The fields in each component are arranged so that an IPMP echo packet is as simple to process as possible.

The echo header signals to IPMP capable nodes how to process the packet. The version field identifies the version of the IPMP protocol being used, to allow for non-backwards compatible extensions to be added in the future if required. The type field, shown in figure 4.5, identifies if the IPMP packet is an echo probe, if the packet is a request or response, and if the echo probe is a one-way singleton probe.

The identifier and sequence number fields allow a source host to match responses with a corresponding request record. If an IPMP packet causes an ICMP message to be sent, the source host can use these fields to match the message with a corresponding request record. The identifier and sequence number fields are placed in the first 8 bytes of the echo packet so that if an ICMP stack is not configured to send more of an original packet than the historical recommendation of the IP header plus the first eight bytes of payload, it will still be able to identify the packet that caused the ICMP message to be returned. The identifier is also used by intermediate systems – such as firewalls and routers – to determine which flow the packet belongs to.

The faux protocol type, faux source port, and faux destination port fields allow an IPMP echo probe to be queued or filtered as another packet type would if a router or intermediate system queues or rate limits packets based on their type. For example, if measurement of TCP traffic is of interest, then the faux protocol field would be set to 6, which is the protocol type of TCP. Similarly, if Hypertext Transfer Protocol (HTTP) traffic is of specific interest, then the sender would set the faux destination port field to 80, which is the most common HTTP port used.

The echo trailer is found in the last four bytes of an echo packet. An echo trailer contains a path record pointer and an IPMP checksum, which are updated when the packet is modified to include an additional path record, or when the echo header is modified at the destination host. These fields are located at the end of the packet so that nodes which process IPMP packets as a bit-stream do not have to buffer the whole packet when inserting a path record; this is discussed in further detail in section 4.4.6.2. The value of the path record pointer field identifies the offset relative to the first byte of the echo header (the version field) at which the next IPMP path record should be inserted, and is used by nodes which process IPMP

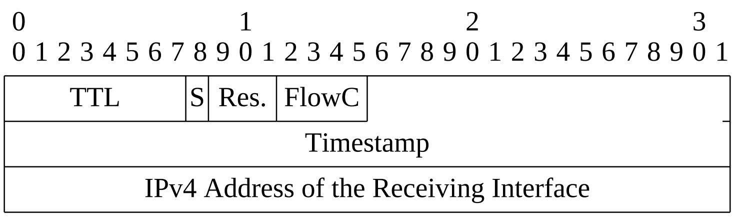


Figure 4.6: Format of the IPMP path record when encapsulated in IPv4

packets in a store-and-forward manner. The checksum is a standard RFC 1009 [66] Internet checksum, and is computed over the echo packet from the IPMP version field to the IPMP checksum field.

Between the echo header and echo trailer is pre-allocated space reserved by the source host for path records. The motivation for using pre-allocated path record space instead of allowing an IPMP echo packet to grow dynamically is two-fold. First, extending the packet requires extra processing; the IP header requires modification to the length and checksum fields, and the packet buffer which holds the probe requires extra memory resources to be dynamically assigned. Second, extending the packet dynamically limits the ability to measure the behaviour of an Internet path with a specific packet size. The space reserved for path records is also initialised in such a way as to reduce the processing burden of intermediate nodes which process IPMP in the forwarding path, and to allow a source to restrict which routers on a path may insert a path record. These details are discussed further in the next section.

4.4.2 Path Record Design

The path record is the structure in the IPMP echo protocol that enables per-hop measurement of Internet packet dynamics to take place. An IPMP path record contains information useful for directly measuring packet dynamics. Like the echo protocol, the path record is designed to be compact and efficient to process. The format of the IPv4 IPMP path record is shown in figure 4.6.

The reason that the path record structure is designed to be compact is to allow as many path records to be included in an echo packet as possible; an IPv4 path record requires 12 bytes, while an IPv6 path record requires 24 bytes. In order to keep the space requirements of a path record to a minimum, a path record does not include information about the path that does not change between probes. A mechanism to retrieve static information is provided with the IPMP information protocol.

The space between the echo packet's header and trailer is reserved for path records. As with the IPMP echo packet design, the path record structure is designed with processing efficiency as a primary design criterion. Most of the space reserved for path records is initialised to zero so that incrementally updating the IPMP checksum [67] is as simple as possible. The exception to this is the first byte of each path record – the TTL field – which is used by the source to restrict which hops may insert a path record. An intermediate node determines if it should insert a path record by comparing the TTL field in the IP header – after the TTL has been decremented as part of the forwarding process – with the TTL field of the next unused path record. If the TTL field in the next unused path record is greater than or equal to the TTL field in the IP header, then the intermediate node may use that path record space, provided the 'S' bit, discussed next, is not set.

When a path record is inserted, the IPMP TTL field in the path record inserted is a copy of the TTL field from the IP header, after the IP TTL has been decremented as part of the forwarding process. Including a TTL field in a path record allows a host to determine the distance, in hops, into an Internet path where the path record was inserted. The set (S) field is a bit which is set when space reserved for a path record is used. This field is useful for systems which process an echo packet as a bit-stream because it allows an intermediate system to determine the next suitable location for a path record to be inserted without waiting for the path record pointer field which comes at the end of the packet. Processing an echo packet as a bit-stream is discussed further in section 4.4.6.2.

The flow counter (FlowC) field records the arrival order of a series of IPMP echo packets which belong to the same flow. An IPMP flow is defined as a series of echo packets with the same source and destination IP addresses, IP protocol type, and IPMP echo ID. Including a flow counter in an IPMP path record was first proposed by Mahajan, Spring, Wetherall, and Anderson in "User-Level Internet Path Diagnosis" [1]. A flow counter provides the ability to determine the arrival order of

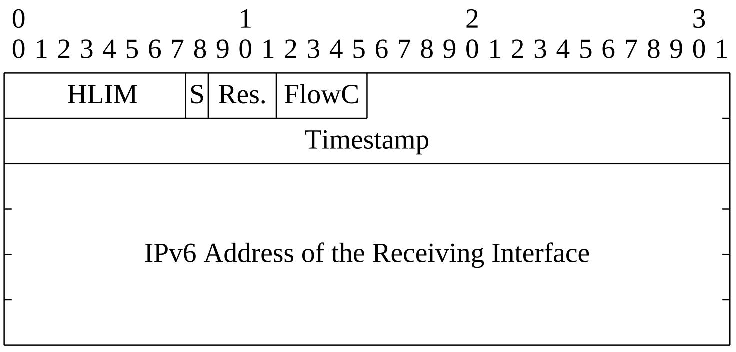


Figure 4.7: Format of the IPMP path record when encapsulated in IPv6

a series of packets and to infer where a previous packet was lost in appropriate conditions. The algorithms for doing so are discussed in section 6.4. The flow counter field is 4 bits wide, which enables 16 sequence numbers to be recorded before the counter wraps. A flow record relates an IPMP flow to a flow counter. Flow records are interface scoped; that is, each flow record records the arrival order of an echo flow at a particular interface. IPMP echo probes that belong to the same flow but arrive at different interfaces therefore have different flow records. Flow counters are an optional part of IPMP, as generating and recording a flow counter requires significant processing and state to be kept compared with other parts of the echo protocol.

The three bits between the S field and the FlowC field are currently reserved; they are set to zero by the sender. A possible use for one of these bits is to include an Explicit Congestion Notification (ECN) Congestion Experienced (CE) bit in the path record, which could be used for determining the congestion status of routers in an end-to-end path.

The timestamp field is a 48 bit field that represents the time the packet was received at the interface. In order to ensure that IPMP is as simple as possible to process in the forwarding path, the timestamp field has no defined format, other than the size and location. Translating the timestamp to real time is handled in a separate packet exchange with the information protocol, discussed in section 4.5. Finally, the IP address field is an IP address of the interface which received the echo probe; the method to choose an appropriate address if the interface has more than one address to choose from is discussed in section 4.4.3.

	IPv4	IPv6
IP header size	20 bytes	40 bytes
IPMP echo header	12 bytes	12 bytes
IPMP echo trailer	4 bytes	4 bytes
Path record size	12 bytes	24 bytes
Minimum IP MTU	576 bytes	1280 bytes
Path record count	45	51
Ethernet IP MTU	1500 bytes	1500 bytes
Path record count	122	60

Table 4.1: Number of path records able to be included for various IP packet sizes

The format of the IPv6 IPMP path record is shown in figure 4.7. The format of the IPv6 path record is very similar to the IPv4 path record. An IPv6 path record varies from the IPv4 path record in two ways. First, it has a 128-bit IPv6 address instead of a 32-bit IPv4 address. Second, it has a Hop Limit (HLIM) field instead of a TTL field, as the IPv6 header has a HLIM field instead of a TTL field. Table 4.1 shows how many path records may be included in an echo probe, depending on the size of the echo probe and whether IPv4 or IPv6 is used.

4.4.3 Path Record IP Address

Path records may be inserted by a source host, a destination host, and intermediate routers. A destination host that inserts a path record uses the destination address of the packet itself so that the source can determine which path record, if any, corresponds to the destination. This is useful for separating the path records which were inserted on the forward path from those which were inserted on the reverse path. A source host that inserts a path record when sending an echo packet chooses an IP address from the outgoing interface; this address will typically be the same address as the source uses for the source IP address in the IP header. An intermediate router that inserts a path record uses an IP address from the interface that receives the probe.

An IPMP-capable node may have to perform an address selection algorithm if the interface has multiple candidate addresses for the path record. The address selection algorithm for path records is similar to the address selection algorithm for IPv6 described in RFC 3484 – Default Address Selection for IPv6 [68]. RFC

3484 details how to determine a suitable source address based on a number of rules where multiple candidate addresses exist. As with RFC 3484, the address selection decision is based on the destination address in the IP header. This is to prevent an destination host from spoofing its source address to learn private addresses along the path. The major difference between the IPMP address selection mechanism and RFC 3484 is that the IP address must be selected from the incoming interface and not the outgoing interface.

There are two sets of candidate path record addresses; one set consists of candidate IPv4 addresses, while the other set consists of candidate IPv6 addresses. Neither of these sets may contain anycast, link-local, or multicast addresses. Anycast addresses are excluded from the candidate sets because an anycast address does not uniquely identify an interface in any scope, and so cannot be used to determine the path an IPMP echo packet took. Link-local addresses are excluded from the candidate sets unless the path record is being inserted into an echo response packet at the destination host; note that this will only occur if an echo probe is sent to another link-local address. A link-local address will never be a candidate address in the IPMP forwarding path because link-local addresses are by definition not routable. A multicast address will never be a candidate address in the IPMP forwarding path, because a multicast address does not uniquely identify a target. If there are multiple candidate addresses remaining after anycast, link-local, and multicast addresses have been removed, then the node should select one of the remaining addresses and use that address consistently.

4.4.4 Path Record Timestamp

There are three approaches to including a timestamp in an IPMP path record. First, if the node does not have a useful or accessible clock, it may choose to not include a timestamp in any path record it inserts, leaving the timestamp field zero. It is recommended that if the cost of reading from a clock is significant, IPMP nodes should not include a timestamp in their path record. Second, if the node has a real-time clock that is easily accessible, then the node could include a timestamp from it in the path record. Depending on the capabilities of the system, a real-time clock

```
struct timeval {
    long tv_sec,      /* seconds */
    long tv_usec      /* and microseconds */
};

struct timespec {
    time_t tv_sec,    /* seconds */
    long tv_nsec      /* and nanoseconds */
};
```

Figure 4.8: The POSIX timeval and timespec structures

should at least present a timeval structure, which has a maximum resolution of one microsecond; more recent systems have a timespec structure with a resolution of one nanosecond. The definitions of the timeval and timespec structures are presented in figure 4.8. Recall that the space reserved for a timestamp in an IPMP path record shown in figure 4.6 is 48 bits. The simplest method of generating an IPMP timestamp in this situation is to use the low-order 16 bits of the tv_sec field and the low-order 32 bits of the tv_usec or tv_nsec field. Third, the node may choose to use some other clock, such as that used to serialise incoming packets. One example of using such a clock is to divide the serialisation clock by 8, in order for the clock's accuracy to be able to measure delay per-byte, as discussed in section 3.3.4.

4.4.5 Creating an Echo Request

An application running on a source host is responsible for creating and initialising an echo request packet and sending it to a destination host. The nature of the measurement is set by the source host, and is reflected in how the echo request is constructed. The characteristics that can be controlled are:

- the size of the echo probe;
- which hops on the path may insert path records;
- how routers and middle-boxes should queue the probe;
- and if the packet is a one-way probe or an echo exchange.

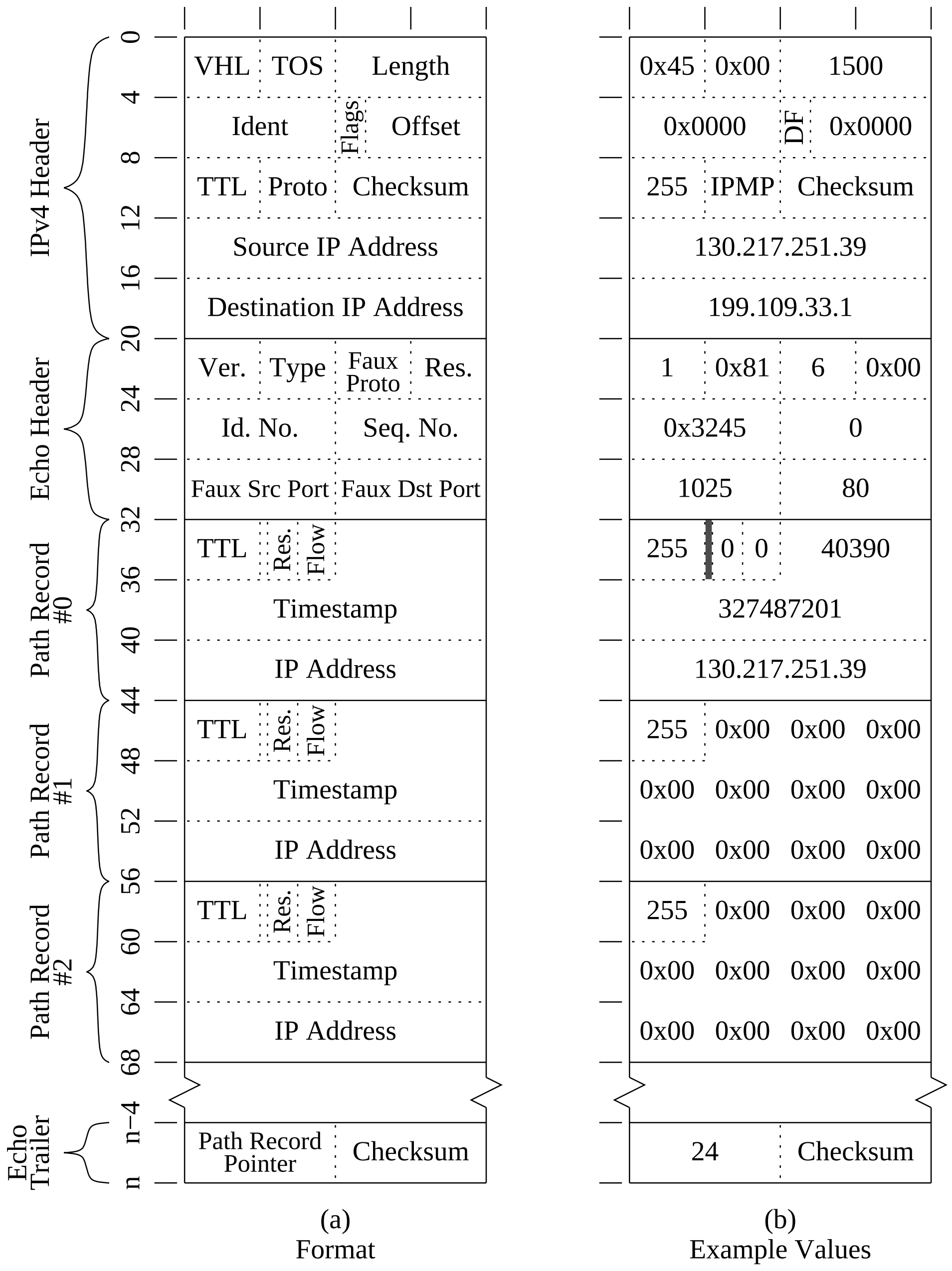


Figure 4.9: Creating an IPMP echo request packet

The source host is required to initialise the space reserved for path records so that intermediate nodes can determine if they should insert a path record. The source host does this by initialising all bytes reserved for path records to zero, except for those that correspond to the TTL field in each path record. In the simplest case where the source host would like to collect path records from all intermediate nodes on the path that support IPMP, it pre-initialises every path record's TTL field to be the same TTL it sets in the IP header, so that the TTL test will succeed on all IPMP capable intermediate nodes and they will each insert a path record.

Figure 4.9 shows the layout of a complete IPMP echo request probe encapsulated in IPv4, alongside an initialised echo request probe. As indicated by the IP length field, this particular probe is 1500 bytes in size; the first 68 bytes and last 4 bytes of this packet are shown. The source host allocates a packet buffer and initialises the source and destination IP addresses, sets the IP TTL field to 255, and sets the IP protocol type to the value assigned to IPMP. The source host also sets the Don't Fragment (DF) bit in the IP header so that the packet is prevented from being fragmented at any point in the path, as the entire echo probe is required when inserting a path record. So that processing an IPMP echo probe is as simple as possible, particularly on routers which may process IP options out of the fast path, it is recommended that an echo probe does not include any IP options, unless the effect of including IP options is of interest. The IPMP echo header begins immediately after the IP header. The version of IPMP described in this thesis is version one. The IPMP type field is set to 0x81, which corresponds to an echo request, according to figure 4.5. The faux IP protocol, faux source port, and faux destination port correspond to an HTTP packet. The IPMP-ID number is 0x3245, which corresponds to the low-order 16 bits of the process ID of the application which created the probe. The sequence number is set to zero, as this is the first echo request sent to the particular destination with this IPMP-ID.

In this example, the source host inserts the first path record so it has a convenient record of when the probe was sent. The 'S' bit is set, indicating that the path record space has been used. The flow counter field is set to zero as it is the first packet sent from the host. In this example, the host forms an IPMP timestamp from a clock with nanosecond resolution; the first 16 bits represent the number of seconds since midnight, while the second 32 bits hold the fractional portion of the timestamp. The IP address in the path record is the same as the source address in the IP header.

The rest of the pre-initialised space for path records is initialised to zero, except for the TTL fields, which are set to 255 so that all intermediate nodes will insert a path record if they can. Finally, the path record pointer is set to 24, which corresponds to the offset of the first byte of the second path record from the first byte of the IPMP header.

4.4.6 Echo Processing at an Intermediate Node

An intermediate node is responsible for determining if it should insert a path record, and then inserting a path record if appropriate. One of the design goals of the IPMP echo probe format is to make the path record insertion process as efficient as possible so that per-hop measurement with IPMP accurately reflects normal packet behaviour, is implementable in the forwarding path, and does not introduce a vector for denial of service. This section discusses how an intermediate node should process an IPMP echo packet.

There are two fundamentally different ways to receive and process an IPMP echo packet, as there are two fundamentally different architectures for forwarding packets. The first is store-and-forward, where a packet is received in its entirety before the forwarding decision is made. The packet undergoes any updates before being forwarded. The second is the faster notion of bit-stream forwarding, where the packet is buffered only long enough for the forwarding decision to be made. Any packet updates subsequent to the routing decision must be applied to the packet in-line. The path record decision processes for both store-and-forward and bit-stream processing are presented in the next two subsections. The most significant architectural difference is in determining if a path record should be inserted.

4.4.6.1 Store-and-Forward Decision Process

The decision process to insert a path record when the packet is stored and then forwarded is shown in figure 4.10. In this diagram, the packet is laid-out right-to-

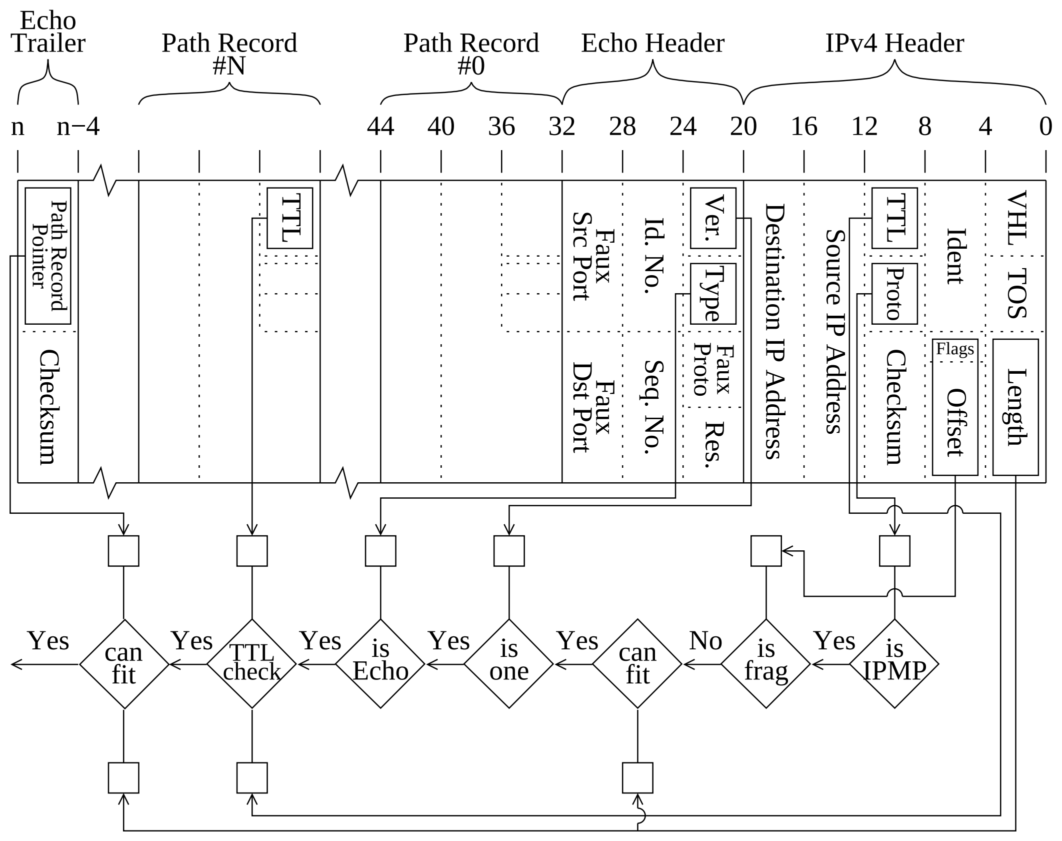


Figure 4.10: The decision process taken to insert an IPMP path record when the packet is stored and then forwarded

left, as if the packet was moving across the page left to right. Each decision is shown under the packet; each small square represents a register used to temporarily hold information used in making the decision. The IPv4 header, the IPMP echo header, and the first path record are shown in the first 44 bytes; this is followed by the N^{th} path record, located somewhere between the echo header and echo trailer; the echo trailer is shown in the last 4 bytes of the packet.

First, the node determines if the packet is an IPMP packet by checking the IP protocol field. If it is an IPMP packet, then the node ensures that a complete IPMP packet is available by ensuring the More Fragments (MF) flag is not set, and the offset field is zero. If the IPMP packet is complete, then the node then calculates the length of the IP header in bytes, so that it can determine where the IPMP header begins; this step is not shown in figure 4.10. The node then loads the IP header's length field and ensures the packet is large enough to contain a complete IPMP echo header and trailer. These first steps may be done in parallel in a digital design.

Next, the node determines if the IPMP packet is an echo probe. It does this by first checking the version field, ensuring that the value of the version field is one. Then, it checks the IPMP type field to see if the ‘E’ bit is set; if set, it signals that the packet is an echo probe. Then, the node loads the path record pointer field, located in the echo trailer, to determine where the next path record should be inserted. These three steps may also be done in parallel in a digital design.

Finally, the node determines if it should insert a path record by loading the TTL field corresponding to the path record – which was pre-initialised by the source host and pointed to by the path record pointer – in order to determine if it should insert a path record. If the pre-initialised TTL field is greater than or equal to the TTL in the IP header after it has been decremented, then the node determines that it should insert a path record. Then, the node does a sanity check to ensure that there is enough space left in the packet to insert the path record. These two steps may be done in parallel in a digital design.

4.4.6.2 Bit-stream Decision Process

The decision process to insert a path record when the packet is presented as a bit-stream is shown in figure 4.11. First, the node keeps a record of the length field in the IP header. When the IP flags and offset fields arrive, the node ensures that a complete IPMP packet is available by ensuring the MF flag is not set, and the offset field is zero. When the IP TTL field arrives, it keeps a record of it for use in forming a path record at a later time, if required. The node then checks the IP protocol field to see if the packet is an IPMP packet. If it is IPMP, then the node waits for the first byte of the IPMP header to arrive.

When the IPMP version field arrives, the node checks that the version is one, to ensure that it can understand the format of the rest of the probe. If it is, it then checks the IPMP type field to ensure that the packet is an echo probe. If it is an echo probe, the node then begins to search the incoming bit-stream for an appropriate place to insert a path record. It does this by checking the pre-initialised path record space, beginning at the first byte after the echo header. The node may choose to insert a path record in the first pre-initialised path record space it finds with the ‘S’ bit not

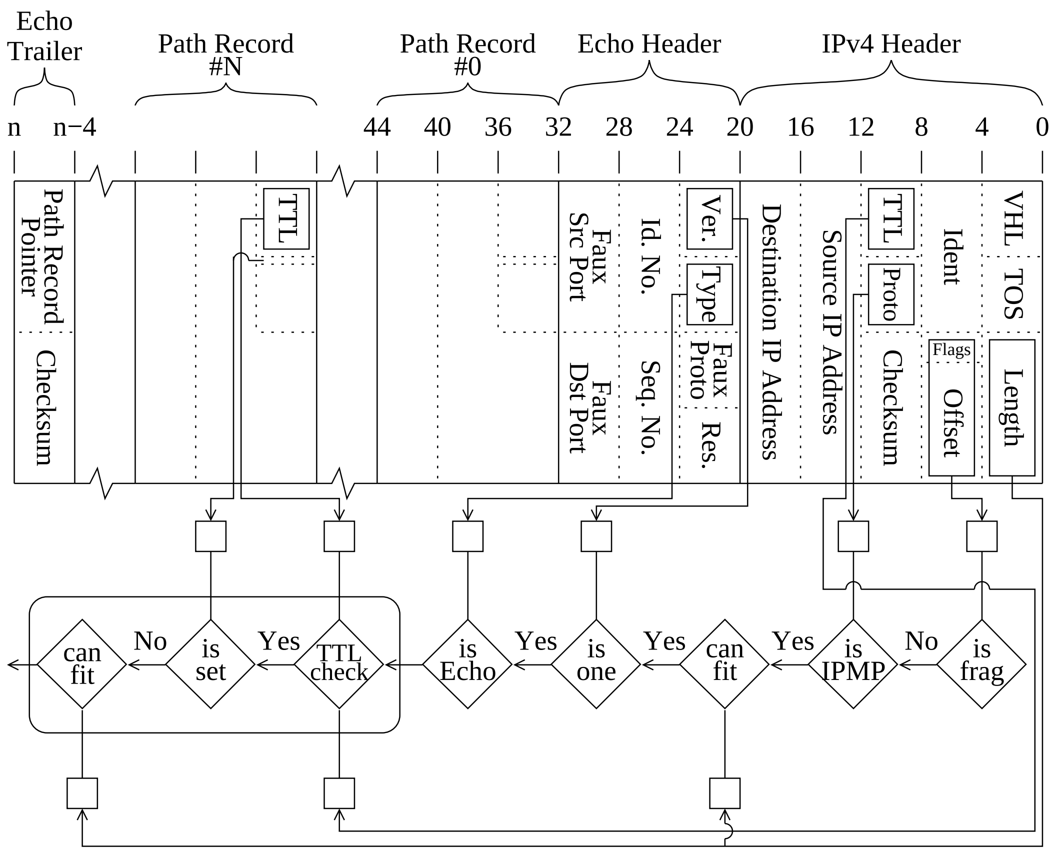


Figure 4.11: The decision process taken to insert an IPMP path record when the packet is presented as a bit-stream

set, so long as the pre-initialised TTL field for that path record is greater than or equal to the IP-TTL stored earlier.

