

Compromised and Degraded Network Simulation Final Report

Submitted for the BSc in
Computer Science

May 2, 2018

by

Aidan Fray

Word Count: 13835

Contents

| | |
|--|-----------|
| 1. Introduction | 5 |
| 2. Aims and Objectives | 6 |
| 2.1. Objective One | 6 |
| 2.2. Objective Two | 7 |
| 2.3. Objective Three | 7 |
| 3. Background | 8 |
| 3.1. Problem Context | 8 |
| 3.1.1. Protocols | 9 |
| 3.1.1.1. Transport Control Protocol (TCP) | 9 |
| 3.1.1.2. User Data Protocol (UDP) | 14 |
| 3.1.1.3. Address Resolution Protocol (ARP) | 15 |
| 3.1.1.4. Custom Router - Raspberry Pi | 16 |
| 3.1.1.5. Router alternatives | 17 |
| 3.2. Comparison of Technologies | 18 |
| 3.2.1. Protocol Comparison | 18 |
| 3.2.1.1. TCP Based Protocols | 18 |
| 3.2.1.2. UDP Based Protocols | 19 |
| 3.2.1.3. Other protocols | 19 |
| 3.2.2. Operating Systems | 20 |
| 3.2.2.1. Linux | 20 |
| 3.2.2.2. Windows | 21 |
| 3.3. Alternative Solutions | 22 |
| 3.3.1. clumsy | 22 |
| 3.3.2. TMNetSim | 22 |
| 3.3.3. Comcast | 23 |
| 4. Technical Development | 24 |
| 4.1. Tool Architecture | 24 |
| 4.1.1. Parameter Handling | 25 |
| 4.1.2. Effect Choice & NQUEUE Creation | 26 |
| 4.1.3. Incoming Packets | 27 |
| 4.2. Tool Class Design | 28 |
| 4.2.1. Effect Class Diagram | 28 |
| 4.2.2. Tool Activity Diagram | 29 |
| 4.3. Tool UI Design | 30 |

Contents

| | | |
|-----------|--|-----------|
| 4.4. | Tool Implementation | 30 |
| 4.4.1. | NFQUEUE | 30 |
| 4.4.2. | Degradation Effects | 33 |
| 4.4.2.1. | Latency | 33 |
| 4.4.2.2. | Packet Loss | 34 |
| 4.4.2.3. | Bandwidth | 34 |
| 4.4.2.4. | Out-of-order | 35 |
| 4.4.2.5. | Connection Simulation | 35 |
| 4.4.2.6. | Jitter | 36 |
| 4.4.3. | ARP Spoofing | 36 |
| 4.4.4. | Network Attacks | 37 |
| 4.4.4.1. | UDP Flooding | 38 |
| 4.4.4.2. | ARP Spamming | 38 |
| 4.4.5. | Router Design | 39 |
| 4.4.6. | Router Implementation | 40 |
| 4.5. | Testing | 41 |
| 4.5.1. | Traffic simulation programs | 41 |
| 4.5.2. | Packet Script | 42 |
| 4.5.3. | Testing correctness | 42 |
| 4.5.4. | Testing methodology considerations | 43 |
| 4.6. | Experiential design | 43 |
| 4.6.1. | Protocol Degradation | 43 |
| 4.6.1.1. | HTTP Experiment | 43 |
| 4.6.1.2. | FTP Experiment | 43 |
| 4.6.1.3. | UDP Experiment | 44 |
| 4.6.2. | Latency Accuracy | 44 |
| 4.6.3. | Visualising effects | 45 |
| 4.6.4. | UDP Flooding | 46 |
| 5. | Evaluation | 47 |
| 5.1. | Experimentation Results | 47 |
| 5.1.1. | Test Environment | 47 |
| 5.1.2. | Effect of Packet Loss on Download Speeds | 47 |
| 5.1.3. | Effect of Latency on Download Speeds | 52 |
| 5.1.4. | Latency effect on Slow Start | 54 |
| 5.2. | Project Achievements | 56 |
| 5.2.1. | Objective One - <i>Achieved</i> | 56 |
| 5.2.2. | Objective Two - <i>Achieved</i> | 56 |
| 5.2.3. | Objective Three - <i>Achieved</i> | 57 |
| 5.3. | Project Management | 57 |
| 5.3.1. | Issues | 57 |
| 5.3.2. | Commits | 57 |
| 5.4. | Further work | 58 |
| 5.4.1. | More protocols | 58 |

