Analysis of Voice Over IP Traffic

James Curtis

October 6, 1999

Abstract

Voice over IP applications allow telephone conversations over a standard Internet connection. This is an exciting new use of the Internet for most users, and can result in large cost savings for long distance toll calls. This is resulting in a high growth rate in the use of these applications. Unfortunately due to the implementation of these applications, a large growth in voice traffic could seriously degrade the performance of other common Internet applications.

This report outlines the progress this project has made in investigating these effects. This has involved a large study of the ways of measuring network traffic. This study has encountered multiple difficulties and problems present in many of the systems in use today. This report explains these issues and introduces some of the techniques currently being used to correct these problems.

The project has focused on the use of one measurement system and this report goes on to detail the the analysis of data from this system. The reliable detection and filtering of the voice traffic present in these data sets is the most important area of this analysis. Finally this report outlines the initial analysis of the voice traffic found using these methods.

Contents

List of Figures

1 Introduction

Today the Internet is experiencing one of the fastest growth rates of any technology seen before. Its audience has quickly spread from academics and computer scientists to the general public, and is now seen by many as an indispensable research tool. For many users however, both connection and machine speeds have limited the potential uses of the Internet. This is now starting to change with the recent increases in machine performance and decreasing costs of broadband connections to the home. With these improvements it is becoming possible to provide a method of communication very similar to the standard telephone network using an Internet connection. The increase in bandwidth, when combined with the advances of audio compression and machine speed has allowed these services to be capable of hight quality, often above the current telephone network. Many users are also likely to accept a drop in quality for the benefit of free long distance calls that voice calls using the Internet often provide. This being the case, an even wider range of people are able and likely to use these systems. It is not surprising therefore that Voice Over IP (VoIP), is an increasing use of the Internet today.

In a communication system, the most important feature is the ability to carry out a conversation over it. The ease of this will decrease as delays between speakers hearing each other increases. Even more distracting to the human ear however is an irregularity in this delay. This variation is called jitter and can inserts pauses into speech, possibly breaking up words.

These properties of voice make VoIP a real-time service. A real-time service refers to a process where data must be dealt with as it is created, or arrives. Any delays or lags break this requirement. Few Internet applications applications, such as WWW, FTP and e-mail are real-time, and so the design focus of Internet protocols is not oriented towards real-time traffic. VoIP applications have to use a different protocol to the major current Internet applications. A significant increase in the this traffic such as could be caused by VoIP use, gives cause for concern. It is possible that the performance of todays applications could be seriously effected by this increase. An explanation of the reasons for this follow in section 2.1.

The motivation behind this project is to investigate these effects. This report

details the progress that this project has made towards this goal.

1.1 How this report is organised

This report is divided into 7 major sections.

Section 2: This section provides a more detailed insight into the motivations and aims of this project. This section introduces the need for simulation, and the requirements for measurement systems to aid this.

Section 3: In this section the technique of passive measurement is introduced. This section then details how these systems can be created, and the problems and limitations associated with them.

Section 4: This section details the specific measurement system that has been used for this project. This system has created a large set of traces, and the initial analysis that was carried out on these traces is also detailed.

Section 5: Once suitable network trace files have been obtained it is necessary to find VoIP traffic within them. This section outlines the techniques this project has used to achieve this. Then this section provides the analysis that has been carried out on this data.

Section 6: This section outlines the many areas of this project that future work can build on.

Section 7: In this section the conclusions that have come from this project are detailed. This also provides an evaluation of the project.

2 Project aims and motivations

2.1 UDP vs TCP

The Internet runs on a hierarchical protocol stack. A simplified version of this is shown in figure 1[1]. The layer common to all Internet applications is the IP (Internet Protocol) layer. This layer provides a connectionless, unreliable packet based delivery service. It can be described as connectionless because packets are treated independently of all others. The service is unreliable because there is no guarantee of delivery. Packets may be silently dropped, duplicated or delayed and may arrive out of order. The service is also called a best effort service, all attempts to deliver a packet will be made, with unreliability only caused by hardware faults or exhausted resources.

As there is no sense of a connection at the IP level there are no simple methods to provide a *quality of service (QoS)*. QoS is a request from an application to the network to provide a guarantee on the quality of a connection. This allows an application to request a fixed amount of bandwidth from the network, and assume it will be provided, once the QoS request has been accepted. Also a fixed delay, i.e. no jitter and in order delivery can be assumed. A network that supports QoS will be protected from congestion problems, as the network will refuse connections that request larger resources than can be supplied. An example of a network that supports QoS is the current telephone network, where every call is guaranteed the bandwidth for the call. Most users at some point have heard the overloaded signal where the network cannot provide the requested resource required to make a call.

The application decides which transport protocol is used. The two protocols shown here, TCP and UDP are the most commonly used ones. TCP provides a reliable connection and is used by the majority of current Internet applications. TCP, besides being responsible for error checking and correcting, is also responsible for controlling the speed at which this data is sent. TCP is capable of detecting congestion in the network and will back off transmission speed when congestion occurs. These features protect the network from congestion collapse.

As discussed in the introduction, VoIP is a real-time service. For real-time

[1]It should be noted that AAL-5 is shown in the physical layer of this diagram. AAL-5 is an encapsulating layer for use with ATM. This report often refers directly to ATM as the physical layer, assuming AAL-5 data encapsulation.

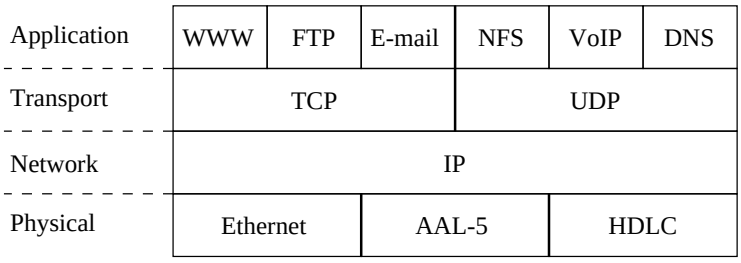

Figure 1: Simplified IP protocol stack

properties to be guaranteed to be met, a network with QoS must be used to provide fixed delay and bandwidth. It has already been said that IP cannot provide this. This then presents a choice. If IP is a requirement, which transport layer should be used to provide a system that is most likely to meet real-time constraints.

As TCP provides features such as congestion control, it would be the preferred protocol to use. Unfortunately due to the fact that TCP is a reliable service, delays will be introduced whenever a bit error or packet loss occurs. This delay is caused by retransmission of the broken packet, along with any successive packets that may have already been sent. This can be a large source of jitter.

TCP uses a combination of four algorithms to provide congestion control, slow start, congestion avoidance, fast retransmit and fast recovery [Ste97]. These algorithms all use packet loss as an indication of congestion, and all alter the number of packets TCP will send before waiting for acknowledgments of those packets. These alterations affect the bandwidth available and also change delays seen on a link, providing another source of jitter.

Combined, TCP raises jitter to an unacceptable level rendering TCP unusable for real-time services. Voice communication has the advantage of not requiring a completely reliable transport level. The loss of a packet or bit error will often only introduce a click or a minor break into the output.

For these reasons most VoIP applications use UDP for the voice data transmission. UDP is a thin layer on top of IP that provides a way to distinguish among multiple programs running on a single machine. UDP also inherits all of the properties of IP that TCP attempts to hide. UDP is therefore also a packet

based, connectionless, best-effort service. It is up to the application to split data into packets, and provide any necessary error checking that is required.

Because of this, UDP allows the fastest and most simple way of transmitting data to the receiver. There is no interference in the stream of data that can be possibly avoided. This provides the way for an application to get as close to meeting real-time constraints as possible.

UDP however provides no congestion control systems. A congested link that is only running TCP will be approximately fair to all users. When UDP data is introduced into this link, there is no requirement for the UDP data rates to back off, forcing the remaining TCP connections to back off even further. This can be though of as UDP data not being a "good citizen". The aim of this project is to characterise the quantity of this drop off in TCP performance.

2.2 Simulation systems

The simplest way of measuring the effect UDP data has on TCP is via simulation. Simulation provides both a way of controlling such parameters as connection bandwidth, router queue lengths and traffic levels, and ways to record such statistics as queue utilisation, packet loss and connection throughput. Attempting to obtain these sorts of results from an actual network would be both extremely costly and difficult.

Simulation however can only produce results as reliable and as accurate as the models employed. For simulation results to have any meaning, the component models in the virtual network, (such as switches, routers and most importantly, traffic models) must be as close to the real-world as possible. A traffic model provides a method for a simulator to create the data to flow through the network. Without accurate models to do this, the results of simulations would have no reference to real networks, creating pointless results.

This project luckily has access to a simulator of this quality. This is a discrete event simulator and is based on the ATM-TN simulator [ACGW95], with a range of modifications. The main change provides a more generic serial interface compared to the full ATM infrastructure present in ATM-TN. ATM-TN provides an actual TCP stack, derived from 4.4 BSD Lite, and a powerful simulation engine. More details including investigations completed using this

can be found in [MPC98]

A traffic model for this simulator exists for HTTP traffic which makes use of TCP for transmission. These models will provide the reference traffic, the performance of which will be tested as UDP data rates are increased into the network.

Traffic models are based around network traces. The simulator takes data recorded from these network traces replicates data during simulation to provide loads for the network. If there is insufficient data to create the required loads the simulator offsets and overlays multiple copies of the recorded data.

For this project a set of traces is required to form a model for UDP traffic. The levels of the UDP traffic would then be increased over simulation runs, and the throughput achieved on the existing TCP traffic models can be measured.

2.3 Data source

Two options are available to create the traces required by the simulator. The first option is to accurately measure VoIP data, created in a lab environment using sample conversations. The advantages of this approach is complete control of the systems and protocols used. This means that there is no question what the traffic represents, what programs and compression techniques were used, or the speed and load of the machines and Internet connections involved.

The second option is to use network trace files that contain IP and transport layer headers (either TCP or UDP), for every packet seen at a specific point in the network. The more traffic that passes this point, the greater likelihood of VoIP data being present in the trace. This process of 'snooping' on traffic is known as passive measurement, and is further discussed in section 3. Once collected the trace files can be checked for data that has the signatures of VoIP traffic. This filtered data then forms the basis for simulation.

