Notes on Setup of Hosts and Dummynet Machine for Simulation of Satellite Transfers and World-Wide-Web Experiments

Allan Bruce

29th August 2001

University of Aberdeen, Electronics Research Group
Allan Bruce

Table of Contents

1 Introduction

A small practical exercise was undertaken to observe properties of worldwide-web transfers, and how to set up a simulated satellite link. This exercise comprised of some relatively simple tasks including learning basic Unix skills. These tasks provided basic experience in using some useful network analysis tools. The project involved web transfers, particularly those over satellite networks. Ways in which these slow, lossy networks can be improved will be discussed. One way in which these transfers could be improved was thought to use persistent-http connections – a default part of the HTTP/1.1 standard. This is an extension of the keep-alive connection as found in HTTP/1.0. This was looked at in some detail and some conclusions drawn.

This report examined various HTTP standards, particularly the way they are implemented and interact with the TCP/IP protocol stack. Several tools were needed to analyse the transfers taking place. These tools included tcpdump[1] and snoop[2]. To make sense of these results, tcptrace[3] and xplot[4] were used to show, graphically, the transfers against a timescale. A timing diagram was also drawn by hand to help understand the transfers.

For satellite links, a simulation program known as dummynet[5] was used. The setup and configuration of this is detailed in Chapter 5. In the next Chapter, Unix skills are discussed as this Operating System gave easiest access to network packets transferred.

2 Basic Unix Skills

The tools required for this practical were all programs intended for Unixoid systems. Some basic knowledge of the Unix operating system was required to use these tools successfully. Knowledge required included the understanding and navigating of the directory structure and network topology used. Computers used throughout this exercise were Douglas and James. Douglas used Solaris 2.6 on a Sparc Processor and was connected via an Ethernet LAN running at 10Mb/s to all other machines. The /usr/local/ directory was stored on a single machine so a directory was made (using the 'mkdir' command) on Douglas for additional programs required. This directory was simply called /usr/local-douglas/. The reason for doing this was so that tcpdump and snoop would collect local traffic and not traffic on a different area of the network. All log files were stored within a 'logs' directory in the authors homespace. A relevant README file describing each of the log files was kept.

2.1 Navigating Unix

To use Unix, several basic commands were required including those to navigate the directory structure. Commands included:

ls	Lists all files in current directory
cd	Changes directory to that specified
rm	Removes file specified
cp	Copies files
mv	Used to move files or rename them
more	A basic text editor (also used 'vi')

Once the OS could be navigated successfully, the next obstacle was to understand how Unix carried out commands or executed programs. For somebody to view files and directories or execute programs, they must have permission. Permissions may be set by the owner or the superuser. Generally the owner is not available to change permissions, therefore superuser mode is required. In this mode, care must be taken as severe damage may be done to the OS. To become a superuser, the 'su' command was used and a valid password was required. To change permissions, the 'chmod'

command is used along with an octet number and the filename. This sets permissions for root, owner, and guests. A book[6] was used for help with the directory structure and various commands.

2.2 Advanced Unix Techniques

A useful technique used in Unix is called piping. This links multiple commands together, allowing users more flexibility. An example of a pipe used is shown below

ls | more

The vertical line is called the pipe. In this example, the user requested to view the files contained in the current directory and used the 'more' editor to view them. This may be useful when a directory contains many files – too many to fit on one screen. Another useful technique used was the 'grep' command. This filters lines of text files to output only ones those specified. An example of this is shown in Chapter 3.2.

To configure Unix for a user, config files must be modified. The shell used on Douglas was the basic 'csh', so the configuration file was '.cshrc'. This file contains details about default paths and lots more.

2.3 Program Installation

The final piece of knowledge required was to know how to install additional programs. Programs are often downloaded as source code within compressed archives. The types downloaded for use in this project were .tar.gz. These files are tar backups (which keep file/directory permissions and ownerships intact) and then compressed using the gzip algorithm. To decompress these archives, two methods can be used depending on which commands are available. The simplest method is to merely type

tar xzvf *filename*.tar.gz (x – eXtract files, v – Verbose output, f – output to File, z – gunZip file first)

This creates a directory called *filename* and extracts all files in the archive to this directory. Unfortunately, the tar command on Douglas did not support the z option

(the option to gunzip the tar file). This meant that to extract the files, two commands had to be used.

gunzip *filename*.tar.gz

and

tar xvf *filename*.tar

The first command extracted the gzip file to a tar file (*filename*.tar), and the last command extracted the tar file to a directory called *filename*.

Once the source code files had been extracted they then needed to be compiled. Files need to be compiled if binary executable files are not available. Generally source code files only are available due to the many different kernel versions and directory structures available. Compiling source code files produces a binary file to be executed with the appropriate kernel version being used. To compile the programs, two commands were used. The first command used was the 'configure' command, typed as follows

./configure

This made a 'Makefile' specific to the machine being run on. This Makefile contained information about the type of compiler available and directory structures as well as many other variables. Once a valid Makefile exists, the 'make' command was executed to compile the source to give binaries. Some programs require options added to the make command, such as

make depend

make install

make all

These make file dependencies, place a copy of the binary into the system path or compile many files at once respectively.

In this chapter, we have looked at how to use the Unix operating system and how to install additional programs. In the next chapter, additional programs for network analysis are discussed.

3 Tools used

The tools used in this practical were tcpdump[1] and snoop[2]. Tcpdump is a freely available program as source code, so compiling was necessary. Snoop is a utility provided free with Solaris Operating Systems. Both programs capture packets transferred on a network interface. These packets are only displayed if they are intended for the machine that tcpdump or snoop is run on unless the interface is run in promiscuous mode. This mode allows the interface to capture all packets on the network passing the interface. Both programs sound as though they do the same job but they have subtle differences. Tcpdump operates at layer 3 of the OSI reference model (the network layer) and as such only captures TCP segments or UDP datagrams using the IP protocol. Snoop, on the other hand works on multiple layers of the OSI reference model and captures many types of packets. A typical webpage packet uses HTTP->TCP->IP->Ethernet (If on an Ethernet LAN) protocols. Snoop is capable of capturing the headers of all of these for analysis. The reason tcpdump was also used was because snoop doesn't offer as much detail as tcpdump for TCP, the main protocol studied in this practical.

3.1 Tcpdump

Tcpdump was downloaded from the internet (http://www.tcpdump.org) along with a necessary library called libpcap. This library was required to filter packets for use with tcpdump. Libpcap was installed by unarchiving it, configuring then compiling as explained in section 2.3. In this case, tcpdump would not install easily on Douglas as the superusers configuration file was invalid. A binary copy was obtained from another Sparc machine using the same operating system within the network. Tcpdump was installed with ease on the authors home machine running RedHat Linux 7.1[7] using the method detailed in section 2.3.

To run tcpdump the following was typed at the prompt

tcpdump *options expression* > *filename*

There are many options for use with tcpdump, but only 5 were used in the duration of this practical, these included:

-s96	this option captures the first specified number of bytes of the TCP segment. 56 bytes is the default as this covers the TCP header, but 96 bytes were used to capture TCP header options also
-S	this option prints absolute TCP sequence numbers instead of relative ones. This makes analysis of transmitted segments easier when multiple connections are in use
-tt	this option prints an unformatted timestamp on each line rather than in human readable form. The reason for this was because xplot[4] prefers this format
-n	this option prints IP address instead of domain names. This is much faster as tcpdump does not need to look up addresses for each capture
-w filename	this option prints packets in raw format. This was required for a tcptrace input file.

Expressions in tcpdump are for filtering packets to capture only relevant ones. There are many options available to the user here but the ones used in the practical were

port domain or 80 and host *hostname*

Port 'domain' is an alias for the known port used for DNS queries so that they can be captured by tcpdump. The program also captured packets which are intended for or transmitted from *hostname* and use port 80 (the default port used for web-servers). James was used for transfers as the Netscape[8] browser was not installed on Douglas. The browser was loaded on James and the X-windows screen was diverted to a network connection. This was done using the following script

rlogin james

<insert password>

setenv DISPLAY Douglas:0.0

netscape

This makes it appear that Netscape is running on Douglas.