4.4.6.3 Inserting a Path Record

Figure 4.12 shows the high-level process of inserting a path record and then incrementally updating the IPMP checksum. If the node implements IPMP flow counters, then it looks up the flow counter record to include in the path record which matches the particular source and destination IP addresses and the IPMP ID. The path record generator takes four items as input. They are: a timestamp, an IP address, the IP TTL of the packet after it has been decremented, and a flow counter. The path record generator uses this input to form the path record, and then inserts it into the echo packet. As output, it produces a sum of the 16 bit words which make up the path record, which is then passed to the checksum update process.

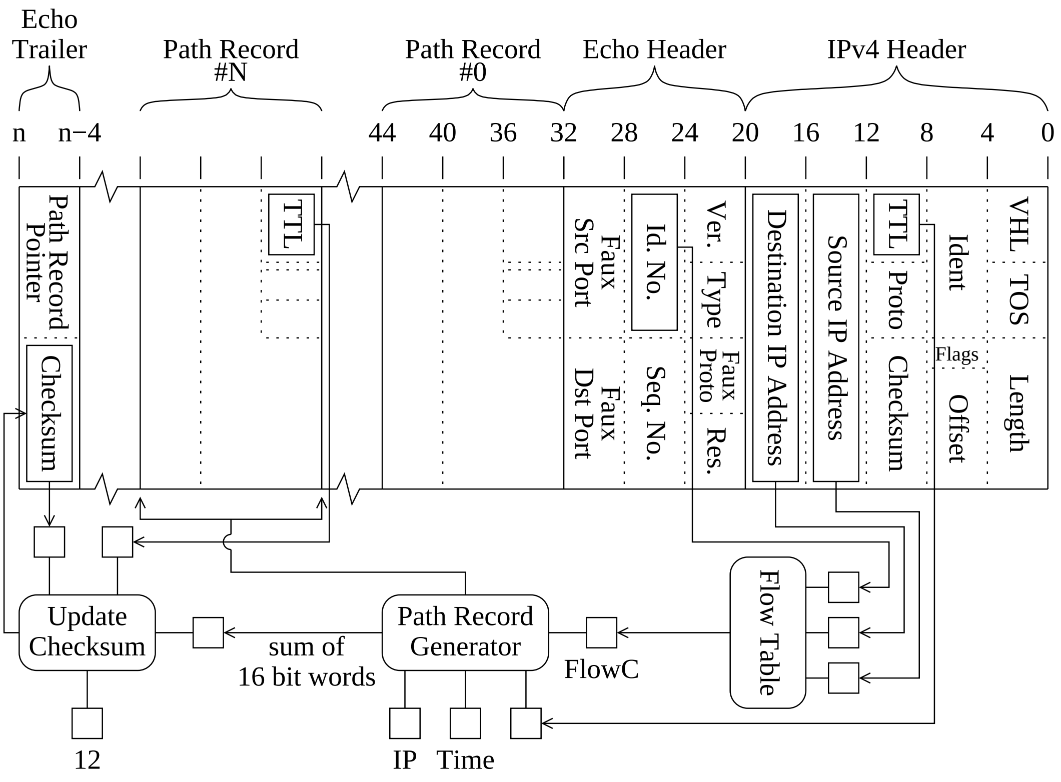


Figure 4.12: The packet update process when inserting an IPMP path record

The checksum update process takes four inputs; the original IPMP checksum, the sum of 16 bit words from the path record, the TTL field from the pre-initialised path record, and the constant of 12 which corresponds to the adjustment of the path record pointer field when an IPv4 path record is inserted. As output, it produces a replacement checksum value for the IPMP echo trailer.

Figure 4.12 does not show the update applied to the IPMP path record pointer field. The value in this field is incremented by 12, which corresponds to the size of an IPv4 path record.

4.4.7 Echo Processing at a Destination

Figure 4.13 shows the processing of an IPMP echo probe at a destination host. This diagram does not include the decision process that determines the node has received a packet addressed to it, as this is the same for all protocol types and is done before a particular protocol handler is invoked. First, the node determines if the packet is an IPMP echo probe by checking the IP protocol field, as well as the IPMP version

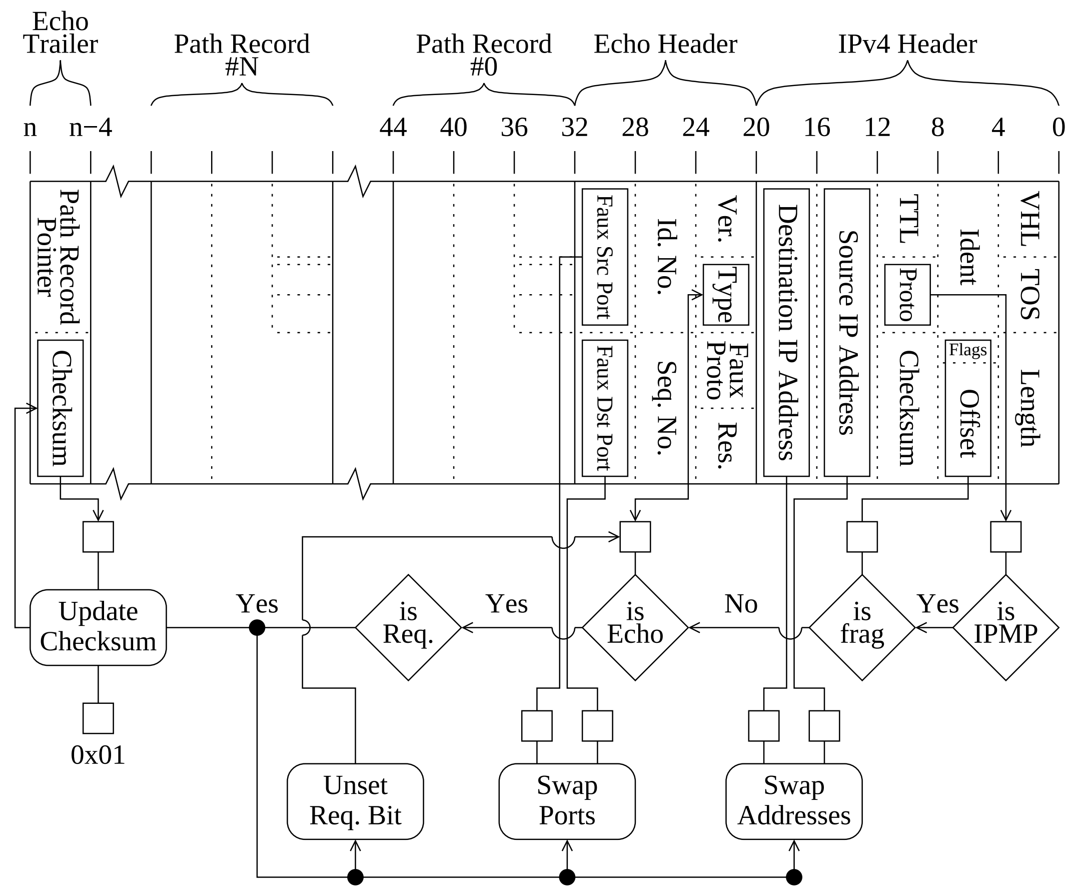


Figure 4.13: Creating an IPMP echo reply packet from an echo request

and type fields. Then, the node checks the IPMP type field to see if the request bit is set; if the request bit is not set, then the packet is placed in a socket queue for processing by a user-space application.

If the request bit is set, the node swaps the IP addresses in the IP header, swaps the the IPMP faux port fields, unsets the request bit in the IPMP type field, adjusts the IPMP checksum, and then sends the IPMP packet back to the source. The node does not reset the TTL field in the IP header. Instead, it decrements it by one. The IPMP echo checksum update is simple, as swapping the locations of 16 bit quantities does not change the sum of the 16 bit words, and the adjustment made to the IPMP type field is constant. The checksum update consists of adding 0x01 to the checksum field, to correct for clearing the request bit. After determining that the packet is an echo request, all of these operations may be done in parallel in a digital design.

Figure 4.13 does not show the target inserting a path record, although it may do so if the echo probe permits. This process is the same as that described in section 4.4.3.

4.5 The Information Protocol

The IPMP information protocol is the part of IPMP designed to query an IPMP-capable node for information useful to measurement. As with the IPMP echo protocol, the information protocol is a simple request/response exchange. However, the information protocol is not used to measure packet dynamics. Instead, it is used to obtain information that is useful to measurement. This information includes the properties of each clock used to generate timestamps as part of a path record, which IP addresses belong to the same router, and other information that the system's administrator believes is useful for measurement.

In order to measure absolute one-way delay, timestamps from different clocks must be calibrated. Similarly, in order to measure variation in packet delay between two points, clock drift and adjustments must be accounted for. The IPMP echo protocol does not define the resolution, the frequency, or the epoch of a path record timestamp. The information protocol provides this information by returning an information response which includes a series of real-time reference points. A real-time reference point consists of a local clock timestamp, a real-time timestamp which matches this local clock timestamp, and a limit of the maximum error of the real-time timestamp. A source host can then convert a timestamp reported in a path record to real time by using linear interpolation between two adjacent real-time reference points.

4.5.1 Information Packet Design

The format of an IPMP information request packet is shown in figure 4.14. An information request packet consists of a simple header which identifies the IPMP packet as being an information request, and an optional body which includes specific reported times of interest, should the sender be concerned with determining clock properties for a specific time period. As with the echo protocol, the identi-

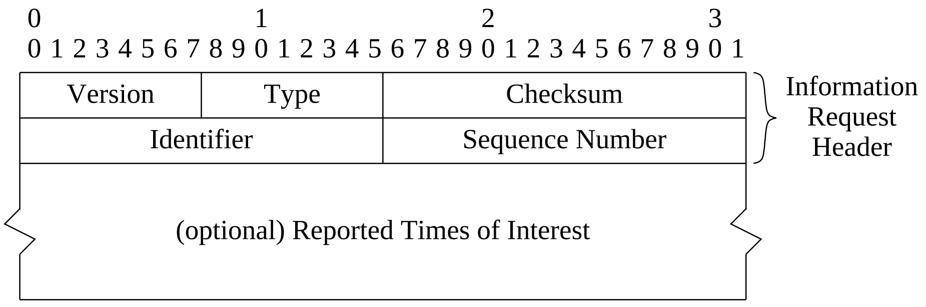


Figure 4.14: Format of an IPMP information request packet

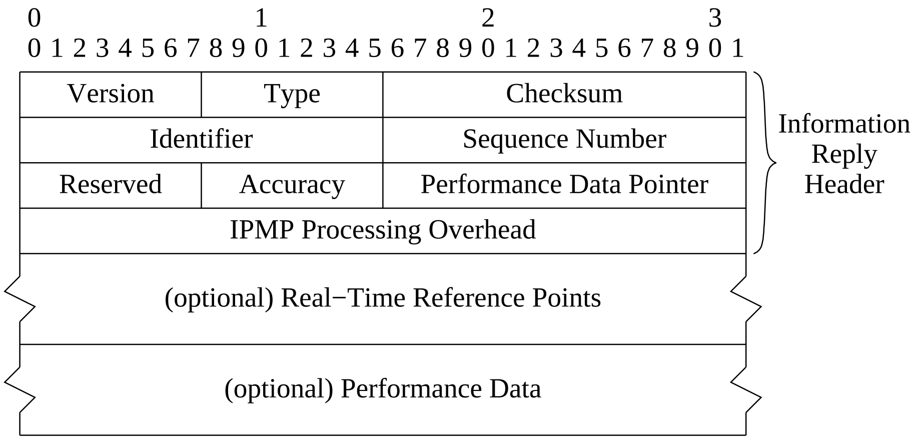


Figure 4.15: Format of an IPMP information reply packet

fier and sequence number fields in an information request packet are used by an application to determine which information replies are of interest.

The format of an IPMP information reply packet is shown in figure 4.15. An information reply packet contains three extra compulsory fields in addition to those found in an information request. They are the accuracy field, the performance data pointer field, and the IPMP processing overhead field. The accuracy field specifies the number of valid bits in the real-time timestamp of each real-time reference point; this is discussed further in section 4.5.2, but for now it is sufficient to note that the field is used to determine the accuracy of the node's real-time clock. The performance data pointer field specifies the offset of the performance data from the first byte of the IPMP header. It also implicitly specifies the length of the real-time reference points embedded in the packet. Finally, the IPMP processing overhead field specifies the extra delay, if any, of forwarding an IPMP echo packet over any other packet. If the overhead is not known, then all bits in the field are set to one.

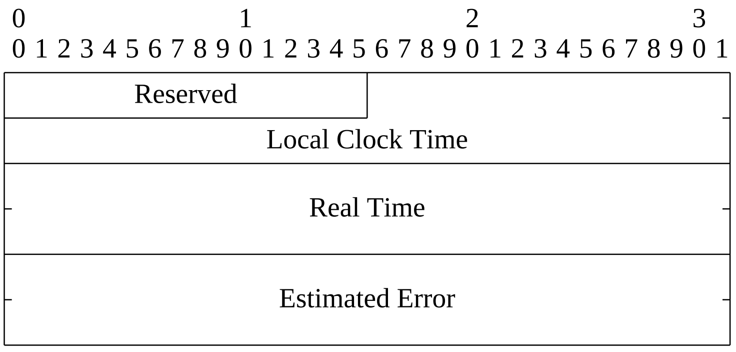


Figure 4.16: Format of an IPMP real-time reference point

4.5.2 Real-Time Reference Point Design

After the information reply header is space for real-time reference points and performance data. The format of an IPMP Real-Time Reference Point (RTRP) is shown in figure 4.16. Each RTRP contains a 48-bit timestamp generated by the local clock, a 64-bit real-time timestamp, and a 64-bit estimate of any error component. A RTRP associates a locally-generated timestamp – such as that used in a path record – with a real-time timestamp, with the error bounds of the real-time timestamp. The real-time timestamp and error bounds are NTP [31] formatted timestamps. The first 32 bits count the number of seconds since midnight January 1st 1900 Universal Time Coordinated (UTC). The second 32 bits are a fractional timestamp, which allows for a maximum resolution of 200 picoseconds. The left-most bit is the most significant bit; any unused bits are not set. A source host determines the number of significant bits in the fractional portion of the timestamp by examining the accuracy field in the information reply.

4.5.3 Resolving Router Aliases

When an IPMP-capable node replies to an information request, it must do so using a common source IP address. The purpose of this is to allow router aliases to be resolved. Therefore, a router should select a single globally-routable address, where possible, and use that address to reply to all information requests.

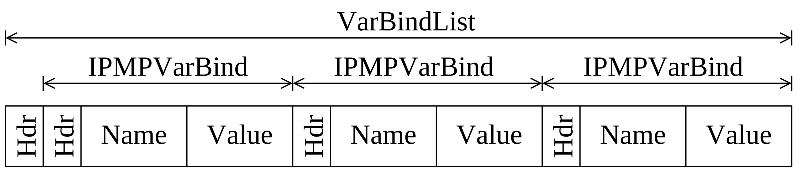


Figure 4.17: Format of an IPMP VarBindList

4.5.4 Performance Data

The performance data field is an optional part of the IPMP information protocol that an operator can use to return information which may be of interest to the source. For example, an operator may wish to report information about the interface that was queried, or report information about the system, or report other information useful for identifying who is responsible for the system. The performance data field contains data formatted as Abstract Syntax Notation One (ASN.1), and encoded with Basic Encoding Rules (BER) which Simple Network Management Protocol (SNMP) uses. The rest of this section gives a general overview of how some example data is formatted in the performance data field.

Performance data is formatted as a VarBindList from the SNMPv2 Protocol Data Unit (PDU) defined in RFC 3416 - Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMPv2) [69]. A VarBindList is a simple arrangement of variable names and their values, as shown in figure 4.17. The variable names and the range of types and values which are valid are defined in various SNMP Management Information Base (MIB) documents. For example, the MIB that defines variable names for network interfaces (IF-MIB) is found in RFC 2863 – The Interfaces Group MIB [70], and the MIB that defines variable names for BGP (BGP4-MIB) is found in RFC 1657 [71]. The variable name is declared as an object identifier, which is a hierarchical arrangement of integers that are assigned by an authority such as IANA. The headers are simple fields that specify the size, in bytes, of the variable and its value; the first header specifies the size of the entire VarBindList.

Table 4.2 shows sample MIB entries from the BGP4-MIB and IF-MIB. These sample MIB entries are selected as they are items of information that are useful for Internet measurement. The data type column specifies the type of the variable;

MIB	Variable	Object identifier	Data type	Example value
BGP4	bgpLocalAs	1.3.6.1.2.1.15.2.	Integer32 (2)	64512
IF	ifType	1.3.6.1.2.1.2.2.1.3.	Integer32 (2)	Ethernet (6)
IF	ifMtu	1.3.6.1.2.1.2.2.1.4.	Integer32 (2)	1500
IF	ifSpeed	1.3.6.1.2.1.2.2.1.5.	Gauge32 (42)	100,000,000

Table 4.2: Sample MIB entries

the number in parentheses corresponds to the identifier used to denote the data type when encoded in ASN.1. A Gauge32 type is an integer that records the maximum value which the variable has held since it was reset. The bgpLocalAs variable allows a router to report the AS to which it belongs; this is a much simpler method of determining who is responsible for operating the system, and potentially more accurate than using a routing table view. The ifType, ifMtu, and ifSpeed variables allow a system to report the type, MTU, and the speed of an interface in bits per second. As a router has multiple interfaces, the details of the interface returned are for the interface which was queried with the information request. As an IP hop may have an intermediate layer 2 device which has a lower serialisation rate or lower MTU than the interface whose details are reported in an information response, the values reported should be considered the upper limit of each particular value that the hop can process.

Figure 4.18 provides an example byte encoding of the variables in table 4.2. For illustrative purposes, the encoding of the VarBindList header and the bgpLocalAs variable is now described; an exhaustive review of BER encoding and ASN.1 representation is outside the scope of this chapter. The first two bytes correspond to the VarBindList header and signify that a sequence of variables follows; the first byte (0x30) signifies that a sequence follows; the next byte (0x3F) specifies that 71 bytes of data follows. The next 16 bytes encode the bgpLocalAs variable, and its value of 64512. The first two bytes specify that a sequence follows, of 14 bytes. The type of the variable is declared as an ObjectID (0x06), whose length is 8 bytes. The encoding of the ObjectID in this PDU is 2B.06.01.02.01.0F.02.00; this corresponds to the object identifier of 1.3.6.1.2.1.15.2. in table 4.2, except that the first two integers are combined to save space, by multiplying the first integer by 40 and adding the second integer to the result; $(40 * 1) + 3 = 43 =$ 0x2B. The last byte

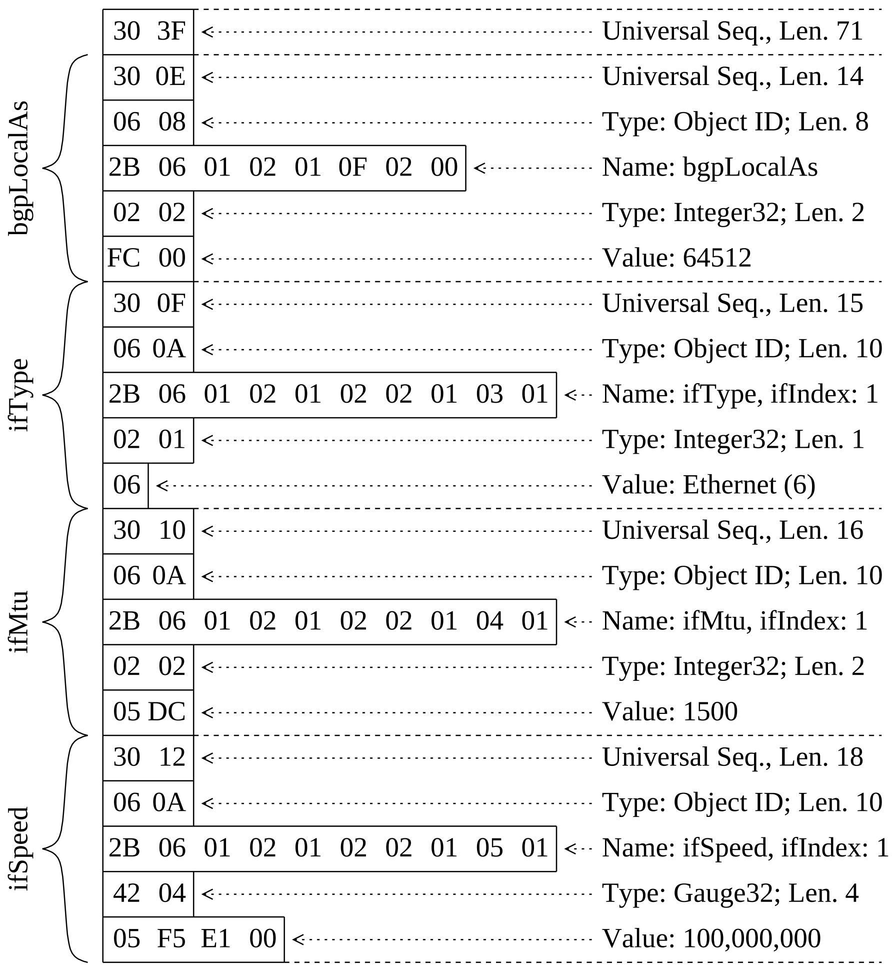


Figure 4.18: Example IPMP performance data

of the ObjectID signifies an index in the case where there are multiple instances of the particular object; because there is only one AS value for a particular router, the index in this case is 0x00. The next two bytes specify the type of the variable, and the number of bytes used in encoding the variable; in this case, the type of the value is an Integer32, and two bytes are used to encode it. Those two bytes follow; 0xFC00 corresponds to the Autonomous System Number (ASN) 64512.

4.6 Relationship to Desirable Features

Chapter 3 looked at overcoming a series of common limitations which arise mainly through the absence of explicit protocol support for per-hop measurement of Internet packet dynamics. A series of desirable features for a measurement protocol for this task were identified. This section examines how the design of IPMP allows for the desirable features to be met.

4.6.1 Integrated Path and Packet Dynamics Measurement

Section 3.3.1 argued that integrated path and packet dynamics measurement is desirable, because the TTL method of progressively probing a path has many limitations. IPMP addresses these limitations by allowing intermediate routers to insert a path record which provides a record of when the router received the packet. With explicit cooperation, integrated path and packet dynamics measurement is possible, as an echo packet is otherwise routed normally to the destination.

4.6.2 Able to Measure the Reverse Path

Section 3.3.2 argued that a measurement protocol for per-hop measurement of Internet packet dynamics should be able to measure the reverse path. It is simple to measure the reverse path with IPMP, as the echo protocol is a request/response-based protocol, and routers on the reverse path are able to insert path records as routers on the forward path do.

4.6.3 Minimal Overhead in Routers

Section 3.3.3 argued that such a protocol should introduce minimal overhead in routers, because these systems are optimised to forward packets as quickly and efficiently as possible. Section 4.4.6 shows that it is possible to process an echo packet as a bit-stream. If an implementation processes an IPMP packet serially amongst other packet operations, then IPMP is tightly constrained and designed to be processed efficiently so that an echo packet will incur negligible additional delay compared to any other IP packet. The performance of software and hardware implementations are measured in the next chapter.

4.6.4 Precise Timekeeping

Section 3.3.4 argued that such a protocol should allow for adequately precise timekeeping suitable for measuring the particular link, and Section 3.3.5 argued that such a protocol should allow a measurement system to determine the clock state of any system which reports a timestamp. IPMP allows the resolution and precision of the clock used to measure packet dynamics to be decided by each system. Conversion of a timestamp to real-time is accomplished in a separate packet exchange, using the information protocol. This packet exchange, described in section 4.5, also allows a source host to determine what the node believes the clock's accuracy is and its offset from real time.

4.6.5 Denial of Service Resistant

Section 3.3.6 argued that such a protocol should be DoS resistant. IPMP is designed with processing efficiency as a primary design criterion. A flood of IPMP echo request packets does not provide any additional vector for overwhelming the CPU of the destination host or intermediate systems compared with processing any other IP packet, if well implemented. In addition, IPMP does not provide any additional vector for amplifying a denial of service attack, because an echo packet does not result in any additional traffic being created.

4.6.6 Measure Different Traffic Types

Section 3.3.7 argued that such a protocol should be able to measure different traffic types, because some systems in a path may prioritise the forwarding of some packets over others. IPMP allows intermediate systems that queue and process packets based on their traffic type to be measured by using the faux IP protocol, faux source port, and faux destination port fields to determine the underlying traffic type, in addition to the fields in the IP header that might also be used.

4.6.7 Partial Deployment Utility

Section 3.3.9 argued that such a protocol should not require full deployment in order to be useful. The minimum support required in order for IPMP to work is protocol support at the source and destination hosts. Even in this circumstance, IPMP is useful, because it allows for simple one-way measurements of delay, jitter, loss, and reordering to take place. As described in section 4.2, the IPMP architecture works well if each AS deploys IPMP at the borders of its network, because it allows a source to determine which network is responsible for adding significant delay, jitter, loss, or reordering to overall packet dynamics.

4.6.8 Resolve Router Aliases

Section 3.3.8 argued that a protocol for per-hop measurement of Internet packet dynamics should be able to resolve router aliases. The IPMP information protocol specifies that all information replies generated by a router should be sent with a single globally routable address where possible. This allows router aliases to be resolved.

4.7 Summary

This chapter presented IPMP, a protocol designed explicitly for per-hop measurement of Internet packet dynamics. IPMP differs from existing measurement protocols by seeking explicit cooperation from intermediate routers. It does so by introducing the concept of a path record. Path records are embedded by intermediate systems into the body of the echo packet as it is forwarded.

In the next chapter, implementation experience with IPMP is presented. Looking further ahead, chapter 6 presents techniques using IPMP for measuring topology, delay, jitter, loss, reordering, and capacity, and chapter 7 discusses application experience in using IPMP in a rural wireless network where it is deployed on all routers.

Chapter 5
Implementation

5.1 Introduction

In the previous chapter, an Internet protocol designed for per-hop measurement of Internet packet dynamics was presented. The protocol, IPMP, seeks the cooperation of intermediate routers. Intermediate routers insert raw data useful for measurement in the form of a path record into an IPMP echo packet as it is forwarded.

This chapter demonstrates that it is feasible to implement IPMP in the forwarding path of a router by presenting and characterising three implementations. Two of the implementations modify the source code of the kernel for two different operating system families. The third modifies the forwarding path of a layer 2 Ethernet switch built for research purposes. Section 5.2 discusses the software implementation of IPMP, first by looking at a high-level model of how IPMP fits into the IP stack in an operating system, and then by discussing the experience of implementing IPMP in 4.4BSD-derived and Linux kernels. Section 5.3 discusses measurements of the forwarding performance of the FreeBSD implementation, and examines the additional processing required to implement optional portions of IPMP, such as flow-counters. Section 5.4 presents the details of a hardware implementation of the IPMP path record insertion process in the forwarding path, and section 5.5 discusses measurements of the forwarding performance of this implementation. Section 5.6 summarises this chapter.