The second option may seem at first more complex and less accurate than the first, but also has advantages. First there are many providers of VoIP applications and these can use different voice compression systems. Secondly different users of VoIP applications may have varied Internet connection and machine speeds. Each user will therefore be able to handle different data rates and compression techniques. Lastly there is no way of knowing what a 'normal'

conversation is like, a pair of research students may have completely different conversation patterns to a business meeting being carried out over the Internet. All of these issues will alter the traffic created, both in speed and size. Trying to cover all of these possibilities in a lab environment would be infeasible.

Identifying actual data in use on real networks however ensures that the data used in simulation is a close representation of the type of voice traffic seen at that point in the network. For these reasons it was decided to try and reliably detect VoIP traffic sessions using only IP header traces to provide sample data.

What follows in section 3 is a general discussion of passive measurement techniques and information on the specific systems used by this project. This discussion includes details on what is passive measurement, how it can be carried out, what problems can arise and what are the limitations of various systems.

3 Passive measurement

The term passive measurement refers to the process of measuring a network, without creating or modifying any traffic on the network. This is in contrast to active measurement, in which specific packets are introduced into the network, and these packets are timed as they travel through the network being measured.

Passive measurement can provide a detailed set of information about the one point in the network that is being measured. Examples of the information passive measurements can provide are:

- Traffic / protocol mixes
- Accurate bit or packet rates
- Packet timing / inter-arrival timing

Passive measurement can also provides a means of debugging a network application, by providing a user with the entire packet contents, as seen on the network.

Active systems provide very little information about a single point of a network. They instead provide a representation of the characteristics of the entire network path between two hosts. Active systems can provide such indications of a networks performance as:

- Packet round trip time (RTT)
- Average packet loss
- Connection bandwidth

Some active systems can also give indications of the following:

- Asymmetric delay times
- Alterations in routing paths between hosts

As stated in section 2.3 this project relies on the use of network traces. My initial expectation when faced with this was that passive measurement systems would be well understood and simple to work with, and that most of my time would be in high level analysis of the traces. This was a large underestimation on my part.

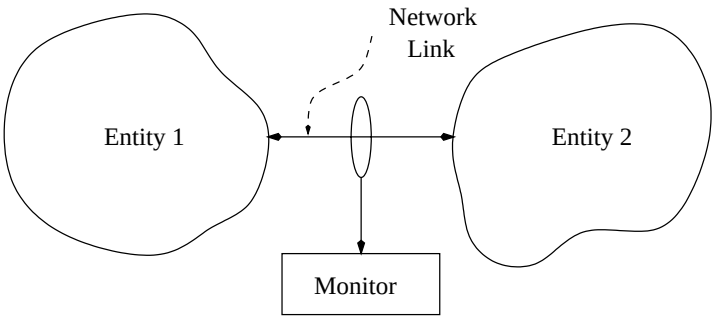

Figure 2: Basic passive measurement setup

Understanding the principals of passive measurement systems has consumed a large portion of my time, and is the area that I have learnt the most about during this project. This includes learning the problems with current implementations, ways of verifying the accuracy of traces and limitations of passive measurement systems. The rest of this section details this information and these experiences.

3.1 Principles of passive measurement

The basic principle of passive measurement is shown in figure 2. There are three main situations that define what the entities represent. In the first situation the first entity represents the entire Internet and the second a single machine. In the next situation the first entity is the outside world, as seen by an organisation. The second entity represents this organisations internal network. A good example of this is is a university's Internet connection and their internal LAN. The final situation is a backbone link where the two entities are just both sections of the Internet.

A monitor 'snoops' on all the traffic flowing between these two entities. What this monitor does with the traffic depends on what the aim of the system is, and also which of the specific situation listed above applies. There are two major categories that passive measurement systems can fall into. The first is to deal with the captured data in real-time. For example by looking at each packet, count the number of bytes passing the monitor every second, or minute etc. These statistics are very small, when compared to the amount of data that could pass the monitor. These values can be used, for example, to see if

available bandwidth is being fully utilised, if saturation is a problem or if there are peak times where more bandwidth could be required. The second type of passive measurement creates files containing copies of all or a proportion of the traffic seen on the link over a certain time period. These trace files can then be post processed. This can allow advanced computation to be carried out that would be impossible in real-time, and also preserves data for further analysis at a later point.

Trace based system have one significant requirement. As the data will be post processed, additional information must be saved with the packet to indicate the time that this packet arrived. The accuracy of this timestamping process will directly relate to the accuracy of the results that can be drawn from a trace file. The issues related to this simple concept of timestamping form some of the hardest problems in passive measurement. A discussion of the problems that I have encountered follows in the next sections.

Real-time analysis suits high speed networks, where the volume of data on the link is too large to record copies of it to disk or even memory. This is likely to occur in the third configuration discussed, where the monitor is on a high capacity network backbone. Real-time analysis also suites projects that want to monitor links for long periods of time, for instance weeks or months. The trade off for long term, or high speed measurement, is detail. The detail provided by these systems is often of limited use for in-depth traffic analysis such as required by this project.

Traced based systems can cover a range of speed of networks. The faster the network the smaller proportion of data that can be saved. One very common subset of the data that is saved is the IP and transport layer headers. The IP header provides information on the source of the datagram, the destination of the datagram, the length of the datagram and which transport protocol is carried in the payload. The transport layer can give an indication of what type of traffic was contained within the packet, but the restrictions of this have to be understood, and care must be taken when claiming a packet contains a certain type of data. These problems will be discussed, with a focus on identifying VoIP data later in section 5.1.

Header traces are commonly used for both of the first two passive measure-

ment configurations discussed, and where ever else network speeds allow traces to be taken. Full capture of all data on a link is normally restricted to the first situation. The data rates created by a single computer are low when compared to backbones and gateways. Full capture allows complete analysis of the actual data passing on the network, which could be used for debugging purposes and also allow later 'playback' of the entire data stream.

One other common subsection of data captured is the physical layer headers. This is used primarily in ATM networks, but this type of capture has limited use for IP level analysis and has not featured as part of this project. As discussed in section 2.3 this project has made use of IP header traces exclusively. For this reason the rest of this section will refer only to trace based systems, although much of the discussion will relate to both trace and real-time analysis.

3.2 Physical layer differences

Passive measurement systems can be implemented in many ways. The physical layer technology has a large influence on the decisions made when designing these systems. Different physical layer technologies will often create different problems and create traces with require quite different interpretations. There have been two types of physical network technology that I have come in contact with through this project. These are ATM and Ethernet.

3.2.1 ATM measurements

Asynchronous Transfer Mode (ATM) is a high-speed cell switched network. An ATM cell is a small fixed sized packet, which an IP packet is broken up into. ATM has become a common choice for high speed (155 Mbit/s to 2.488 Gbit/s), networks. ATM uses a point to point, full duplex connection. This means that one ATM link can only be between two hosts, and each host has a separate path to communicate with the other. In a normal ATM network one host will be an ATM switch, connecting multiple ATM links together, and the other host could either be an end point machine or another switch.

This network design causes problems to a passive measurement system. An example of an ATM monitoring solution is shown in figure 3. As each host has a transmit (Tx) and receive (Rx) line, to monitor the full bi-directional link, a monitor must contain 2 receive interfaces. Data is sent to these interfaces using

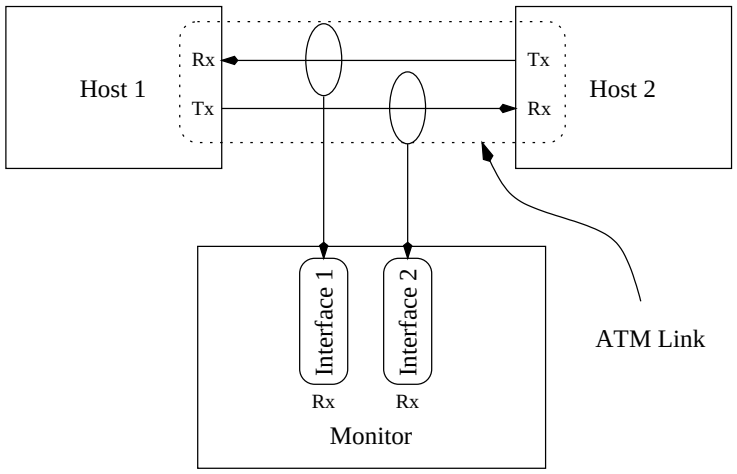

Figure 3: ATM measurement setup

passive devices, either an optical splitter for fiber-optic networks or resistor network for copper media.

A voice conversation is normally two way, both parties both speak to and hear from the other. For this reason VoIP traffic is bi-directional, the traffic in one direction is related to the traffic in the opposite direction. This type of 'flow' analysis requires that each interface has the same clock to timestamp the packets as they arrive. Without this there will be no relationship between interfaces, and so no way of deciding how the two should combine.

3.2.2 Ethernet measurement

Ethernet is one of the most widely spread LAN networking technologies in use today. First introduced in the early 1970s, it now has a range of configurations available, in both media and speed.

In its most basic configuration Ethernet is relatively simple to measure. This base configuration is shown in figure 4. As shown here, Ethernet is a broadcast technology, where all but the sending machine can see all of the traffic on the network. The Ethernet backbone could either be a coax cable string, as in 10base2 and 10base5, or a passive hub as could be in use in 10 or 100baseT. In all of these situations every machine sees all of the traffic at the same time, excluding cable delays. This means that all the monitor needs to do is capture every packet from the network and timestamp it.