Contents

| | |
|--|-----------|
| 5.4.2. More platform support | 58 |
| 5.4.3. Larger test network | 59 |
| 6. Conclusion | 60 |
| Appendix A. Test Plan | 61 |
| Appendix B. HTTP Testing | 65 |
| Appendix C. FTP Testing | 66 |
| Appendix D. UDP Testing | 67 |
| Appendix E. GitHub Issues | 68 |
| Appendix F. Testing Correctness | 70 |
| Bibliography | 72 |

1. Introduction

Networks and their functionality play a crucial role in today's internet infrastructure. Networks relay packets and allow communication between computers from opposite sides of the globe. This paper will discuss and explore the effects of degradation on a live network, and how these effects can be reduced or negated.

The live network will be a simulation of network traffic, this traffic will be routed through a custom router running on a small Raspberry Pi. This Raspberry Pi will run the degradation tool that will allow the control of network conditions.

Issues may arise from various aspects of the project. The first issue being the accuracy in simulation of network traffic may cause inaccuracies in the tools validity. More problems may occur regarding the discovery of effective practises to negate hostile network conditions, and other issues involving the stability and scalability of the tool meaning its max capacity needs defining. These issues will be discussed later.

2. Aims and Objectives

Create a custom simulated network that can demonstrate and visualise network degradation and common DoS attacks, this is so network engineers can identify weak spots and points of strain

The above aim will be achieved by reaching these targets:

2.1. Objective One

Develop a fully working client server pair that communicate using HTTP/1.1 (Fielding et al., 1999) and FTP over the TCP protocol alongside a client and server that use the UDP protocol

- The client will use HTTP/1.1 over TCP and will need to be able to send PUT and GET requests to the server.
- The server will need to be able to deal with HTTP/1.1 requests and act accordingly. This means when the server receives a PUT request it will alter data. A GET request will therefore mean a retrieval of data.
- The client will need to be able to upload and download files using FTP.
- The server should be able to receive uploaded files and send data when a download is requested.
- The UDP client will need to send a set number of packets each containing a single value that increments with each packet sent.
- The UDP server will display a grid of received and missing packets where the positions are denoted by the packets value, this will give a visual representation of missing packets.

The functionality above will be contained in one executable that will take parameters to define which client or server will be loaded. For example the parameters '-f c' will load the FTP client.

Note: Authentication for any of these channels is outside the scope of this project

2. Aims and Objectives

2.2. Objective Two

Create a program that runs on a Linux based OS that can be used to simulate degradation and attacks. This program will then be run on the router.

- Degradation factors will be:
 - Packet loss
 - Latency
 - Rate of transfer (Bandwidth)
- Attacks will compose of some small and simple attacks; possible attacks could be:
 - UDP Flooding (Xiaoming et al., 2010), where the network is filled with erroneous packets that will attempt to clog up the network.
 - ARP (Address Resolution Protocol) spamming and poison, the network is filled with ever changing ARP requests that cause computers to not be able to correctly resolve the valid locations of each other (Whalen, S., 2001).
 - There may also be space for other attacks if time allows.

2.3. Objective Three

Create a working custom router that can be used to simulate degradation in conjunction with the program outlined in Objective 2

- This router should be able to act in place of a real commercial router.

Note: The speed that the router can handle is not a concern.

- This router will have to be able to deal with internet connections over Ethernet and a wireless connection, with multiple devices connected at the same time.

Note: The security of the router is also outside of the scope of this project

3. Background

3.1. Problem Context

The main focus of this project is to create a network degradation tool that can be used to simulate imperfect network conditions and common attacks. To understand the goal better it is important to know what criteria reduce network quality.