All output filenames were recorded and logged with the options used and web-server used. All logs were kept in /home/allan/logs/

An example tcpdump command used is shown below

tcpdump –s96 –S –tt –n port domain or 80 and host james > /home/allan/logs/001.tcpdump

These output options were used for analysis with 'xplot' using a 'perl' script for analysis. For analysis using 'tcptrace'[3], the –w option was required. The output of tcpdump shows the time of packet, sequence numbers, acknowledgment numbers, flags and window sizes. These TCP properties are required knowledge for understanding transfers. A detailed description can be found in TCP/IP Illustrated, Volume 1[9]

3.2 Snoop

At the same time as running tcpdump, another terminal was opened to run snoop, thus capturing the same packets at the same time. The reason for this was to analyze the types of data transferred. Snoop is executed in the same way as tcpdump however the options are different. The options used were

-ta this option prints a timestamp beside each captured packet

-v this option prints a verbose output – quite extensive

The expressions used were of the same type as tcpdump however the format was slightly different. Snoop requires

port domain or port 80 and host james

Notice the inclusion of the second 'port'. An error is produced if this additional 'port' is omitted. The verbose output means that captures were very large so a pipe and grep were often used to filter relevant inforation. This was done as follows

snoop *options expression* | grep *wantedfilter* > *filename*

The grep command is case sensitive so *wantedfilter* must be typed correctly to filter correctly. Sometimes the 'get requests' only were desired, therefore 'grep GET' was used. Care must be taken here to ensure that necessary information is not omitted. If in doubt, snoop was left to output the full capture. Grep could then be used on the filename at viewing time.

The output form of snoop depends on which options used. With the options used in this practical the output had a form of showing HTTP get request, referrer, connection type, user-agent, host and accepted formats. These HTTP protocol properties are discussed in detail in TCP/IP Illustrated, Volume 3[10].

In the next chapter, web transfers are captured using the tools described above. These web transfers are analysed and some of their properties discussed.

4 Web Transfers

A basic web-page was required to analyze transfers, so Netscape Composer was used. A simple page was made with some text, a jpeg background image and 3 gif images. This gave a total of 5 objects for transfers with a web server. The webpage and its source can be found in Appendix A1.

To start with, this web page was published on a private ISP web-space and a transfer was captured using tcdump. The transfers were analyzed paying close attention to timing and sequence numbers. A timing diagram of this can be seen in Appendix A2. Some acknowledgements were delayed but no segments were retransmitted. The reason for the delayed ACKs was caused by some segments arriving out of order, this is due to a segment being delayed in transport. This is common as IP packets are not sent along the same path - one router may have been congested resulting in a small delay on forwarding a packet.

Mostly, 2 segments are ACKed at once reducing the ACKs by half, however at one point 3 segments were ACKed. This is normal TCP behaviour; the receiver should ACK at least every other full sized segment. At times the client received segments quicker than it was processing them, the server paused transfers due to the receivers window size being reported as zero. The client then processed segments and acknowledged 3 at once and increasing the window size to allow transfer to resume.

Another interesting point no notice was that there were 3 or 4 TCP connections opened for transferring all 5 objects. When 4 connections were transferred, it was observed that the smallest image was embedded with the html page. This was found by examining the sequence numbers in each connection and relating the difference to the object sizes. For example, each connection had an initial sequence number and a final one. The difference between these gave the total number of bytes transferred including overheads from IP protocol. Once the overheads were removed, the actual number of bytes transferred was found. This could then be compared to the object sizes and the specific object may be found. This is a recognised feature of the HTTP/1.0 standard (HTTP/0.9 would use a separate connection for each object resulting in 5 connections. Another method of finding what is transferred is to look

at the 'get' requests within snoop. This method verified the proposed objects using the sequence number method.

Another property noticed was that with the HotJAVA web-browser, there were usually 2 connections open at the same time although only 1 was receiving data at a time (data was sent and the connection remained open although no more data was transferred through it). It appears that either a connection was not closed until another one was required, or the delay of a closed connection was holding up the opening of another. It was thought to be the latter as connections remained open for several segment transfers although were not being utilized at all*.

Some data was transferred from the client to the server. Snoop was run and the transfers were undertaken again. It was found that these transfers from client to server were 'get requests'.

4.1 Worldwide Web Transfers

The example web page (see Appendix A1) was published on several web servers worldwide. These web servers had different properties, such as path delay and hops between client and server (see Appendix A3). Several interesting properties were recorded.

Some connections were closed by the server as expected (immediately after no more data is required, a FIN is sent) however some connections remained open for a long time afterward. The connections that remained open had two different ways of closing the connections. One way, was that no FIN flags were sent and a RST flag was sent after a timeout. In the other method, a connection remained open until the client required other connections, for example looking at another web page. In this method, the client sent FIN flags but no ACKs were returned by the server. The client kept on resending, but no ACKs. Eventually when the client required the connections, a RST flag was sent to the server, requiring no acknowledgement and hence closing the connection. This is a property of keep-alive connections with

* This web browser was the default browser installed on Douglas. The version is not known. It is unknown why only 1 connection was actively transferring data and why there was a limit of two connections opened at one time.

HTTP/1.0 although headers within the protocol set timeouts, they were not always obeyed.

Netscape 6.1 was installed on an iMac running MacOS 9.x. This uses HTTP/1.1 with persistence and keep-alive connections. With these keep-alive connections, it was found that they did adhere to the conditions set in the http header. The two conditions were 'max' and 'timeout'.

It was also noted that one connection to a site contained a hop delay of over 600ms. It was thought that this could have been a satellite hop.

4.2 Viewing Results

One method used to view the results was running tcpdump using the –r option which reads a raw tcpdump capture and presents a readable form. The standard readable form also used the –s96, –S, –tt, and –n options as described in Chapter 3.1. These readable forms were input to tcptrace[3] and several plots were produced. Tcptrace is a network analysis tool written by Shawn Ostermann. It takes inputs from several packet capturing tools and produces sets of graphs of common properties of the transfers including the MSS, window size, RTT and sequence number plots. These properties are discussed in detail in TCP/IP Illustrated Volume 1[9]. Tcptrace was executed by the following command

```
tcptrace –S filename
```

This takes input from a tcpdump readable format as discussed above. These plots were then plotted graphically by using xplot[4]. With these simple transfers, these plots were not very informative as each connection had only a few segments of TCP data. An example ftp transfer of a text file was made and tcptrace was run to view a transfer of several hundred segments. This can be seen in Appedix A4. This allowed better use of the program as retransmissions and delays were much more apparent. The plot was relatively linear and the slope gave the throughput of the connection. There was one retransmission occurrence. This was seen by a negative slope between two points. Here the sequence number dropped instead of increasing due to the retransmission.

The round trip times (RTT) were recorded for all web transfers using 2 different connection styles. One connection was on an Ethernet LAN with access to the internet via a high speed 100Mb/s link. The other was via a V90 class modem using an ISP to access the internet. The following table shows these results

Site	RTT (ms) High Speed	RTT (ms) V90 modem	RTT after connection setup (ms) High Speed 100Mb/s	RTT after connection setup (ms) V90 modem	Tcpdump output filename
www.erg.abdn.ac.uk	6	347	12	887	x-07-01-01-tcpdump
www.yodadrinkslager.screaming.net	34	378	42	847	x-07-01-02-tcpdump
www.smirnoff.fresserve.co.uk	34	366	64	761	x-07-01-03-tcpdump
pegasus.phys.uh.edu	180	364	204	766	x-07-01-04-tcpdump
www.cs.pdn.ac.lk	1150	827	2187	1294	x-07-01-05-tcpdump

Table 1: Site Statistics

Each RTT was calculated 3 times and an average value obtained. Table 2 shows the server software and operating system used on each server

Site	Server	OS
www.erg.abdn.ac.uk	Apache 1.3.11	Solaris
www.yodadrinkslager.screaming.net	Zeus 3.3	FreeBSD
www.smirnoff.fresserve.co.uk	Microsoft IIS 5.0	Windows2000
pegasus.phys.uh.edu	Apache 1.3.6	Linux 2.2.6
www.cs.pdn.ac.lk	Apache 1.3.20	Solaris 2.7
www.burbank.co.uk	Microsoft IIS 4.0	Windows NT4
www.test.globalweb.co.uk	Apache[†]	Linux

Table 2: Server Software and OS

As can be seen from the results in table 1, a site may have a lower RTT on the slow link which is not expected. This is affected by the route taken, i.e. how many hops and different geographical route. Table 2 is useful to find out if the web servers

[†] The version of Apache and Linux was undeterminable by the website used (www.netcraft.com) for finding software and OS versions

support HTTP/1.1. All of them did but Microsoft IIS 4.0 does not use keep-alive connections. Table 3 below shows the sites and there corresponding traceroute outputs for both connection speeds.