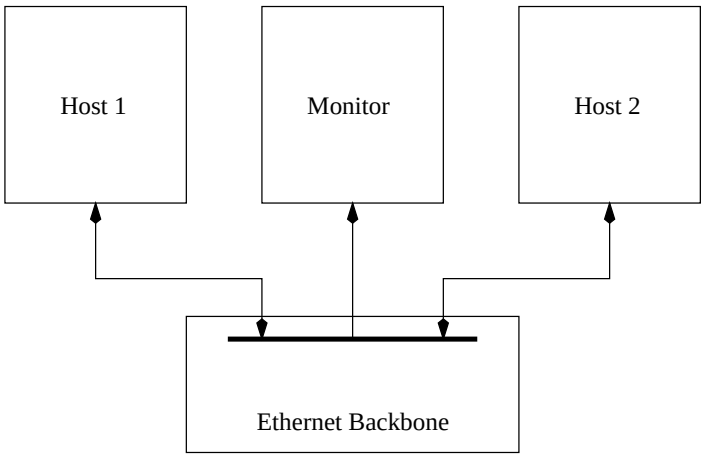

Figure 4: Basic Ethernet measurement setup

Unfortunately fewer and fewer networks are using such a simple broadcast system any more. In order to obtain higher levels of performance and security, most networks use some sort of active device for the backbone. A common example of this is a network switch, where packets are sent only to the destination machine. This has the advantage of being much more secure, only the destination machine can look at the traffic, and can have higher performance. This is because if two pairs of machines are using the network at once, in a broadcast system they all have to share the available bandwidth equally. In a switch however each pair have the full available bandwidth between them. The possible bandwidth between hosts only drops off when multiple hosts all want to transmit to the same destination.

Both of these properties make passive measurement difficult. If the network traffic is no longer broadcast, how can the monitor watch the traffic ? There are two methods that can allow this. First, it is often the case that a Ethernet switch will contain one port that is provided a copy of all of the data. The problem with this port is that the switch must interfere with the data. Normally two packets could flow through the switch at exactly the same time, if the source and destinations are all different. If this occurs then the switch must delay and queue one of the packets before it is sent to this port, introducing inaccuracies to the timestamps on these packets. One other problem is that the data rates inside a switch can exceed the data rates of a single line. In this case the switch

must 'clip' the data that is sent to the accumulated port, dropping packets from the trace.

The second alternative is to insert a passive electronic tap into the wire between a host and switch. This situation ends up looking very similar to the ATM situation already discussed due to the full duplex host to hub connection present on modern 100baseT systems, requiring 2 receive interfaces on the monitor.

3.3 Implementations

A passive measurement system will often be designed with one type of analysis in mind. Different types of analysis require different levels of accuracy and the design will focus on the area most important to the project.

Take for example a research project interested in the protocol mixes present on the Internet today. This type of project requires limited accuracy to obtain results. The systems need not have accurate timestamps, or require 100% packet capture. A random packet loss is unlikely to effect bulk statistics such as protocol mixes. However to obtain a complete picture of the Internet, this project would need to install monitoring positions at as many points of the network as possible. This means that spending large amounts of money to obtain accurate results would provide limited benefit, while buying more machines, providing more monitoring points across the network is likely to improve the project.

As the ultimate aim of this project is simulation, the accuracy of the trace files is extremely important. A poor set of trace files could end in a poor simulation with dubious results. For these reasons each possible solution must be evaluated, and the solutions used must be well understood so the accuracy of any results drawn from simulations can be stated.

3.3.1 Software based measurement

Quite possibly the simplest and quickest passive measurement system is to run the Unix program `tcpdump`. This program is supported under most Unix systems. It will listen on a specified network interface, and capture all traffic seen on that link. Contrary to the packages name, it will dump all data sourced from and destined for that interface, as well as any other traffic seen on that section of the network, and is not restricted to TCP. `tcpdump` can run in two modes,

either capture traffic to a file, or display text information on every packet on the screen. Also built in are filtering rules that allow capture of a specified subset of data, for instance TCP traffic, on port 80, between host x and y.

`tcpdump` is a not only a very common application, but provides a good example of techniques used in software based solutions. A software based system does not all systems that contains software components. All measurement systems will require some software components, the important distinction is that the in a software system, the software is provided no special assistance for measurement by any hardware.

A software based monitor required assistance from the operating system kernel. In most operating systems the kernel contains the network stack that provides an interface between the applications and the network card. In normal operation there are two levels of filtering occurring. First the network card will only transmit to the network stack any packets that have to be dealt with by this machine. For point to point network systems, this will be all packets, for broadcast systems such as Ethernet, this will be only a subset of packets. Next the kernel splits up packets, deals with some itself, and sends on the others to the appropriate application to deal with them.

A software monitor such as `tcpdump` will run as an application program. Without a special interface to the network stack, this program would be unable to capture any packets that were destined for other machines or other applications on the monitor. This would mean that passive measurement would be impossible.

Figure 5 shows a representation of the network layering present in many Unix kernels. The interface that allows programs such as `tcpdump` to work, is the packet capturing interface. This interface, referred to often as the `pcap` interface, is accessed through a C library called `libpcap`. This library also allows reading and writing to files, providing a uniform access to a program for trace files and live networks. `tcpdump` is a higher level interpretation and filtering program that sits on top of `libpcap`. For this reason it is common to refer to files created by `libpcap` as `tcpdump` files.

`libpcap` provides the method for a software program to capture packets that are destined for other applications. To allow capture of all packets seen

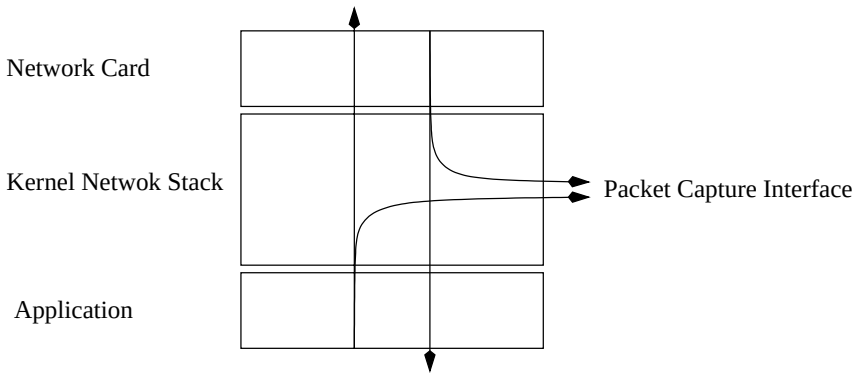

Figure 5: Software monitor setup

on a network, it is often required to put a network card in promiscuous mode. In this mode a network card will provide the network stack with all packets on the network, not only the ones to be dealt with locally. These packets will be normally discarded at the kernel level, but will be forwarded to the `pcap` interface if requested.

The problem with software monitoring solutions is the fact that none of the components were designed or optimised for this use. There are many delays, and buffers in a software solution that reduce the accuracy of timestamps and increase the possibility of packet loss.

The first place this occurs is at the hardware level. A network card may not be reliable enough to capture every packet on a network. In normal use these dropped packets would be corrected for by the TCP stack for a reliable connection. In a monitoring situation, undetected packet loss could cause problems with analysis, or just reduce the accuracy of results. Another common problem is in an attempt to increase performance, many network cards will buffer a stream of packets. The card will then only interrupt the host once and deliver a collection of packets, opposed to once per packet. This will increase host performance, but will mean that the timestamps on successive packets may not represent the actual inter-packet time.

The timestamping process on a `libpcap` system occur in the kernel. This timestamp can be delayed if the host system is under load, and the interrupt service for a packet is delayed. Also the buffers that store the packet before

delivery to the `libpcap` application are unchecked. If an application program does not clear the buffer quickly enough, then packets can be silently dropped.

Software based systems are not limited to using the `pcap` interface, but could write a custom network card driver to provide an interface to the network card for use with passive measurement only. This solution removes the ability to use the network card as a standard network interface, but can still suffer from many of the problems of a `libpcap` system.

Software solutions have one major advantage. They are likely to be much cheaper and quicker to setup than a specialist hardware solution. Software solutions work well for a project that needs a prototype implementation or has only limited demands on the accuracy of the data measured.

3.3.2 Hardware based measurement

Hardware solutions try to correct many of these limitations of a software system. In a full hardware solution, as much of the monitoring system is carried out on the custom interface, and use the system for storage and formatting of the data.

The important features that can be carried out on the card are:

- Timestamping
- Clock synchronisation
- Dropped packet counters
- Traffic filtering

Timestamping before delivery to the host ensures as close to the wire time as possible. Ideally the packet will be timestamped as soon as it has arrived on the card, before any buffering or queuing takes place.

Clock synchronisation is important if there are multiple interfaces in a machine. As discussed in section 3.2.1 if packets are being captured by two interfaces, the timestamps need to be consistent between interfaces.

In a hardware system, a check can be placed on any buffers to detect packet loss. A hardware solution will not necessarily be perfect, but a well designed system will be able to provide a definite counter of any drops that have occurred in the system. For this reliability to carry through to the final network trace,

checks must be placed at every point in the system that packet loss can occur. This includes any software components that run on the host PC to deal with the storage of the data as it is delivered by the monitor card.

Hardware solutions also allow a monitor to deal with larger data rates than software solutions could. It is quite often the case that a limited amount of data will be saved in a monitor, however in a software solution the operating system and possibly monitoring application have to see all of the traffic, then decide if it should be kept. If the hardware becomes intelligent, then it can carry out this function, discarding unwanted data at the earliest possible point.

The major disadvantage of hardware monitoring systems has always been cost. Until recently there have been limited amounts of hardware designed specifically for network monitoring. This meant that any group wanting high accuracy network monitoring had to design from scratch such a system. Such a solution was decided on by the DAG group [DAG99], at the University of Waikato. The cards created by this group, are in constant development and currently support multiple network technologies, speeds and media. Different cards exist for Ethernet, OC3 and OC12 ATM, and DS3 ATM. It is the aim of the WAND group [WAN99], under which this project has been carried out, to take advantage of these custom solutions.

These solutions are often too costly for some measurement projects, with the cost restricting there use to projects requiring very high accuracy or high speed systems.

3.4 Problems and limitations

Through this section many different problems have been introduced. These and several other fundamental issues fix limitations on passive measurement systems, and the results that can be drawn from them. This section attempts to collect all these together, and provide some general discussion related to these problems. This section forms a basic list of issues that must be checked and allowed for when creating or using passive measurement systems.

3.4.1 Timestamps

The simple concept of attaching an arrival time to a packet causes some of the hardest problems faced by passive measurement systems. As seen so far in this

section, the issue of timestamping comes up again and again. The problems discussed so far mainly condenses to two issues, multiple interfaces and delays in the timestamping process.

A delay is introduced whenever a packet is buffered between reception and timestamping. This is most likely to occur in a software based system, but can also occur outside of the monitor in such active devices as switches. It is unlikely that for a given system these delays can be altered, they just have to be known, and accounted for when interpreting results.