Latency, packet loss and bandwidth are the main factors that distinguish between good and bad internet connections. Latency is the delay between sending a message and seeing its result. Packet loss is the percentage of packets that are lost in a transmission, the higher the percentage, the more effect it has on the speed and stability of the network. Bandwidth, is the measure of how many bits/bytes are being transferred per second, the larger the bandwidth the more data can be transferred in less time, thus making the overall connection faster. A combination of favourable values from these criteria defines a good network connection. One way to visualise the throughput of a network connection is a download and upload speed test, the result of these is measured in Mbps and can be used to quickly compare the speed of two connections.

Other small signs of poor network conditions are Jitter and Error rate. Jitter is simply the variance in inter packet gaps (time gap between two packets). The gap is normally an issue that exists in large packet switched networks. TCP based communications are not normally effected heavily by jitter due to how the TCP protocol deals with these issues. However, the UDP based VoIP (Voice over IP) protocol is visibly degraded by heavy jitter. Error rate is the number of bit errors that have occurred in a data stream over a period of time. This can again effect UDP based communications heavily but the TCP protocol mitigates the effects and therefore remains relatively unaffected.

It is also necessary to mention in more detail the protocols that are to be used in the network simulation section. TCP (Postel et al., 1981) and UDP (Postel, 1980) are the transport layer protocols up for discussion.

3. Background

3.1.1. Protocols

3.1.1.1. Transport Control Protocol (TCP)

TCP's main design choices are centred around reliability, where it has a few techniques designed to make sure packets arrive correctly.

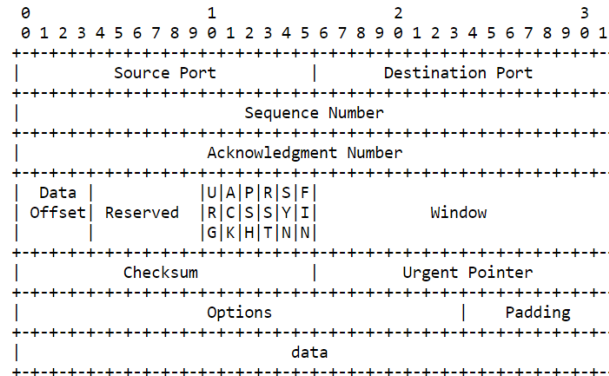


Figure 3.1.: TCP Header (RFC 793)

Initiating a connection

The connection is initiated with a series of signals carried as packets. These signals are abbreviated as ACK (Acknowledgement) and SYN (Synchronise). The initiating party begins by sending an empty packet with the SYN flag set, this starts the synchronization process. Then once the target computer receives this SYN packet it sends back an ACK plus its own SYN packet (This is normally sent as a single ACK + SYN packet). Where finally the initial computer sends back an ACK packet in response to the previous SYN packet, this connection has now been initiated and the communication channel has been set up. This process is known as a '3-way handshake'.

| | | |
|-----|----|--|
| TCP | 66 | 51122 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| TCP | 58 | 53 → 51122 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| TCP | 54 | 51122 → 53 [ACK] Seq=1 Ack=1 Win=64240 Len=0 |

Figure 3.2.: The output from Wireshark showing the handshake

3. Background

Sequence Numbers and Acknowledgement Numbers

One of the main mechanisms for detecting missing and incorrect packets is the sequence and acknowledgement numbers. The sequence number is increased by the length of every correctly transmitted packet. A packet's length can be calculated by taking the total size of the packet and subtracting the size of the header. For example, if a packet with a length of 100 was sent it would move the sequence value up by 100, this gives the protocol the information to judge what it should be expecting and what it has received, if these values do not add up, TCP can flag for an error and retransmit.

To guarantee arrival of a packet the sender needs to know when packets have been received, this is where the acknowledgement number comes in. Packets carrying acknowledgements are known as "ACKs". For a packet with a sequence number of 100 being sent, it will require a packet to be returned containing the ACK number of 100 to verify that exact packet has been transferred correctly. If the corresponding ACK packet is not received before the the time-out value has been reached the packet is resent, this is how TCP guarantees retransmission of packets. TCP's guaranteed arrival is one of the reasons why packet loss does not have the same devastating effect it does on UDP with TCP.