Site	Traceroute output high speed	Traceroute output V90 modem
www.erg.abdn.ac.uk	5ms : 2 hops	undeterminable : 19 hops
www.yodadrinkslager.screaming.net	32ms : 15 hops	293ms : 12 hops
www.smirnoff.fresserve.co.uk	35ms : 16 hops	290ms : 11 hops
pegasus.phys.uh.edu	150ms : 27 hops	414ms : 21 hops
www.cs.pdn.ac.lk	1800ms : 24 hops	775ms : 24 hops
www.burbank.co.uk	122ms : 26 hops	417ms : 17 hops
www.test.globalweb.co.uk	48ms : 18 hops	332ms : 17 hops

Table 3: Traceroute Statistics

Site 5 had large delays from the high speed link, one hop containing over 600ms – this was thought to be due to a satellite link. The reason for the very low RTT on site 1 with the high speed access is due to the web server being located within the same VLAN. A possible explanation for the delays greater than 1000 ms and 30 hops is due to external packets being discarded by a firewall. The traceroute output details the total number of hops. These were then split into domain hops, shown in Appendix A3.

4.3 Perl Script

An alternative method for analyzing the web transfers was to use a perl script supplied with the xplot source code. This script produced sequence number plots for all connections on the same graph allowing the transfers to be analyzed more easily. To run the script, the following was typed at a prompt

perl tcpdump2xplot.pl *filename*

This then produced files with .xplot suffix. These could then be plotted using xplot as normal for analysis.

4.4 HTTP/1.1

Further web transfers were undertaken using Microsoft Internet Explorer 5.0[11] and Netscape Communicator 6.1. This allowed the transfers to use HTTP/1.1 connections if the server was capable. Results showed that with this standard, fewer TCP connections were required to transfer objects. This reduced overheads, and increased the throughput of the connection. Transfers took differing times to complete with Netscape and Internet Explorer. One reason for this is due to file types being transferred. In the HTTP header, the web client software lists several filetypes it can use and in a certain order of preference. The web server then transfers an object in this order meaning that Internet Explorer may download a *.png file whereas Netscape may download a *.tif.

In general it was found that Internet Explorer was the quickest for browsing. It displayed incomplete objects on screen from early on in the transfer. Netscape seemed to wait until files were almost completely downloaded before displaying anything on screen.

4.5 HTTP Headers

Several options are present in an http header. Originally, in HTTP/1.0 there were very few options but now there are many more implemented in HTTP/1.1. The most common options of use throughout this project were the keep-alive options. These are 'max' and 'timeout'. 'Max' sets the maximum number of 'get' requests within a single connection – usually set to 100. 'Timeout' sets the idle time in seconds after transfer that a connection will automatically close, usually 15 seconds. The close is usually initiated by the server utilising a 'standard' 4-way close as determined in the TCP protocol.

When a get request is made, the client asks for a specific object and HTTP standard. Details of which type of preferred connection exist here also. The header tells the server what version of browser/OS it is running and accepted object formats in order of preference. The client will also detail what character set it accepts and what language it is using.

On response to an HTTP request, the server replies with the HTTP standard it will use for the transfer and what object will be sent depending on those available (and compatible with the client). The header also contains the length of the object and details of connection types.

Some other headers are contained but it is not known what these were used for.

5 Dummynet

As a further study to the project, it was decided to simulate a satellite hop to analyse behaviour. To setup a satellite simulation, an Intel 486 DX2 66 workstation was used running FreeBSD 2.2.8-release. This distribution implements an IP firewall which is fully configurable. This firewall is known as dummynet. Set up of this is not difficult but many options are required before any transfers will work with satellite simulation.

To setup the dummynet, two network interface cards were installed on the dummynet machine and each connected to a different computer (see Appendix A5). First, the kernel had to be recompiled so the machine would act as a bridge (in this mode it would forward appropriate packets between interface cards). This is done by editing the kernel file. A copy of the generic kernel was made using

```
cd /usr/src/sys/i386/conf
cp GENERIC <newname>
```

The new copy was then edited and the following options added

```
options BRIDGE
options dummynet
options IPFIREWALL_DEFAULT_TO_ACCEPT
options IPDIVERT
options IPFW_DIVERT_RESTART
```

Once completed, the new kernel was then recompiled by

```
/usr/sbin/config <newname>
cd ../../compile/<newname>
make depend ; make ; make install
```

This procedure took around one hour to complete on the dummynet machine. A reboot was required for the machine to be used. Finally, two more options were required to enable configuration of dummynet. These were activated by

```
sysctl –w net.link.ether.bridge=1
sysctl –w net.link.ether.bridge_ipfw=1
```

With these options enabled, the dummynet was then set up for use but needed to be configured to allow simulation of packets.

To allow traffic flow between test machine and outside, the following must be used

ipfw add 100 pass ip from <test machine> to any

ipfw add 200 pass ip from any to <test machine>

Where <test machine> is the IP address of the machine being used to browse web pages and NOT the dummynet machine. (If traffic not intended for test bed is not wanted then the following command must be entered

ipfw add 65500 deny ip from any to any)

This allowed the dummynet machine to merely act as a bridge. Pipes were required to configure delays, losses and bandwidth limitation. For full configuration, the following was needed

ipfw add pipe 1 ip from <test machine> to any

ipfw add pipe 2 ip from any to <test machine>

This configuration is shown in Appendix A6. At the start it was thought that 4 pipes were needed and each pipe required a direction specified, in other words outgoing (out) or incoming (in). It could be set up in this way, but ambiguities arose when calculation of RTT was considered. Each pipe would be set up with a delay of X, therefore with 4 pipes the RTT was thought to be 4X but was actually found to be only 2X. Due to this, only 2 pipes were used.

Next, the pipes were configured using

Ipfw pipe N config bw XXXKbit/s delay YYY plr ZZZ

Where N was the number of pipe to be configured, XXX was the limiting bandwidth, YYY was the delay in milliseconds and ZZZ was the packet loss rate expressed as a normalised percentage, e.g. 4% was written as 0.04.

HTTP/1.1 uses persistence as default for transferring objects. Many people have discussed advantages and disadvantages of this. The next chapter looks at how persistence affects the transfer of web pages, looking closely at number of connections used and the amount of time taken to transfer all the data.

6 Persistent Connections

Most new web browsers and web server software support the HTTP/1.1 standard. Persistent connections are a default part of this new standard and as such, a web server administrator has to go to large lengths to disable this feature, increasing the likelihood of persistent transfers. Persistence is very similar to the keep-alive connections in HTTP/1.0 and in many cases is thought to be merely an addition to the old standard.

Persistence was developed specifically with small transfers in mind as most transfers require 3 segments for connection and 4 for disconnection resulting in a large percentage of overhead. With small objects on unique connections, the transfer rarely gets beyond the TCP slow-start stage (if TCP is used although it is by far the most popular protocol for web transfers). Persistence allows many objects to be transferred in a single connection. If many objects can be transferred on the same connection then overhead is reduced as less connection/disconnection segments are required. With a single connection for multiple objects, the TCP protocol can then extend beyond the slow-start phase thus network utilisation is increased.

There are many other features of persistence not discussed in this practical. These can be found in RFC 2616[12] or in the book 'Web Protocols + Practice'[13].

6.1 Testing Persistence

To see the effects of persistence, some basic tests were undertaken. These test included transferring the web-page from previous experiments and the BBC main webpage (http://www.bbc.co.uk). These transfers were made using the dummynet to simulate different scenarios. These scenarios included no delay, a fast satellite and a slow satellite[‡]. The satellites were given higher bit error rates due to the small size of webpage to ensure some packets were lost throughout the transfer.

The main reason for the development of persistence was to reduce times taken to transfer web pages and therefore increase utilisation of the network medium. To test

[‡] A slow satellite was simulated with a 2000ms RTT, a bandwidth of 136Kbit/s and a packet loss rate of 4%. A fast satellite was simulated with a 1200ms RTT, a bandwidth of 544Kbit/s and a packet loss rate of 4%.

this, several of the test web-pages were transferred in each conditions mentioned above using HTTP/1.0 and HTTP/1.1 with persistence. Each transfer was carried out 3 times and an average transfer time obtained. The results of the slow satellite scenario can be seen below in table 4.