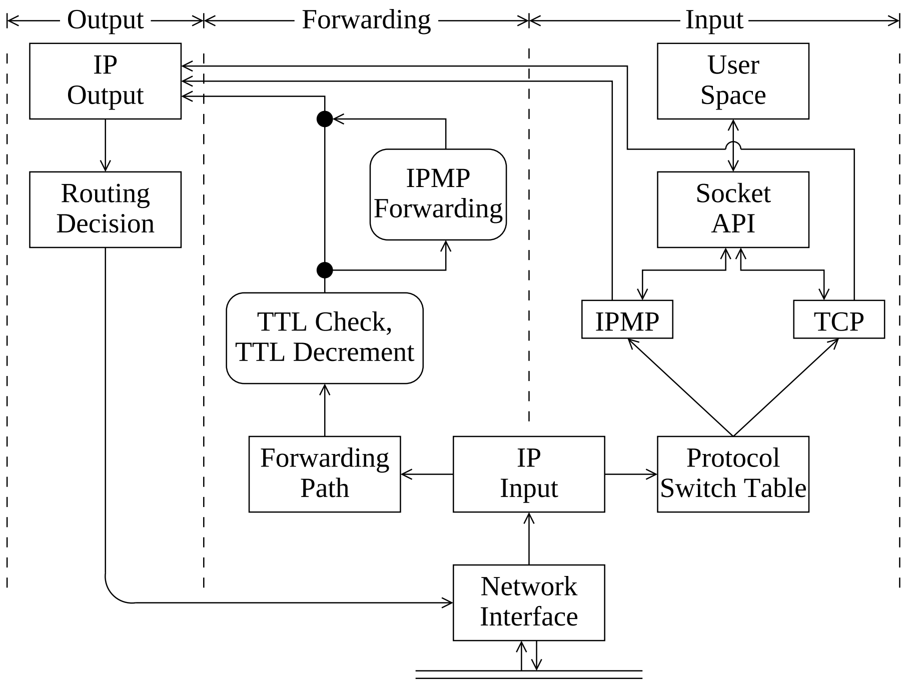


Figure 5.1: A model of interactions between IPMP and a kernel

5.2 Software Implementation

This section discusses the software implementation of IPMP in the kernel of two different operating system families. The first family is the BSD family, which consists of a number of operating systems derived from 4.4BSD, such as FreeBSD, NetBSD, and OpenBSD. The second family is the Linux family. The network stacks of these operating system families are substantially different in their design and implementation, and so the interactions that IPMP has with these systems are quite different at the source-code level.

5.2.1 Model

While most of IPMP could feasibly be implemented entirely in user-space or entirely in kernel-space, some of the protocol is most suited towards a kernel-space implementation. For example, time critical operations such as modifying an echo packet in the forwarding path or replying to an echo request are suited to a kernel-space implementation, while processing an information request may be done at low-priority in user-space. This section focuses on the kernel-space implementation of the echo protocol because the performance and efficiency of IPMP is critically important in the context of a kernel.

Figure 5.1 presents a model of how a software implementation of IPMP might interact with a kernel. The figure is divided into three sections, each separated by dotted lines. These sections are the output path, the forwarding path, and the input path. The output path is conceptually simple. The output path takes a packet to be sent, determines the route to the destination, and passes the packet to the appropriate network interface for transmission. The implementations of IPMP discussed in this section do not modify the output path.

When an IP packet arrives in the IP input module from a network interface, the host has to determine if the packet is addressed to it, or if it should forward the packet on. If the packet is for the forwarding path, the IP TTL will be checked, and if greater than one, it will be decremented. If the packet is an IPMP packet, the packet is passed to the IPMP forwarding routine for further processing. After any IPMP-specific processing has taken place, the packet is then passed to the IP output path for transmission. This process was discussed in detail in section 4.4.6.

If the packet is addressed to this host, the packet is passed to the input path for further processing. Software-based IP stacks maintain a table of IP protocol modules to further process IP packets of a particular type; in this case, IPMP is simply another protocol module in this table. The IPMP protocol module is responsible for determining if the IPMP packet is an information or echo probe. Information requests, information replies, and echo replies are passed to the socket queue for reading and further processing by user-space applications. Echo requests are modified and then passed to the IP output path for transmission. The process of modifying an echo request was discussed in detail in section 4.4.7.

The rest of this section details the kernel implementations for the BSD and Linux kernels, and how they relate to the model shown in figure 5.1. For each implementation, the effect of the system's packet storage facilities, timestamp facilities, and kernel facilities for sending an echo request are examined, because these have the most impact on the design of an implementation.

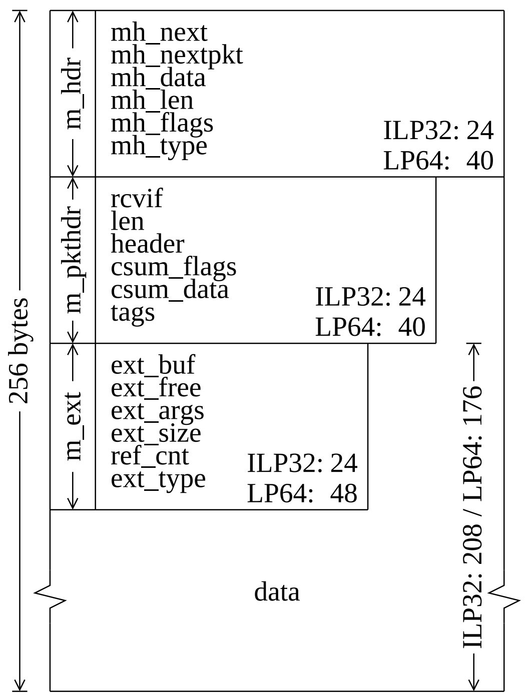


Figure 5.2: The BSD mbuf structure

5.2.2 BSD Implementation

The BSD implementation of IPMP works on FreeBSD versions 3.0 through to 6.0, and NetBSD 1.6. This section discusses how the design of the BSD kernel affects the implementation of IPMP.

5.2.2.1 Packet Storage

Internally, BSD kernels store packets in mbuf structures. An mbuf is a small buffer of fixed size designed to allow packet headers and trailers to be efficiently added to and removed from the packet without creating a copy of the packet. Figure 5.2 shows the layout of the mbuf structure found in FreeBSD 6.0. All mbuf structures begin with a header, m_hdr which is used by the kernel to manage mbuf structures, and to link to any other mbuf structures that combine to form the packet. The first mbuf of each packet also has a second header, m_pkthdr, which records the length of the packet and the interface the packet was received on where applicable, among other items. The interface field is used by the BSD implementation to determine an appropriate IP address for an IPMP path record.

Most BSD kernels use 256-byte mbuf structures. On systems with 32-bit integers, longs, and pointers (ILP32) the first mbuf has 208 bytes for the actual packet, and on systems with 64-bit longs and pointers (LP64) the first mbuf has 176 bytes for packet data. Packets that require more storage than provided in the first mbuf link extra storage to the mbuf, either by linking additional mbuf structures to the first mbuf, or by attaching external buffer space to the mbuf with an m_ext structure.

While most network drivers will ensure that a received packet is available in a single buffer, this is not guaranteed. Therefore, the BSD implementation of IPMP is required to be careful when accessing or modifying the echo trailer or a path record, because these structures could be split across two mbuf structures. The BSD implementation makes working copies of the echo trailer and any relevant path record using m_copydata, and then copies these structures back into the packet with m_copyback when finished. These necessary copy operations reduce the ability to efficiently process an IPMP packet in the forwarding path.

IPMP is designed to allow an echo request packet to be efficiently transformed into an echo reply and returned. When replying to an echo request packet, the system is not required to allocate a new mbuf structure and then copy the IPMP packet in, because the input routine has complete control over the mbuf. This is desirable from an implementation perspective.

5.2.2.2 Timestamps

As described in section 4.3, the precise format of the timestamp field in the IPMP path record is not defined. The BSD implementation of IPMP provides the ability for the system administrator to choose between a real-time timestamp and a timestamp based on the system's CPU cycle counter where available.

FreeBSD and NetBSD provide similar real-time timestamp facilities in the kernel. Both systems have a microtime function that provides a timestamp with microsecond precision representing real-time as accurately as the system can with the hardware available. FreeBSD provides a number of additional time routines, including the nanotime function, found in FreeBSD releases from 3.0 onwards. The FreeBSD implementation of IPMP uses nanotime where available, because this function provides a higher-resolution timestamp than the microtime function.

Modern CPUs include a CPU cycle counter that can be useful for timing purposes, as described by Pásztor and Veitch [64]. On Pentium-class CPUs, this register is known as the TSC register. Modern Pentium CPUs operate at speeds in excess of 1Ghz; therefore, the TSC register provides the necessary primitives for a clock capable of measuring time differences with at least a nanosecond precision [64]. However, using the TSC to measure time intervals presents two significant challenges. First, if the rate of the CPU changes in order to save power, then so does the rate of the clock. Second, the TSC values from two different CPUs in an Symmetric MultiProcessor (SMP) system cannot be directly compared without some effort to first calibrate the TSC registers from the various CPUs, because the TSC is local to the CPU it is read from. As calibrating TSC registers across CPUs is outside the scope of this protocol, the real-time clock is used by default. However, a system can be configured to use the TSC if the system is a single CPU machine whose clock rate does not abruptly change.

5.2.2.3 Sending an Echo Request

The BSD implementation also has a loadable kernel module that may be used by user-space applications to create and send an IPMP echo request packet in kernel-space. Sending an echo request in kernel-space has a number of advantages over sending an echo request in user-space if the source host inserts a path record, because it is possible to generate the path record's timestamp closer to when the packet is transmitted; that is, after the packet has been assembled in kernel-space and the appropriate route has been determined. A user-space application passes a structure to an IPMP system call that contains parameters used to form an echo request packet, such as the destination address and the size of the packet to send.

The kernel module is accessed by using a system call number which is dynamically assigned by the kernel when the kernel module is loaded. The ability to dynamically load a system call is convenient compared to compiling a kernel with the functionality built in, because that would require a system call number to be permanently assigned for this purpose.

5.2.3 Linux Implementation

The Linux implementation of IPMP runs on various releases from the 2.4 branch of the Linux kernel. The implementation of IPMP on Linux is logically similar to the implementation of IPMP for the BSD kernel. The main differences are in manipulating a packet in the packet buffer, recording a timestamp in a path record, and sending an echo request probe from kernel-space.

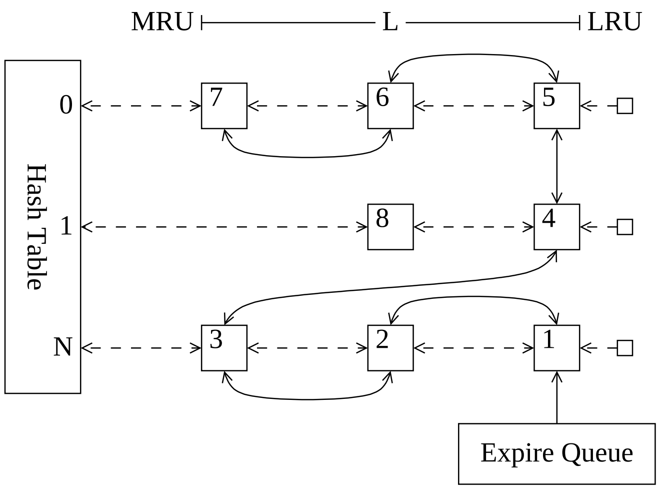
5.2.3.1 Packet Storage

The Linux kernel stores packets in skbuff structures, which are designed to hold an entire packet in a single buffer complete with any appropriate layer 2 headers. In order to allocate an appropriately sized skbuff, the system has to determine the outgoing route in order to determine the size of any layer 2 headers to reserve space for. This differs from the logical approach in figure 5.1, where the routing decision is made after the IP packet has been constructed.

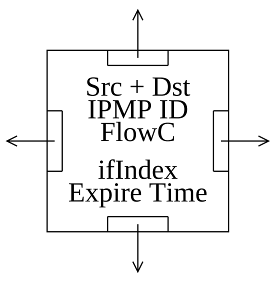
The skbuff structure is designed to be shared throughout the network stack to reduce unnecessary copying of the contents of the packet. Care must be taken to avoid creating a race condition with other consumers of the packet when modifying an echo packet. One example of such a race condition is in modifying a packet in the receive path in parallel to a user-space application using a packet filter to capture incoming packets. If a packet is modified before the packet filter has captured the packet, then the packet filter will not have an accurate record of the contents of the packet when it was received. Consequently, when replying to an echo request modifications must be made to a copy of the original packet.

5.2.3.2 Timestamps

As with the BSD implementation, the Linux implementation of IPMP is able to either use a real-time clock or a CPU cycle counter for timestamp purposes. The Linux 2.4 kernel provides a real-time clock with a microsecond resolution, via do_gettimeofday. In order to obtain a real-time clock with higher resolution, the Pulse Per Second Kit (PPSKit) patch [72] to the Linux kernel has to be used. The



(a) Data structure



(b) Node structure

Figure 5.3: The structure of the IPMP flow counter implementation

PPSKit patch provides a clock with nanosecond resolution using the do_clock_get-time function. The PPSKit is a patch maintained externally to the Linux kernel.

Therefore, the Linux implementation of IPMP uses do_clock_gettime on systems where the PPSKit patch is used, and do_gettimeofday on systems where it is not. The Linux implementation also has the ability to use the TSC register as a clock source, though it is susceptible to the same SMP and clock-rate limitations that the BSD implementation is.

5.2.3.3 Sending an Echo Request

The Linux implementation also contains code to construct and send IPMP echo request packets in kernel-space. The implementation, however, is compiled into the kernel and accessed by using an Input/Output (I/O) control function (ioctl) on an IPMP socket. This is because Linux does not provide the ability to dynamically load system call modules into the kernel. This is less convenient than using a loadable system call as the BSD implementation does, because it requires a pre-defined magic value to be assigned, and modification of the Linux socket code so that the IPMP-specific ioctl is then handled by the IPMP code.

5.2.4 Flow Counters

The structure of the IPMP flow counter implementation for both BSD and Linux is shown in figure 5.3. The flow counter implementation in this work is based on the implementation of syncache in the FreeBSD operating system [73], which uses a similar technique to maintain temporary records of incoming TCP SYN packets. The structure consists of a hash table, a queue of flow counters ordered by expiry time, and the flow entries themselves. In figure 5.3(a), each square box represents a flow entry, the dashed lines represent links for joining flow entries with the same hash index, the solid lines represent links for maintaining the expire queue, and the number in each flow entry represents the next flow entry to expire. In figure 5.3(a), the next entry to expire is the entry marked 1.

The flow counter implementation shown in figure 5.3(a) consists of a hash table of N entries; each entry in the table is a queue of IPMP flow records of maximum length L, ordered from Most Recently Used (MRU) to Least Recently Used (LRU). The implementation described allows for O(1) insertion, modification, and deletion of entries in the flow table. The search operation is O(L) worst-case, although this is mitigated by restricting the length of each queue, by choosing a hash table with more entries to reduce index collisions, and by ordering the flow counters based on their last use. This implementation strategy was chosen because balanced tree data structures are not as widely available among the various BSD kernels as queues are, and the syncache provided a suitable model for implementing the flow counter data structures.

The structure of the IPMP flow record structure is shown in figure 5.3(b). Each flow record structure contains the source and destination IP addresses and the IPMP ID field which identify the echo flow, as well as a flow counter that records the number of packets seen for the particular flow. As the hash table is held centrally while flow records are scoped to an interface, a flow record also includes the interface index that the flow is scoped to. An interface index value uniquely identifies each network interface in the system.

A flow record also contains the time that it is due to expire, which is a fixed amount of time after the interface last received a packet which belonged to the flow.

In addition to this data, a flow record also contains four fields for managing the flow counter hash table. Two of these fields link the flow record to other flow records that share the same hash key value, while the other two fields link to the records that will expire before and after the flow record.

All flow records are allocated out of the same *zone*, which allows the operating system to allocate all IPMP flow records out of a small group of memory pages. Allocating flow records out of a single zone increases the chance that other flow entries required subsequently will be found in the CPU's cache. Whenever a new flow record is created, or a flow counter of an existing record is incremented, the particular flow record is moved to the front of the list corresponding to the appropriate hash index. The list of entries is searched from MRU to LRU so that the most active flow records are located faster. The flow records are locked with a mutex, restricting the use of flow counters to one thread of execution at a time, because the kernel may have multiple threads of execution contending for flow records which are stored centrally.

5.3 Software Forwarding Performance

This section measures the additional forwarding delay incurred by an IPMP echo probe where a path record is inserted as the packet is forwarded. In a software implementation of IPMP, additional processing delay defines additional CPU processing load. The software-based system measured is a FreeBSD router. In order to determine the additional processing delay incurred, the difference in forwarding time of a FreeBSD router that makes IPMP modifications is measured and compared to one that does not.

5.3.1 Apparatus

Figure 5.4 illustrates the apparatus of this experiment. The source sends IPMP echo request packets to a destination machine that is a single hop away. The router is a FreeBSD 5.3-RELEASE system with a Pentium3 800Mhz CPU and five physical network interfaces. Four of the interfaces are Intel EtherExpress interfaces on a

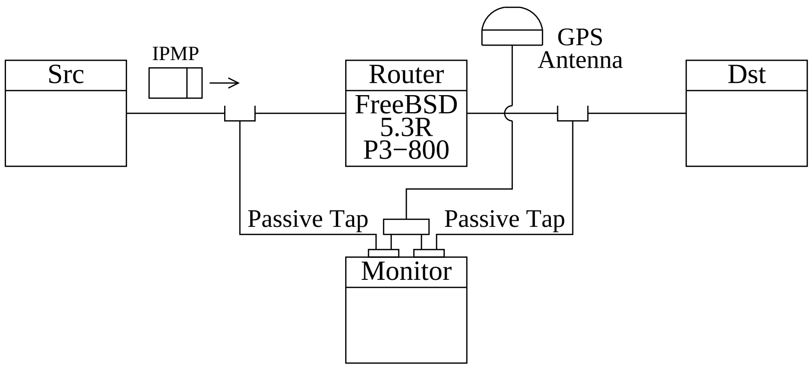


Figure 5.4: Apparatus of the IPMP forwarding performance experiment

single PCI board, two of which are enabled and used to connect the source and destination hosts, respectively. In this experiment; both interfaces operate at 100baseT full-duplex. The fifth interface is a 3Com 3c905C-TX, which is used as out-of-band access to control the configuration of the router.

A fourth machine is used to passively monitor and timestamp each packet it observes entering and exiting the router on both links. Two Dag 3.5e cards [74] were used to passively monitor the two links in this machine. The Dag cards were used in an in-line manner, where a host connects to one port, and the router is connected to the other port. The clock of each Dag card was synchronised from a single Trimble Palisade Global Positioning System (GPS) receiver, whose RS422 output was replicated to each card. This synchronisation enables absolute time differences to be measured between when a packet is seen entering and then exiting the router. The Dag cards used in this experiment have a specified precision of 60ns, which is sufficient to measure packet delay on a per-byte basis. The forwarding delays in both the forward and reverse directions can be measured, as the path in this experiment is symmetrical.

5.3.2 Methodology

Four experiments were conducted in order to measure the performance of various IPMP echo forwarding components. These experiments are summarised in table 5.1. First, the forwarding delay of the router without IPMP echo forwarding operations is measured, and is used as the control. Second, the IPMP forwarding

Experiment	Name	IPMP	TSC	FlowC
1	No-IPMP			
2	IPMP #1	×	×	
3	IPMP #2	×		
4	IPMP #3	×		×

Table 5.1: Summary of the software forwarding performance experiments

delay when a path record is inserted was measured. In this experiment, the timestamp was derived from the least significant 48 bits of the TSC register and flow counters were disabled. Third, this experiment was repeated, but using nanotime to derive the path record timestamp. Finally, this experiment was repeated, but with flow-counters enabled. Each of these experiments should show additional processing delay compared to the previous experiment because each additional forwarding operation requires additional CPU time.

To measure if IPMP processing delay is affected by the size of an IPMP echo probe, the source sent 17600 IPMP echo request packets of varying size to the destination. The packets varied in size between 96 and 1500 IP packet bytes; each size was a multiple of 4 bytes, and 50 packets of each size were sent in random order. 96 bytes is the smallest IPMP packet with space for 5 path records, and 1500 bytes is the largest IP packet supported by the network interfaces.

Echo request packets were sent each time an echo reply for a previous request was received. There was no other traffic on the network, and no IPMP packets were lost in any of the experiments. Each IPMP echo request belongs to a different IPMP flow so that the flow counter implementation is required to manage a number of flow records in order to gain a worst-case understanding of the flow counter overhead.

5.3.3 Results

Figure 5.5 plots the minimum forwarding delay measured against packet size on the forward path. The minimum delay observed is plotted because that corresponds to the forwarding time of the packet when the system was otherwise idle and not interrupted by other unrelated system tasks or activity. Figure 5.5 suggests that inserting a path record adds little additional processing delay, although the processing delay

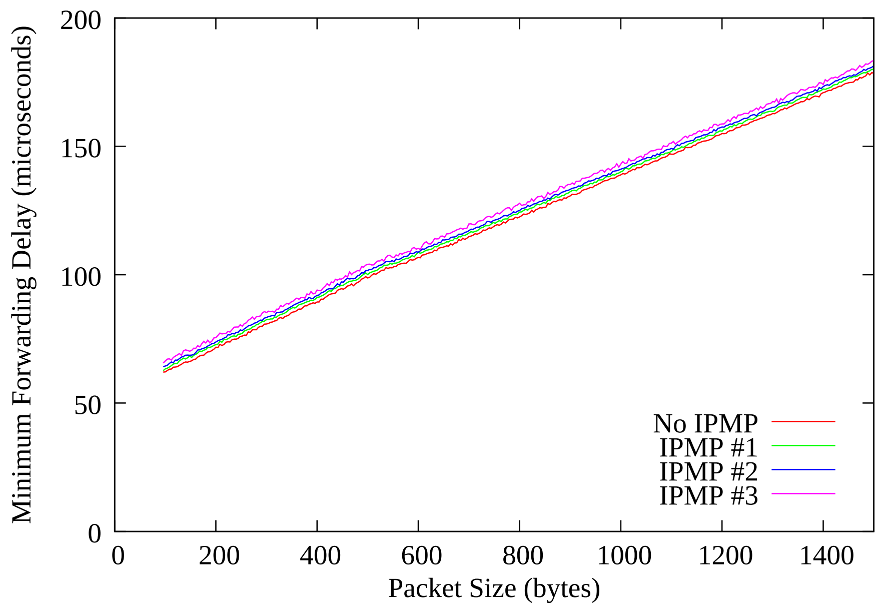


Figure 5.5: Forward-path minimum forwarding delay of IPMP packets against packet size

grows with the additional per-packet processing required for experiments IPMP #2 and IPMP #3.

Table 5.2 shows the minimum, lower quartile, median, and upper quartile values of processing delay inferred for each of the four experiments. As forwarding delay grows linearly with packet size, serialisation delay is excluded by computing the linear least squares fit for each statistic. That is, processing delay is inferred by using the intercept of the linear least squares fit with the Y-axis.

The first three rows in table 5.2 show the minimum processing delays inferred for each of the four measurements, the additional processing delay measured compared to no IPMP processing, and that additional delay as a percentage. Without IPMP processing, the processing delay is 56.4μs. A path record with a timestamp derived from the value of the TSC register (IPMP #1) requires an additional 1.5μs processing delay, which corresponds to 2.6% additional processing delay compared to not doing any IPMP processing. A path record with a timestamp derived from nanotime (IPMP #2) incurs an additional 1.0μs processing delay, which corresponds to 4.4% additional processing delay compared to not doing any IPMP processing. Finally, a path record with a timestamp derived from nanotime and a flow counter (IPMP #3) requires an additional 1.8μs processing delay, which corresponds to

Statistic	Processing Delay (μs)			
	No IPMP	IPMP #1	IPMP #2	IPMP #3
Minimum:	56.383	57.845	58.836	60.666
		+1.462	+2.453	+4.283
		+2.6%	+4.4%	+7.6%
Lower Quartile:	57.443	58.649	59.723	62.024
		+1.206	+2.280	+4.581
		+2.1%	+4.0%	+8.0%
Median:	58.403	59.194	60.262	62.643
		+0.791	+1.859	+4.240
		+1.4%	+3.2%	+7.3%
Upper Quartile:	62.912	59.861	60.947	63.454
		-3.051	-1.965	+0.542
		-4.8%	-3.1%	+0.9%

Table 5.2: Forward-path IPMP echo processing delay

7.6% additional processing delay. In this case, the minimum additional processing delay corresponds to a hash table index with no other flow records.

The last three groups in table 5.2 show the median processing delay and the inter-quartile range of the processing delays measured for each experiment. The third group, corresponding to the median processing delay values of the four experiments, shows that the median additional processing delay is less both as a percentage and an absolute value compared to the minimum observed delays. This indicates that, on average, the additional processing delay will be less than when a packet is processed with the minimum forwarding time, because the minimum forwarding case is not the usual case even on a machine that is otherwise idle.

The first column in table 5.2, corresponding to the forwarding delays measured where no IPMP forwarding operations took place, shows a greater interquartile range than the experiments where IPMP forwarding operations took place. In addition, the upper quartile values are less for the experiments where IPMP was enabled than where it was not. One possible reason for this result is by requiring additional IPMP processing in cases where the system is otherwise unable to forward the packet, the system will be ready to forward the packet when IPMP processing finishes.

Figure 5.6 shows the minimum forwarding delay measured for each packet size in the reverse direction, and table 5.3 shows the processing delay inferred for the

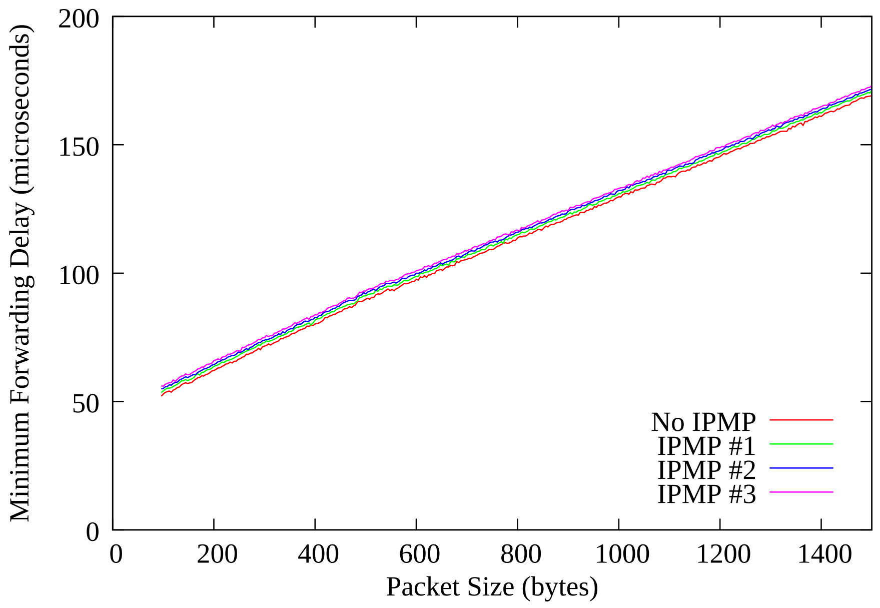


Figure 5.6: Reverse-path minimum forwarding delay of IPMP packets against packet size

Statistic	Processing Delay (μs)			
	No IPMP	IPMP #1	IPMP #2	IPMP #3
Minimum:	47.039	48.405	49.430	50.475
		+1.366	+2.391	+3.436
		+2.9%	+5.1%	+7.3%
Lower Quartile:	47.956	49.219	50.277	51.308
		+1.263	+2.321	+3.352
		+2.6%	+4.8%	+7.0%
Median:	48.854	49.726	50.783	51.821
		+0.872	+1.929	+2.967
		+1.8%	+3.9%	+6.1%
Upper Quartile:	52.745	50.234	51.419	52.391
		-2.511	-1.326	-0.354
		-4.8%	-2.5%	-0.7%

Table 5.3: Reverse-path IPMP echo processing delay

reverse path. While the graphs in figures 5.5 and 5.6 appear similar, the minimum processing delay through the router is 9.3μs less on the reverse path than that on the forward path. This difference is most likely due to additional latency in the quad port network interface card, because the Dag cards used in the experiment were synchronised. It is important to note that the difference in forwarding delay on the reverse path is nearly seven times larger than the additional processing delay of IPMP in its minimum configuration. This indicates that IPMP processing delay is not significant, and that other factors can have more effect than adding IPMP to the forwarding path. Experiments IPMP #2 and #3 show similar additional delays to the corresponding experiments for the forward path. The 4th experiment – the IPMP configuration where the flow counter implementation is enabled – shows significantly less additional processing than the same measurements made in the forward path.