In situations where connections are created and recreated quickly in succession or are being quickly re-established after errors there cannot be initial sequence numbers (ISN) that clash with previous segment that may still exist on a network connection. Therefore TCP solves this issue by generating new ISNs using a ISN Generator that uses a 32 bit clock. The 32 bit clock has a life cycle of 4.55 hours where this time period is commonly known as the Maximum Segment Lifetime (MSL). If an ISN is generated within this MSL, it can be guaranteed to be unique.

Receiving Window

In the TCP header there is a two byte value that represents the receiving window size, this value defines the maximum size of unacknowledged packets the receiving end has space for. If a client sends a packet with a 65535 window size value it tells the receiving computer not to send more than 65535 bytes before receiving corresponding acknowledgements. This value can also be extended using a value in the area allocated in the 'TCP Options' called 'Window Scaling', this allows the value in the window size to be multiplied by this scale value. For example, a window size of 65535 with a window scale of 5 would have a total window size of 327675 Bytes. A larger window size means more data can be sent before the protocol needs to wait for acknowledgements and can normally mean faster transfer speeds. It is worth noting; this exchange of window scaling values is only performed in the handshake and can only be present in the initial exchange of packets.

3. Background

Flow Control

Flow control is a mechanism used by the protocol to ensure that the receiving party does not become overwhelmed by the incoming transmission of packets. TCP has two types of buffers the 'send' and 'receive', where the send buffer collects what data is going out and the receive buffer collects what is coming in. The purpose of the flow control is to prevent the sending of packets that will not fit in the destination receive buffer and therefore will prevent these packets from being dropped. The protocol achieves this by allowing each party to advertise its available receive buffer space through the 'Window Size' section in the header (This is included in each ACK packet) and is commonly known as the 'Advertised Window'. Once the receive buffer is full the receiving party will advertise what is known as a 'zero window' where it sets the window size to zero and transfer will stop until there is adequate space in the receive buffer.

Sliding Window

To control the number of packets the protocol has in flight at any one time the protocol utilises the 'sliding' window. The size of the sliding window is altered by two factors: The size of the send buffer and the size of the destinations receive buffer. Therefore, the maximum amount of data the protocol can send is the smallest of the two previous values.

$$SlidingWindowSize = \text{Min}(LocalSendBuffer, DestinationReceiveBuffer)$$

The sliding window then dynamically works out how many packets it can send by keeping track of the number of packets in flight (unacknowledged) and the amount left of the current window size. This whole process gives the protocol the ability to dynamically adapt to changing conditions and aims to prevent buffers from being overfilled.

Closing a connection

The closing of a TCP connection is very similar to the initial construction (Postel et al., 1981), but in this case the protocol utilises the 'FIN' flag defined in the TCP header. In this example it will be a client disconnection from a server. After all the data is transferred the client no longer requires a connection to the server and sends an empty packet with the FIN bit set, the server receives the packet and returns an ACK. The server now sends its own FIN and this is then acknowledged by the client. This middle step with the server sending an ACK and FIN separately is normally performed in a single packet in live networks. The client and server now knows the connection is closed and will free up any resources allocated to that connection.

3. Background

Congestion control

When there are situations where multiple clients are connected to a single receiver, the bandwidth needs to be split equally between the clients, this is performed by the inbuilt TCP congestion control. The clients send data and keep increasing their transfer rate until they lose a packet. Packet loss is therefore a sign of congestion - this is because it assumes that the packets are being lost due to the router having insufficient packet buffering space. When the number of clients drops, more bandwidth then becomes available and the congestion algorithm utilises this naturally because it has been periodically probing the network to check for available bandwidth. This probing is done by increasing its transfer rate until packet loss is encountered, where it then quickly slows its transfer rate and repeats the process. This results in TCP auto balancing when new clients are introduced and removed and thus utilizing as much bandwidth as is available.

The rate of transfer is indirectly defined by two values "Congestion Window" (Cwnd) and "Advertised Window". The congestion window is the amount of packets a connection stream allows in flight and the advertised window is used to inform the sending end how much buffer space is available to hold incoming packets. These both define the overall "WinSize" where a small value means less packets can be sent without waiting for corresponding acknowledgements and therefore reducing the transfer speed. The equation below is used to define window size:

$$WinSize = \min(CongestionWindow, AdvertisedWindow)$$

Later this report will discuss the effects of packet loss, latency and other degradation signs on this congestion control.