Site	Time taken to transfer using Netscape 4.7	Time taken to transfer using Netscape 6.1
www.smirnoff.freeserve.co.uk	6.687 seconds	6.026 seconds
pegasus.phys.uh.edu	8.996 seconds	9.103 seconds
www.burbank.co.uk	10.175 seconds	9.946 seconds
www.test.globalweb.co.uk	12.342 seconds	11.682 seconds

Table 4: Average Transfer Times of Slow Satellite Scenario

As can be seen from the table, transfer times were reduced by using Netscape 6.1. This was thought to be a direct result of using persistent connections. There is one exception to this – pegasus.phys.uh.edu transfers were quicker when Netscape 4.7 was used. This was due to one long transfer time when using Netscape 6.1 which affected the average considerably. This long time was thought to be due to many packets lost in this transfer (the PLR was consistent for all tests however dummynet uses a random process to determine which packets will be lost which will approximate to the specified PLR over a range of transfers). To get accurate results, these tests would have to be repeated many more times.

Another reason for the implementation of persistence is to reduce the amount of connections to reduce server load. It has been seen that the use of persistence is faster, so to ensure this is due to several objects being transferred down a single connection, a further test was carried out.

This test carried out transfers on the test web-pages again and the amount of connections per transfer was recorded. It was found that using persistence, each transfer used less connections. For no satellite simulation three connections were mostly required but sometimes only one was required. When the delay was increased, the amount of connections also increased. For the transfer of a single web-page over the slow satellite, often ten connections were required. Some of these connections were reset after a SYN packet was received from the server. It is unknown why this was the case.

Another test carried out included loading the BBC website to see how many connections were required over the different conditions. For no satellite simulation, 35 connections were opened and transfer took around 5 seconds. For slow satellite simulation, the number of connections increased to between 85 and 90 connections and transfer times took up to 72 seconds. This was much longer than Netscape 4.7 took, it required 88 connections but only 53 seconds.

Finally, one of the test web-pages was reloaded 10 times in quick succession to see how many connections would be needed overall. For no satellite simulation, it was found that only 4 connections were opened throughout the duration of this test. This is one of the features of persistence; although there is no more data required for this transfer, the hosts will keep the connection open for the transfer of more data if required.

Due to restrictions in time, these tests were not carried out as extensively as hoped but this sets a good foundation for some future work.

7 Future Work

One aim of this project was to setup a satellite simulation and observe some transfers of worldwide-web transfers. This looked at the use of persistence to reduce transfer times across satellite networks. Time did not allow extensive research in this area so future work can be recommended.

It is recommended to continue research on the use of persistence to reduce transfer times across these networks. It has been seen that transfer times can be reduced for small web-pages but larger transfers appear to take more time than without using persistence. This would include many transfers of the test web-pages and an average obtained. Once completed, a test to transfer much larger web-pages should be undertaken and an average obtained. This would give an idea of the advantages/disadvantages of persistence in a real scenario.

The use of persistence when using proxy connections would be another area of study which would be of benefit. Proxy servers are using persistence to allow users to have only one connection to them but transferring multiple objects.

8 Conclusion

An aim of this project was to use network tools to analyse transfers. This included installing and understanding how to run the tools. By using these tools, it is possible to see many breakdowns in transmissions and how connections are utilized. This leads the path for useful analysis of much more complex transfers.

Another aim was to be able to set up a reliable satellite simulation to carry out tests on. This was done using dummynet, an implementation of the FreeBSD operating system and configuring all necessary parameters. All properties of this dummynet are programmable, e.g. the 'packet loss rate' (PLR) and the delay. Once this was setup, transfers could commence and analyses undertaken.

Finally, the last aim was to look at various HTTP standards and how they affect transfers. This allowed several tests to be carried out to see if using certain standards resulted in quicker communication.

All of these aims were met and some conclusions can be drawn. The suite of tools were used successfully and offered very good methods to analyse transfers. The satellite simulation was set up and was found to be very stable and accurate. The use of persistence in HTTP/1.1 was looked at and found to have positive effect on some transfers. Further work has been recommended to take this last step further.

9 References / Bibliography

[1] tcpdump, http://www.tcpdump.org

[2] snoop, Sun Microsystems, http://www.sun.com

[3] tcptrace, Shawn Ostermann, http://www.tcptrace.org

[4] xplot , ftp://mercury.lcs.mit.edu/pub/shep/

[5] dummynet, FreeBSD, http://www.freebsd.org

[6] Teach Yourself Linux In 24 Hours, Sams

[7] Redhat Linux, http://www.redhat.com

[8] Netscape Navigator, http://www.netscape.com

[9] "TCP/IP Illustrated, Volume I",

[10] "TCP/IP Illustrated, Volume III",

[11] Microsoft Internet Explorer, http://www.microsoft.com/ie/

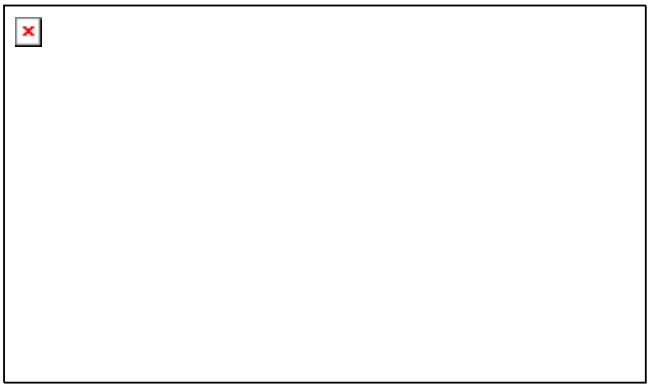
[12] http://www.ietf.org/rfc/rfc2616.txt

[13] "Web Protocols and Practice",

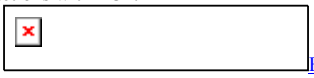
10 Appendix

Appendix A1:-Example Web Page (with no background)

Analysing the behaviour of http protocol variants



There are few variants of http protocol used in various commercial web clients and servers. The characteristics of these http protocols can be significant to specific network conditions. The main aim of this study is to understand the behaviour of http protocol variants and their interactions with TCP.

Study is being undertaken at the  Electronics Research Group. The data collected will be analysed and will be used to build a web traffic model in a network simulator. This is done as a partial requirement for a project evaluating the TCP protocol performance in next generation satellite systems.

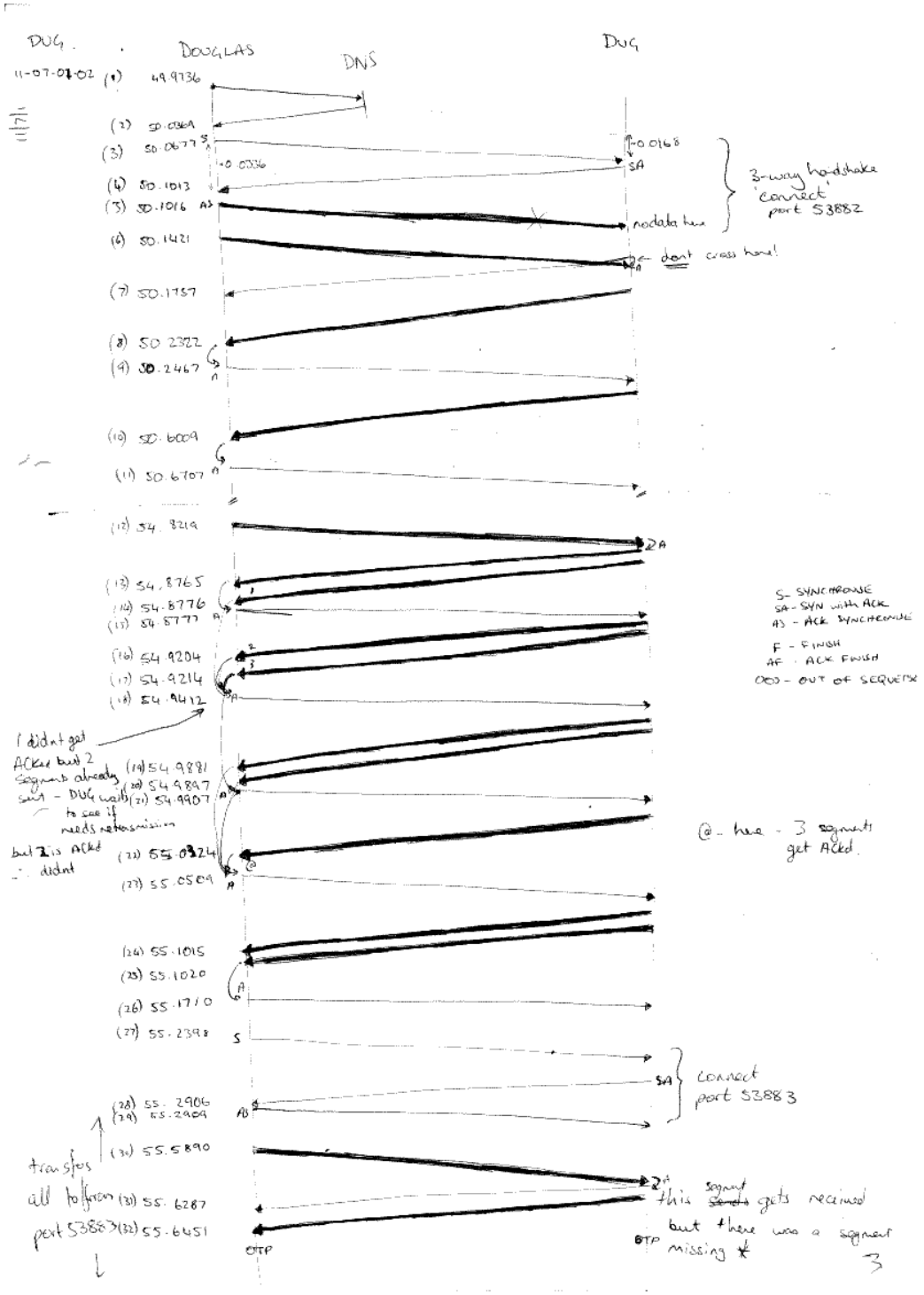
Some useful tools being used for this project include tcpdump and snoop - tools for monitoring network traffic on a specified interface. "tcptrace" and "xplot" will be used for trace analysis.