5.4 Hardware Implementation

The fundamental difference between a software implementation and a hardware implementation is that hardware is limited by physical constraints while software is often constrained more critically by execution time. A software implementation of IPMP is required to execute a series of instructions serially as quickly as possible, while a hardware implementation is limited by the number of logic gates and flip-flops. Therefore, when examining the ability to implement IPMP in hardware, an important metric to consider is the number of logic gates and flip-flops required. This section describes a hardware implementation.

5.4.1 Overview

The IPv4 forwarding component of IPMP, described in section 4.4.6, was implemented in a layer 2 Ethernet switch that was built for research purposes by the WAND network research group. The WAND group is a network measurement and research group at the University of Waikato, of which I am a member. The switch has a customised data-path which is non-blocking and does not have conventional

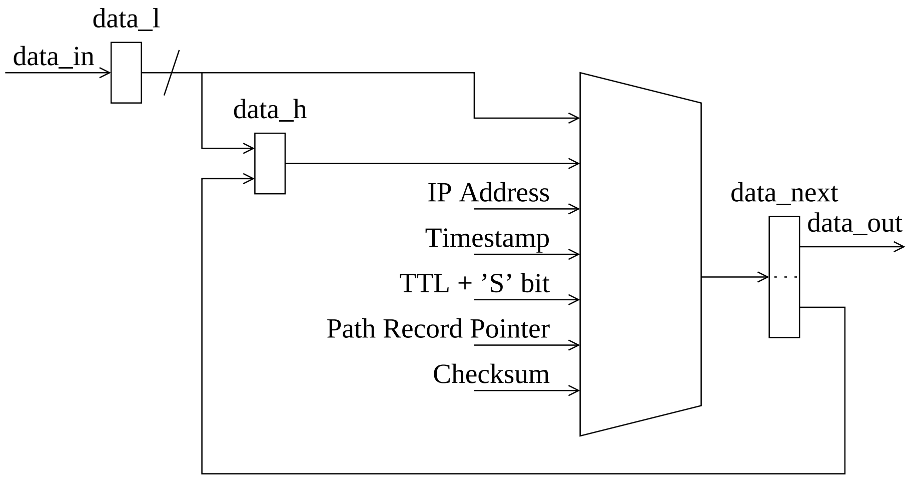


Figure 5.7: Illustration of the IPMP forwarding component in hardware

Ethernet Media Access Control (MAC) functions. Instead, the ports on the switch are configured in a cross-bar fashion, where bits that arrive on one port are transmitted out another port after a momentary delay. This is different to the traditional Ethernet notion of store-and-forward behaviour where the entire packet is buffered before being forwarded. As discussed in section 4.4.6, IPMP is well suited to either packet forwarding strategy.

The data-path of the switch is implemented in a Field-Programmable Gate Array (FPGA) and is therefore well suited to customisation. The IPMP forwarding component was implemented in 492 lines of liberally commented VHSIC Hardware Description Language (VHDL) code. The code is provided in appendix B. Compared to a forwarding path without IPMP, the IPMP forwarding component requires an additional 6592 Application-Specific Integrated Circuit (ASIC) gate equivalents.

An illustration of the IPMP forwarding component is shown in Figure 5.7. The IPMP forwarding component receives data in 8-bit quantities, labelled as data_l and data_h in figure 5.7. Two bytes at a time are buffered so that the IPMP path record pointer and IPMP checksum fields can be modified in-line as they arrive. Incoming data is multiplexed with potential replacement values. The replacement values, listed beside the multiplexor in figure 5.7, are either provided externally such as the IP address and timestamp, or working values from the current packet such as the TTL, path record pointer, and checksum. A state machine combined with a count of the number of bytes of the current packet already seen is used to determine the

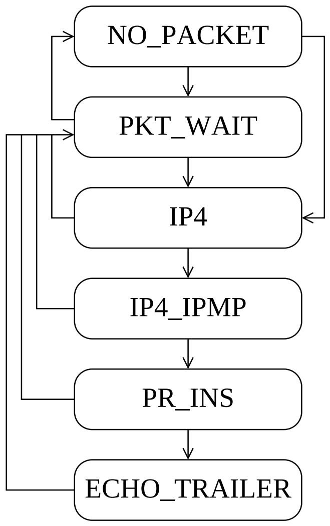


Figure 5.8: State machine of the IPMP forwarding component

current field and what data should be placed in data_next from the possible inputs. One byte at a time is read from the data_next and output to the Ethernet MAC.

5.4.2 Checksums

Computing a replacement checksum value is a potentially difficult computation because it depends on the replacement values inserted into the packet as it is forwarded. If a path record is inserted, then seven 16-bit words in the IPMP packet are modified, not including the checksum field itself. In order to avoid a final time-consuming computation of the replacement checksum field, the replacement checksum field is computed iteratively and stored in a 32-bit word using the seven replacement values as they are written out. When the checksum field arrives, the final operation required is to sum the value of the checksum to the working computation, and then fold the 32-bit working value into a 16-bit checksum and write the replacement checksum out.

5.4.3 State Transitions

The state machine transitions are shown in figure 5.8. The state machine has six possible states and eleven possible state transitions. The state machine begins in the NO_PACKET state. When the first byte of an incoming packet is received, the current time is read and is stored for use in a potential path record, as is the interface's IP address. The state machine stays in this state until the Ethernet type field arrives. If the Ethernet type field signals an IPv4 packet, then the state machine is moved into the IP4 state for further processing; otherwise, the state machine is moved into the PKT_WAIT state.

The PKT_WAIT state is used whenever it is determined that the packet will not have a path record inserted. In this state, each incoming byte passes through the IPMP forwarding component without modification. Once in the PKT_WAIT state, the IPMP forwarding component will wait until the end of the current packet before moving back into the NO_PACKET state.

When in the IP4 state, the IPMP forwarding component may process up to five operations. First, in order to determine where the IPMP header begins, the length of the IPv4 header is calculated by shifting the IP header length field 2 bits to the left, which is equivalent to multiplying by four. Second, in order to determine the total size of the IP packet, the IP length field is recorded when it arrives. The length field is used immediately to determine if the packet is large enough to include an IPMP echo header, space for an additional path record, and an echo trailer. If it is not large enough, then the state machine is moved into the PKT_WAIT state. Third, it determines if the packet is a complete, unfragmented IP packet by ensuring the IP MF bit is not set, and the IP offset field is zero. If the packet is a fragment, then the state machine is moved into the PKT_WAIT state because a complete IPMP echo probe is required in order to insert a path record and modify the echo trailer. Fourth, the value of the IP TTL field is recorded so that it can be used later when forming a path record. Finally, the IP protocol field is examined; if the type is IPMP, then the state machine is moved into the IP4_IPMP state for further processing, otherwise the state machine is moved into the PKT_WAIT state.

When in the IP4_IPMP state, the IPMP forwarding component checks that the IPMP version field is one, and that the echo bit in the IPMP type field is set. If these two conditions hold, then a calculation is made to determine where the first path record in the packet will be located, and the state machine is moved into the PR_INS state. Otherwise, the state machine is moved into the PKT_WAIT state, because the IPMP packet is not an echo probe.

When in the PR_INS state, the IPMP forwarding component compares the current byte offset with a value pre-calculated to determine when the first byte of the next path record has arrived. Additionally, the forwarding component ensures that with each byte read there would be enough space left in the packet for a path record should one be inserted. Each time the current byte offset matches the value denoting where the next path record begins, the IP TTL value stored earlier is compared with the pre-initialised TTL value in the path record. If the stored IP TTL value is greater than the path record TTL, then the state machine is moved into the PKT_WAIT state, because a node may only insert a path record in the first available space reserved for a path record. Otherwise, it then checks if the 'S' bit is set; if it is set, then the pre-calculated path record offset is adjusted to determine where the next path record begins, as this space has already been used. Otherwise, the checksum calculation records the existing value of the TTL, and then the path record begins to be written out. This process continues for each incoming byte until the complete path record has been written. The state machine then moves into the ECHO_TRAILER state.

The final state the IPMP forwarding component may enter is the ECHO_TRAILER state. Prior to entering this state, a path record has been inserted. The purpose of this state is to wait until the echo trailer arrives and then to update the path record pointer and checksum fields found in it. The IPMP forwarding component determines when the trailer has arrived by comparing the current byte counter with the length of IP payload reported in the IP header. When the path record pointer field arrives, the existing value is read and replaced with a value 12 greater than the current value. When the checksum field arrives, the existing value is read and merged into the incremental calculation of the final checksum value. The only operation left is to fold the checksum and write the replacement value, as discussed in section 5.4.2, because the checksum alteration has been iteratively calculated as each field was modified.

5.5 Hardware Forwarding Performance

5.5.1 Methodology

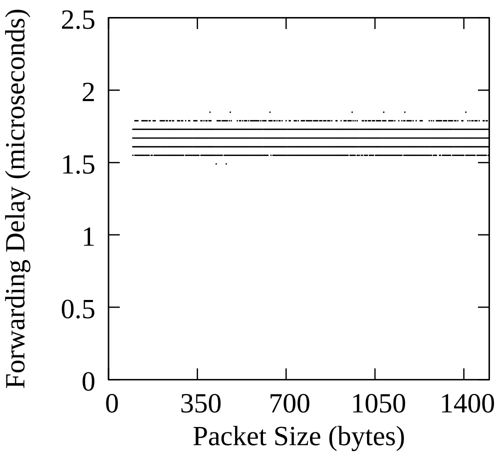
This section measures the forwarding delay of the switch with and without IPMP. The experiment used is similar to the apparatus and methodology used for the software forwarding performance measurement described in section 5.3, except that the software router is replaced by the switch hardware. The other difference is that only two forwarding modes of the switch are measured; with IPMP, and without IPMP.

The configuration of the switch in these experiments is slightly different to how the switch would normally be configured. Normally, the switch would not have to calculate the 4-byte Ethernet Frame Check Sequence (FCS) as part of forwarding a packet, because it does not modify any packet it forwards. However, the Ethernet FCS is required to be calculated when an IPMP packet has a path record inserted. Both measurements of the switch with and without IPMP include the FCS being recalculated, even though this step is unnecessary in the second experiment. This is done for two reasons.

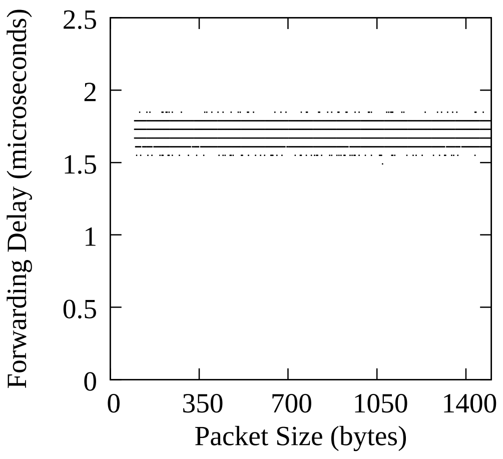
First, the FCS calculation occurs at layer 2, beneath the IP layer. The FCS calculation is only of interest to Ethernet links; other layer 2 technologies may have different checksum functions, or may not have a layer 2 checksum at all. Second, an IP router will have to calculate the FCS even if it does not implement IPMP, because an IP router modifies a packet when it decrements the TTL field in the IP header. Therefore, the switch was configured to recalculate the FCS in both experiments, though ordinarily it would not recalculate the FCS unless the packet was modified. In a separate measurement of the switch, it was determined that the FCS calculation adds 360ns $\pm$ 60ns additional delay.

5.5.2 Results

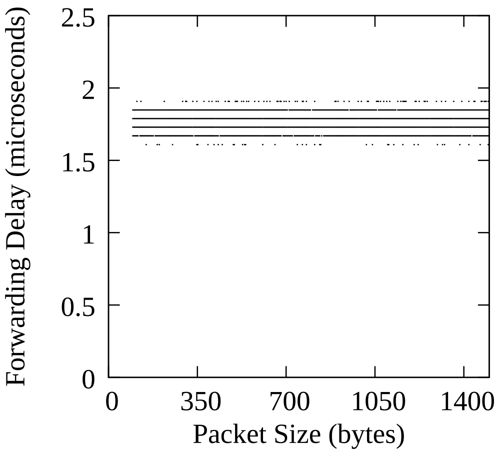
The results of the hardware forwarding performance measurement are shown in a series of six graphs in figure 5.9. Each graph plots forwarding time against packet size. The first two graphs show the forwarding delay measured through the switch for both the forward and reverse directions where no IPMP support is present. The



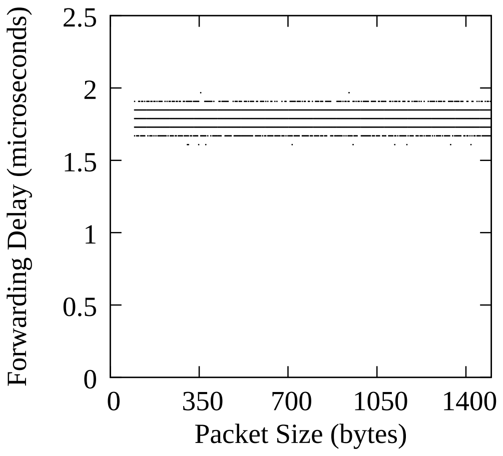
(a) No IPMP (fwd)



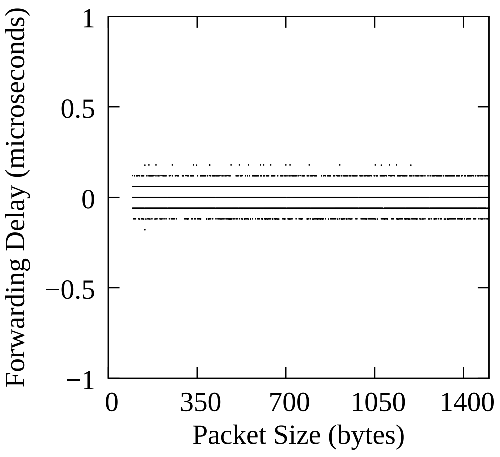
(b) No IPMP (rev)



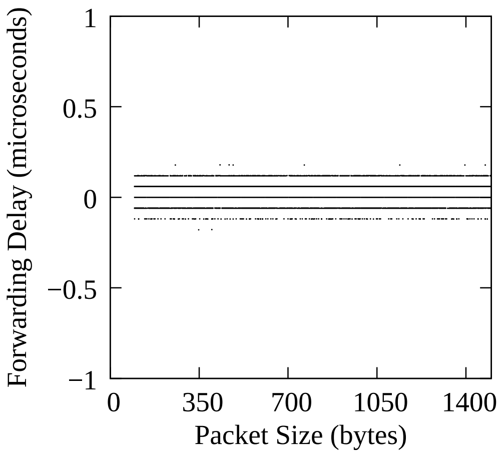
(c) With IPMP (fwd)



(d) With IPMP (rev)



(e) Dag to Dag (fwd)



(f) Dag to Dag (rev)

Figure 5.9: Forwarding delay through non-blocking crossbar switch

Forwarding Delay	Forward Path		Reverse Path	
	Frequency	Percentage	Frequency	Percentage
1.490μs	2	0.01%	1	0.01%
1.550μs	810	4.60%	78	0.44%
1.609μs	5685	32.30%	1286	7.31%
1.669μs	8200	46.59%	6171	35.06%
1.729μs	2593	14.73%	7732	43.93%
1.788μs	303	1.72%	2271	12.90%
1.848μs	7	0.04%	61	0.35%
Total:	17600	100%	17600	100%

Table 5.4: Forwarding delay without IPMP through switch

Forwarding Delay	Forward Path		Reverse Path	
	Frequency	Percentage	Frequency	Percentage
1.609μs	40	0.23%	10	0.06%
1.669μs	1333	7.57%	470	2.67%
1.729μs	6657	37.82%	4687	26.63%
1.788μs	7801	44.32%	8960	50.91%
1.848μs	1676	9.52%	3437	19.53%
1.907μs	93	0.53%	304	1.73%
1.967μs	0	0%	2	0.01%
Total:	17600	100%	17600	100%

Table 5.5: Forwarding delay with IPMP through switch

middle two graphs show the forwarding delay measured through the switch where IPMP support is present. The final two graphs show the forwarding delay measured between the two Dag cards when they are directly connected to each other with an Ethernet crossover cable. In this situation, each packet arrives instantaneously at both Dag cards. As the cards are synchronised, they should report identical time-stamps for each packet; that is, the difference in packet timestamps at each Dag card should be zero.

The graphs have a number of similarities. First, the first four graphs show that the size of the packet does not affect the forwarding time of the packet through the switch with or without IPMP support. Second, all graphs have bands offset by 60ns from each other. These bands are artifacts of the Dag cards used, which have a specified timestamp precision of 60ns. Figures 5.9(e) and 5.9(f) show the Dag artifacts with the switch removed; there are three strong bands at -0.060μs, 0μs, and 0.060μs, with two weaker bands a further 60ns from the outer bands.

Timestamp	Forward Path		Reverse Path	
Difference	Frequency	Percentage	Frequency	Percentage
-0.179μs	1	0.01%	2	0.01%
-0.119μs	390	2.22%	191	1.09%
-0.060μs	4725	26.85%	1846	10.49%
0.000μs	8463	48.09%	7808	44.36%
0.060μs	3595	20.43%	6531	37.11%
0.119μs	403	2.29%	1214	6.90%
0.179μs	23	0.13%	8	0.05%
Total:	17600	100%	17600	100%

Table 5.6: Accuracy bounds of IPMP forwarding delay experiments

Tables 5.4, 5.5, and 5.6 show the frequency of each band for the three measurements. Table 5.6 shows the frequency of each timestamp difference when the Dag devices are directly connected, and shows that the most common timestamp difference measured between the Dag interfaces is no timestamp difference. Approximately 46% of all packet timestamps are identical between the Dag cards, while another 47% of all packet timestamps are ± 60ns. This result indicates that the mode is the best measurement of delay for the switch.

On the forward direction, the most common delay measured without IPMP is 1.67μs and 1.79μs with IPMP, indicating an IPMP forwarding delay of 120ns $\pm$ 60ns. However, for the reverse direction, the most common delay measured without IPMP is 1.73μs and 1.79μs with IPMP, indicating a forwarding delay of 60ns $\pm$ 60ns. These conflicting results indicate the actual forwarding delay added by including IPMP in the switch is between 60ns and 120ns, and that the margin of error is in the same order of magnitude as the delay itself.

5.6 Summary

This thesis argues that a protocol for per-hop measurement of Internet packet dynamics is feasible and useful. In this chapter it was proven, through implementation, that IPMP is feasible to implement. The software implementations showed that the overhead of inserting a path record is minor compared with the other packet processing that a host does when forwarding any other packet, and that other factors such as the model of network interface can have a larger effect than adding IPMP.

The hardware implementation showed that the IPMP forwarding component only needs to buffer two bytes of the packet. Decrementing the IPv4 header's TTL field and then incrementally updating the checksum also requires two bytes of the packet to be buffered; therefore, if implemented in the same processing stream as the TTL decrement, inserting a path record into an IPMP echo probe will not further delay the packet.

In order to facilitate the incremental deployment of IPMP, an IPMP hardware dongle could be developed and deployed so that IPMP could be deployed without requiring routers to be upgraded. Such a device would attach to a router's port and process packets in-line as they are received. The VHDL implementation provided in appendix B provides the digital logic required for processing an IPMP packet in an Ethernet device.

Chapter 6

IPMP Measurement Techniques

6.1 Introduction

In this chapter, the application of IPMP to existing packet probing techniques is discussed. The use of IPMP to measure topology, delay, loss, reordering, and capacity is considered in three different deployment scenarios. The first case considered is the optimal case where all nodes have IPMP support on the forward and reverse paths. In this case, each IPMP echo packet has a path record inserted at each hop in the network. This enables the packet's dynamics to be measured on a per-hop basis. The second case considered is the case where some routers have IPMP support and those routers are at strategic locations such as at AS boundaries. In this case, each IPMP echo packet has a path record inserted at these strategic locations, which allows the packet's dynamics to be measured on a per-AS basis. The third case considered is the minimum case where only the end hosts have IPMP enabled. In this case, each ICMP echo packet has a path record inserted at the source when it is transmitted, at the destination when it is received, and then back at the source when it is received. This allows the packet's one-way dynamics to be measured.

6.2 Topology

6.2.1 All Nodes with IPMP

An IPMP echo packet that has a path record inserted at each hop contains the complete IP topology of the path followed by the packet for both the forward and reverse paths, because an IPMP path record includes an IP address. Converting the IP topology to a router-level topology is done by sending an IPMP information request to each IP address, and then noting which replies have the same source IP address. Converting the IP topology to an AS-level topology requires some way of determining which AS is responsible for the IP address. IPMP provides an additional facility for this, through a node providing the AS number to which it belongs in an information response, as discussed in section 4.5.4.

Compared with the TTL-limited technique that `traceroute` uses, IPMP offers a number of useful features, especially if all nodes have IPMP deployed on the forward and reverse paths. First, it allows the forward and reverse path IP topologies to be determined in a single packet exchange. A separate probe for each hop is not required as it is with the TTL-limited technique. The IP path reported is not susceptible to being a false path, because an IPMP echo packet is routed normally to its destination. Second, it allows for a robust method of definitively determining which IP addresses are aliases of the same router, because a mechanism for doing so is provided for in the IPMP information protocol. When replying to an information request, the router uses a constant source address to send each information reply. In order to resolve router aliases, the IP addresses included in path records must each be sent an information request. Third, if each router is configured to return the AS to which it belongs in an IPMP information reply, the AS path can also be determined without requiring access to routing tables or other separately managed data.

6.2.2 Some Nodes with IPMP

If IPMP is deployed at strategic locations, such as at AS boundaries, then the sequence of ASes on the forward and reverse paths between two systems can be determined. The TTL-limited technique can be used to infer the IP addresses for the remaining hops on the forward path which did not insert a path record, if required. The appropriate TTL values to use when probing to infer the remaining hops are those that correspond to routers that did not insert a path record. This can be determined by examining the TTL fields of each path record which was inserted. In this case the number of probes required to infer the forward path is less than if TTL-limited technique alone was used.

6.2.3 End Hosts with IPMP

If IPMP is only deployed at the source and destination, then it provides the ability for the length of the forward and reverse paths, in IP hops, to be determined by using the TTL value embedded in a path record at the destination. This can be useful for measuring path symmetry in IP hops, but is unable to provide additional data to explain why.

6.3 Delay

6.3.1 All Nodes with IPMP

An IPMP echo packet that has a path record inserted at each hop allows, at a minimum, per-hop jitter to be measured for all hops on the forward and reverse paths, because an IPMP path record includes a timestamp. If two probes P and P' are sent, and the path records inserted at hop h are notated as $P_{[h]}$ and $P'_{[h]}$ respectively, then the relative delay jitter for these two probes is $P'_{[h]} - P_{[h]}$. Per-hop queueing delay can be measured by a technique similar to that used by `cing` or `tulip`. If enough probes are sent, then each hop should have at least one probe encounter minimum queueing delay through it, allowing the other probes to infer the queueing delay they experienced.

When compared with the ICMP timestamp techniques used by `cing` and `tulip`, IPMP offers a number of useful features for measuring delay and jitter, especially when IPMP is deployed on all hops on the forward and reverse paths. First, the precision of the IPMP path record timestamp can be much better than the maximum 1ms resolution of the timestamp provided by the ICMP timestamp protocol. Second, because IPMP echo packets are designed to be efficiently processed in the forwarding path, the timestamps can be a more reliable measure of the behaviour of the path leading to the router.

Third, as an echo packet is otherwise forwarded normally and can go on to collect additional timestamps from other systems in the path, it is possible to definitively determine the queueing behaviour between two arbitrary systems that both insert a path record. Using the timestamps in an echo packet separates the behaviour between the particular systems from the behaviour prior and subsequent to the hop. This is a more robust method than inferring the queueing behaviour of the path prior to the hop being measured, and then inferring the per-hop queueing behaviour from the additive effect of travelling the particular hop.

Fourth, it requires no extra effort to measure the queueing delay for hops on the reverse path, because path records are also inserted on the reverse path. Finally, because each hop in the path is not probed individually, the number of probes required to measure queueing delay does not grow with the path's length.

As discussed in section 4.5, determining the relationship that a particular timestamp has to real time is accomplished using the IPMP information protocol. In the event where the clocks on two different interfaces are determined to be synchronised, it is possible to measure absolute one-way delay between the particular interfaces. Suitable calibration sources might be a high quality real-time source such as a GPS or Code Division Multiple Access (CDMA) time receiver, or a recovered line clock from a Synchronous Optical Network (SONET) link provided the master clock is sufficiently accurate.

6.3.2 Some Nodes with IPMP

If IPMP is deployed at strategic locations, such as at AS boundaries, then it is possible to determine queueing delay statistics through particular ASes. If a path exhibits significant jitter, then being able to reduce the problem domain into a series of ASes is useful for determining who the responsible ASes are, how they contribute to the jitter, and where in the topology they are. If the respective systems at the AS boundaries have synchronised clocks, then absolute one-way delay between the systems can be determined. If a path exhibits significant absolute delay, then being able to measure the absolute delay contributed by each AS is useful for determining which

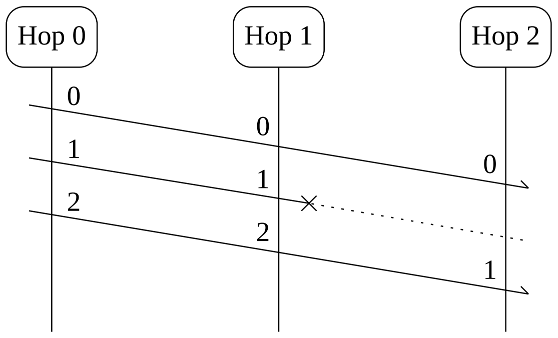


Figure 6.1: Example IPMP flow counter values after single probe loss

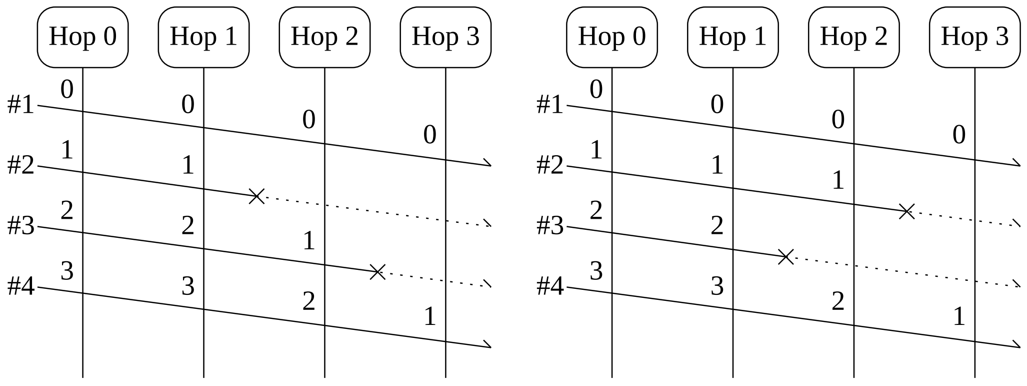
ASes contribute the most delay, and for determining a more appropriate topology if one exists.