The difference in clocks between interfaces is a difficult problem to solve. The ideal solution is to combine the two interfaces to use a single clock. This could occur if the interfaces are combined to a single network card, or the packets get timestamped at the software level, using the system clock. Using a software timestamp can greatly reduce accuracy, and combining interface can often just not be a practical solution. So other systems must be designed. The DAG hardware discussed previously aims to provide methods for a master and slave clocking system. One card provides a clock which the other corrects to. Other advanced clocking methods implemented in DAG hardware are discussed briefly later in this section.

A less accurate, but simple system is for software to reset the clocks on both cards at the same time. Some times one reset could be delayed, causing the cards to start out of synchronisation, but more of an issue in this situation is clock drift.

Clock drift occurs in any simple timing system, where the actual rate of oscillation is slightly different to the stated time. This difference in time will not only be different from actual time, but also different between each card. This causes two effects. First the cards will drift apart from each other over time, loosing synchronisation. This can be corrected by the master–slave system described above. What this will not fix is the relationship between the timestamps recorded and actual time. Over the period of a long trace, the apparent length of the trace recorded will be different to the actual length.

This drift will often be a linear drift, but can also be effected by temperature. The amount of clock drift can be reduced by more accurate timing systems on the card, but will always be present to some degree. An external clock source

provides the best method of correcting for this drift. Possible clock sources include the SONET clock present in ATM over optical fiber, cellphone clock systems, and most importantly the *Global Positioning System (GPS)*.

GPS uses an accurate clock system which can be used by the monitor card to synchronise its clock. One other important advantage of the GPS system is it allows monitors in multiple positions around the globe to synchronise all of their clocks. This allows timing to be carried out on packets traveling between these two points, not just a packets round trip time.

The DAG card supports the use of GPS, providing a solution that eliminates, or greatly reduces the problems associated with timestamping.

3.4.2 Packet loss

Especially in a software based solution, but also in a hardware solution, packet loss is possible. Conditions such as machine load and network data rates will have large effects on the amount of packet loss seen in any system.

Any packet loss is obviously undesirable. As previously discussed, any point in a system where packet loss could occur, must be checked for this. The effect of packet loss on analysis will have varying effects. Some bulk statistics will be hardly effected while some types of detailed analysis, that pull out small numbers of specific packets, could be seriously effected.

If documentation of packet loss is not preserved, how can future analysis be confident of any archived trace files. A trace may be sufficient for initial analysis, but may not be for this later, more detailed analysis. This is again an example of why packet loss must be detected and also shows the importance of documentation of traces.

3.4.3 Internet routing

This report has detailed the reasons why two interfaces are needed for a full-duplex connection, and why timestamps must be synchronised to allow recombination of the traffic seen on each of the interfaces.

One fundamental issue with this however is that the Internet routing algorithms provide no guarantee, and in fact is quite often not the case, that packets flowing between two hosts will travel along the same path.

This means that especially if the monitor is positioned on a backbone link, only half of the traffic flowing between two hosts may be seen. Also as routes can change dynamically, suddenly traffic may no longer flow through the link being monitored.

These problems are fundamental, and there is no way of correcting them. If a project requires that all traffic to and from a host are definitely monitored, then the measurement point itself must move to a position on the network where this is guaranteed, such as on one hosts LAN.

These issues must be kept in mind when using measurement traces taken from positions in the network where these sorts of events can happen.

3.4.4 Fixed sized capture

Both IP and TCP protocols provide methods for adding options to the header fields of the packets. This means that both the IP and TCP layer packet headers can be of variable length. To avoid having to inspect every packet, to decide the length of capture, it is regular practice to capture a fixed length of data per packet. To avoid wasted space, this often fits only minimum sized headers, as options are not often used, especially at the IP level.

The possible lack of the entire header is just another issue that must be allowed for when analysing network traces, and can provide a nasty source of segmentation faults to the unwary programmer.

3.5 Verification

With so many possible situations for errors to occur in a trace file, care must always be taken to verify the accuracy of trace files. This process can be very subjective, for example, without a definite count of dropped packets, how can packet loss be looked for. A full exploration of this subject is beyond the scope of this project, but there are a number of simple techniques that have been used through this project that this section explains.

Through the course of this project I have mainly concentrated on testing the correctness of the timestamps in the trace files I have used. Most checks for dropped packets require extra monitoring equipment to run back to back trials of the monitors. These trials therefore require larger amounts of time to

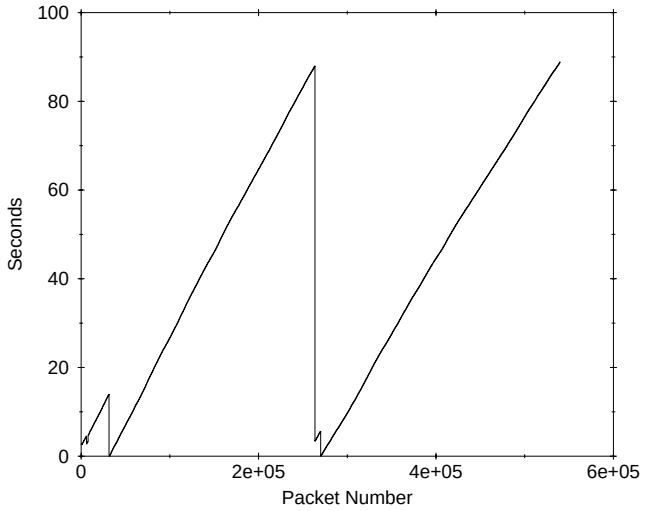

Figure 6: Timestamp vs packet index

analyse, putting them outside of the time-frame of this project. Work has been carried out in this area by other members of the WAND group however.

The primary way of verifying trace files is to use graphical techniques. The first graph commonly plotted of a data set is the timestamp verses the packet number. The packet number is a number assigned to each packet, in order of arrival. For a monitor that is positioned on a backbone or high volume link, it is likely that traffic will be flowing consistently. This means that a graph of timestamp verses packet number should produce a relatively straight line. This line should be constantly increasing, as the time a packet arrives should never be before the previous packet.

An example of a trace where this is not the case is shown in figure 6. It is quickly obvious that there is something wrong with the timestamps in this case. The trace that created this graph came from a group in America. A summary of this group, and their project follows in section 3.7.

When a problem such as this is found, it normally suggests some major misinterpretation or a fundamental problem with the monitoring software. These problems are not likely to be simple to fix, and often the data must be artificially fixed in software at a later point.

A much less obvious problem is shown in figures 7 & 8. At first inspection

of these graphs it would easily be assumed that both systems are correct, and the data used as such. This is not the case however. This becomes apparent if the forward difference of each set is taken. These results are shown in figures 9 & 10. A forward difference is a operation that replaces a data point with the difference between the successive data point, and itself. For example, the value at point 1 will be replaced with the current value at point two minus the current value at point one. This provides the packet inter-arrival time, the length of time between successive packets arriving.

Transforming the data in this method provides much finer detail on the timestamping process. This is shown by the differences in scales on the timestamp axis, around 400 times. By inspecting figures 9 & 10 it can be seen that in the first graph, there are often large inter-arrival times, in the order of 100 to 200 milliseconds. These sorts of inter packet times would be very unlikely to occur on a busy network. Compare this to the corrected version, where the inter-arrival times are all under 5 milliseconds.

Not only does this graph pick up the large inter-arrival times that occur but also picks up a number of packets with negative inter-arrival times. This is not obvious from the first set of graphs, and could be completely overlooked.

This set of graphs are taken from two different revisions of a software monitoring system that this project has made use of. The problem causing the large inter-arrival time was a factor of machine load and a buffer overflow. These problems are discussed, along with the setup of this system in section 4

These graphs provide a good example of the fact that one inspection of the data will not provide assurance of its reliability. It can also be seen that graphs provide a much more useful view of large data sets. The files used to create these graphs are often no more than a text file, containing hundreds of thousands of lines, with a data value per line. Identifying problems by only inspecting these files would be impossible.

The one other form of verification that has been used in this project is a constant sanity check on all data used. By this for example, I mean, never interpret data as being IP unless everything that IP has to conform to is checked.

A list of the current sections of an IP packet that all programs used for this project check for are:

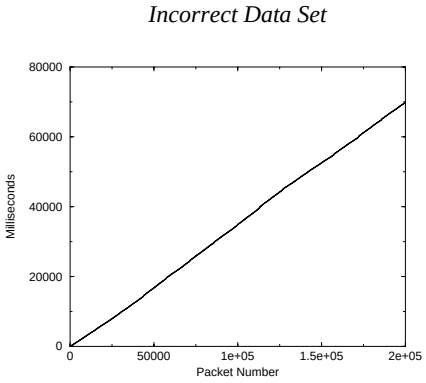

Figure 7: Timestamp vs packet index

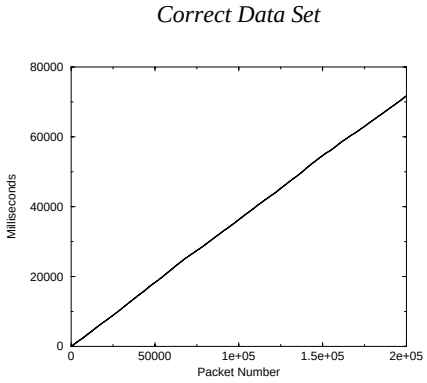

Figure 8: Timestamp vs packet index

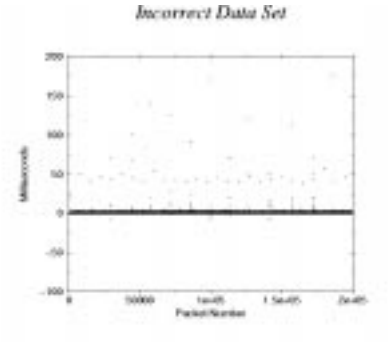

Figure 9: Timestamp forward difference

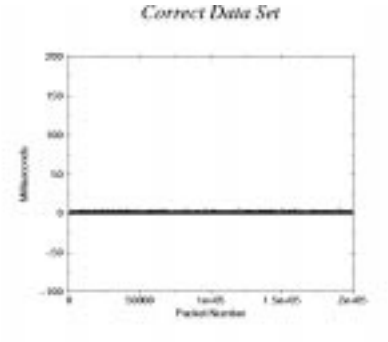

Figure 10: Timestamp forward difference

- Link layer type. For example Ethernet type 0x800 for IP.
- IP version number. All programs test for IP version 4
- IP header length. Must be equal to 5 or greater than 5 if options are in use.