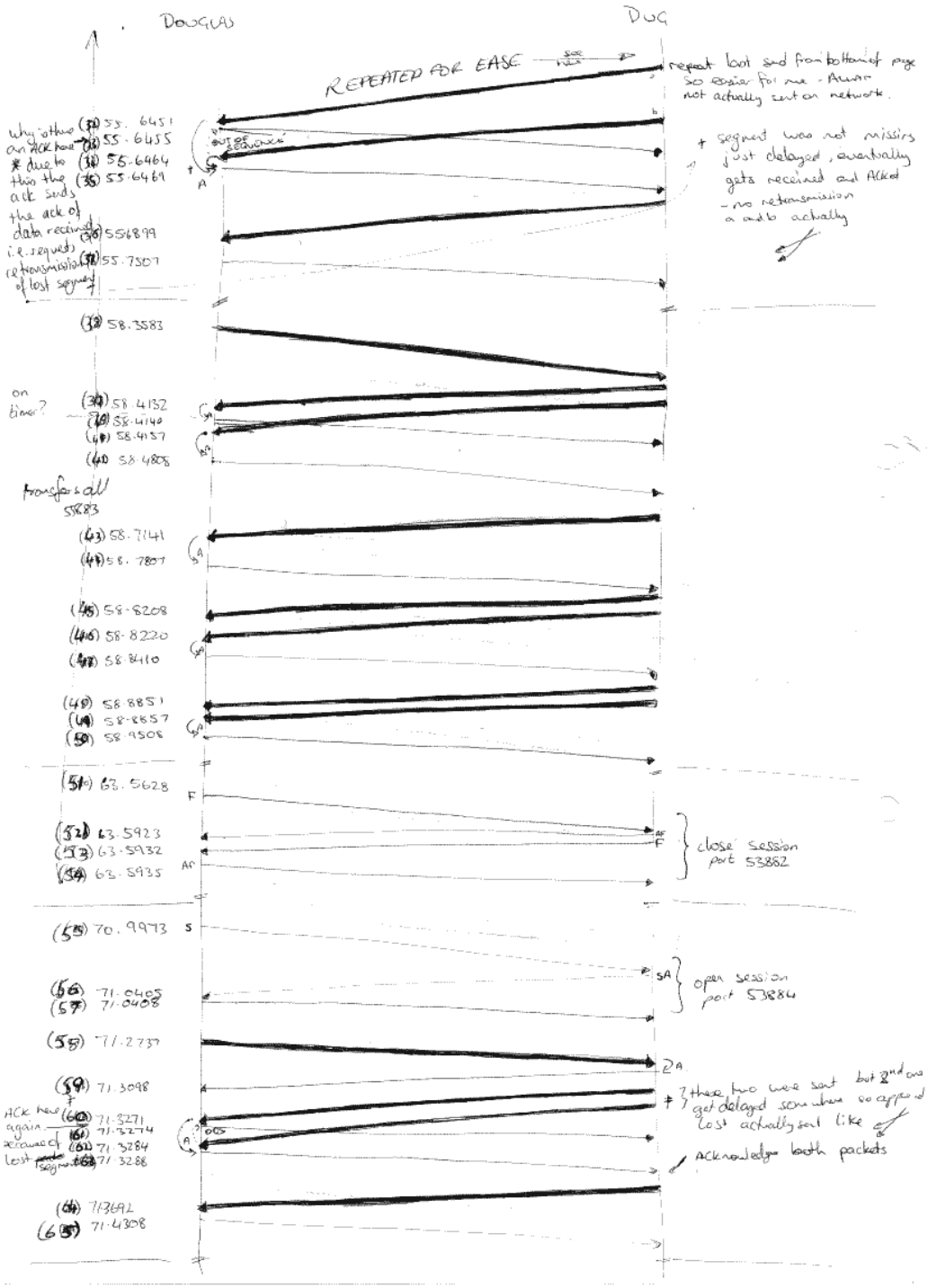
Main research is by Mahesh Sooriyabandara and Allan Bruce and supervised by Dr. G. Fairhurst.
Date : 06-07-2001

Example Web Page (source code)

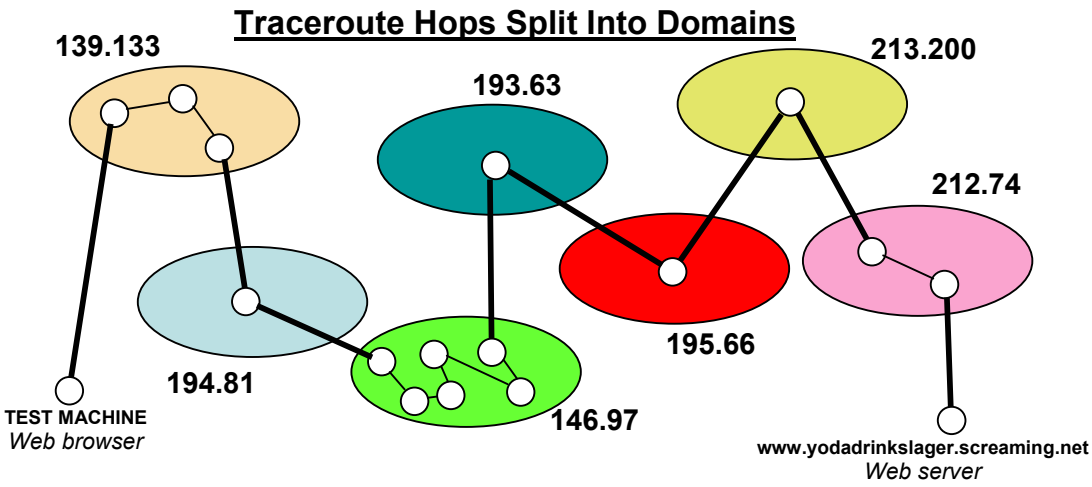
```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.7 (Macintosh; I; PPC) [Netscape]">
  <title>webtest.htm</title>
</head>
<body text="#000000" bgcolor="#FFFFFF" link="#0000FF" vlink="#990066" alink="#FF0000"
background="bathroom_tile_texture.jpg">
<center><img SRC="erglogo.gif" height=100 width=400>
<br><b><u><font size=+4>Analysing the behaviour of http protocol variants</font></u></b>
<br> 
<p><img SRC="satellite.gif" height=257 width=436></center>
<p><br>
<br>
<br>
<br>
<br>
<br>
<p>There are few variants of http protocol used in various commercial web
clients and servers. The characteristics of these http protocols can be
significant to specific network conditions. The main aim of this study
is to understand the behaviour of http protocol variants and their interactions
with TCP.
<br>Study is being undertaken at the <a href="http://www.abdn.ac.uk"><img
SRC="aberdeenuniversitylogo.gif" height=50 width=229 align=CENTER></a><a
href="http://www.erg.abdn.ac.uk">Electronics
Research Group</a>. The data collected will be analysed and will be used
to build a web traffic model in a network simulator. This is done as a
partial requirement for a project evaluating the TCP protocol performance
in next generation satellite systems.
<p>Some useful tools being used for this project include tcpdump and snoop
- tools for monitoring network traffic on a specified interface. 
"tcptrace" and "xplot" will be used for trace analysis.
<p>Main research is by Mahesh Sooriyabandara and <a href="mailto:allan@erg.abdn.ac.uk">Allan
Bruce</a> and supervised by Dr. G. Fairhurst.
<br>Date : 06-07-2001
</body>
</html>
```