6.3.3 End Hosts with IPMP

If IPMP is deployed only at end hosts, then it is useful for determining one-way jitter on the forward and reverse paths, and absolute one-way delay if the clocks are synchronised. While the raw data provided in this scenario is limited, it is still useful for diagnosing end-to-end performance problems, and in measuring path asymmetry.

6.4 Loss

It may be possible to determine where an echo packet was lost using subsequent echo probes, because a system which implements IPMP may also keep flow state and report the value of the flow counter in a path record. Figure 6.1 outlines a simple example of determining where a packet was lost using IPMP flow counters. In this example, the source sends three packets towards a destination, though the middle packet is lost. The first packet is returned to the source with all flow counters having the value 0. The second packet is lost between hops 1 and 2; because hops 0 and 1 forward the packet before it is subsequently lost, they increment the flow counter corresponding to the packet's flow record. The third packet is not lost and is returned to the source, and can be used to determine that the second packet was lost after hop 1 by comparing the flow counter fields in the packets received.



(a)
Probe 2 is lost after Hop 1,
Probe 3 is lost after Hop 2.

(b)
Probe 2 is lost after Hop 2,
Probe 3 is lost after Hop 1.

Figure 6.2: Example IPMP flow counter values when two consecutive probes are lost

6.4.1 All Nodes with IPMP

When compared with the IP-ID technique that `tulip` uses, IPMP offers a number of useful features for measuring packet loss, particularly if IPMP is deployed on all routers in the path, and they all keep flow state. First, the behaviour of the flow counter is defined, unlike the IP-ID field, so determining if a previous packet was received by a router is more reliable. Second, diagnostic tools do not need to send the triplet of packets formed in the `tulip` style where the outer packets are used to infer forward loss, because the primitive for measuring loss is embedded in subsequent measurement packets. Third, because a single packet collects the necessary information from each router, the number of probes required to infer the location of packet loss does not grow with the path's length.

Using IPMP to infer per-hop loss does have some limitations, however. If more than one packet is lost between two received packets and the packets are lost at different hops, while it is possible to infer how many packets were lost at each hop, it is not possible to determine the order in which the packets were lost. Figure 6.2 shows an example where two consecutive packets are lost at two different hops in the path. In case (a), probe #2 is lost between hops 1 and 2, and probe #3 is lost between hops 2 and 3. The IPMP-ID signature in probe #4 – which is not lost and is used to infer where loss occurred – is the same as the IPMP-ID signature in probe #4 in case (b), where probe #2 is lost between hops 2 and 3, and probe #3 is

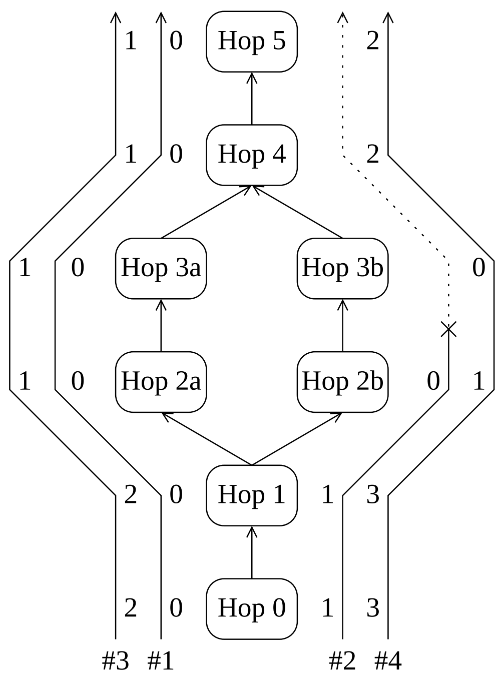


Figure 6.3: Example IPMP flow counter values when probes follow alternate IP paths

lost between hops 1 and 2. While fine-grained measurement of where each packet is lost might be useful in some scenarios, the ability to infer where each packet was lost is adequate to identify lossy and congested links.

A second limitation arises in measuring per-hop loss where multiple paths exist towards a destination, and packets are routed round-robin through alternate paths towards the destination. Figure 6.3 shows an example case where probes #1 and #3 are forwarded through the path on the left, while probes #2 and #4 are forwarded through the path on the right. Probe #2 is lost between hops 2b and 3b. When probe #3 is sent, the flow counters in the packet report that this is the second packet that hop 2a has received. At this point, unless a subsequent packet is routed over the same hops where probe #2 was lost, the source might incorrectly infer that probe #2 was lost between hops 1 and 2a. Probe #4 allows the source to infer that a previous probe belonging to the same flow was seen at hop 2b but was subsequently lost, ruling out packet loss between hops 1 and 2a. Unless the underlying topology of the path is known, the IPMP technique is restricted to inferring where packet loss did not occur, because in the example presented, there could be multiple active paths between hops 2b and 4.

6.4.2 Some Nodes with IPMP

If IPMP is partially deployed at strategic locations through a path, then a diagnostic tool could infer the AS or ASes where packet loss occurs. If IPMP is deployed only at end hosts, then diagnostic tools can reliably determine whether the loss occurred on the forward path or the reverse path.

6.5 Reordering

IPMP provides two facilities to measure packet reordering: timestamps, which record when an echo packet packet was seen; and flow counters, which record the arrival order of a series of echo packets. It is recommended that measurement techniques use flow counters where possible, and timestamps where flow counters are not reported, because two consecutive packets may have the same timestamp inserted if the clock used is not of sufficient resolution.

6.5.1 All Nodes with IPMP

The technique for measuring reordering is simple. If two packets arrive at one particular hop in a different order compared to the arrival order at a prior hop, then reordering has occurred between those hops.

When compared with the IP-ID technique used by `tulip`, IPMP offers a number of useful features for measuring packet reordering, especially when IPMP is deployed on all hops on the forward and reverse paths. First, it is possible to directly measure where reordering occurs on a per-hop basis. If reordering occurs multiple times in the path, then each instance can be detected by comparing the path records inserted at each hop. Using the addresses inserted in the relevant path records, measurement tools can report the path segment or segments where reordering occurred. This is a more robust method than inferring the reordering behaviour of a particular hop by inferring the reordering behaviour of the path leading to the hop and subtracting that from the reordering behaviour measured of the hop. Second, the number of probes required to characterise packet reordering does not grow with the length of the path, because each probe collects information for the complete forward and reverse paths. Third, if reordering occurs due to the probe packets taking different IP paths, it is possible to identify the two paths by the IP addresses that make up those paths.

6.5.2 Some Nodes with IPMP

In the case where IPMP is partially deployed at strategic locations through the path, such as at AS boundaries, then diagnostic tools can narrow the search for the reordering segment to the applicable AS. Using the TTL values in the relevant path records, diagnostic tools can report on the length of the path upon which the reordering occurred. In the case where IPMP is deployed only at end hosts, diagnostic tools can identify forward path versus reverse path reordering as `tulip` does.

6.6 Capacity

Capacity estimation requires the ability to measure the delay incurred by a packet of a particular size. With IPMP, it is possible to measure the minimum dispersion of a packet-pair at a particular hop and use this dispersion measurement to infer capacity, because an IPMP path record includes a timestamp. IPMP offers capacity estimation techniques the ability to directly measure the dispersion of a packet pair at each hop that implements IPMP. The dynamics of the packet-pair after it leaves a hop which implements IPMP does not affect the measurement of packet dispersion at that hop, because a path record contains the time each packet was seen at that hop. For these reasons, error does not accumulate in per-hop capacity estimation as it does in techniques discussed in section 2.6.3.

6.6.1 All Nodes with IPMP

If all hops on the forward path implement IPMP, then it is simple to determine the capacity of the path and where the capacity limiting link is positioned. If enough packet pairs are sent through the network, at least one packet pair should traverse the capacity-limiting link back-to-back. The capacity-limiting link can then be identified as the hop with the largest minimum measured packet dispersion. The following two equations further explain the two parts of the capacity estimation technique. The first part, shown in equation 6.1, consists of determining $\delta_{\min[h]}$ – the minimum packet-pair dispersion measured for each hop h in a path given N packet pairs each consisting of P and P'.

$$\delta_{\min[h]} = \min \left\{ P_{[h,i]} - P'_{[h,i]}, i \in [1, N] \right\} \tag{6.1}$$

The second part, shown in equation 6.2, determines the capacity c of the path by dividing the size S of each packet in the packet pair by the maximum $\delta_{\min[h]}$ value recorded for the H hops on the forward path.

$$c = \frac{S}{\max(\delta_{\min[h]}), h \in [0, H]} \tag{6.2}$$

When compared to the packet-pair technique itself, IPMP offers three main advantages. First, each dispersion measurement is taken in-place in the path. The effect of cross traffic on the dispersion of the packet pair subsequent to a particular hop does not affect the measurement of packet dispersion at that particular hop because the dispersion is recorded in the packet-pair. For this reason, a technique to filter overestimations of capacity is not required. Second, it is possible to determine where in the path the capacity limiting link is positioned and use the IP addresses inserted in path records to denote the link. Third, specialised software is not required to be run at the destination host to measure the dispersion of arriving packets.

Estimation of the capacity of the reverse path still presents measurement challenges if the capacity of the reverse path is greater than that of the forward path, because the packet pair must queue back-to-back at the capacity limiting link. One possible approach to this problem is to use a tailgater packet ahead of a packet pair of smaller probes, which is not echoed back to the source by the destination. If the packet-pair is sent back-to-back from the destination, then it is possible to use the packet-pair to estimate the capacity of the reverse path using the technique discussed in this section.

6.6.2 Some Nodes with IPMP

If IPMP is deployed at strategic locations throughout a path, such as at AS boundaries, then a slightly different technique is required in order to estimate the capacity between the locations. This is because if links of differing capacities exist between the points, the packet-pair could compress after it has been sent over the capacity-limiting link, and lead to an over-estimation of the path's capacity. Instead, a technique similar to CapProbe[49] can be used, where the dispersion of a packet pair with the minimum combined delay between the two points is used to estimate the capacity, because this packet pair is likely to have encountered no cross traffic between the two points. The main advantage this technique brings to CapProbe is the ability to determine approximately where in the path the capacity-limiting link lies.

6.6.3 End Hosts with IPMP

If IPMP is deployed only at end-hosts, then it can be used to infer the capacity of the forward path using the CapProbe technique. In this case, IPMP allows the capacity of the path to be measured without requiring a specialised program at the destination host to be run.

6.7 Summary

This chapter described IPMP-based measurement techniques to measure topology, delay, loss, reordering, and capacity, and compared these measurement techniques to currently available techniques. The IPMP-based measurement techniques described in this chapter are more robust, require less probes to be sent, and are potentially more accurate and convenient than corresponding measurement techniques that do not use IPMP, as IPMP allows routers to embed information useful to measurement of per-hop packet dynamics into IPMP probes as they are forwarded.

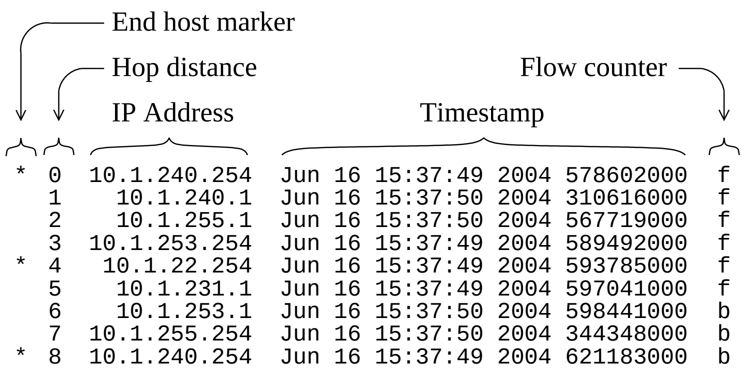
IPMP allows each link to be measured independently of the behaviour seen by the probe on prior links. If IPMP were standardised and deployed, operators would be able to diagnose path faults in greater detail than at present, using fewer probes.

Chapter 7
Applications of IPMP

7.1 Introduction

In the previous chapter, measurement techniques for measuring delay, jitter, loss, reordering, and capacity on a per-hop, one-way, and end-to-end basis were presented and discussed. This chapter provides actual examples where having IPMP available and deployed on all routers in a network has been useful in practice for understanding network behaviour. The network measured, Connecting Rural Communities Network (CRCnet), is a rural wireless network operated by members of the WAND network research group, where all routers are software-based and are able to be customised. CRCnet itself is used to provide higher capacity paths to remote schools than is otherwise available, and therefore has some real-world use. This chapter is not an exhaustive review of possible IPMP measurement applications on CRCnet. Rather, it provides three case studies where IPMP enabled per-hop measurement of packet dynamics to take place.

This chapter begins by describing a general-purpose IPMP-based measurement utility. The utility, `ipmp_ping`, assembles and transmits IPMP echo request packets and prints out replies which are received in response. The output from the utility is designed to be simple to parse so that it is suited for use in simple network measurement and monitoring scripts. Then, this chapter presents the CRCnet network and three measurement case studies. The first case study measures packet loss on a symmetrical wireless path which was reported by CRCnet operators to exhibit substantial packet loss that significantly reduced the quality of service provided. The



```
* 0  10.1.240.254  Jun 16 15:37:49 2004 578602000  f
  1    10.1.240.1  Jun 16 15:37:50 2004 310616000  f
  2    10.1.255.1  Jun 16 15:37:50 2004 567719000  f
  3  10.1.253.254  Jun 16 15:37:49 2004 589492000  f
* 4   10.1.22.254  Jun 16 15:37:49 2004 593785000  f
  5    10.1.231.1  Jun 16 15:37:49 2004 597041000  f
  6    10.1.253.1  Jun 16 15:37:50 2004 598441000  b
  7  10.1.255.254  Jun 16 15:37:50 2004 344348000  b
* 8  10.1.240.254  Jun 16 15:37:49 2004 621183000  b
```

Figure 7.1: Sample output from `ipmp_ping`

second case study measures path capacity and identifies the capacity limiting link on a different symmetrical wireless path. The last case study examines in greater detail the data collected in the second study, but instead of using the data to identify the capacity of the path, the data is used to reverse engineer the interactions the packets had as they were contending for the same resources on routers at the end of the symmetrical path.

7.2 IPMP Ping

`ipmp_ping` is a simple yet powerful application that combines the behaviour of `traceroute` with the behaviour of `ping`. That is, the IP path discovered is reported with each probe. `ipmp_ping` allows the user to specify, among other options, a target address, the probe size, how many probes to send, and how long to wait between sending probes.

Figure 7.1 shows an example of the output from `ipmp_ping`. Each line of output prints the IP address, timestamp, and flow-counter values from the path record, the distance in hops into the path where the path record was inserted, and an asterisk if a hop's IP address corresponds to either the source or the destination. The timestamp is formatted similarly to the output from the ctime function; the month, date, time, and year of the timestamp is displayed. The timestamp also includes the number of nanoseconds elapsed in the current second.

`ipmp_ping` expects the path record timestamp to have a specific format; the first 16 bits of the 48-bit path record timestamp field are the least-significant 16-

bits of a seconds-clock which counts from midnight on the 1st of January 1970, while the remaining 32-bits are used to count the number of nanoseconds that have elapsed in the current second. The timestamps in figure 7.1 are generated with the do_gettimeofday function in Linux, which has a microsecond resolution; the intermediate nodes have multiplied the microsecond timestamp by 1000 to obtain a timestamp with a nanosecond resolution.

7.3 Overview of CRCnet

As described in the introduction to this chapter, CRCnet is a rural wireless network designed to offer higher-capacity paths than what is otherwise available. These paths typically are offered to rural schools which use the network for commodity Internet access. Therefore, while CRCnet is used for research and development of network protocols, it also carries real-world Internet traffic.

Node	Description
HSK	Router at Hosking's house
MCG	Router at Tony McGregor's house
MSB	Router at Waikato management school
MWP	Router at Murray Pearson's house
PIR	Router at Mount Pirongia
PWS	Router at Pirongia woolshed
TTK	Router at Te Taka's house
WTU	Router at Waitatuna school

Table 7.1: List describing CRCnet routers used in chapter 7

Table 7.1 provides a list describing CRCnet routers used in this chapter. Many of these routers are used as relay points to reach networks that are further than the 802.11b signal could otherwise propagate. As these routers operate outdoors, most routers are small, low-powered devices. For example, the router at PIR is located near the top of a mountain where power is not readily available. This router is powered by a collection of batteries buried in the ground, which are charged by solar cells.

Time of Day	Link	Loss	Time of Day	Link	Loss
15:37:42	PWS-MWP	5	15:57:56	PWS-MWP	2
15:37:49	PWS-MWP	4	15:58:02	PWS-MWP	4
15:42:48	PWS-MWP	3	16:03:02	PWS-MWP	3
15:42:54	PWS-MWP	6	16:03:13	PWS-MWP	4
15:46:25	PWS-MWP	2	16:08:44	PWS-MWP	10
15:48:15	PWS-MWP	2	16:12:56	PWS-MWP	4
15:48:18	PWS-MWP	2	16:13:04	PWS-MWP	5
15:48:21	PWS-MWP	5	16:17:52	PWS-MWP	3
15:52:41	PWS-MWP	2	16:18:01	PWS-MWP	2
15:52:44	PWS-MWP	5	16:23:09	PWS-MWP	6
15:52:53	PWS-MWP	2	16:23:15	PWS-MWP	3
Total:					62

Table 7.2: Consecutive packets lost on CRCnet between MWP and PWS

7.4 Per-Hop Loss Measurements

This measurement-driven case study determines the symptoms of pathological loss reported on a symmetrical CRCnet path. At the time, the link in question was a recently installed wireless link that reached PSC via a relay at PWS. As IPMP was installed at each router in the path, flow counters were enabled, and only one path between GTW and PSC exists, a series of IPMP echo packets provided the necessary primitive to determine where packets were being lost on this path, using the technique described in section 6.4.

Every 1.5 seconds, a single echo request was sent from GTW to PSC. This low rate of probes reduced the probability that IPMP packets would cause congestion to occur and artificially induce packet-loss. Of the 2015 echo probes sent over the course of 50 minutes, 62 were lost, corresponding to a loss rate of 0.97%. Table 7.2 shows when the echo probes were lost, how many were lost in between received packets, and which link was determined to have lost the packets. This table shows that all probes were lost on the PWS-MWP link, that probes were lost approximately every 5 minutes, and 9 packets were typically lost at each 5 minute interval – or an effective outage of 13.5 seconds.

With this information, CRCnet operators were told which particular link caused the packet loss. Interestingly, the link only lost packets on the reverse path back to GTW. The actual cause of the loss was an antenna which was not correctly aligned.

Link	Length	Prop.
TTK – MSB	0.2km	0.67ns
MSB – MWP	13.6km	45.36μs
MWP – PIR	13.1km	43.70μs
PIR – MCG	17.4km	58.04μs
MCG – HSK	2.8km	9.34μs
HSK – WTU	0.4km	1.30μs

Table 7.3: Length and propagation delay of 802.11b point-to-point wireless links between TTK and WTU

The 5 minute intervals coincided with SNMP polling of CRCnet interfaces by Nagios [75].

7.5 Per-Hop Packet Dispersion Measurement

802.11 is an interesting layer 2 MAC on which to measure per-hop packet dispersion, as it is a half-duplex shared medium with layer 2 acknowledgements and retransmissions. Each packet sent has a layer 2 acknowledgement returned; if the acknowledgement is not received, then the packet is retransmitted. The acknowledgement is sent in the same band as any other data packet, so a gap is inserted between any packets sent which arrive at the node back-to-back. The nominal serialisation rate of 802.11b equipment is 11Mbps, although the capacity of the links seen by experimentation on CRCnet is approximately 4.2Mbps. This section investigates the capacity of a path on CRCnet.

7.5.1 Methodology

In order to gain an empirical understanding of how the behaviour of the half-duplex wireless links is related to the path's capacity, a series of 1500-byte IPMP echo request packet-pairs were sent across a 12-hop symmetrical path in CRCnet composed entirely of 802.11b point-to-point wireless links. In this experiment, all CRCnet routers had IPMP code enabled and used the PPSKit patch for more precise timing. No routers in the path had synchronised real-time clocks. The length of each wireless link and their propagation delays are shown in table 7.3. The propagation delay

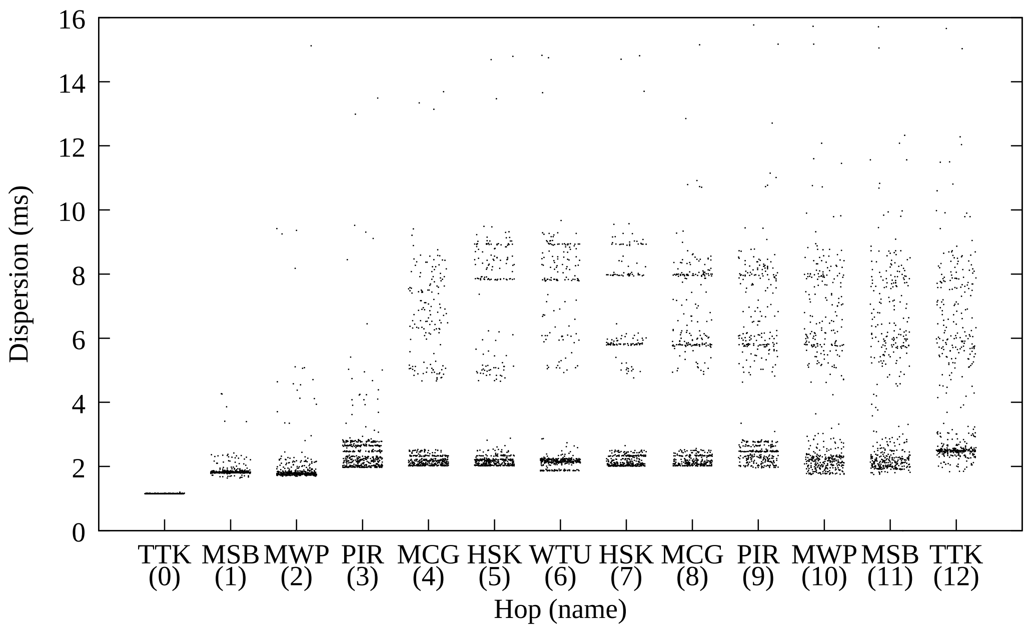


Figure 7.2: Dispersion measured for 1500-byte packet-pairs traversing 802.11b point-to-point wireless links on CRCnet between TTK and WTU

of a radio signal is approximately the speed of light when the signal takes a direct path from the transmitter to the receiver.

7.5.2 Capacity Estimation Results

Figure 7.2 shows the observed dispersion for 3000 intact packet-pairs at each hop through the path. The time stamps inserted in path records at the source host (TTK) show that the packet-pairs are queued to be sent an average of 1.06ms apart, which is slightly less than the serialisation time of 1.091ms for a 1500 byte packet at 11Mbps. Therefore, it can be stated that the packet-spacing requirement of a packet-pair – that they are sent back-to-back into the network – is fulfilled.

Table 7.4 shows the minimum observed packet-pair dispersion for all hops on the CRCnet path. The capacity limiting link is the hop in the path with the largest minimum observed packet pair dispersion, or if there is more than one hop with this characteristic, the capacity limiting link is the first hop in the path. In this case that hop is PIR to MCG, with a minimum dispersion measured of 2.013ms, which equates to a capacity of 5.961Mbps. This link is also likely to be the link that limits

Hop	Link	Min. Disp.
1	TTK – MSB	1.579ms
2	MSB – MWP	1.654ms
3	MWP – PIR	1.677ms
4	PIR – MCG	2.013ms
5	MCG – HSK	2.013ms
6	HSK – WTU	1.857ms
7	WTU – HSK	2.007ms
8	HSK – MCG	2.010ms
9	MCG – PIR	1.958ms
10	PIR – MWP	1.695ms
11	MWP – MSB	1.611ms
12	MSB – TTK	1.813ms

Table 7.4: Minimum dispersion measured for 1500-byte packet-pairs traversing 802.11b point-to-point wireless links on CRCnet between TTK and WTU

TCP throughput, because it introduces the most packet dispersion variation, which indicates queueing delay variation.

7.6 Reverse Engineering CRCnet

The previous section estimated the capacity of the CRCnet path measured by using the minimum packet dispersion at each hop and reporting the capacity of the path using it. This section investigates what can be learned from the packet-dispersion dynamics captured with IPMP on CRCnet. Taking the minimum packet dispersion of each hop from figure 7.2 does not explore the rest of the data, and ignores potentially interesting information in it. This section examines the ability to reverse engineer properties of the CRCnet path using packet interactions captured with the IPMP packet-pairs used to infer capacity in section 7.5.

First, the nominal serialisation rate of an 802.11b link is 11Mbps, yet the maximum capacity available from this path is 5.961Mbps, which corresponds to 54.2% of the nominal serialisation rate. Using information independently known about the length of each link, including the capacity-limiting link, it is possible to separate the 802.11b propagation delay from router processing delay. Second, between MCG and PIR, very few packets are observed to have a packet pair dispersion between 2.6ms and 5.0ms, and there is a significant number of packet-pairs observed to have

Link	Prop.	Data	ACK	Total
TTK – MSB	0.67ns	1262μs	116μs	1.378ms
MSB – MWP	45.36μs	1307μs	161μs	1.468ms
MWP – PIR	43.70μs	1306μs	160μs	1.466ms
PIR – MCG	58.04μs	1320μs	174μs	1.494ms
MCG – HSK	9.34μs	1271μs	125μs	1.396ms
HSK – WTU	1.30μs	1263μs	117μs	1.380ms

Table 7.5: Serialisation and propagation delay for 802.11b data and acknowledgement packets at 11Mbps

a dispersion greater than 5.0ms. Using packet dynamics measured with IPMP, determining causes of this variation in packet delay might allow TCP performance to be improved. Third, a number of bands of packet dispersion values exist at 5.8ms, 8ms, and 9ms between hops 5 and 8 that indicate some property of the network is causing these to occur.

7.6.1 Forwarding Overhead

Forwarding overhead comes in MAC-layer overheads and router capability to forward packets. As discussed in the introduction to section 7.5, 802.11b has numerous well-known performance limitations due to comparatively large MAC headers, preamble, and in-band acknowledgements. Excluding propagation delay, a data frame at 11Mbps includes an additional delay of 171μs for pre-amble and framing, and an additional delay of 116μs for the layer 2 acknowledgement frame [76]. As the length and thus propagation delay of each link is known, it is possible to account for the 802.11b MAC-layer overheads and infer the system's capability for forwarding packets for links where the packet-pair is able to queue back-to-back.

Table 7.5 shows the theoretical minimum 802.11b MAC inter-packet dispersion for the six links on the forward path. At hop 1 (MSB) the minimum packet dispersion measured is 1.579ms, which is 0.201ms more than the minimum packet dispersion possible with 802.11b. The remaining 0.201ms is difficult to precisely quantify; however, the most significant contributors to the remaining delay are likely to be processing delay in bringing the packet from the network interface into the

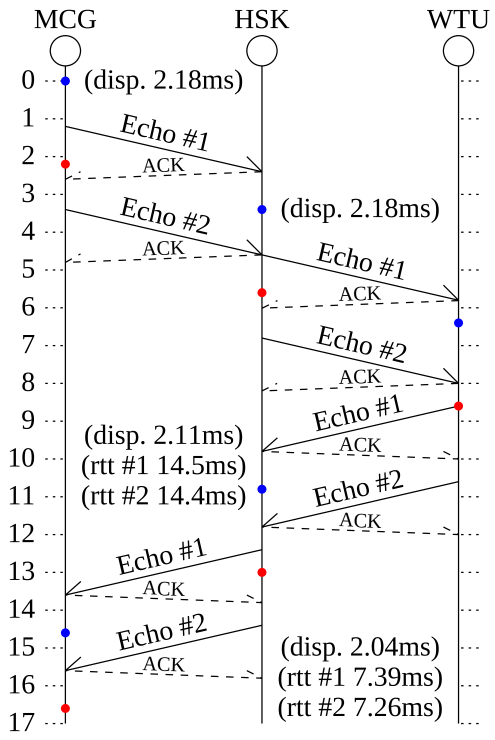


Figure 7.3: Packet-pair sequence diagram for a pair that maintains a 2ms dispersion through MCG, HSK, and WTU

IP forwarding path where the timestamp is generated, and in the 802.11b radios switching from receive to transmit and vice versa.