Any packet that breaks any of these rules is discarded. Unfortunately the only points who know exactly how the data should be interpreted are the hosts involved in sending the data. This means that it is still possible to misinterpret data, but the likelihood of this is low if a packet confirms to the above rules.

3.6 Trace analysis

Once the data has been collected, what format should it be stored in for analysis ? As this project focuses on the IP layer up, there should ideally be no concern to any programs about the physical layer technology that the trace was recorded from. Unfortunately this is often not the case. As this project wanted to make use of multiple sources of data, both collecting data from different physical technologies, and using different software, a common format to write tools for was needed. The format decided on was the previously discussed `libpcap` file format. This format was at the time the only common file format for these sorts of applications. The decision to use `libpcap` and discussions of the file format itself is covered more in section 4.2.1. Through the time this project has been running, another option has become available. This solution provides a very substantial base for writing network monitoring systems. It is my opinion that this system provides an extremely powerful solution for any passive monitoring projects.

3.6.1 CoralReef API

This system is called CoralReef [CR99]. It has been created by the CAIDA research team in San Diego, USA [CAI99]. The aim of this software is to provide an *application program interface (API)* that allows a common programming interface to different low level passive monitoring systems. Many network technologies are supported by CoralReef, which then presents a common interface to these interfaces to the application. CoralReef provides the same API for use

on both live networks, and a range of trace files formats, including the `libpcap` file format.

The DAG group are aiming for support under CoralReef for their range of hardware solutions. This should provide CoralReef with the ability to use high accuracy hardware solutions. This makes CoralReef a comprehensive system to write monitoring applications for. The ability to write software once, that will run on a large range of monitoring systems is a large benefit to a research project. Once fully completed CoralReef, in my opinion, will provide a interface that allows extremely quick development of specialist monitoring software, with out requiring the need to redesign or re-implement the low level monitoring system.

3.7 Case study : MOAT, NLANR

The Measurement and Operations Analysis Team (MOAT) are a division of the NLANR research team, in San Diego, USA [MOA99]. The main aim of the MOAT team is to create a network analysis infrastructure, providing a platform to build a better understanding of the Internet. One major part of this is a large scale passive monitoring system, that consists of multiple measurement points spread across the USA. These monitors are mostly connected to the vBNS, a high speed ATM American research network.

Unfortunately the majority of the traces produced by MOAT suffer from a number of the problems detailed within this section. The first major problem of these traces is a general lack of documentation. The exact file format that they are stored in has changed over time, seemingly with no obvious way of detecting this fact.

A lot of problems also stem from a lack of decent hardware to monitor with. The hardware consists of two standard ATM network cards, that happen to be able to provide some indication the arrival time of a packet. For this reason this system would probably fall under a hardware assisted software solution.

The clock on the network card can only contain a small value, and there is no simple way of detecting if in the time between packets this counter has actually wrapped around, missing an entire period of the clock. The software running on the host must provide the upper range of the clock, and this system is prone

to being confused by missing a hardware clock wrap. This, combined with the lack of a reliable system to reset the clocks on the cards, leads to timestamps of very limited accuracy.

For these reasons, attempts to use this data for this project were abandoned, after considerable time was spent investigating the problems seen in these traces.

It is hoped that a new passive monitoring system currently being designed and deployed by MOAT will correct many of these problems, leading to a much more reliable system. The new system is based around the latest version of the DAG ATM monitoring cards, providing a timestamping system that technology used for the current hardware simply could not provide.

4 The New Zealand Internet Exchange

It was originally proposed that this project would use two sources of trace files, one taken on the vBNS by MOAT and the other from the *New Zealand Internet Exchange (NZIX)*, conducted by the WAND group. Due to the failure to find any reliable way to interpret the MOAT traces, this project has focused on the NZIX data set. At the start of this project an initial exploration of the data set was necessary, as this was the first time this data set had been used. This section aims to both give an overview of the measurement system, and details some of the results of this initial exploration.

The NZIX is a 10/100 Mbit Ethernet switch located at the University of Waikato, New Zealand. It forms an inter-connection point for multiple New Zealand bandwidth providers and also provides the University's Internet connection. The WAND group have been permitted to collect trace files of the traffic flowing through this switch.

4.1 Measurement system

The measurement makes use of the *Switch Port Analyser (SPAN)* port present on the NZIX switch. This port can be configured to mirror traffic on some or all of the other ports of the switch. In its current configuration the switch is copying all traffic flowing through the switch to the SPAN port. As discussed in section 3.2.2 the process of copying data to this port can interfere with the timestamping process and can also drop packets if the switch has more traffic flowing through it than the SPAN port can accept. As the data rates seen flowing through this switch are significantly lower than the available bandwidth on the SPAN port it is not expected that packet loss should be a problem. It is unknown what timestamp inaccuracies may be introduced by the switch. It is currently accepted that these are present, and no special action is taken.

The SPAN port is attached to a standard network card in a PC running Linux. Custom software, written by Steven Donelley at Waikato University, is used to capture and timestamp the packets. This software runs using the `pcap` interface to the kernel network stack, discussed in section 3.3.1. This software based system unfortunately also inherits all of the problems discussed in this section.

The first trial of the software, called `ethdev`, produced less than ideal results. An example of this has already been seen in figure 9. The unexpectedly large inter-arrival times seen in this graph were due to packet loss. During testing it became obvious that this packet loss occurred during disk writes, which were normally caused by the writing of the captured data. It was believed that during disk writes the machine was simply not powerful enough to cope with capturing the data and managing the hard drive and so the easiest solution was attempted, use a more powerful computer. This provided limited improvements. The rate of packet loss dropped, but was still at an unacceptable level.

The last option was to work through each section of the software involved, including the Linux kernel network stack and the `pcap` libraries to find any place where packets could be dropped. This task was carried out by Steven who identified the buffer in question. The buffer formed the connection between the `pcap` libraries and the kernel network stack. During disk writes the machine was too busy to schedule the user process that was responsible for clearing the `pcap` buffer. Once filled this buffer silently discarded packets. Two fixes were applied to correct this problem. First the size of the buffer was increased. Second, a test was placed around the insertion to this buffer so if it ever became full, any loss would be recorded.

With these fixes in place, the system was placed under stress test, and no packet loss was detected. Even with this large amount of work, and testing, complete faith in this system would not be sensible. There are still many areas that are not checked for packet loss, and some of these, such as at the hardware level, are impossible in this system. These problems serve as a good example of the difficulty involved in accurate measurement using software solutions.

`ethdev`, while a `libpcap` application, does not use the standard `libpcap` file format. `ethdev` has been designed to be one part of a larger passive measurement, and one way delay system. Because of this, GPS support is included in the `ethdev` system. At precisely one second intervals a zero length packet is inserted, along with the software timestamp at this point. This allows post processing of the files to correct for clock drift. While this greatly reduces the inaccuracy of the clock on the system, it does not reduce the possible delays before a packet is timestamped. A general layout of the `ethdev` measurement

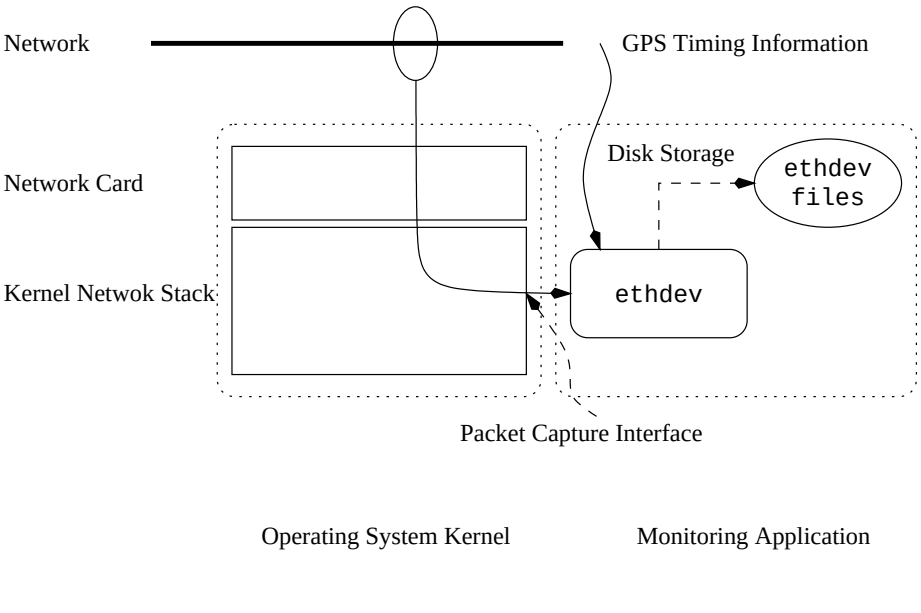

Figure 11: Ethdev measurement layout

system is shown in figure 11.

It should be noted at this point that while there is no complete faith in the measurement system, or the exactness of the timestamps, it does not mean that the produced traces are unusable. This is not the case, it only means that care must be taken when drawing results from these traces. The accuracy of the derived results must take into account the margin of error present in the original measurement system.

It is hoped that in the near future a trial will be run comparing the accuracy of the software measurement to a hardware system, based around the Ethernet version of the DAG. This aims to provide accurate values on the errors contained in the existing software based traces.

4.2 File description

An `ethdev` file consists of multiple, fixed length entries of the C structure shown in figure 12. There is one entry of this structure for every packet received including GPS pulses. There are no headers or trailers so this format is not self identifying.

The timestamp variable (`ts`) is a 32 bit integer, that represents a 4 MHz clock. This means that the clock increments four times per microsecond. The

```
struct queue_entry {
        long ts;
        long crc;
        long length;
        u_char section1[14+12]; /* ethernet + IP */
        long srcip, dstip;
        u_char section2[20];
}
```

Figure 12: `ethdev` file format

maximum range of the `ts` variable is therefore approximately 17.9 minutes. All programs dealing with `ethdev` files must cope with this.

The CRC stores a 32bit CRC that is calculated over the entire IP packet. The `length` variable stores the length of the entire captured packet. As previously discussed, if this is set to zero, this is a GPS interrupt, to allow for clock correction.