Appendix A2:- Timing Diagram


DUG
DOUGLAS
DNS
DUG
11-07-01-02
(1) 49.9736
(2) 50.0064
(3) 50.0677 S
(4) 50.1013
(5) 50.1016 A3
(6) 50.1421
(7) 50.1757
(8) 50.2322
(9) 50.2467
(10) 50.6009
(11) 50.6707
(12) 54.8219
(13) 54.8765
(14) 54.8776
(15) 54.8777
(16) 54.9204
(17) 54.9214
(18) 54.9412
(19) 54.9881
(20) 54.9897
(21) 54.9907
(22) 55.0324
(23) 55.0509
(24) 55.1015
(25) 55.1020
(26) 55.1710
(27) 55.2398
(28) 55.2906
(29) 55.2909
(30) 55.5890
(31) 55.6287
(32) 55.6451
+0.0168
SA
+0.0336
nodata here
3-way handshake 'connect' port 53882
dont cross here!
S- SYNCHRONISE
SA- SYN with ACK
A3 - ACK SYNCHRONISE
F - FINISH
AF - ACK FINISH
OOO - OUT OF SEQUENCE
I didn't get ACKed but 2 segments already sent - DUG waits to see if needs retransmission
but 2 is ACKed ∴ didnt
@ - here - 3 segments get ACKed.
connect port 53883
transfers all to/from port 53883
this segment gets received but there was a segment missing *
OTP
OTP


DOUGLAS
DUG
REPEATED FOR EASE
repeat last send from bottom of page so easier for me - Answer not actually sent on network.
+ segment was not missing just delayed, eventually gets received and ACKed - no retransmission and ack actually
why is there an ACK here
* due to this the ack sends the ack of data received
i.e. requested retransmission of lost segment
on time?
transfer is all 55883
close session port 53882
open session port 53884
ACK here again
received
lost
ACKnowledge both packets

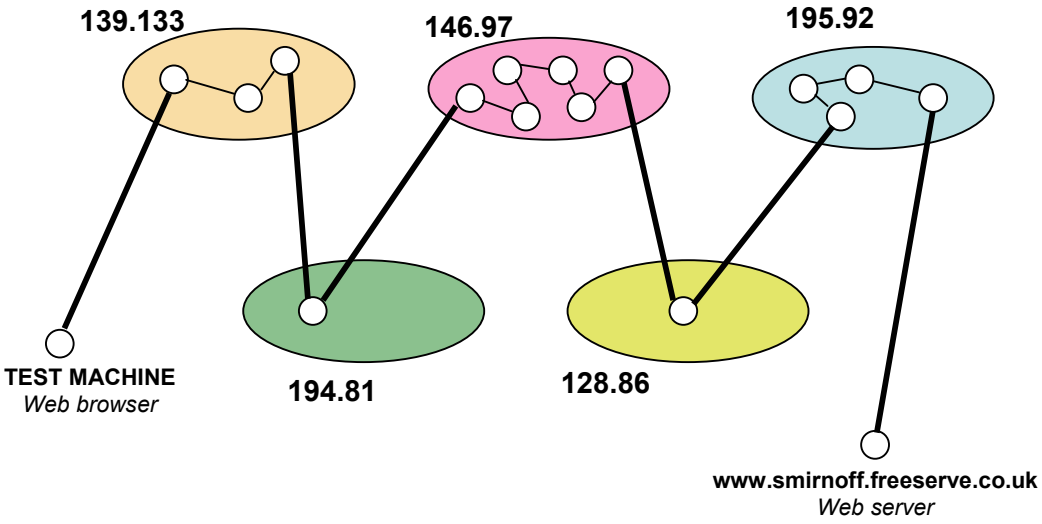
Appendix A4:- Traceroute diagrams




```
 1  milliways (139.133.204.64)  2.861 ms  2.160 ms  2.164 ms
 2  gw34.abdn.ac.uk (139.133.34.1)  4.812 ms  5.777 ms  4.753 ms
 3  gwkccs.abdn.ac.uk (139.133.7.4)  7.865 ms  16.373 ms  5.130 ms
 4  aclarke-gw.abman.net.uk (194.81.60.94)  5.820 ms  5.660 ms  5.780 ms
 5  146.97.250.17 (146.97.250.17)  13.479 ms  30.888 ms  28.902 ms
 6  146.97.37.29 (146.97.37.29)  14.096 ms  10.927 ms  19.769 ms
 7  pos9-0.edin-scr.ja.net (146.97.35.61)  23.848 ms  13.276 ms  17.556 ms
 8  pos0-0.leed-scr.ja.net (146.97.33.26)  19.888 ms  24.736 ms  17.311 ms
 9  pos2-0.lond-scr.ja.net (146.97.33.30)  23.376 ms  21.598 ms  30.466 ms
10  146.97.35.2 (146.97.35.2)  23.357 ms  27.087 ms  28.268 ms
11  linx-gw.ja.net (193.63.94.249)  23.704 ms  28.047 ms  32.986 ms
12  fe3-0.lon0.nacamar.net.uk (195.66.224.32)  22.672 ms  22.702 ms  24.028 ms
13  pos4-1-0.lon1.worldonline.net.uk (213.200.77.38)  33.415 ms  25.656 ms  32.012 ms
14  ge10-0-4.lon8.as9105.net (212.74.111.202)  29.022 ms  37.305 ms  24.084 ms
15  pos10-0.mk0.as9105.net (212.74.111.129)  30.608 ms  24.179 ms  29.786 ms
```

Route can be seen as 15 hops in total over 7 domains (5 intermediate)

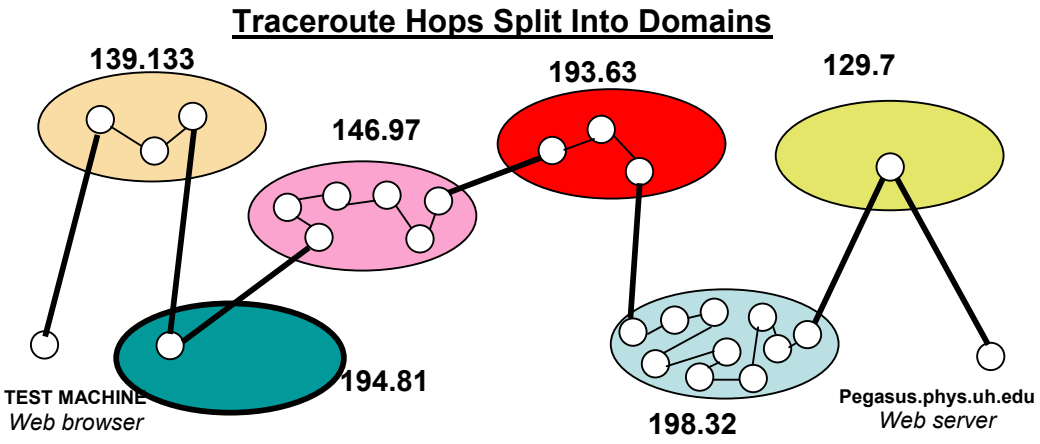
Traceroute Hops Split Into Domains




```
 1  milliways (139.133.204.64)  3.036 ms  2.604 ms  2.105 ms
 2  gw34.abdn.ac.uk (139.133.34.1)  4.210 ms  4.241 ms  5.937 ms
 3  gwkccs.abdn.ac.uk (139.133.7.4)  4.493 ms  14.606 ms  4.840 ms
 4  aclarke-gw.abman.net.uk (194.81.60.94)  5.186 ms  5.371 ms  4.981 ms
 5  146.97.250.17 (146.97.250.17)  11.828 ms  9.563 ms  8.884 ms
 6  146.97.37.29 (146.97.37.29)  12.928 ms  20.469 ms  12.093 ms
 7  pos9-0.edin-scr.ja.net (146.97.35.61)  11.437 ms  14.512 ms  14.121 ms
 8  pos0-0.leed-scr.ja.net (146.97.33.26)  30.714 ms  17.493 ms  23.295 ms
 9  pos2-0.lond-scr.ja.net (146.97.33.30)  27.018 ms  23.170 ms  25.790 ms
10  146.97.35.2 (146.97.35.2)  25.651 ms  23.836 ms *
11  uk-gw.ja.net (128.86.1.240)  34.364 ms  29.274 ms  28.189 ms
12  195.92.202.73 (195.92.202.73)  28.300 ms  30.954 ms  26.050 ms
13  195.92.201.2 (195.92.201.2)  32.232 ms  22.139 ms  39.443 ms
14  195.92.201.99 (195.92.201.99)  30.249 ms  32.050 ms  33.738 ms
15  195.92.200.136 (195.92.200.136)  41.142 ms  33.795 ms  35.176 ms
16  alteonG1.svr.pol.co.uk (195.92.195.141)  35.086 ms  30.538 ms  31.456 ms
```