7.6.2 MCG to WTU Minimum Dispersion

Figure 7.2 shows that most packet pairs are forwarded through MCG to WTU and back with a dispersion of between 2ms and 2.6ms. A packet-pair through hops 4, 5, 6, 7, and 8 contains adequate information to infer the interaction the first echo packet has with the second echo packet when the packet is turned around at WTU, and is a useful place to begin before examining the causes of the variation and banding in packet dispersion. Even though the clocks at MCG, HSK, and WTU are not synchronised, it is possible to determine the round trip time between MCG and WTU, as well as HSK and WTU, using the timestamps inserted at hops 4, 8, 5, and 7. It is then possible to infer the packet sequence between hops 4 and 8 using the arrival dispersion of the packet pair at each hop.

Figure 7.3 is a packet-pair sequence diagram which shows *how* the packet-pair maintained a dispersion of approximately 2ms as it was forwarded through these hops. The sequence diagram is constructed using the round-trip and dispersion information contained in a single packet pair returned to the source host. The blue and red dots represent the timestamps the echo packets had embedded in path records at each of the hops in the path. Using the embedded timestamps, the RTT of the echo packets between MCG and WTU is measured to be approximately 14.5ms, the RTT of the packets between HSK and WTU is measured to be approximately 7.3ms, and the echo packets maintain their dispersion of approximately 2ms. The sequence diagram shows that in order for the packets to maintain this dispersion, both packets have to simultaneously queue at WTU. The diagram shows that by the time the first echo packet is able to be returned, the second echo packet is being serialised on the half-duplex wireless link. Therefore, the first echo packet has to queue at WTU until the second echo packet is deserialised.

7.6.3 PIR to MCG Dispersion

Between PIR (hop 3) and MCG (hop 4), very few packet-pairs are measured to have a dispersion value between 2.6ms and 5.0ms. This 2.4ms gap could be caused by a number of factors, including cross traffic in the form of another 1500 byte packet increasing the dispersion of the packet-pair. However, the cause of the extra dispersion is unlikely to be cross traffic, because cross traffic would cause the packet-pair to show a similar dispersion at earlier hops too.

Similarly, it is not likely to be contention for access to the link between the reply to the first echo packet and the second echo request, as for that to occur the first echo packet must be serialised at four different links ahead of the trailing packet. The minimum RTT measured in the data set for the first echo packet in a packet-pair between MCG on the forward path (hop 4) and MCG on the reverse path (hop 8) is 13.9ms. The implication of this is that a significant number of probes at PIR would be required to be dispersed by at least this amount in order for the PIR to MCG link to be under contention. Figure 7.2 shows this is not the case, as very few probes at PIR are dispersed by more than 13.9ms.

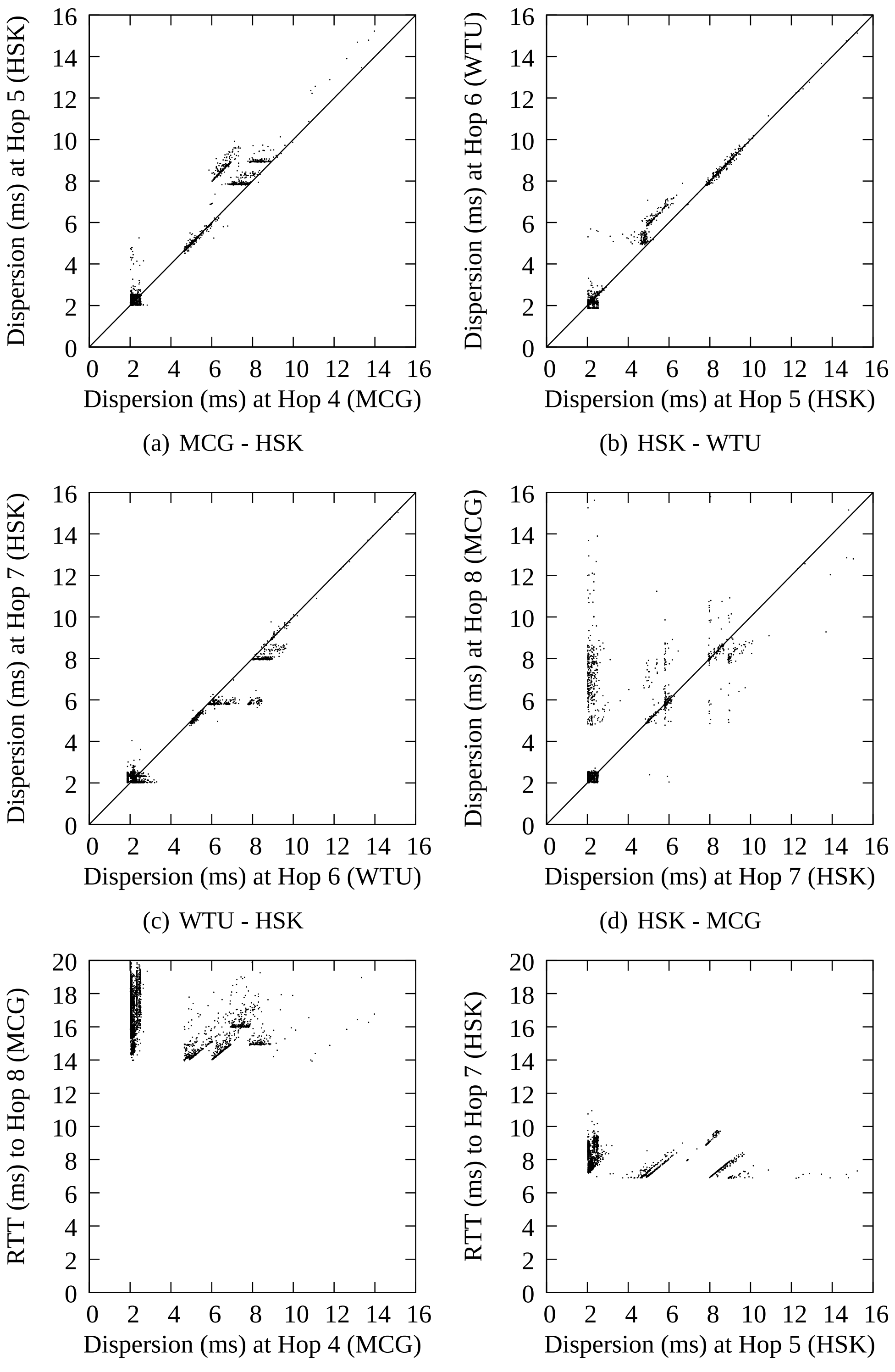
In this case the 2.4ms gap is most likely caused by the second packet in the pair being lost or corrupted and then requiring layer 2 retransmission. The link between PIR and MCG corresponds to the longest link in the path, as shown in table 7.3, increasing the probability that the signal may not be strong enough in some scenarios for the packet pair to be received back-to-back at MCG.

7.6.4 MCG to WTU Dispersion Banding

In figure 7.2, banding can be observed between HSK (hop 5) and MCG (hop 8). Between these hops, there are bands of measured packet dispersion at 5.8ms, 8ms, and 9ms, indicating some property of the network is causing this to occur. The dispersion banding weakens past hop 8 on the reverse path. As WTU is the point where the echo request packets are returned, it is possible that the first echo packet is contending for the same half-duplex wireless link as the second echo packet. This section examines if this is the case by reconstructing packet dynamics based on the information contained in the IPMP echo packets for these hops with these dispersion values.

Figure 7.4 contains six scatter plots of packet-pair dispersion values. The first four plots show the relationship between the measured dispersion of packet-pairs at adjacent hops. Dispersion at the first router for the hop is plotted on the x-axis, and dispersion at the second router is plotted on the y-axis. The last two plots show the relationship between the arrival dispersion of a packet-pair and the RTT as measured for the path measured back to that hop.

In the first four plots, points above the diagonal line through $y = x$ show the dispersion of the packet pair increases; that is, the second packet queued for longer at the previous hop than the first does. Points below the diagonal line show the dispersion decreases; that is, the second packet caught up to the first by queueing for less time at the previous hop. Diagonal clustering on the line through $y = x$ shows that the dispersion remains constant for a range of arrival dispersion values. Horizontal or vertical clustering indicates that the dispersion of the packet pair is more likely to change for a range of arrival dispersion values than for other dispersion values around the cluster. Horizontal clustering further indicates that the dispersion



(a) MCG - HSK

(b) HSK - WTU

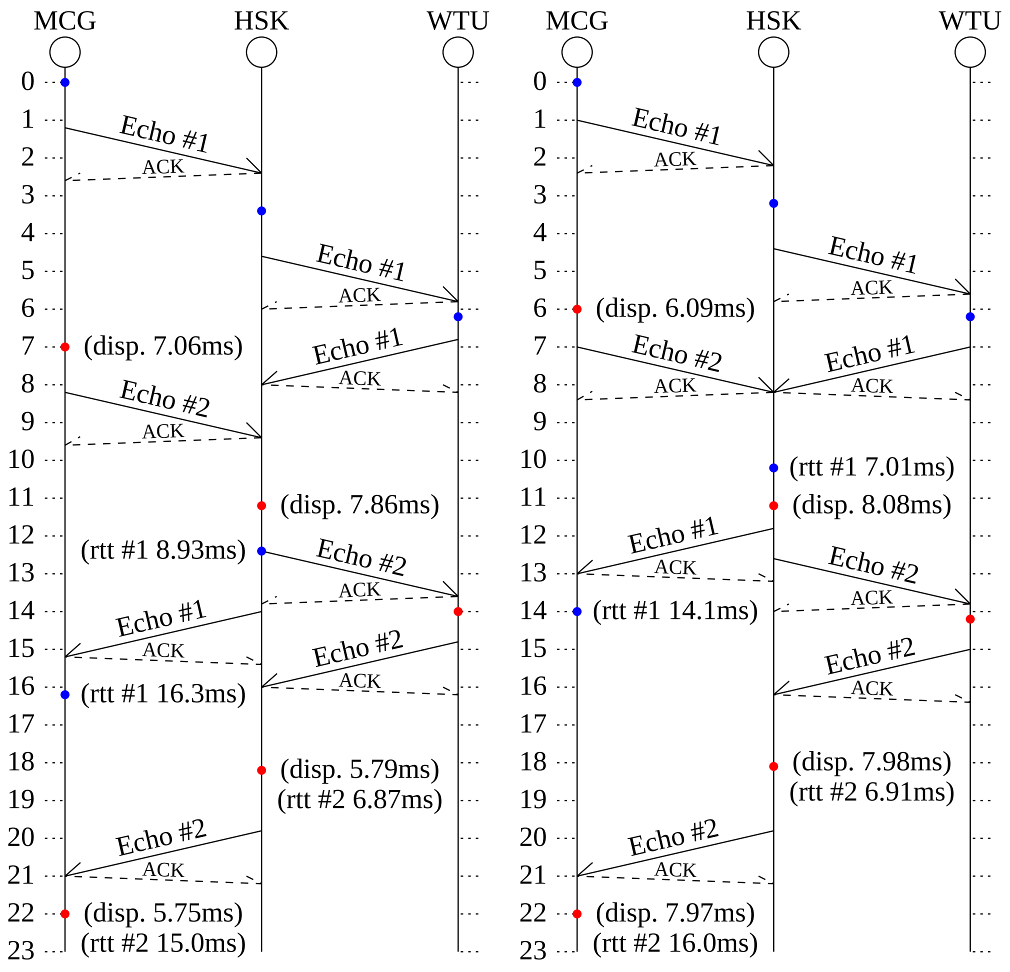
(c) WTU - HSK

(d) HSK - MCG

(e) MCG - WTU RTT

(f) HSK - WTU RTT

Figure 7.4: Packet-pair dispersion scatter plots between MCG and WTU



(a) 8ms dispersion band (b) 8-9ms dispersion

Figure 7.5: Packet-pair sequence diagrams for MCG, HSK, and WTU

of the packet pair at the next router is able to be predicted by the dispersion at the first router.

Consider the horizontal band of 8ms at HSK (hop 5) in figure 7.2. Figure 7.4(a) shows that packet-pairs arriving at MCG with a dispersion of between 7ms and 8ms are observed to have an approximate dispersion of 8ms at HSK; that is, their measured dispersion increases by up to 1ms for this hop. Figure 7.4(f) shows that the minimum RTT measured for the round trip from HSK to WTU is 7ms, which suggests that the if the first echo packet incurs minimum delay through HSK and WTU, it is likely to interfere with the second echo packet. Figure 7.5(a) is a packet-sequence diagram of a packet pair selected from the dataset because it arrives at MCG with a dispersion of 7.06ms and at HSK with a dispersion of 7.86ms. In this example, echo #1 has the same packet dynamics on the round trip to HSK as it did in figure 7.3. However, while echo #1 is being processed by HSK, echo #2 arrives from MCG to HSK. As HSK is a low-powered 486-based system which is only capable of processing a single packet at a time, the diagram suggests that echo #2 is required to queue at HSK for some time while echo #1 is processed.

Figure 7.5(b) shows a similar case, except the packet-pair is dispersed by 6.09ms at MCG. This causes the echo packets belonging to the packet-pair to arrive almost simultaneously at HSK. Figure 7.4(a) shows that packet-pairs which arrive at MCG with a dispersion between 6ms and 7ms then arrive at HSK with a dispersion increased by 2ms. The main subsequent difference between the two packet-sequence diagrams in Figure 7.5 is that in the first example the second echo packet is forwarded first, while in the second example the first packet is forwarded first. In both examples, the second echo packet is returned to MCG 22ms after the first echo packet entered MCG. As the second echo packet is delayed more in the first example than in the second, this allows the second echo packet to catch up while the first is processed; examining figure 7.5(a) once again, one of the two distinct causes of the band of packet-dispersion values at 5.8ms in figure 7.2 can be explained by this.

7.7 Summary

IPMP is well suited to per-hop measurement, particularly when all hops in the network have IPMP deployed and enabled. This chapter presented three case studies of measurements of per-hop packet dynamics in detail not possible with existing Internet protocol support for measurement. The first case study not only directly determined *where* packet loss occurred, it was able to report which direction through the hop the loss occurred. The second case study not only inferred the capacity of the path, it determined where in the path the capacity limiting link is, and which link is most likely to cause TCP to not reach capacity. The final case study re-created packet-pair interactions for hops close to a destination in a symmetrical path, and revealed the system capability to forward packets for some of these hops. The next chapter reviews recent, related work, in the area of Internet measurement protocols.

Chapter 8
Related Work

8.1 Introduction

This thesis describes a protocol designed for the per-hop measurement of Internet packet dynamics, and shows how the protocol is feasible and useful. This chapter places this thesis in the context of existing, related research, and compares the solution described in this thesis (IPMP) with these works. This chapter reviews two related pieces of work in the area of measurement-specific Internet protocols. In section 8.2, IPMP is compared with One-way Active Measurement Protocol (OWAMP), another protocol designed specifically for active measurement of the Internet. Then, in section 8.3, IPMP is compared with trajectory sampling, an approach to per-hop measurement of packet dynamics that works by passively sampling a subset of packets as they pass through a set of routers in the same administrative domain.

8.2 One-way Active Measurement Protocol

8.2.1 Overview

The only Internet protocol specific to active measurement to be standardised in the Internet Engineering Task Force (IETF) at the time of writing is OWAMP. The goals and requirements of OWAMP are defined in RFC 3763 [36] – "One-way Active Measurement Protocol Requirements." The goal of OWAMP is to provide interoperability in the one-way measurement of Internet packet dynamics. Chapter 2 discussed the use of OWAMP to measure one-way packet dynamics. This section compares OWAMP with IPMP.

8.2.2 Requirements

RFC 3763 splits OWAMP into two parts. The first half of the protocol, OWAMP-control, is used to negotiate and control OWAMP measurements. OWAMP-control is required to authenticate and schedule measurement sessions, to report measurement results, and to provide fine-grained control over when probes are sent and how they are formed. The second half of the protocol, OWAMP-test, is used to measure one-way Internet packet dynamics. The requirements specify that test traffic:

- be simple, lightweight, and easy to implement;
- should be indistinguishable from other traffic on the network so that routers are unable to prioritise its forwarding;
- should be resistant to tampering by intermediaries;
- should be able to be encapsulated in a single Asynchronous Transfer Mode (ATM) cell if possible.

8.2.3 Specification

The OWAMP specification [37], currently work in progress, meets these requirements with a TCP-based control protocol used to negotiate and control a measurement, and a UDP-based test protocol to measure one-way packet dynamics. The OWAMP-control protocol allows a pair of hosts to:

- negotiate the UDP ports, the packet size, and the destination of the test traffic;
- negotiate authentication and encryption for both the control session and test probes in order to prevent intermediary nodes from tampering with a measurement or prioritising measurement traffic ahead of other traffic;
- retrieve packet timings from an end host when the forward path is measured.

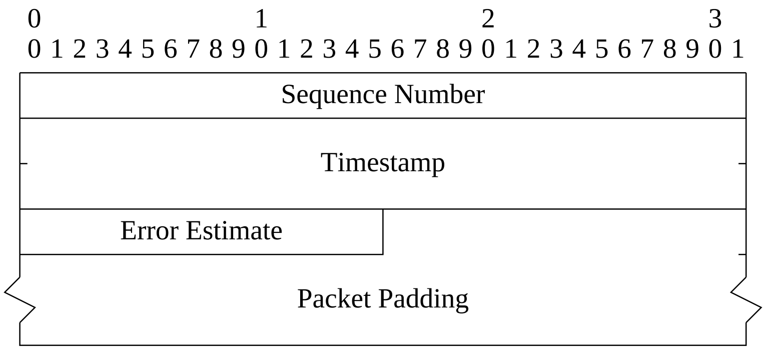


Figure 8.1: OWAMP-test probe without authentication support

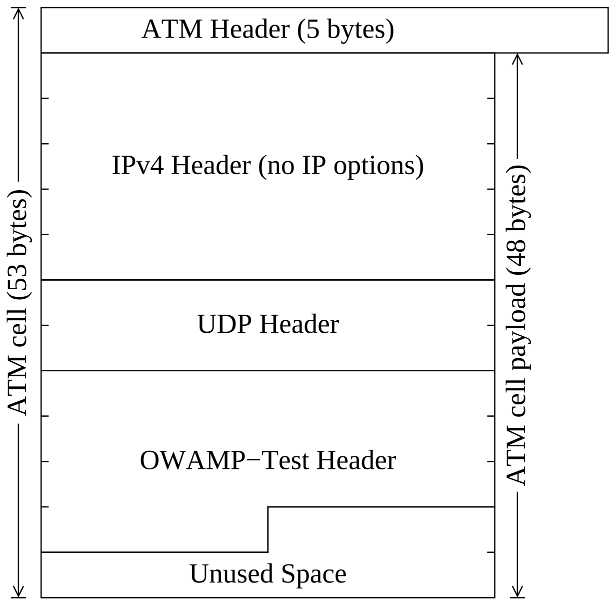


Figure 8.2: OWAMP-test probe encapsulated in a single ATM cell

The format of a OWAMP-test probe differs if the test traffic is authenticated or encrypted by the sender. Figure 8.1 shows the format of OWAMP-test traffic without support for authentication. In this context, a 14-byte header is included in each probe, and consists of a sequence number, transmit timestamp, and an estimate of the error of the timestamp.

Figure 8.2 shows how this format allows a OWAMP-test probe to fit within a single ATM cell, so that a OWAMP-test probe can avoid measuring ATM Segmentation and Reassembly (SAR) overhead, if desired. An ATM cell has a fixed size of 53 bytes, and each cell has a fixed header of 5 bytes, leaving 48 bytes for payload. Therefore, an IPv4 OWAMP-test probe can be encapsulated in a single ATM cell provided no more than 6 bytes of IP options or OWAMP packet padding is included.

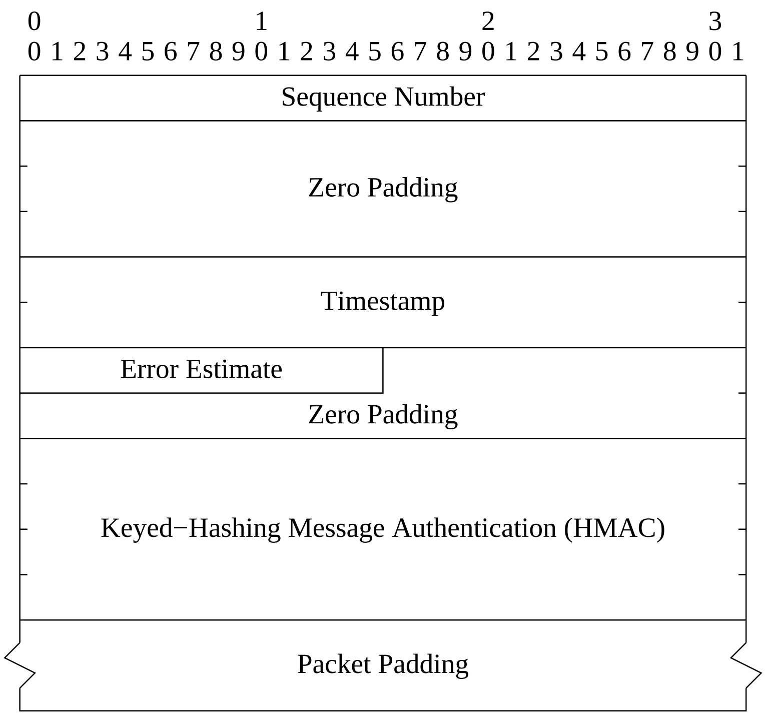


Figure 8.3: OWAMP-test probe with authentication and encryption support

Figure 8.3 shows the format of OWAMP-test traffic with support for authentication and encryption. In this context, a 48-byte header is included in each probe. In addition to the fields it has in common with the header without support for authentication, this header contains a 16-byte Keyed-Hashing for Message Authentication (HMAC) field for authentication, and 18 extra bytes of zero padding.

In authenticated mode, only the sequence number and the first block of zero padding are protected; the timestamp is sent in the clear and alterations by intermediaries will not be detected by the receiver. This mode permits a sender to pre-compute the authentication fields ahead of generating a timestamp for the packet, so that the timestamp is generated as close as possible to when the probe is sent. In encrypted mode, the packet's protection is extended to include the timestamp, error estimate, and second area of zero padding.

8.2.4 Comparison with IPMP

While both IPMP and OWAMP are protocols designed for measurement of Internet packet dynamics, they are significantly different, partly because IPMP is for measurement of per-hop Internet packet dynamics, whereas OWAMP is for one-way measurement of a path. This section compares two areas of significant difference: the OWAMP support for authentication and encryption, and the OWAMP-control protocol.

8.2.4.1 Authentication and Encryption

OWAMP contains many features designed to prevent an intermediary from interfering with OWAMP-test probes. OWAMP-test probes can run on UDP ports negotiated out-of-band between the hosts, and can be further authenticated and encrypted to prevent intermediaries from falsifying or modifying measurement probes, respectively. In contrast, IPMP echo probes are obvious so that a router can efficiently process them in the forwarding path.

It is possible for an intermediate node to modify existing path records in an IPMP echo packet, or to forward IPMP packets with a higher priority. These scenarios are possible in theory, but not likely in practice, in part because they would require a more complicated forwarding path. In the first scenario, such a system would be required to know the timestamp format used in the path record in order to correctly adjust it, or it would have to be able to intercept and modify information requests for the node.

In both cases, it is likely that cheating could be inferred through measurement with other techniques or other protocols. If it is determined that another party is maliciously modifying another system's path record, the operator of that system might seek legal redress. In addition, these modifications may result in a reduced ability for the organisation that implements them to measure, diagnose, and understand their own network.

8.2.4.2 Control Protocol

OWAMP contains a control protocol with support to negotiate, among other things, UDP ports, probe packet size, probe frequency, authentication, and encryption. This thesis does not describe a control protocol for IPMP, as it focuses on router-modifications required in order to efficiently measure per-hop Internet packet dy-

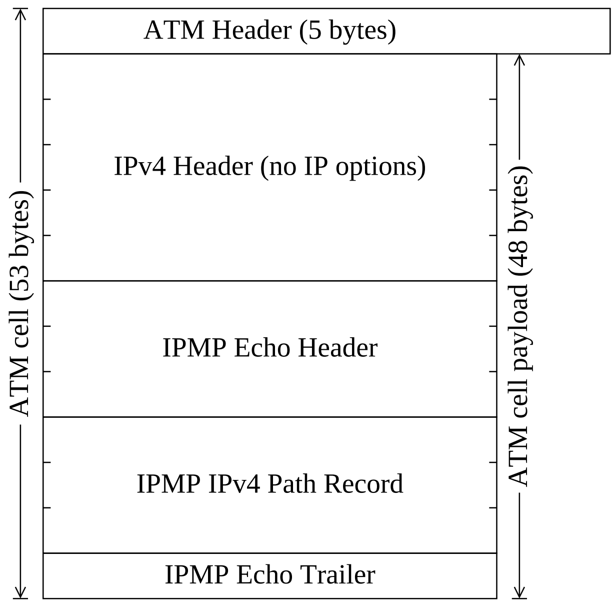


Figure 8.4: IPMP echo packet encapsulated in a single ATM cell

namics. IPMP is not as dependent on a control protocol as OWAMP, and is useful for measurement of per-hop Internet packet dynamics without one.

A control protocol could be defined for IPMP that builds on the work done with the OWAMP-control protocol. An IPMP-control protocol, like OWAMP-control, could provide for precise control over when probes are transmitted by a remote host, authentication and encryption of a control session, and the ability to retrieve information about the dynamics of each IPMP echo probe if they are sent to a third-party destination. In addition, an IPMP-control protocol may allow a source to negotiate the contents of the pre-initialised path record space to restrict which routers are permitted to embed path records.

8.2.4.3 Small Packets

One of the design goals of OWAMP is to provide a probe packet that could be encapsulated in a single ATM cell. OWAMP provides this capability with unauthenticated OWAMP-test traffic. Figure 8.4 shows that it is also possible for an IPMP echo packet to be encapsulated in a single ATM cell, provided IPv4 is used, and the IP header contains no options. In this scenario, a probe provides enough space for a single path record to be included in the packet.

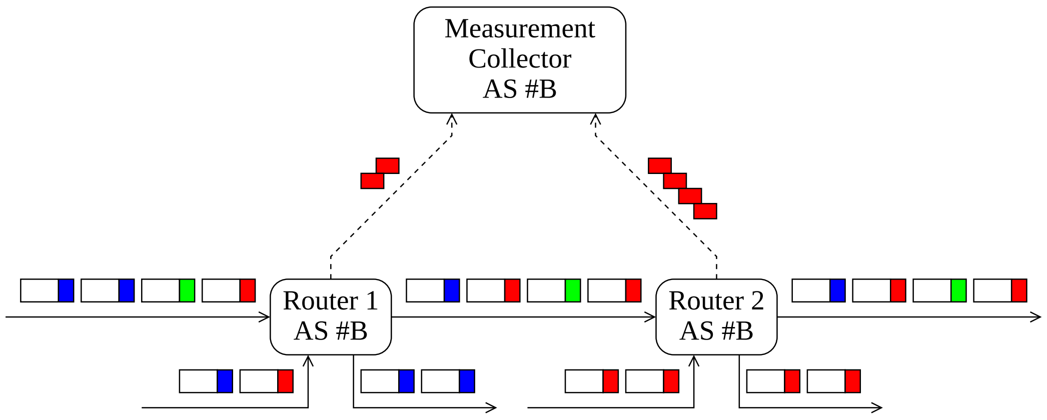


Figure 8.5: Overview of trajectory sampling

8.3 Trajectory Sampling

8.3.1 Overview

In “Trajectory Sampling for Direct Traffic Observation” [77], Duffield and Grossglauser describe an architecture for passively sampling a series of packets at each router as they are forwarded through a domain. This is accomplished by deploying an *identification hash function* on routers through the domain that selects packets based on values in the IP, TCP, and UDP headers, so that the same series of packets is selected at each router. Each router then forwards packet details to a measurement host which can then process and analyse the information. The Packet Sampling working group (PSAMP) in the IETF is currently defining an interoperable protocol for trajectory sampling.