The actual data captured is stored in the remaining variables. 48 bytes of data are captured, enough for the Ethernet header, and minimum IP and TCP headers. Any options in either of these will result in truncated headers.

For security and privacy reasons, the source and destination IP addresses in the recorded trace files are encoded. The encoding is a replacement system, where every IP address is replaced by a false substitute. These substitutions are consistent over all of the traces. A database of the translation between substitute and original is held by the maintainers of the NZIX switch. Translations are released on request, when the owner of that IP address has provided consent for this to occur.

4.2.1 `libpcap` file conversion

As it was expected that multiple trace sources would be used, a common file format was required programs for. Due to the traffic filtering abilities, and the popularity of `tcpdump` the `libpcap` file format was decided on as this common point. Due to this a file converter had to be written to convert `ethdev` files to `libpcap` format. The process of conversion and analysis is shown in figure 13, assuming `ethdev` files created as in figure 11.

A `libpcap` file consists of one file header, and repeated entries of a packet header followed by the captured data for that packet. The file header contains

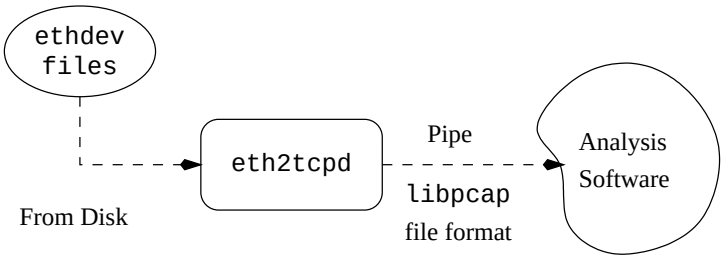

Figure 13: Analysis setup

sections to allow self identification of the file, identify the version of `libpcap` used, the maximum capture length for this trace and an identifier for the type of physical link this data was captured from. Each packet is then saved in the file, with an extra generic header prepended. This header contains the timestamp of the packet, the length captured of this packet, and the length of the packet on the wire. The length captured must be equal to or less than the maximum capture length from the file header. The exact format of these headers can be found in the `pcap.h` header file that is normally located in `/usr/include` on most systems.

To store the timestamp `libpcap` uses a Unix `timeval`. A `timeval` is the basic time structure for Unix and contains a 32 bit seconds variable and a 32 bit microseconds counter. This stores the time in seconds, from the first of January 1970, and wraps approximately once every 68 years.

`ethdev` timestamps give no indication of the actual time or date that the trace was taken. For this reason, all traces converted with the `eth2tcpd` program written, appear to have been taken on the first of January, 1970.

`eth2tcpd` supports conversion both to and from `libpcap` to `ethdev`. As `libpcap` has no concept of the GPS pulse, by default this program will omit these packets from the conversion. An option is provided to override this.

4.3 Trace characteristics

The NZIX measurement project was started on the first of December 1998 and ran until the end of March 1999. Each day, at 10am and 2pm NZST a 10 minute capture was taken. This has resulted in 211 trace files that, compressed, consume approximately 6.4GB of hard disk space.

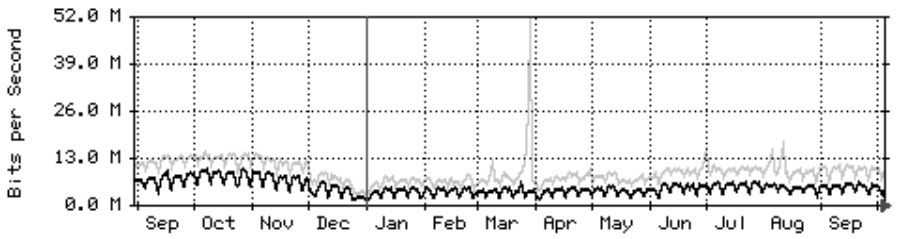

Figure 14: Years traffic plot of NZIX

Shown in figure 14 is a plot of the daily traffic average, for the year covering the period of the project. This plot was taken from the SNMP data provided by the switch, related to the traffic on the SPAN port. This, and graphs of other ports of the switch can be accessed publicly from the WWW [ITS99].

The black line of the plot represents the daily average of the data rates. The gray line represents the maximum 5 minute data rate per day. The spike seen around late March was possibly captured by a trace taken around that time. This trace shows different characteristics to all other NZIX traces, but the reasons for this are yet unexplained. It is also possible that it is coincidence that the spike and the unusual trace occur around the same time of March. This will be looked into further when time permits.

It can be seen from this plot, and also from the trace sizes, that the data rates on NZIX have changed over the time of the project. They initially dropped, due to a breakup of the Kawahiko organisation, an organisation created for bulk bandwidth purchase, and a large contributor to NZIX traffic. Data rates dropped to a low around the Christmas period, and have then stayed reasonably consistent for the rest of the project.

To collect statistics from NZIX traces, a program called `tcpd_stats` was written. This program allows various bulk statistics to be extracted from any `libpcap` file. All statistics quoted for the rest of this section are from the 10am trace dated 31st March, 1999. The rest of this section details the modes this program can run in, and the information provided by each.

4.3.1 Text based statistics

This mode presents summary statistics of the types of traffic seen in the trace file. An example of the header of this file is shown in figure 15. This shows

```
============================================================
                    TRACE DURATION
00:09:57.662371
============================================================
                    PACKET COUNTS
Total Packets   923770

TCP             751537          Percentage of packets: 81.36
UDP             108959          Percentage of packets: 11.80
ICMP            43935           Percentage of packets: 4.76
============================================================
                    TRAFFIC CREATED
Total Traffic   352161806

TCP             331008282       Percentage of traffic: 93.99
UDP             14327199        Percentage of traffic: 4.07
ICMP            2483873         Percentage of traffic: 0.71
============================================================
```

Figure 15: Text statistics header

a common result. The trace was running for approximately 10 minutes, as expected. TCP was responsible for a large percentage of the total packets, and an even larger percentage of total traffic. The combined data rate was approximately 4.5 Mbits per second. The traffic count is listed in bytes.

The rest of the output file details the traffic on each port number for both TCP and UDP. This allows for identification of the major types of traffic. A small section of this is shown in figure 16. First the port number is listed, then the number of packets seen on this port, the amount of data on this port, and the percentage of the total data see on this port. WWW traffic runs on port 80, and from this figure it can be seen that almost 50% of traffic in this trace was web traffic. A more detailed explanation of traffic identification methods follow in section 5.1.

A script has also been written to compact and present the output as a web page. This allows creation of a quick summary web page for each individual trace, without human intervention.

4.3.2 Packet and data rates

This mode will create data sets suitable for creating graphs of data or packet rates. Examples of these are shown in figures 17 & 18. The packet rate has

```
============================================================
                TCP SOURCE PORT USEAGE

Port    20:   11609 Data :    6091952 Percentage : 1.84
Port    21:    1175 Data :     106918 Percentage : 0.03
Port    22:       3 Data :        120 Percentage : 0.00
Port    23:    5210 Data :     738330 Percentage : 0.22
Port    25:   76578 Data :    3711758 Percentage : 1.12
Port    37:       8 Data :        336 Percentage : 0.00
Port    42:       5 Data :        373 Percentage : 0.00
Port    53:    2141 Data :    2387734 Percentage : 0.72
Port    79:      47 Data :       3828 Percentage : 0.00
Port    80:  235292 Data :  156214139 Percentage : 47.19
Port    81:     120 Data :      26685 Percentage : 0.01
```

Figure 16: TCP port usage

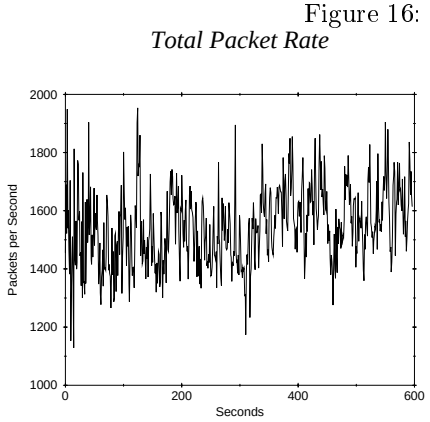

Figure 17: Combined packet rate

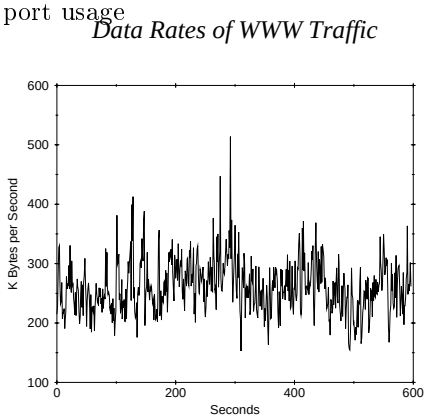

Figure 18: WWW data rate

been calculated from all traffic in the trace, while the data rate graph has been filtered to only include web traffic. This filtering can be applied to any protocol, or service required. These graphs show the traffic to be very 'bursty', with a reasonably stable mean, but large variation around this.

4.3.3 Packet size distribution

This is the final mode of this program. This mode provides a list of every possible packet size, and the number of packets that were seen of that size. Examples of the graphs that can be created with this data are shown in figures 19 & 20. The first graph shows the percentage of packets present for each packet size. From this it can be seen that over 50% of packets are very small, under 100 bytes. The second graph shows the proportion of data that each packet size was

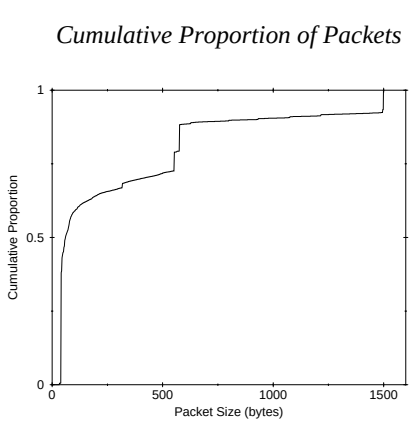

Figure 19: Packet size distribution

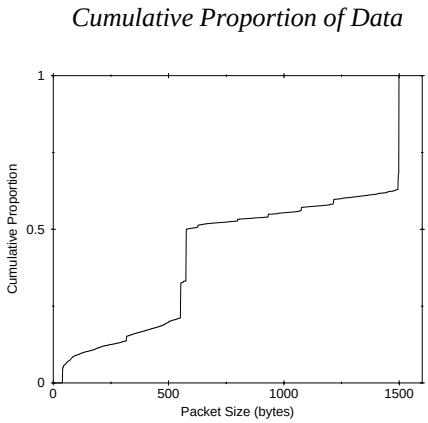

Figure 20: Data distribution

responsible for. From this it can be seen that while maximum sized, 1500 byte packets only make up approximately 10% of packets, they carry approximately 40% of all data.