Route can be seen as 16 hops in total over 5 domains

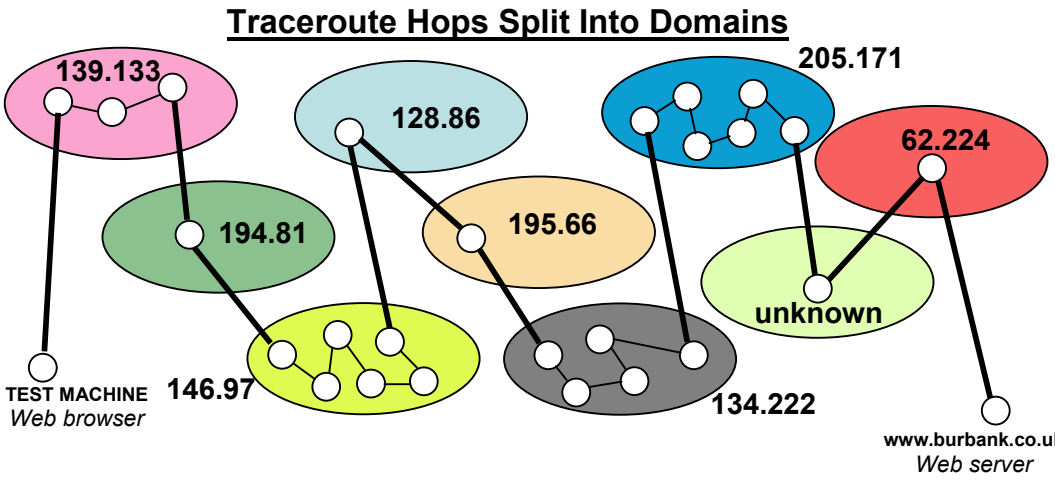
(3 intermediate)




```
 1  milliways (139.133.204.64)  2.956 ms  2.103 ms  2.101 ms
 2  gw34.abdn.ac.uk (139.133.34.1)  4.951 ms  4.891 ms  4.765 ms
 3  gwkccs.abdn.ac.uk (139.133.7.4)  5.255 ms  16.300 ms  5.009 ms
 4  aclarke-gw.abman.net.uk (194.81.60.94)  6.665 ms  5.533 ms  5.623 ms
 5  146.97.250.17 (146.97.250.17)  10.686 ms  10.713 ms  9.235 ms
 6  146.97.37.29 (146.97.37.29)  14.946 ms  20.399 ms  25.039 ms
 7  pos9-0.edin-scr.ja.net (146.97.35.61)  14.822 ms  17.958 ms  20.820 ms
 8  pos0-0.leed-scr.ja.net (146.97.33.26)  26.630 ms  26.973 ms  21.264 ms
 9  pos2-0.lond-scr.ja.net (146.97.33.30)  28.960 ms  25.399 ms  25.840 ms
10  146.97.35.6 (146.97.35.6)  22.284 ms  22.070 ms  23.392 ms
11  us-gw2.ja.net (193.63.94.91)  28.271 ms  22.696 ms  23.660 ms
12  193.62.157.18 (193.62.157.18)  91.709 ms  90.632 ms  87.277 ms
13  ny-pop.i2.ja.net (193.62.157.210)  95.792 ms  95.864 ms  97.488 ms
14  clev-nycm.abilene.ucaid.edu (198.32.8.29)  106.189 ms  116.730 ms  105.653 ms
15  ipls-clev.abilene.ucaid.edu (198.32.8.25)  113.960 ms  109.556 ms  111.011 ms
16  kscy-ipls.abilene.ucaid.edu (198.32.8.5)  129.682 ms  126.980 ms  127.155 ms
17  dnvr-kscy.abilene.ucaid.edu (198.32.8.13)  138.995 ms  130.486 ms  133.434 ms
18  scrm-dnvr.abilene.ucaid.edu (198.32.8.1)  153.115 ms  157.528 ms  160.970 ms
19  losa-scrm.abilene.ucaid.edu (198.32.8.18)  163.873 ms  163.366 ms  162.010 ms
20  hstn-losa.abilene.ucaid.edu (198.32.8.22)  195.996 ms  201.506 ms  198.599 ms
21  LINK2ABILENE.GIGAPOP.GEN.TX.US (198.32.236.13)  194.006 ms  192.559 ms  196.165 ms
22  INTRALINK2IBT.GIGAPOP.GEN.TX.US (198.32.236.37)  199.774 ms  204.746 ms  194.647 ms
23  UH.GIGAPOP.GEN.TX.US (198.32.236.30)  205.365 ms  201.777 ms  197.224 ms
24  vespasian-vlan10.gw.uh.edu (129.7.254.254)  205.210 ms  196.519 ms  204.796 ms
25  Pegasus.Phys.UH.EDU (129.7.2.50)  204.640 ms  197.582 ms  204.103 ms
```

Route can be seen as 25 hops in total over 6 domains

(4 intermediate)