Figure 8.5 illustrates the trajectory sampling technique. This example shows two adjacent routers in AS #B with identification hash functions that sample red packets from incoming links, and a measurement collector where the packet records are sent. In this example, the measurement collector is able to determine the trajectory of the two red packets that enter router 1. In this case, both are forwarded to router 2.

Trajectory sampling is useful in many situations, including traffic engineering, traffic measurement, and measurement of per-hop Internet packet dynamics. The last of these is compared to IPMP in the next section.

8.3.2 Comparison with IPMP

Both trajectory sampling and IPMP allow for direct measurement of per-hop Internet packet dynamics. Both techniques require the cooperation of intermediate routers. Trajectory sampling uses a passive measurement technique to sample traffic as it flows through a domain using a hash function, whereas IPMP uses information embedded in path records to determine the packet's dynamics and its trajectory through the network. This section compares trajectory sampling and IPMP in terms of complexity, security, and practicality issues.

8.3.2.1 Complexity

Trajectory sampling is more complex than measuring per-hop Internet packet dynamics with IPMP. In order to measure per-hop packet dynamics, the same identification hash function is required to be installed on each router in the network. Duffield et al. outline a possible implementation scenario involving a Digital Signal Processor (DSP) to sample the appropriate packets. In addition to this, trajectory sampling requires a language to produce identification hash functions, a protocol to load these functions into a router, and a measurement collector to receive packet trajectories.

In comparison, IPMP requires the forwarding path to be modified so that echo packets will have a path record inserted where appropriate, and some capability to reply to information requests. As described in chapter 5, the modification required to the forwarding path is simple, and a number of possible implementation strategies exist for the information protocol.

8.3.2.2 Security and Privacy

Duffield et al. define trajectory sampling as being used in a single domain, such as an AS, because numerous security and privacy issues arise with passive packet sampling. Access to such a facility is likely to be available only to operational staff inside a single domain because this facility can be used to monitor traffic from third-parties. In comparison, IPMP echo probes can not be used to directly monitor third-party traffic, so they do not present this threat.

8.3.2.3 Practicality

Trajectory sampling is useful, by design, to operators for measuring per-hop Internet packet dynamics inside a single domain. Before measurement commences, the same identification hash function must be deployed amongst all routers in the path. If the applicable routers are not known in advance, then either the path must first be discovered, or the hash function must be distributed amongst all possible routers.

In contrast, IPMP allows any network user to measure per-hop Internet packet dynamics, and allows them to be measured Internet-wide, rather than just in the local domain. No state is required to be kept by routers to allow per-hop Internet packet dynamics to be measured.

8.4 Summary

This chapter reviewed two Internet measurement protocols that were developed recently. While both protocols have different goals and motivations behind their development, contrasting with IPMP allows the design of IPMP to be further scrutinised. Because IPMP is designed to be a general-purpose measurement protocol that seeks the cooperation of routers, IPMP is designed for efficient processing by routers.

Chapter 9
Conclusions

9.1 Summary of Thesis

This thesis presents the IP Measurement Protocol (IPMP) – a protocol designed for per-hop measurement of Internet packet dynamics. An IPMP echo probe directly measures per-hop Internet packet dynamics in the context of an IP path to a destination by allowing routers on the path to embed information into the probe as it is forwarded. IPMP also provides auxiliary functions useful for Internet measurement. The two main goals of this thesis are to show that an Internet protocol designed for per-hop measurement of Internet packet dynamics, such as IPMP, is *useful*, and to show that the modification required to efficiently process IPMP in the forwarding path is *feasible*.

To accomplish the first goal, this thesis motivates the measurement of packet dynamics on an end-to-end, one-way, and per-hop basis in chapter 2. The metrics discussed – delay, loss, reordering, and capacity – are fundamental properties of an Internet path. Measurement of these properties on a per-hop basis is useful in many scenarios, and is particularly valuable in an operational context when diagnosing a poorly performing Internet path, monitoring the behaviour of a network, and in providing data on the properties of Internet paths. Measurement tools and techniques currently available for measuring per-hop Internet packet dynamics have many common limitations. These limitations include being forward-path bound, relying on static path behaviour over the course of the measurement, requiring each hop to be measured with a separate set of probes, and depending on router and firewall implementations and configurations that do not disable, rate-limit, or block the required responses.

This thesis shows how IPMP addresses these limitations by combining the measurement of the path's topology with the packet's dynamics, so that each probe can efficiently measure packet dynamics on a per-hop basis. Per-hop measurement techniques that use IPMP to measure topology, delay, loss, reordering, and capacity are more robust, reliable, convenient, and accurate than currently available measurement techniques. These properties are derived from the ability of each IPMP echo packet to carry a path record inserted by each router on an Internet path. With this feature, it is possible to directly measure an IPMP echo packet's dynamics on a per-hop basis on both the forward and reverse paths. Further, it is possible to isolate the packet dynamics contributed by each hop, because a path record contains information which records when the packet arrived at each router. This provides a more robust method for measurement of per-hop behaviour compared with techniques which probe each hop individually.

For example, packet-pair capacity estimation techniques that use IPMP do not require additional techniques to determine which of the measured dispersion modes corresponds to the underlying capacity of the path because this information is embedded in path records in the returned packets. Continuing with the capacity estimation example, IPMP also provides convenience through being a general-purpose measurement protocol, so explicit support for a packet-pair capacity estimation technique at a target host is not required if IPMP is implemented.

The usefulness and convenience of IPMP was demonstrated by measurement of per-hop packet behaviours on CRCnet with IPMP deployed on each router in the network. Using IPMP packet-pairs, it was possible to determine the capacity limiting link of a path, and the capability of some routers in the path to forward packets by causing the two packets in the pair to contend for the same resources. In addition, through per-hop measurement of variation in observed packet dispersion, it was possible to infer where engineering effort would be best directed if improving TCP performance of a particular path was important. For another CRCnet path, it was possible to determine *where* packets were being lost and to correlate this information to periodic tasks which use the network.

To accomplish the second goal of the thesis, which is to show that IPMP is feasible to implement, this thesis presents three different implementations of IPMP and measurements of the performance of two of these implementations. The first two implementations were constructed by modifying the source code to the kernel for BSD and Linux kernels. Measurement of the BSD implementation shows that the forwarding path modification adds little extra processing delay compared to simply forwarding an unmodified probe. In addition, other host factors were measured to have a larger impact on forwarding performance than IPMP, indicating that IPMP does not have a significant effect on forwarding performance.

The third implementation, written in VHDL, modified the forwarding path of a customised non-blocking cross-bar Ethernet switch. This implementation processed an echo probe in a fundamentally different way to the software implementation and showed that the required modifications can be made to an IPMP echo probe without significantly delaying the packet. The increase in forwarding delay was measured at between 60ns and 120ns; the accuracy of this measurement is limited by the passive measurement hardware used in the experiment.

This thesis also shows how IPMP can be implemented in a hardware-optimised forwarding path, like that of many routers, so that an IPMP echo packet experiences the same dynamics as any other packet, and does not provide a vector for DoS. This is accomplished by processing an echo packet in parallel with other IP forwarding tasks.

9.2 Future Work

The Internet relies on cooperation and interoperability. In order for IPMP to become widely deployed, it must first be formally standardised by a standards body such as the IETF. In conjunction with this, IPMP must also be implemented by router and operating systems vendors.

This thesis has outlined a series of measurement techniques to measure fundamental properties of Internet links in isolation from each other. Future work might examine the use of IPMP in more complicated measurement tools and techniques such as for user-space evaluation of the likely TCP performance of a path similar to that inferred by TReno [78]. In addition, the development of a tool suitable for user-level Internet path diagnosis of jitter, loss, and reordering like that described by Mahajan et al. [1] that uses IPMP could be developed.

If IPMP were widely deployed in the Internet, then large-scale Internet modelling and analysis would be more feasible and lead towards more accurate models of Internet behaviours. Examples of large scale Internet measurement include reverse-engineering the Internet [22] as proposed by Spring, Wetherall, and Anderson. The data captured would be useful for developing more accurate models of per-hop Internet behaviour [3] and would be useful in simulating the behaviour of new protocols in a more realistic manner.

IPMP is both feasible and useful and has the potential to enhance Internet measurement, modelling, and development.

Appendix A

IPMP Internet Draft

This chapter contains a specification of IPMP in Internet Draft form.

```
Individual Submission                                        A. McGregor
Internet Draft                                                 M. Luckie
Intended status: Experimental                         Waikato University
Expiration Date: December 7, 2006                           June 5, 2006
```

IP Measurement Protocol (IPMP)
draft-mcgregor-ipmp-05.txt

Status of this Memo

Copyright Notice

Abstract

The practice and need for active measurement of networks is well established. Current tools are not well suited to this task, primarily because the protocols which they employ have not been designed for measurement of the modern Internet. The IP Measurement Protocol (IPMP) is based on packet-probes. It is designed to allow routers to participate in measurements by inserting path information as the probe passes between a pair of hosts. IPMP is tightly constrained and easy to implement.

Table of Contents

1. Introduction

The practice and need for active measurement of networks is well established. Current tools are not well suited to this task, primarily because the protocols which they employ have not been designed for measurement of the modern Internet.

For example, the Internet Control Message Protocol (ICMP) is widely used for measurements (in part because there is not a better alternative) despite its well known limitations for this task. These limitations include it being treated differently to other IP protocols at routers and hosts, because it is difficult to efficiently generate appropriate response packets. In addition, ICMP has been implicated in denial of service attacks. As a consequence, some Internet Service Providers (ISPs) disable ICMP even though this potentially causes poor performance and does not comply with [RFC1009].

Current packet probing techniques are not suited to measuring packet delay at the router level, for several reasons. Routers often make bad measurement targets because they are optimised for the relatively simple task of forwarding packets. Because they are an opportunity for denial of service attacks, routers may process tasks that are resource intensive at low priority or not at all. Some measurement techniques construct measurement traffic that can be difficult to efficiently detect and respond to amongst other network traffic. This type of measurement traffic precludes measuring to a router and makes the task of identifying where delays occur in the network difficult.

IPMP addresses all of these issues by providing a measurement protocol that is tightly constrained [RFC1925], efficient, and easy to implement. The protocol has been designed so measurement packets can be processed with approximately the same level of computation as needed for IP packet forwarding.

The protocol operates as an echo protocol allowing packet loss, one-way path length, round-trip time, and one-way delay measurements to be taken. The protocol also allows a measurement host to resolve router aliases from the interface IP addresses it collects in the echo exchange by exchanging information packets after the measurement has completed.

Packets are generated by a measurement host and returned by an echoing router or host. An echoing router or host is known as an echoing system in this memo. The translation of router timestamps to real-time timestamps is supported through a separate information request and reply exchange between the measurement system and systems that insert timestamps into the echo request or reply.

2. Terminology and Definitions

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described [RFC2119] and indicate requirement levels for compliant IPMP implementations.

3. Packet Formats

3.1 IPMP Echo Request and Echo Reply

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |    Version    |     Type      | Faux IP Proto |   Reserved    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Identifier           |        Sequence Number        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |       Faux Source Port        |     Faux Destination Port     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

    |                         Path Record(s)                        |
    :                             .....                             :
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |      Path Record Pointer      |           Checksum            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version = 1

Type

```
        0 1 2 3 4 5 6 7
       +-+-+-+-+-+-+-+-+
       |E|  Res  |S|I|R|
       +-+-+-+-+-+-+-+-+
```

E = echo packet
I = information packet (see Section 3.3)
R = request packet = 1 / response packet = 0
S = singleton packet, used for one-way probes, not echoed

Faux IP Proto, Faux Source Port, Faux Destination Port

The Faux IP Proto field mimics the IP header's protocol field. The Faux Source and Destination Port fields mimic the TCP or UDP source and destination port fields (as appropriate). These fields allow an IPMP echo packet to be queued or filtered based on a quintuple of values when combined with the IP source and destination addresses. Intermediate routers schedule an echo packet with these fields if they implement a packet scheduling discipline that is not first-in first-out (FIFO).

When an echo packet is received at the destination, the Faux Source and Destination Ports MUST be swapped to reflect the way that TCP and UDP ports are inverted in reply packets.

Identifier, Sequence Number

The echo request packet should contain enough information to match an echo response packet. The identifier SHOULD be set to the lower 16 bits of the process identifier responsible for sending the packet, and the sequence number SHOULD increment for each echo request packet sent to a unique IP address with a particular identifier value.

Firewalls that keep state SHOULD use the Identifier field - combined with the source and destination IP addresses and IP protocol number (not Faux IP Proto) - to hash a packet in order to decide which echo packets to allow past.

Path Record(s)

A source host is responsible for pre-allocating space for path records to be inserted by intermediate systems. The path record format is shown in Section 3.2. Other systems MUST NOT increase the length of the space reserved for path records if the pre-allocated space is consumed.

The first path record begins at the first byte after the echo header. All path records are contiguous.

The pre-allocated space MUST be initialised to zero, except for the Time to Live (TTL) or Hop Limit (HLIM) fields.
A source host MAY restrict the insertion of path records to specific TTL / HLIM values by pre-initialising the TTL / HLIM fields of the path record space to the TTL / HLIM values of interest. The TTL / HLIM fields MUST be laid out in descending order, except for the wildcard value of 255, e.g. [254, 250, 249, 255, 255].

Path Record Pointer

The position of the next available path record location, in bytes from the beginning of the IPMP header, if there is space. If an intermediary inserts a path record, it MUST increment the path pointer by the size of the path record inserted.

Checksum

The checksum is the 16-bit one’s complement of the one’s complement sum of the IPMP message starting with the IPMP version field and ending with the end of the IPMP packet. For computing the checksum, the checksum is set to zero.

3.2 Path Record Format

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      TTL      | Flags | FlowC |          Timestamp            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     Timestamp <continued>                     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      IP version 4 (IPv4) Address of Receiving Interface       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     HLIM      | Flags | FlowC |          Timestamp            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Timestamp <continued>                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |      IP version 6 (IPv6) Address of Receiving Interface       |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TTL / HLIM

A host or router MUST include in a path record the value of the TTL or HLIM field in the packet’s IP header after it has been decremented as part of the packet forwarding process. This allows an end-host to identify gaps in the network where a path record was not inserted.

Flags

```
 0 1 2 3
+-+-+-+-+
|S| Res |
+-+-+-+-+
```

S = Set bit (S-bit). Set to 1 when the path record space has been used, and initialised to zero by the source host to indicate that the path record has not been set.

FlowC

A host or router MAY include a flow counter which corresponds to the flow counter value for an IPMP echo flow. An IPMP echo flow is defined as a series of echo packets with the same source and destination IP addresses and IPMP echo identifier. A flow counter is scoped to the receiving interface.

Timestamp

A host or router MAY include the time at which the interface completed receiving the packet. If the timestamp is not set in the path record, the default value is zero. The timestamp is allocated 48 bits, although the host or router does not have to use them. The timestamp, if included, SHOULD represent the time that the last bit of the packet was received.

IPv4 / IPv6 Address of Receiving Interface

When an echo request packet is received by the destination host, the destination SHOULD set this field to the destination address of the echo request so that the source host can determine which path record (if any) corresponds to the echo packet arriving at the destination. Similarly, when an echo reply packet is received by the source host, the source SHOULD set this field to the destination address set in the IP header.

Otherwise, this field contains the address of interface at which the echo packet was received. If the receiving interface has multiple addresses to choose from, the router should select one with the same scope as the echo packet’s destination. A router SHOULD NOT use an anycast, link-local, or multicast address in this field, because these addresses either do not uniquely identify the interface or are not globally-routable, preventing the information exchange from taking place.

3.3 IPMP Information Request and Information Reply

Information Request

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |      Type     |            Checksum           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               :
:            (optional) Reported Times Of Interest              :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Information Reply

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |      Type     |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |   Accuracy    |   Performance Data Pointer    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   IPMP Processing Overhead                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             (optional) Real-time Reference Points             |
:                             .....                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  (optional) Performance Data                  |
:                             .....                             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version, Type, Checksum, Identifier, Sequence Number

See Section 3.1

Accuracy

The number of valid bits in the fractional portion of each real-time timestamp in each real-time reference point.

Performance Data Pointer

The position, of the performance data field, in bytes from the beginning of the IPMP packet, if it exists. If there is no performance data field, the default is 0.

IPMP Processing Overhead

The maximum difference between the time taken to process and forward an IPMP packet and the time taken to forward an IP packet with the same characteristics. The echo system may use the values supplied in Faux IP Proto, Faux Source Port, and Faux Destination Port if it implements a queueing discipline that is not FIFO. If the overhead is unknown, then the value recorded is MAX_TIME, i.e., 0xffffffff.

Reported Times of Interest

A measurement system MAY request that the end system constrain the real-time reference points to cover a particular timestamp that it received. In this case, the timestamping system SHOULD return a real-time reference point for just this timestamp or a pair of reference points, one before and one after, the timestamp. In other cases the timestamping system MAY return an arbitrary number of reference points bounded by the size of the packet it can send.

Real-time Reference Points

A real-time reference point (see Section 3.4) gives the relationship between real-time and the timestamp that would have been placed in a path record by the interface at that time. If any reference points are included, there must be at least two. Within the bounds of the overall size of the packet, any number of reference points may be included.

Performance Data

The Performance Data field allows arbitrary information from the MIB of the system or the interface to optionally be included in the Information Reply. It is formatted as a VarBindList from the SNMPv2-PDU defined in [RFC3416]. In this context, ObjectSyntax is the only valid choice within VarBind.

      For example:

```
          IPMP-PERFORMANCE-DATA DEFINITIONS ::= BEGIN

          IMPORTS
              ObjectName, ObjectSyntax,
                  FROM SNMPv2-SMI;

          max-bindings
              INTEGER ::= 2147483647

          -- IPMP simplified list element
          IPMPVarBind ::=
              SEQUENCE {
                  name
                      ObjectName,
                  value
                      ObjectSyntax
              }

          -- variable-binding list
          VarBindList ::=
              SEQUENCE (SIZE (0..max-bindings)) OF
                  IPMPVarBind

          END
```

3.4 Real-time Reference Point Format

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                               |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
   |                       Local Clock Time                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Real Time                           |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Estimated Error                        |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Local Clock Time

     A 48-bit timestamp in the same format as would be used in a path
     record, which correlates a free-running local clock timestamp
     with real time.

Real Time

A Network Time Protocol (NTP) formatted timestamp representing the real time corresponding with the local clock time.

Estimated Error

An NTP formatted timestamp estimating the error bounds of the real-time timestamp.

4. Processing of IPMP Packets

4.1 Source Host Echo Processing

A source host constructs an IPMP request, encapsulates it in an IP datagram and the appropriate link layer protocol and sends it into the network. Performance information is gleaned from the presence or absence of a reply, the delay between the request and the reply, the path record(s) if present, and the presence of errors, if they occur.

When constructing an echo request, a source MUST set all words from the end of the echo header to the beginning of the echo trailer (the space allocated for path records) to zero, except for the bytes corresponding to the TTL/HLIM field for path records.

The source host MAY include a path record when it sends an echo request, and a path record when it receives an echo reply.

IPMP echo packets encapsulated in IPv4 SHOULD be sent with the Don’t Fragment (DF) bit set, because the entire echo packet is required in order to insert a path record. A source host SHOULD assign a unique value to the IPv4 identifer field in case the packet is fragmented despite the DF bit being set.

4.2 Destination Host Echo Processing

The IPMP echo request and echo reply packet formats are designed to make processing at the destination host simple and efficient.

On receipt of the IPMP echo request, the destination constructs the echo reply from the echo request by:

1. exchanging the source and destination IP addresses;

2. exchanging the faux source and destination ports;

3. setting the reply bit in the IPMP type field and incrementally updating the IPMP checksum for this alteration;

4. decrementing the TTL field in the IP header and incrementally updating the IP checksum;

5. optionally inserting a path record as described in Section 4.4; and

6. scheduling the packet for forwarding, taking into account the faux IP protocol, faux source port, and faux destination port, if appropriate.

The destination host does not:

- validate the value of the IPMP checksum present in the header against the contents of the packet;

- recalculate the IPMP checksum from scratch before transmitting the packet;

Processing information request packets requires more resources than processing an echo request. Direct measurements are not made with an information exchange. Consequently, an implementer may choose to process information request packets off the interface card and/or at low priority.

4.3 Forwarding System Echo Processing

A forwarding system does not need to be IPMP aware. In the simplest case, an IPMP packet is forwarded like any other IP packet.

If the forwarding system schedules packets based on the value of any combination of the IP protocol field and the TCP or UDP source and destination ports, then the forwarding system SHOULD use the faux fields in the IPMP header for this purpose in place of the IP, TCP or UDP fields.

A forwarding system MAY include a path record, as described in the following section.

4.4 Path Record Insertion

Inclusion of path records is optional. A path record MAY be inserted by forwarding systems on the forward and reverse paths, and by the source and destination hosts.

A forwarding system SHOULD NOT insert a path record if it cannot modify the packet in the same processing stream that any other packet would take. This is to avoid the measurement path being significantly different than that taken by a regular packet, and to reduce opportunities for denial of service attacks.

A system that can insert a path record MUST ensure the complete IP packet is available. If the echo packet is encapsulated in IPv4, this means that the IP more fragments (MF) bit must not be set, and the IP offset field must be zero. If the packet is encapsulated in IPv6, this means that the packet is not encapsulated in an IPv6 fragmentation header.

A system that can insert a path record MUST check for sufficient space in the echo packet based on the size of the path record inserted. An IPv4 path record requires 12 bytes, while the IPv6 path record requires 24 bytes.

If a path record is inserted, it MUST be inserted in the first path record space where the S-bit is not set. If the system processes the packet in a store-and-forward manner, the first byte of this path record will be indicated by the path record pointer.

A system that can insert a path record MUST compare the TTL/HLIM field in the pre-initialised path record with the TTL/HLIM field in the IP header after it has been decremented as part of the normal IP forwarding process. A system MUST NOT insert a path record unless the path record TTL is greater than or equal to the IP TTL/HLIM.

A system that inserts a path record MUST update the path record pointer as appropriate, MUST incrementally update the IPMP checksum field as described in [RFC1624], and MUST also set the S-bit in the flags field of the path record inserted. A system that inserts a path record MUST NOT increase the length of an echo packet if there is no reserved space left.

A system that adds a path record MAY include a timestamp in the path record. If it does not include a timestamp, the timestamp field in the path record is left unaltered, i.e., remains zero.

A system that adds a path record MAY include a flow counter in the path record. If it does not include a timestamp, the FlowC field in the path record is left unaltered, i.e., remains zero. If a flow counter is included, the flow counter MUST increment by one with each echo packet belonging to the same flow. An IPMP echo flow entry MUST have interface scope, and is defined by the source and destination IP addresses and IPMP identifier.

4.5 Destination Host Information Request Processing

When a destination host replies to an information request, it MUST do so using a constant source address. This provides a mechanism to resolve router aliases.

A destination host MAY include information useful for measurement in the performance data section in an information reply packet. This field is described in Section 3.3. Useful data to include in this field may include bgpLocalAs from the BGP4-MIB [RFC1657], and ifType, ifMtu, and ifSpeed from the IF-MIB [RFC2863].

4.6 Denial of service attacks

Because IPMP echo request packets are processed with approximately the same effort as forwarding an IP packet, they do not introduce any new denial of service opportunities.

IPMP information request packets may require more processing and may be used as the basis of a denial of service attack in the same way as any information request can be used on a router or host. Because information request packets are not used to make measurements, an implementer may implement protection against denial of service attacks made with these packets in the same way as other information requests. This might involve processing IPMP information requests at a low priority, or regulating the maximum flow of packets.

5. Discussion

5.1 Checksums

An IPMP checksum MUST be calculated by the source host when the echo request packet is created. It is incrementally updated by forwarding systems that insert a path record. The checksum is not checked by a forwarding system or the destination. Errors that occur between the source host and the destination host on the forward and reverse paths are detected when the echo reply is received at the source host.

5.2 Real-time Timestamps

32 bit real-time timestamp fields are coded following the conventions described in [RFC1305]. Summarising from [RFC1305]:

> In conformance with standard Internet practice, timestamps are specified as integer or fixed-point quantities, with bits numbered in big-endian fashion from 0 starting at the left, or high-order, position. All quantities are unsigned and may occupy the full field width with an implied 0 preceding bit 0.

Timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 1 January 1900 00:00. The integer part is in the first 32 bits and the fraction part in the last 32 bits. In the fraction part, the nonsignificant low order can be set to 0.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Seconds                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Seconds Fraction (0-padded)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This format allows convenient multiple-precision arithmetic and conversion to UDP/TIME representation (seconds), but does complicate the conversion to ICMP timestamp message representation, which is in milliseconds. The most future time that can be represented is 4,294,967 (some time in the year 2036) with a precision of about 200 picoseconds, which should be adequate for even the most demanding measurements. RFC 2030 (SNTP) contains a proposal for extending timestamps beyond the year 2036.

5.3 Inferred Real Time

The real time of a timestamp in a path record can be inferred when a system provides an IPMP Information Reply with at least one Real-Time Reference Point earlier, and one later, than the timestamp. For the purpose of this inference, the drift of the clock is assumed to be linear. Linear interpolation is used between the two nearest Real-time Reference Points, where one is greater than, and one is less than, the timestamp.

The timestamp in the path record structure may be of any format, as discussed in Section 3.2; potentially, the timestamp can wrap over the course of a series of measurements. It is the responsibility of the measurement host to send information requests to the timestamping systems sufficiently frequently to avoid information loss. The correct frequency can be estimated from an information reply.

5.4 Minimum Implementations

5.4.1 Echoing System

The simplest echoing system implementation returns the IPMP echo request without a path record. As described in Section 4.2, this only requires that the IP source and destination addresses be exchanged, the R bit in the IPMP options field to be set, and the packet scheduled for forwarding. Because of the format of the IPMP echo request and echo reply packets, this can be implemented with a very small number of instructions. A system that does not insert path records does not need to process IPMP Information Requests.

Systems which provide even the minimum level of implementation will allow a number of measurements to be made that are not currently possible, specifically if they are routers that currently process ICMP at a low priority to avoid DOS attacks.

5.4.2 Forwarding System

Forwarding systems do not need to be IPMP aware.

A forwarding system that is IPMP aware MAY include path records with only the Forwarding IP Address set. This requires writing the address to the packet and updating the checksum and Path Pointer in the packet as described in Section 4.4. In this case the forwarding system does not need to process IPMP Information Request packets.

6. Security Considerations

An IPMP echo exchange reveals the path that may be taken between two hosts. Some ISPs may feel that their security is weakened if attackers know of an address in the core of their network, and may choose to enable path record insertion for selected IP addresses.

IPMP can be abused for denial of service attacks disguised as legitimate measurement activity, although we feel that the potential impact of IPMP denial of service attacks has been minimised through not providing a vector for denial of service that is not available with other protocols.

7. Acknowledgements

The comments of Randy Presuhn are appreciated. Maureen C. Curran provides editorial assistance on the IPMP Internet drafts series. James Spooner and Matt Jervis assisted with a VHDL implementation and provided advice on how to make IPMP more hardware-friendly.

8. References

8.1 Normative References

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and analysis", RFC 1305, March 1992.