5 Voice detection and analysis

In the proposal for this project, it was expected that the most work would focus on identifying all different types of UDP traffic, and pick out as much VoIP data as possible. This would require an understanding of as many different VoIP packages as possible, and attempt to identify any data they create in the available traces.

This unfortunately has not been the case. As represented by this report, a large amount of time has been spent understanding the use of passive measurement systems, and the issues that arise in this area. This in effect has resulted in a lower level study, of the systems that underlay any trace based analysis.

Because of the restriction in time, analysis of VoIP traffic has been restricted to one protocol. This protocol is called H.323 and is a standard created by the *International Telecommunications Union (ITU)* [ITU99]. The standard, titled "Packet-based Multimedia Communication Systems", describes a general method for real-time communication over a packet-based network such as IP. The standard allows an H.323 entity to provide real-time audio, visual and data services. Support for audio is mandatory however, while visual and data services are optional.

Identifying only one protocol for VoIP may not be as restrictive as could be thought. Since its introduction in 1996 H.323 has become widely supported, with both large software vendors such as Microsoft and hardware vendors such as Cisco, shipping products based around it. With such large support, many older proprietary systems are now either including support for, or converting to H.323. This means that while detecting H.323 only will not provide all possible VoIP data, it should provide sufficient amounts for simulations to use.

This section first details the methods for finding H.323 sessions that have been used in this project, then gives some information on the possible H.323 sessions that was found on the NZIX traces.

5.1 Traffic identification

TCP and UDP both use port numbers to identify the application that 'owns' the traffic that arrives. Every time a TCP or UDP port is requested, a unique port number will be returned. TCP/IP uses a well known port system to provide

most services. All of the well known, historic TCP/IP applications, such as WWW, FTP, SMTP and DNS, have an assigned port number that they will listen on. When one of these services is started, it will request the specific port it should run on, and open that port to listen from. When a client attempts to open a connection to that service, they will request a port to connect out on. This port will be a random unassigned number. Then knowing the address of the machine, and the default address for the service, they can open a connection.

It is unusual, but not impossible for two machines to violate this port system. If a machine knows the port a service is running on, it can connect that port, and not need to assume the default port number. All of the 'well known' port numbers are below 1024, and are protected from being opened by any non-privileged services. It is therefore unusual to find a service running in the restricted port range, but not on the specified port. It is not so uncommon to run a service on an unrestricted port number, that does have a specified restricted port. For example, a WWW server is specified to run on port 80. It would be extremely unusual to run a web server on port 25, the default port for e-mail services. It is possible however that a WWW server could run on any high number port (greater than 1024), if for instance a machine is running two servers, one for internal access and one for external access.

In header trace files, only the source and destination port number is available, no user data. If one of the ports is a restricted port number it is normal to assume that the data carried in the packet is for that service. If both port numbers are high however, only in restricted cases can any conclusion be drawn as to the type of data carried in the packet. As we will see, H.323 is an area in which this is the case. None of these methods are perfect however, and there is always an element of guess work in identifying traffic from trace files.

5.1.1 H.323 detection

H.323 is a generic standard that incorporates multiple other ITU standards to form a complete system. The specification also allows for implementations over multiple network technologies, each implementation is specified in a separate document. For this project we are only interested in the details for H.323 over IP. All ITU specifications also have to be paid for to obtain them. For these

reasons the detection systems that have been used for this project have come from a practical investigation, without in-depth research into the protocol itself. However all IP implementations of H.323 have to be able to interact with each other, and so correctly identifying the signature of one H.323 application should allow detection of all IP implementations of H.323.

As H.323 is a new protocol, it does not run using the privileged section of port numbers. It does however use the same well-known idea. When a H.323 application is started, it will request two specific TCP port numbers to listen on. These are ports 1503 and 1720. These ports are used for call setup and call control, one area of VoIP applications that do need reliable delivery. A H.323 application that wishes to connect to another H.323 user, will connect to that machine on both ports 1503 and 1720. Using these two connections the applications negotiate UDP ports to use for data transfer.

H.323 specifies the use of the *Real Time Protocol (RTP)* [SCFJ96] for data transfer. This is a general purpose protocol for real-time IP applications. RTP uses two UDP ports, one for data transfer, and one for RTP control information. The data transfer port is defined to be an even port number, and the control port number will be the successive port number. The data port will have large numbers of small, and often fixed sized packets flowing over it. The control port will have much lower data volumes, and will not necessarily have regular arrival times or sizes.

The actual port numbers that are negotiated by H.323 are indeterminable, but conforming to the RTP standard, data will be carried on an even port and control information on the next port. Using both this property and the well-known ports, 1503 and 1720, it is possible to attempt to detect H.323.

The process used in this project takes two steps. The first step uses a program written for this task. It takes a `libpcap` file, and looks for pairs of machines that both communicate on ports 1503 and 1720. It then prints the IP addresses of these machines.

The second step uses `tcpdump` to filter the traffic between these two machines. By human inspection it can be easily seen if there are the indications of RTP data between these two machines. Improvement of this second step of detection, to a system that can be carried out by a computer is one section of the future

work possible on this project.

5.1.2 H.323 traces from NZIX

Using this system on NZIX resulted in 22 traces identified as possible H.323 connections. The first step identified 71 sessions from the 211 NZIX traces. Of these 71 there were multiple 'port scans' where a host attempts a connection to every port on a destination machine. This will result in connections being attempted on both port 1503 and 1720, misleading the detection. The second stage easily picks up these as not H.323 connections.

One other circumstance occurred, where a connection was seen on both ports 1503 and 1720, but no more data passed between these hosts. It is possible, although not confirmed, that this could be caused by an attempted H.323 connection that was refused by the destination machine. These were also discarded as not being H.323 connections, as it was the UDP data in particular that was wanted.

5.2 Voice trace characteristics

All of the 44 hosts involved in the 22 traces were unique. There were no duplicated IP addresses. This means that we have no definite indication that there any regular H.323 users on this link. This could mean two things, there are no regular H.323 users or we are not detecting them. As the traces are only run for 20 minutes out of 1440 minutes in a day, the likelihood of missing repeated H.323 conversations is large. Also as the IP addresses are encoded there is no way of telling if a user is on a dynamic IP, where their IP address changes ever connection. With unencoded IP addresses it would be possible to tell if multiple addresses all come from the same ISP, indicating that this could be a possibility.

Without knowledge of repeated H.323 use, all traces could be simply experimentation, and it is possible that the quality of the results are not sufficient for a user to persist.

Much longer trials need to be run, possibly over days to weeks to investigate properly the level of H.323 use on NZIX. Currently this is not possible due to storage constraints, and so a move must be made to real-time detection of H.323. This is discussed further in the future work section (§ 6).

```
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C=  4, HL= 5, TL=   88
0.5.211.254:49607 ->-> 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=   92
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C=  6, HL= 5, TL=   88
0.5.211.254:49607 <-<- 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=  100
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C= 10, HL= 5, TL=   88
0.5.211.254       ->-> 0.0.251.56       RSVP C=  1, HL= 6, TL=  160
0.5.211.254       <-<- 0.0.251.56       ICMP C=  1, HL= 5, TL=   60
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C= 28, HL= 5, TL=   88
0.5.211.254:49607 <-<- 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=   76
0.5.211.254:49607 ->-> 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=   72
0.5.211.254:1843  <-<- 0.0.251.56:1503  TCP  C=  1, HL= 5, TL=  136
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C= 10, HL= 5, TL=   88
0.5.211.254:1843  <-<- 0.0.251.56:1503  TCP  C=  1, HL= 5, TL=  136
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C=  1, HL= 5, TL=   88
0.5.211.254:1843  ->-> 0.0.251.56:1503  TCP  C=  1, HL= 5, TL=   40
0.5.211.254:49607 <-<- 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=   80
0.5.211.254:1843  ->-> 0.0.251.56:1503  TCP  C=  1, HL= 5, TL=   40
0.5.211.254:49606 ->-> 0.0.251.56:49608 UDP  C=  2, HL= 5, TL=   88
0.5.211.254:49607 <-<- 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=   80
0.5.211.254:49606 ->-> 0.0.251.56:49608 UDP  C=  1, HL= 5, TL=   88
0.5.211.254:49607 ->-> 0.0.251.56:49609 UDP  C=  1, HL= 5, TL=  104
0.5.211.254:49606 ->-> 0.0.251.56:49608 UDP  C= 18, HL= 5, TL=   88
0.5.211.254:49606 <-<- 0.0.251.56:49608 UDP  C= 11, HL= 5, TL=   88
```

Figure 21: H.323 packet flow

The characteristics seen in each of the available traces are all reasonably similar. To understand the flow of traffic between the two machines, a program was written to print out a text representation of this flow. Sample output of this can be seen in figure 21. Printed in one column is one IP address followed by the port on that machine. Next is an indication of the direction of the packet and the other host's IP and port. Then the protocol is listed, count field, the header length and total length of the packet in bytes. The count field is the number of identical headers that were seen. This compacts long flows of identical packets to a single line. It can be seen that this occurs on a regular basis with the UDP data channels, seen on the even UDP port numbers. It can be seen in this case that both sides are using the same data packet size, 88 bytes. This section does not represent the 'normal' output of this program. This section has been chosen due to the large number of different packets and protocols seen in a short time. More often all that is seen is large flows of UDP data packets between the two hosts with some RTP control data also.