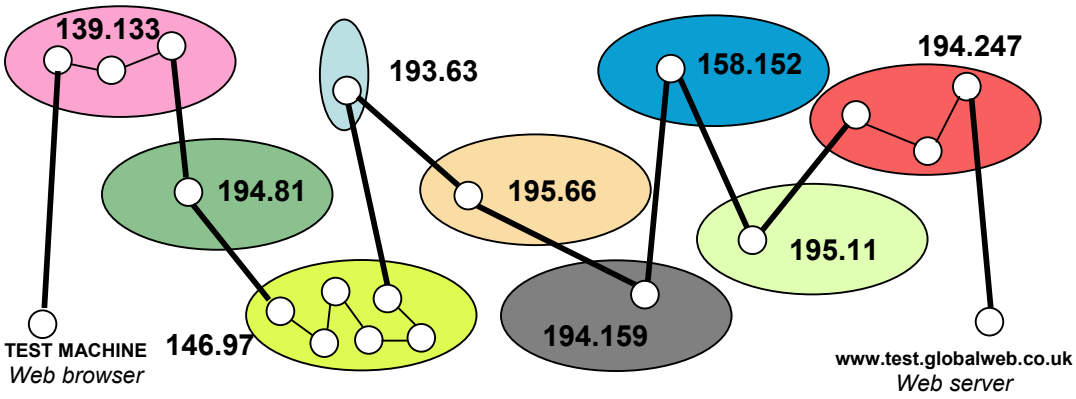



```
 1  milliways (139.133.204.64)  2.879 ms  3.009 ms  2.127 ms
 2  gw34.abdn.ac.uk (139.133.34.1)  6.856 ms  4.753 ms  4.805 ms
 3  gwkccs.abdn.ac.uk (139.133.7.4)  5.002 ms  16.683 ms  5.202 ms
 4  aclarke-gw.abman.net.uk (194.81.60.94)  10.044 ms  14.902 ms  5.488 ms
 5  146.97.250.17 (146.97.250.17)  10.227 ms  9.398 ms  9.278 ms
 6  146.97.37.29 (146.97.37.29)  14.264 ms  19.135 ms  23.052 ms
 7  pos9-0.edin-scr.ja.net (146.97.35.61)  12.255 ms  20.273 ms  13.253 ms
 8  pos0-0.leed-scr.ja.net (146.97.33.26)  18.192 ms  18.343 ms  18.011 ms
 9  pos2-0.lond-scr.ja.net (146.97.33.30)  26.353 ms  24.332 ms  21.553 ms
10  146.97.35.2 (146.97.35.2)  23.363 ms  24.924 ms  21.483 ms
11  linx-gw.ja.net (128.86.1.249)  22.995 ms  21.206 ms  22.164 ms
12  r13-Gi4-0.Ldn-KQ4.UK.KPNQwest.net (195.66.224.54)  29.550 ms  28.995 ms  23.123 ms
13  r4-Gi1-0-0.200.ldn-KQ4.uk.kpnqwest.net (134.222.109.242)  21.956 ms  29.106 ms  25.063 ms
14  r1-Se1-3-3.ldn-KQ1.UK.KPNQwest.net (134.222.231.85)  26.493 ms  24.813 ms  30.160 ms
15  r2-Se0-1-0.0.ledn-KQ1.NL.kpnqwest.net (134.222.230.170)  30.916 ms  35.625 ms  34.892 ms
16  r1-PO4-0.obl-KQ1.NL.kpnqwest.net (134.222.96.34)  29.009 ms  29.055 ms  34.766 ms
17  r1-PO1-0.wdc.US.kpnqwest.net (134.222.228.34)  111.767 ms  111.815 ms  113.234 ms
18  wdc-brdr-03.inet.qwest.net (205.171.24.113)  102.676 ms  103.510 ms  102.513 ms
19  wdc-core-03.inet.qwest.net (205.171.24.69)  105.120 ms  102.578 ms  106.935 ms
20  atl-core-03.inet.qwest.net (205.171.5.243)  124.529 ms  119.954 ms  122.574 ms
21  atl-core-02.inet.qwest.net (205.171.21.157)  120.883 ms  124.904 ms  121.224 ms
22  atl-edge-03.inet.qwest.net (205.171.21.46)  123.278 ms  120.556 ms  121.136 ms
23  205.171.51.234 (205.171.51.234)  117.826 ms  123.912 ms  117.826 ms
24  * * *
25  64.224.0.100 (64.224.0.100)  128.347 ms  126.500 ms  126.081 ms
26  burbank.co.uk (209.35.163.129)  118.541 ms  125.294 ms  123.538 ms
```

Route can be seen as 26 hops in total over 9 domains

(7 intermediate)

Traceroute Hops Split Into Domains



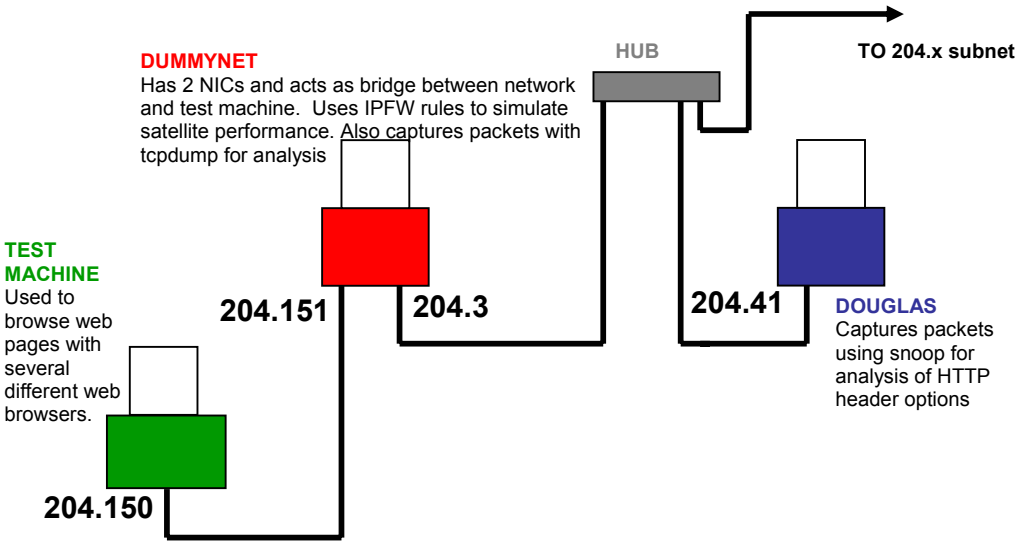

```
 1  milliways (139.133.204.64)  2.831 ms  2.077 ms  2.167 ms
 2  gw34.abdn.ac.uk (139.133.34.1)  4.828 ms  4.955 ms  4.865 ms
 3  gwkccs.abdn.ac.uk (139.133.7.4)  16.989 ms  15.510 ms  5.331 ms
 4  aclarke-gw.abman.net.uk (194.81.60.94)  7.769 ms  5.545 ms  5.734 ms
 5  146.97.250.17 (146.97.250.17)  9.785 ms  12.061 ms  9.347 ms
 6  146.97.37.29 (146.97.37.29)  13.904 ms  16.689 ms  11.144 ms
 7  pos9-0.edin-scr.ja.net (146.97.35.61)  11.492 ms  16.527 ms  21.450 ms
 8  pos0-0.leed-scr.ja.net (146.97.33.26)  18.450 ms  27.231 ms  19.766 ms
 9  pos2-0.lond-scr.ja.net (146.97.33.30)  32.023 ms  35.862 ms  28.696 ms
10  146.97.35.6 (146.97.35.6)  26.864 ms  25.046 ms  24.458 ms
11  linx-gw.ja.net (193.63.94.249)  23.115 ms  32.644 ms  21.848 ms
12  linx-2.router.demon.net (195.66.224.13)  26.371 ms  26.082 ms  22.430 ms
13  tele-backbone-1-ge020.router.demon.net (194.159.252.54)  24.510 ms  32.792 ms  23.929 ms
14  anchor-core-2-fxp1.router.demon.net (158.152.0.178)  36.384 ms  33.346 ms  36.909 ms
15  demon-gw-2.sol.co.uk (195.11.50.130)  37.791 ms  33.314 ms  38.483 ms
16  atm1-0-0-1.core2.scotland.net (194.247.77.34)  50.325 ms  56.771 ms  55.608 ms
17  fe12-0-0.core1.scotland.net (194.247.67.41)  44.368 ms  46.100 ms  41.028 ms
18  ABZ-Sci-Park.LL.scotland.net (194.247.71.109)  50.041 ms  51.625 ms  44.770 ms
```

Route can be seen as 18 hops in total over 9 domains

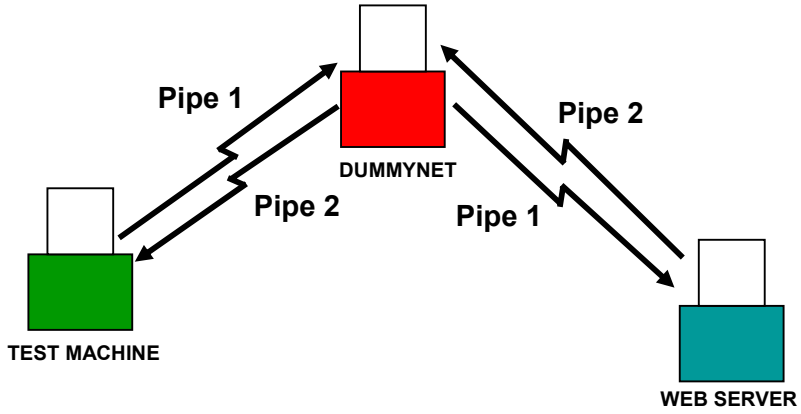
(7 intermediate)

Appendix A4:- FTP Example Transfer

Appendix A5:- Dummynet Topology



Appendix A6:- Dummynet Configuration



These 4 pipes are configured to simulate satellite conditions. Typical conditions are 544 Kbit/s bandwidth, 1200ms RTT, and 10^{-5} BER

Setup:
ipfw add pipe 1 pass ip from 139.133.204.150 to any
ipfw add pipe 2 pass ip from any to 139.133.204.150

Configuration:
ipfw pipe 1 config delay 500 ms bw 136Kbit/s plr 0.04
ipfw pipe 2 config delay 500 ms bw 136Kbit/s plr 0.04