[RFC1624] Rijsinghani, A., editor. "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.

[RFC2030] Mills, D. "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 2030, October 1996.

[RFC2119] Bradner, S. "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[RFC3416] Presuhn, R., editor. "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMPv2)", RFC 3416, STD 62, December 2002.

8.2 Informative References

[RFC1009] Braden, R., and Postel, J. "Requirements for Internet Gateways", STD 4, RFC 1009, USC/Information Sciences Institute, June 1987.

[RFC1657] Willis, S., Burruss, J., and Chu, J., editor. "Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2", RFC 1657, July 1994.

[RFC1925] Callon, R., editor. "The Twelve Networking Truths", RFC 1925, April 1996.

[RFC2863] McCloghrie, K., and Kastenholz, F. "The Interfaces Group MIB", RFC 2863, June 2000.

9. Authors’ Addresses

Anthony J. (Tony) McGregor
Department of Computer Science
Waikato University
Private Bag 3105
Hamilton
New Zealand
Phone: +64 7 838 4651
EMail: tonym@cs.waikato.ac.nz

Matthew J. Luckie
Department of Computer Science
Waikato University
Private Bag 3105
Hamilton
New Zealand
EMail: mjl@luckie.org.nz

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

Appendix B

VHDL code for IPMP Forwarding

This appendix provides a listing of VHDL code used to implement the path record insertion process when forwarding an IPMP echo packet. The implementation is described in section 5.4.

```
-------------------------------------------------------------------------
-- This file is part of the NBCS project undertaken by the WAND Network
-- research group, Dept of Computer Science, The University of Waikato.
--
-- (C) 2003-2006 The University of Waikato
--  All rights reserved
--
-- Authors:
--      James Spooner  < jbs3@cs.waikato.ac.nz >
--      Matthew Jervis < mgj3@cs.waikato.ac.nz >
--      Matthew Luckie < mjl@wand.net.nz >
--
-------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ipmp_8 is
  port (

    -- Options bus interface
    reset_n : in std_logic;
    clk     : in std_logic;

    data_in   : in std_logic_vector(8 downto 0);
    data_in_v : in std_logic;
    cnt_in    : in std_logic_vector(11 downto 0);

    data_out   : out std_logic_vector(8 downto 0);
    data_out_v : out std_logic;
    cnt_out    : out std_logic_vector(11 downto 0);

    -- Additional signals for IPMP Option
    time_in : in std_logic_vector (47 downto 0);
    ip_in   : in std_logic_vector(31 downto 0)

    );

end ipmp_8;

architecture rtl of ipmp_8 is

  type IPMP_STATE is (NO_PACKET, PKT_WAIT, IP4, IP4_IPMP, PR_INS,
                      ECHO_TRAILER);

  signal currstate : IPMP_STATE;
  signal nextstate : IPMP_STATE;

  signal ip4_addr : std_logic_vector(31 downto 0);

  signal data_h : std_logic_vector(8 downto 0);
  signal data_l : std_logic_vector(8 downto 0);
  signal data_v : std_logic;

  signal cnt : std_logic_vector(11 downto 0);

  signal length_i : std_logic_vector(15 downto 0);
  signal length   : std_logic_vector(15 downto 0);
  signal length_l : std_logic;

  signal pathp_i : std_logic_vector(15 downto 0);
  signal pathp   : std_logic_vector(15 downto 0);
  signal pathp_l : std_logic;

  signal ttl_i : std_logic_vector(7 downto 0);
  signal ttl   : std_logic_vector(7 downto 0);
  signal ttl_l : std_logic;

  signal plst_i : std_logic_vector(5 downto 0);
  signal plst   : std_logic_vector(5 downto 0);
  signal plst_l : std_logic;

  signal carry      : std_logic_vector(31 downto 0);
  signal carry_wrap : std_logic_vector(15 downto 0);

  signal csum_i : std_logic_vector(31 downto 0);
  signal csum   : std_logic_vector(31 downto 0);
  signal csum_l : std_logic;

  signal abus : std_logic_vector(15 downto 0);
  signal bbus : std_logic_vector(15 downto 0);

  signal timestamp : std_logic_vector(47 downto 0);

  signal advance : std_logic;

  signal data      : std_logic_vector(15 downto 0);
  signal data_next : std_logic_vector(15 downto 0);

  -- Aliases
  alias eth_type    : std_logic_vector(15 downto 0) is data;
  alias ip4_v       : std_logic_vector(3 downto 0) is data(15 downto 12);
  alias ip4_hl      : std_logic_vector(3 downto 0) is data(11 downto 8);
  alias ip4_proto   : std_logic_vector(7 downto 0) is data(15 downto 8);
  alias ip4_offset  : std_logic_vector(12 downto 0) is data(12 downto 0);
  alias ip4_length  : std_logic_vector(15 downto 0) is data;
  alias ip4_ttl     : std_logic_vector(7 downto 0) is data(15 downto 8);
  alias ipmp_ver    : std_logic_vector(7 downto 0) is data(15 downto 8);
  alias ipmp_type   : std_logic_vector(7 downto 0) is data(7 downto 0);
  alias ipmp_pathp  : std_logic_vector(15 downto 0) is data;
  alias ipmp_chksum : std_logic_vector(15 downto 0) is data;
  alias pr_ttl      : std_logic_vector(7 downto 0) is data(15 downto 8);

  -- Constants

  constant ETH_TYPE_IP4      : std_logic_vector(15 downto 0) := X"0800";
  constant IP_VERSION_FOUR   : std_logic_vector(3 downto 0)  := X"4";
  constant IP_PROTO_IPMP     : std_logic_vector(7 downto 0)  := X"A9";
  constant IPMP_VERSION_ONE  : std_logic_vector(7 downto 0)  := X"01";
  constant IP_OFFSET_ZERO    : std_logic_vector(12 downto 0)
                                                   := B"0000000000000";

  -- Structure Lengths

  constant ETH_HDR_LEN       : integer := 22;
  constant IPMP_PR_LEN       : integer := 12;
  constant IPMP_ECHO_LEN     : integer := 16;
  constant ECHO_HEADER_LEN   : integer := 12;
  constant ECHO_TRAILER_LEN  : integer := 4;

  -- Packet Offsets

  constant ETH_TYPE_O        : integer := 21;
  constant IP4_VHL_O         : integer := ETH_HDR_LEN + 1;
  constant IP4_LENGTH_O      : integer := ETH_HDR_LEN + 3;
  constant IP4_OFFSET_O      : integer := ETH_HDR_LEN + 7;
  constant IP4_TTL_O         : integer := ETH_HDR_LEN + 9;
  constant IP4_PROTO_O       : integer := ETH_HDR_LEN + 10;
  constant IPMP_VER_O        : integer := 1;
  constant IPMP_TYPE_O       : integer := 2;
  constant IPMP_PATHP_O      : integer := 13;
  constant IPMP_CKSUM_O      : integer := 15;

begin

  data    <= data_h(7 downto 0) & data_l(7 downto 0);
  advance <= data_in_v;

  ATLATCH : process (clk, reset_n)
  begin
    if reset_n = ’0’ then
      timestamp   <= (others => ’0’);
      ip4_addr    <= (others => ’0’);
    elsif rising_edge(clk) then
      -- Latch IP addresses into registers
      -- Latch timestamp into register until first byte
      if cnt = 0 then
        timestamp <= time_in;
        ip4_addr  <= ip_in (31 downto 0);
      end if;
    end if;
  end process;

  --
  --    Data (input) Latch
  --

  DLATCH : process( clk, reset_n )
  begin
    if reset_n = ’0’ then
      data_l   <= (others => ’0’);
      data_h   <= (others => ’0’);
      data_v   <= ’0’;
      cnt      <= (others => ’0’);
    elsif rising_edge(clk) then
      data_v   <= data_in_v;
      if advance = ’1’ then
        data_l <= data_in;
        data_h <= data_l(8) & data_next(7 downto 0);
        cnt    <= cnt_in;
      end if;
    end if;
  end process;

  cnt_out <= cnt;

  CKSUM : process (clk, reset_n)
  begin
    if reset_n = ’0’ then
      csum     <= (others => ’0’);
    elsif rising_edge(clk) then
      if advance = ’1’ then
        if cnt = "000000000000" then
          csum <= (others => ’0’);
        elsif csum_l = ’1’ then
          csum <= csum_i;
        end if;
      end if;
    end if;
  end process;


  --
  --    IPMP Packet Detector
  --

  carry_wrap <= carry(15 downto 0) + carry(31 downto 16);

  PDETECT : process ( data, cnt, timestamp, data_l, carry,
                      pathp, ip4_addr, currstate, ttl,
                      data_v, data_h, data_next, plst,
                      csum_i, csum, length, abus, bbus)
  begin

    carry <= (others => ’0’);
    csum_i <= (csum + bbus - abus);
```

```
data_next <= data_h(7 downto 0) & data_l(7 downto 0);

data_out_v <= data_v;
data_out   <= data_h(8) & data_next(15 downto 8);

nextstate <= PKT_WAIT;

-- Latch indicators for various values we need to remember
ttl_l <= ’0’;
ttl_i <= (others => ’0’);

pathp_l <= ’0’;
pathp_i <= (others => ’0’);

length_l <= ’0’;
length_i <= (others => ’0’);

plst_l <= ’0’;
plst_i <= "000000";

csum_l <= ’0’;

abus <= (others => ’0’);
bbus <= (others => ’0’);

if cnt = "0000000000" then
  -- Reset the statemachine if we get another packet
  nextstate <= NO_PACKET;
else
  case currstate is

    when NO_PACKET =>

      nextstate <= NO_PACKET;

      if cnt = ETH_TYPE_O then
        -- check that the ethernet type indicates IPv4
        if eth_type = ETH_TYPE_IP4 then
          nextstate <= IP4;
        else
          nextstate <= PKT_WAIT;
        end if;
      end if;

    when IP4 =>

      nextstate <= IP4;

      if cnt = IP4_VHL_O then
        -- check that we’ve got IP version 4
        if ip4_v = IP_VERSION_FOUR then
          -- If so, grab the header length (in case we have options)
          -- and continue; ip4_hl is word addressed
          plst_l <= ’1’;
          plst_i <= (ip4_hl & "00") + ETH_HDR_LEN;
        else
          -- Abort on NON-IPV4 packets
          nextstate <= PKT_WAIT;
        end if;
      end if;

      if cnt = IP4_LENGTH_O then

        -- if this packet does not hold enough for echo header, echo
        -- trailer, and a path record, then skip.
        if ip4_length < plst + IPMP_ECHO_LEN + IPMP_PR_LEN then
          nextstate <= PKT_WAIT;
        end if;
        length_l <= ’1’;
        length_i <= ip4_length;

      elsif cnt = IP4_OFFSET_O then

        -- Abort if a fragment
        if ip4_offset /= IP_OFFSET_ZERO then
          nextstate <= PKT_WAIT;
        end if;

        -- Abort if the MF bit is set, since the whole packet
        -- needs to be here for the echo trailer
        if data(13) = ’1’ then
          nextstate <= PKT_WAIT;
        end if;

      elsif cnt = IP4_TTL_O then
        -- Keep TTL for path record
        ttl_l  <= ’1’;
        ttl_i  <= ip4_ttl;

        -- Load a potential ttl replacement into checksum
        csum_l <= ’1’;
        bbus   <= ip4_ttl & X"00";

      elsif cnt = IP4_PROTO_O then
        -- skip if ip protocol type is not IPMP
        if ip4_proto /= IP_PROTO_IPMP then
          nextstate <= PKT_WAIT;
        end if;

      elsif cnt = plst then
        -- got to the end of the IP header,
        -- go onto the IPMP header now
        nextstate <= IP4_IPMP;

      end if;

    when IP4_IPMP =>

      nextstate <= IP4_IPMP;

      if (cnt - plst) = IPMP_VER_O then
        -- skip if an unknown IPMP version
        if ipmp_ver /= IPMP_VERSION_ONE then
          nextstate <= PKT_WAIT;
        end if;
      elsif (cnt - plst) = IPMP_TYPE_O then
        -- skip if not an IPMP echo
        if (data(15) = ’0’) then
          nextstate <= PKT_WAIT;
        else
          pathp_l <= ’1’;
          pathp_i <= X"000C";
          nextstate <= PR_INS;
        end if;

      end if;

    when PR_INS =>

      nextstate <= PR_INS;

        if (cnt - plst) = pathp + 1 then
          if cnt > length + ETH_HDR_LEN -
            (ECHO_TRAILER_LEN + IPMP_PR_LEN) then
            -- check for room for a path record and echo trailer
            nextstate <= PKT_WAIT;

          elsif ttl > pr_ttl then
            -- skip if pre-initalised PR TTL is larger than IP TTL
            nextstate <= PKT_WAIT;

          elsif (data(7) = ’0’) then
            -- subtract the existing PR TTL value from the checksum
            csum_l    <= ’1’;
            abus      <= pr_ttl & X"00";
            bbus      <= X"0080";
            -- write the replacement TTL value out
            data_next <= ttl & X"80";

          else
            -- wait until the next path record now
            pathp_l <= ’1’;
            pathp_i <= pathp + 12;

          end if;

        elsif (cnt - plst) = pathp + 3 then
          -- First 16 bits of timestamp
          csum_l    <= ’1’;
          bbus      <= timestamp(47 downto 32);
          data_next <= bbus;

        elsif (cnt - plst) = pathp + 5 then
          -- Second 16 bits of timestamp
          csum_l    <= ’1’;
          bbus      <= timestamp(31 downto 16);
          data_next <= bbus;

        elsif (cnt - plst) = pathp + 7 then
          -- Third 16 bits of timestamp
          csum_l    <= ’1’;
          bbus      <= timestamp(15 downto 0);
```

```
        data_next <= bbus;

      elsif (cnt - plst) = pathp + 9 then
        -- First 16 bits of IP address
        csum_l    <= '1';
        bbus      <= ip4_addr(31 downto 16);
        data_next <= bbus;

      elsif (cnt - plst) = pathp + 11 then
        -- Second 16 bits of IP address
        csum_l    <= '1';
        bbus      <= ip4_addr(15 downto 0);
        data_next <= bbus;
        nextstate <= ECHO_TRAILER;

      end if;

  when ECHO_TRAILER =>

    nextstate <= ECHO_TRAILER;

    -- if the path record pointer has arrived, then increment it
    if cnt = ETH_HDR_LEN + (length - 4) + 1 then
      csum_l    <= '1';
      abus      <= ipmp_pathp;
      bbus      <= ipmp_pathp + 12;
      data_next <= bbus;

    -- sneaky in-between path record pointer and checksum cycle to
    -- fold the checksum
    elsif cnt = ETH_HDR_LEN + (length - 3) + 1 then
      csum_l <= '1';
      csum_i <= (X"0000" & csum(15 downto 0)) +
                (X"0000" & csum(31 downto 16));

    -- if the checksum field has arrived, incrementally update it
    elsif cnt = ETH_HDR_LEN + (length - 2) + 1 then
      carry  <= (not ((X"0000" + csum(31 downto 0)) +
                      (X"0000" & csum(31 downto 16))))
                + ipmp_chksum + 1;

      if carry_wrap = X"FFFF" then
        data_next  <= X"0000";
      else
        data_next <= carry_wrap;
      end if;

    -- got to the end of the IPMP packet, go back into PKT_WAIT
    elsif cnt = ETH_HDR_LEN + length then
      nextstate <= PKT_WAIT;

    end if;

  when PKT_WAIT =>

    -- Wait the packet out
    nextstate <= PKT_WAIT;
```

```
          if data_h(8) = ’0’ then
            nextstate <= NO_PACKET;
          end if;

        when others =>
          nextstate <= PKT_WAIT;

      end case;
    end if;
  end process;


  STATEM : process ( clk, reset_n )
  begin
    if reset_n = ’0’ then
      length    <= (others => ’0’);
      pathp     <= (others => ’0’);
      ttl       <= (others => ’0’);
      plst      <= (others => ’0’);
      currstate <= NO_PACKET;
    elsif rising_edge(clk) then

      -- This is important
      -- We don’t advance the state machine unless the word is valid.

      if advance = ’1’ then

        currstate <= nextstate;

        if length_l = ’1’ then
          length <= length_i;
        end if;

        if pathp_l = ’1’ then
          pathp <= pathp_i;
        end if;

        if ttl_l = ’1’ then
          ttl <= ttl_i;
        end if;

        if plst_l = ’1’ then
          plst <= plst_i;
        end if;

      end if;
    end if;
  end process;

end rtl;
```

Bibliography

[1] R. Mahajan, N. Spring, D. Wetherall, and Anderson. T., "User-level Internet path diagnosis," in *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP)*, Bolton Landing, NY, Oct. 2003, pp. 106–119.

[2] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, Apr. 2001, pp. 905–914.

[3] S. Floyd and E. Kohler, "Internet research needs better models," *ACM SIGCOMM Computer Communications Review*, vol. 33, no. 1, pp. 29–34, Jan. 2003.

[4] M. Muuss, "Ping," `http://ftp.arl.mil/~mike/ping.html`.

[5] V. Jacobson, "Traceroute," `ftp://ftp.ee.lbl.gov/traceroute.tar.Z`.

[6] K. Lai and M. Baker, "Measuring bandwidth," in *Proceedings of IEEE INFOCOM*, New York, NY, Mar. 1999, pp. 235–245.

[7] V. Jacobson, "pathchar - a tool to infer characteristics of Internet paths," Apr. 1997, `ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf`.

[8] A.J. McGregor, "The IP Measurement Protocol," `http://moat.nlanr.net/AMP/AMP/IPMP/`, 1998.

[9] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 37–52, Apr. 2005.

[10] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," RFC 4271, IETF, Jan. 2006.

[11] J. Hawkinson and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)," RFC 1930, IETF, Mar. 1996.

[12] B. Huffaker, D. Plummer, D. Moore, and k. claffy, "Topology discovery by active probing," in *Symposium on Applications and the Internet (SAINT)*, Nara, Japan, Jan. 2002, pp. 90–96.

[13] L. Subramanian, V.N. Padmanabhan, and R.H. Katz, "Geographic properties of Internet routing," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, Monterey, CA, June 2002, pp. 243–259.

[14] V. Paxson, *Measurement and Analysis of End-to-End Internet Dynamics*, Ph.D. thesis, University of California at Berkeley, 1997.

[15] H. Burch and W. Cheswick, "Mapping the Internet," *IEEE Computer*, vol. 32, no. 4, Apr. 1999.

[16] "Netdimes project," `http://www.netdimes.org/`.

[17] A. Broido and k. claffy, "Analysis of RouteViews BGP data: policy atoms," in *Proceedings of NRDM workshop*, Santa Barbara, CA, May 2001.

[18] A. Broido and k claffy, "Internet topology: Connectivity of IP graphs," in *Proceedings of SPIE International symposium on Convergence of IT and Communication*, Denver, CO, Aug. 2001, pp. 172–187.

[19] H. Burch, *Measuring an IP Network* in situ, Ph.D. thesis, Carnegie Mellon University, 2005.

[20] M. Fomenkov, k. claffy, B. Huffaker, and D. Moore, "Macroscopic Internet topology and performance measurements from the DNS root name servers," in *Proceedings of USENIX LISA*, San Diego, CA, Dec. 2001.

[21] E. Cronin, S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1369–1382, Sept. 2002.

[22] N. Spring, D. Wetherall, and T. Anderson, "Reverse-engineering the Internet," in *ACM SIGCOMM Workshop on Hot Topics in Networks*, Cambridge, MA, Nov. 2003, pp. 3–8.

[23] N. Spring, *Efficient discovery of network topology and routing policy in the Internet*, Ph.D. thesis, University of Washington, 2004.

[24] J. Postel, "Internet Protocol," RFC 791, IETF, Sept. 1981.

[25] K. Papagainnaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, "Measurement and analysis of single-hop delay on an IP backbone network," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 908–921, Aug. 2003.

[26] G. Malkin, "Traceroute using an IP option," RFC 1393, IETF, Jan. 1993.

[27] M. Luckie, K. Cho, and B. Owens, "Inferring and debugging path MTU discovery failures," in *Proceedings of Internet Measurement Conference 2005*, Berkeley, CA, Oct. 2005, pp. 193–198.

[28] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proceedings of ACM/SIGCOMM '02*, Pittsburgh, PA, Aug. 2002, pp. 133–145.

[29] S. Bellovin, "A technique for counting NATted hosts," in *Proceedings of the 2nd ACM SIGCOMM conference on Internet measurement*, Marseille, France, Oct. 2002.

[30] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links," RFC 1144, LBL, Feb. 1990.

[31] D.L. Mills, "Network Time Protocol (version 3): Specification, implementation and analysis," RFC 1305, University of Delaware, Mar. 1992.

[32] R. Govindan and V. Paxson, “Estimating router ICMP generation times,” in *Proceedings of the PAM2002 workshop on Passive and Active Measurements*, Fort Collins, CO, Mar. 2002.

[33] G. Jin and B.L. Tierney, “System capability effects on algorithms for network bandwidth effects,” *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 27 – 38, Oct. 2003.

[34] J. Postel, “Internet Control Message Protocol,” RFC 792, IETF, Sept. 1981.

[35] K.C. Claffy, G.C. Polyzos, and H-W. Braun, “Measurement considerations for assessing unidirectional latencies,” *Internetworking: Research and Experience*, vol. 4, no. 3, pp. 121–132, 1993.

[36] S. Shalunov and B. Teitelbaum, “One-way active measurement protocol (OWAMP) requirements,” RFC 3763, IETF, Apr. 2004.

[37] S. Shalunov, B. Teitelbaum, A. Karp, J.W. Boote, and M. Zekauskas, “A one-way active measurement protocol (OWAMP),” Internet Draft draft-ietf-ippm-owdp-16.txt, IETF, Feb. 2006, work in progress.

[38] K. Anagnostakis, M. Greenwald, and R. Ryger, “cing: Measuring network-internal delays using only existing infrastructure,” in *Proceedings of IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 2113–2124.

[39] S. McCanne and V. Jacobson, “The BSD packet filter: A new architecture for user-level packet capture,” in *Proceedings of the Winter 1993 USENIX Conference*, San Diego, CA, Jan. 1993.

[40] M. Luckie, “Scamper,” `http://www.wand.net.nz/scamper/`.

[41] E. Blanton and M. Allman, “On the impact of bursting on TCP performance,” in *Passive and Active Network Measurement: 6th International Workshop, PAM 2005*, Boston, MA, Mar. 2005, pp. 1–12.

[42] S. Savage, “Sting: a TCP-based network measurement tool,” in *Proceedings of USITS ’99: The 2nd USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, Oct. 1999, pp. 71–79.

[43] J.C.R. Bennett, C. Partridge, and N. Shectman, “Packet reordering is not pathological network behavior,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 789–798, Dec. 1999.

[44] E. Blanton and M. Allman, “On making TCP more robust to packet reordering,” *ACM Computer Communication Review*, vol. 32, no. 1, Jan. 2002.

[45] S. Keshav, “A control-theoretic approach to flow control,” in *Proceedings of SIGCOMM ’91*, 1991, pp. 3–15.

[46] R.L. Carter and M.E. Crovella, “Measuring bottleneck link speed in packet-switched networks,” Tech. Rep. BU-CS-96-006, Boston University, 1996.

[47] V. Paxson, “End-to-end Internet packet dynamics,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, 1999.

[48] C. Dovrolis, P. Ramanathan, and D. Moore, “Packet-dispersion techniques and a capacity-estimation methodology,” *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 963–977, Dec. 2004.

[49] R. Kapoor, L-J Chen, A. Nandan, M. Gerla, and M.Y. Sanadidi, “CapProbe: A simple and accurate capacity estimation technique,” in *Proceedings of ACM SIGCOMM 2004*, Portland, OR, USA, Aug. 2004, pp. 449–460.

[50] A.B. Downey, “Using pathchar to estimate Internet link characteristics,” in *Proceedings of SIGCOMM ’99*, Cambridge, MA, Aug. 1999.

[51] R.S. Prasad, C. Dovrolis, and B.A. Mah, “The effect of layer-2 store-and-forward devices on per-hop capacity estimation,” in *Proceedings of IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 2090–2100.

[52] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *Proceedings of SIGCOMM '00*, Stockholm, Sweden, Aug. 2000.

[53] K. Harfoush, A. Bestavros, and J. Byers, "Measuring bottleneck bandwidth of targeted path segments," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 2079–2089.

[54] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000, pp. 1371–1380.

[55] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 601–615, 1997.

[56] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of Internet path properties," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, San Francisco, CA, Nov. 2001, pp. 197–211.

[57] J. Aikat, J. Kaur, F.D. Smith, and K. Jeffay, "Variability in TCP round-trip times," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet Measurement*, Miami Beach, FL, Oct. 2003, pp. 279–284.

[58] F. Baker, "Requirements for IP Version 4 routers," RFC 1812, Cisco, June 1995.

[59] P. Almquist and F. Kastenholz, "Towards requirements for IP routers," RFC 1716, IETF, Nov. 1994.

[60] R. van den Berg and P. Dibowitz, "Over-zealous security administrators are breaking the Internet," in *Proceedings of LISA '02: Sixteenth Systems Administration Conference*, Berkeley, CA, Nov. 2002, pp. 213–218.

[61] S. Casner, "The mtrace (8) manual page," `http://ftp.parc.xerox.com/pub/net-research/ipmulti/`.

[62] S.F. Donnelly, *High Precision Timing in Passive Measurements of Data Networks*, Ph.D. thesis, University of Waikato, 2002.

[63] C. Hornig, “A standard for the transmission of IP datagrams over Ethernet networks,” RFC 894, IETF, Apr. 1984.

[64] A. Pásztor and D. Veitch, “PC based precision timing without GPS,” in *Proceedings of SIGMETRICS*, Marina Del Ray, CA, June 2002, pp. 1–11.

[65] D. Moore, G.M. Voelker, and S. Savage, “Inferring Internet denial-of-service activity,” in *Proceedings of the 2001 USENIX Security Symposium*, Washington D.C., Aug. 2001.

[66] R. Braden and J. Postel, “Requirements for Internet gateways,” RFC 1009, IETF, June 1987.

[67] A. Rijsinghani, “Computation of the Internet Checksum via incremental update,” RFC 1624, Digital Equipment Corporation, May 1994.

[68] R. Draves, “Default address selection for Internet Protocol version 6 (IPv6),” RFC 3484, Microsoft Research, Feb. 2003.

[69] R. Presuhn, “Version 2 of the protocol operations for the simple network management protocol (SNMP),” RFC 3416, IETF, Dec. 2002.

[70] K. McCloghrie and F. Kastenholz, “The interfaces group MIB,” RFC 2863, IETF, June 2000.

[71] S. Willis, J. Burruss, and J. Chu, “Definitions of managed objects for the fourth version of the border gateway protocol (BGP-4) using SMIv2,” RFC 1657, IETF, July 1994.

[72] U. Windl, “Implementation of nanosecond time and a PPS API for the Linux 2.4 kernel,” `http://www.kernel.org/pub/daemons/PPS/`.

[73] J. Lemon, “Resisting SYN flood DoS attacks with a SYN cache,” in *Proceedings of BSDcon02*, San Francisco, CA, Feb. 2002, pp. 89–98.

[74] Endace Measurement Systems, “DAG network monitoring interface cards,” `http://www.endace.com/`.

[75] E. Galstad, “Nagios - network monitoring software,” `http://www.nagios.org/`.

[76] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, “Performance anomaly of 802.11b,” in *Proceedings of IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 836–843.

[77] N.G. Duffield and M. Grossglauser, “Trajectory sampling for direct traffic observation,” *IEEE Transactions on Networking*, vol. 9, no. 3, pp. 280–292, June 2001.

[78] M. Mathis and J. Mahdavi, “Diagnosing Internet congestion with a transport layer performance tool,” in *Proceedings of INET’96*, Montreal, Canada, June 1996.