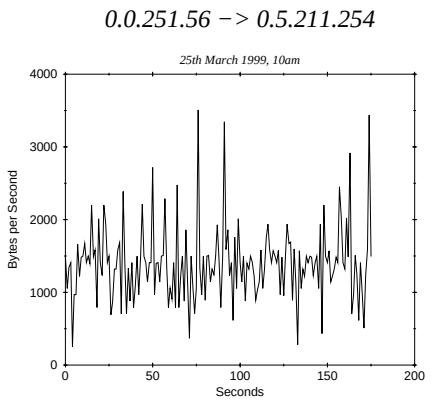

Figure 22: UDP data rates

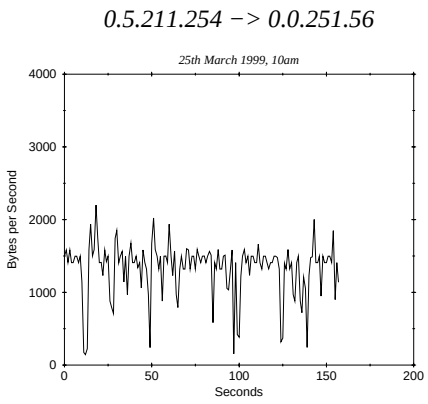

Figure 23: UDP data rates

In figures 22 & 23 the data rates for each of the directions are plotted. These rates only include the UDP traffic seen between these two machines. The data rates and packet rates will both produce approximately the same graphs as all data is sent in fixed sized packets, and only the control information is slightly different.

The data rates shown on these graphs are achievable by a 33.6kbit per second modem link, the lowest common Internet connection speed in use today. It can be seen that one graph is more erratic that the other, a feature that is unlikely to be reliably explained without more knowledge of each of the users.

Two more graphs are included to provide another example. These, shown in figures 24 & 25, show that it is not a requirement to have both directions of traffic aim for the same average bit rate. It is obvious that both of these directions have a fixed mean bit rate that they are aiming for but these are very different for each direction. Again the bit rates are both relatively low, with both just being within range of a standard modem connection.

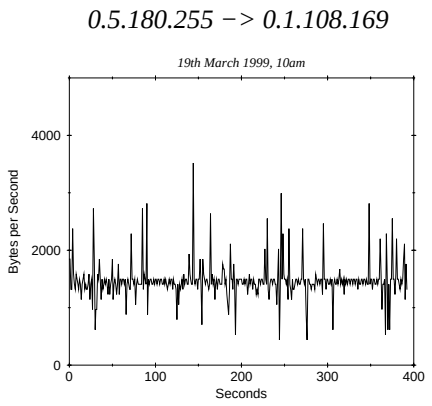

Figure 24: UDP data rates

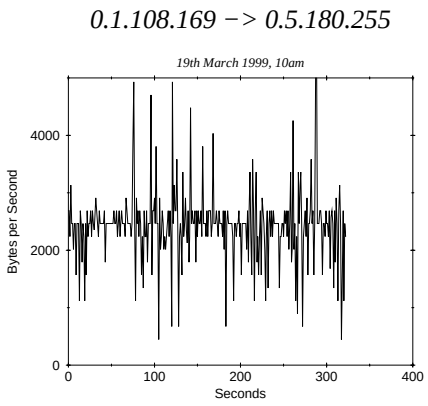

Figure 25: UDP data rates

6 Future work

There are multiple areas that this project can be continued from. The most obvious area is to conduct simulations using the H.323 sessions found in the NZIX traces. This step does not require large amounts of work as both the simulator and the traces now exist. This will then be able to provide a preliminary answer to the questions this project has been aiming towards. However there are a number of other areas that can be developed on to assist in a more accurate and complete simulation run.

The first extension is a more in-depth study into the H.323 protocol. This will allow verification of the methods used here, and may also provide more reliable ways of detecting H.323 traffic. One other significant area should be looked at. It has been assumed in this project that the added overhead of congestion control for an application is not worth the possible benefits. This is not necessarily the case. More study should be done on investigating if H.323 supports applications responding to congestion, and if any actually do this. A study of this type would verify the accuracy of the current methods to create traffic models.

Other improvements could extend the traffic identification to allow machine identification of RTP traffic. This would first be in the context of H.323. If two machines communicate on the appropriate ports how can it be determined if RTP traffic was also seen between them. Then a more general purpose system for finding RTP traffic without any other indications could be attempted. This would be made difficult because of the fact that the RTP ports are adjacent.

As many UDP stacks allocate port numbers in sequence, it is quite likely that adjacent UDP ports would be seen between hosts. To achieve RTP detection it may require more artificial intelligence to detect the properties of RTP such as regular fixed sized data packets and irregular control packets.

Once detection for H.323 has become more reliable, it may be worth investigating a real-time method for this detection. This could run on a DAG card consistently, looking at all traffic on a link, and save only traffic that appears to be H.323 sessions. Moving away from the requirement for trace files to real-time detection would allow much larger volumes of traffic to be inspected. As the storage requirements are for only the H.323 sessions, and not the entire trace file, many more sessions could be stored at one time.

Limiting all work done on VoIP detection to H.323 traffic obviously omits any other VoIP systems in use from simulations. Work could be carried out in investigation of the signatures of any other VoIP application that could be in use. One method for this has already been suggested. The use of RTP is not restricted to H.323 systems and a generic detection for this could enable larger volumes of data to be found. However, as already mentioned, H.323 has become a very popular protocol, and there seems to be no obvious reason currently why this may change. Extra work on detecting other protocols may provide limited gains as more and more applications start to use H.323.

7 Conclusion

Due to an underestimation of the complexities involved in using passive measurement techniques, this study has focused at a much lower level than originally intended. Through the study of these systems a number of points have been raised.

The most obvious conclusion that can be drawn from this project is a feel for the difficulty involved in passive measurement. It has been stated multiple times that this simple concept presents some very large problems. It has been seen that different network technologies each create a set of new problems adding on to a core set of difficulties and limitations inherent to passive measurement.

Problems such as timestamps are slowly being corrected by new generations of custom hardware solutions, assisted by technologies such as GPS. In other areas, such as increasing data rates, the problems are continuing to get worse, and new ideas and methods for dealing with these are required.

This project has also encountered the lack of any standard platform on which analysis can be carried out on. This has been seen in the requirements for file converters and the multiple interpretations and various accuracies for such variables as timestamps. It is hoped that the hardware improvements, such as being created by the DAG group, will provide the necessary support software require for high accuracy measurement. When combined with CoralReef, it will be then possible to write analysis software that will portable over multiple network technologies without change.

Once available these systems can not only be used to create new, accurate data sets to work with but also verify the accuracy of the existing traces. This will provide a way to state any current results to an appropriate level of accuracy.

The analysis techniques that have been created and used in this project can still be applied once such systems as CoralReef, supported by DAG hardware, are in use. These analysis techniques, while not as comprehensive as would have been liked, achieve the goal of detecting VoIP traffic with no more information than traces of packet headers.

This project has therefore achieved the main aim of the proposal, to work towards simulation to investigate the effects of VoIP on existing traffic. As seen in section 6 there are many areas that can be worked on from this project to

further understand these effects. This report provides a base on which future projects can be built upon to finally provide reliable answers to the questions this project asked.

References

[ACGW95] M. Arlitt, Y. Chen, R. Gurski, and C. Williamson. Traffic modeling in the ATM-TN TeleSim project: Design, implementation, and performance evaluation.". In *Proceedings of the 1995 Summer Computer Simulation Conference*, Ottawa, Ontario, July 1995.

[CAI99] CAIDA. http://www.caida.org, October 1999. Cooperative Association for Internet Data Analysis, San Diego, USA.

[CR99] CoralReef Software Suite. http://www.caida.org/Tools/CoralReef, October 1999. CAIDA Research Group, San Diego, USA.

[DAG99] DAG Group. http://dag.cs.waikato.ac.nz, October 1999. Computer Science Department, University of Waikato, New Zealand.

[ITS99] University of Waikato Network Overview. http://operations.waikato.ac.nz/mrtg, October 1999. ITS, University of Waikato, New Zealand.

[ITU99] International Telecommunication Union (ITU) Home Page. http://www.itu.int, October 1999. International Telecommunication Union.

[MOA99] NLANR Network Analysis Infrastructure. http://moat.nlanr.net, October 1999. National Laboratory for Applied Network Research, San Diego, USA.

[MPC98] Andrew McGregor, Murry Pearson, and John Cleary. The effect of multiplexing HTTP connections over asymmetric high bandwidth-delay product circuits. In *Proceedings of SPIE, Internet Routing and Quality of Service*, Boston, Massachusetts, November 1998.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederic, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *Internet RFC 1889*, January 1996.

[Ste97] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. *Internet RFC 2001*, January 1997.

[WAN99] WAND Group. http://atm.cs.waikato.ac.nz/wand, October 1999. Computer Science Department, University of Waikato, New Zealand.

A Glossary

10/100base* Multiple standards for different media options and speed grades of Ethernet networks.

AAL-5 ATM Adaptation Layer–5. An encapsulation layer to enable large data packets to be split into ATM cells.

ATM Asynchronous Transfer Mode.

CAIDA Cooperative Association for Internet Data Analysis

CRC Cyclic Redundancy Code. A very reliable error detection system often used in networking.

DAG Research Group at the University of Waikato, concerned with hardware monitoring solutions. Consult [DAG99] for an explanation of the acronym.

DNS Domain Name System. Online service used to map human readable names to IP addresses.

FTP File Transfer Protocol. Common way of transferring files across an IP network.

GPS Global Positioning System. International guidance system, that can also provide a distributed, highly accurate clock system.

IP Internet Protocol. The base protocol of the TCP/IP suite of protocols. Provides the basis of all Internet connections.

ISP Internet Service Provider. A company providing Internet access, normally to homes and small business.

LAN Local Area Network. A physical network technology designed to span short distances.

MOAT Measurement Operations and Analysis Team. A research team based in the USA.

NLANR National Laboratory for Applied Network Research. The parent research group to MOAT.

RTT Round Trip Time. A measure of the delay between hosts. Consists of the time taken for a single packet to leave one machine, reach the other and return.

SNMP Simple Network Monitoring Protocol. A protocol used to monitor the health of, and provide some statistics from, routers and hosts on a network.

SONET Synchronous Optical Network. A high speed optical fiber transport standard.

TCP Transmission Control Protocol. The reliable, connection orientated transport service on which most applications depend. Sits on top of IP.

UDP User Datagram Protocol. Thin layer on top of IP to provide multiple application support on one host.

WWW World Wide Web. Large information service that allows a user to browse information using a hyper-link system.