AIMD

AIMD, standing for Additive Increase Multiplicative Decline, is a feedback control algorithm known for its use in TCP's congestion control. It works by having a linear increase and exponential decline and it will naturally mean that if set up correctly all entities using AIMD will converge on equal usage of a shared resource - in this case network bandwidth. This famously produces a 'saw-tooth' pattern when the maximum capacity is reached (Huston, 2006).

Slow Start

Slow start is designed for defining suitable initial transfer rates of a TCP connection. The slow start mechanism is required because a initial huge transfer rate will have a high chance to cause buffers to overflow and will revert the connection to a crawl. Therefore,

3. Background

the solution to this is to start with a small-medium transfer rate and increase it per acknowledgement received.

Congestion Algorithms

There are various algorithms used to manage congestion on a network and they work in slightly different ways. All algorithms have situations where they work most effectively.

Congestion Algorithm - TCP Reno

TCP Reno was first released in the 4.3BSD OS and is normally grouped up with TCP Tahoe as they share many features. TCP Reno uses AIMD where it performs its additive increase when a full window is transferred correctly and its multiplicative decline when a packet is lost. It introduces two new ideas "Fast Retransmit" and "Fast Recovery":

Fast Retransmit

This mechanism uses 3 duplicated acknowledgements (These can be caused by re-ordered or missing segments) to signal the need for a retransmission. Fast Retransmit can repair single segment loss and does not involve the use of any time-outs, this lack of time-outs can involve a lot less idle link time. After this the congestion window size is reduced by half because of detection of loss.

Fast Recovery

For every duplicate ACK a segment must arrive, this can therefore be used to assume that the receiving of a segment means the buffer in the receiving end has more space available. Therefore, the sending of more packets has a high chance not to cause the buffer on the receiving end to overflow - new segments are then therefore put onto the network when acknowledgements are received (even duplicated acknowledgements).

These techniques are both used together and can result in effective mitigation of packet loss.

Reno defines congestion problems in terms of packet loss and can be defined either of two ways:

- 3 duplicated acknowledgements is considered non-severe and the algorithm moves into congestion avoidance with Fast Retransmission and fast recovery coming into effect.
- Retransmission defined by a time out is defined as **severe** and will set the Congestion Window to the minimum value and initiate a Slow Start. Time outs are usually caused by multiple occurrences of packet loss.

3. Background

TCP NewReno

NewReno (Henderson et al., 2012) as the name suggests is the successor to TCP Reno that was discussed in the previous section. It is the version of Reno used in recent versions of the Linux Kernel.

TCP NewReno uses further heuristics to recover from multiple packet loss without the need for time-outs, this therefore means it outperforms Reno at higher error rates and is the algorithm used in the later experimentation section of this report.

3.1.1.2. User Data Protocol (UDP)

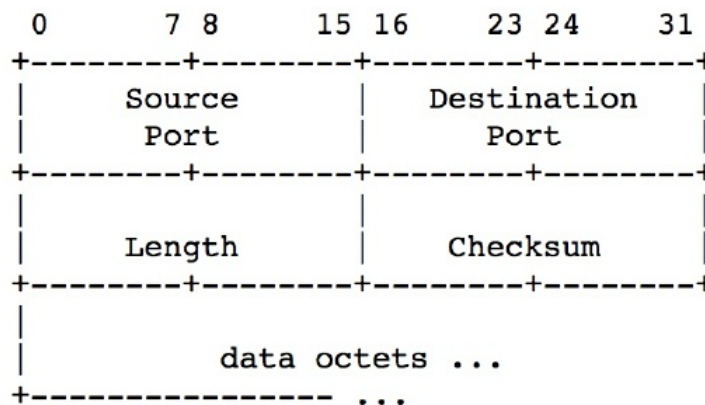


Figure 3.3.: UDP Header (RFC 768)

UDP is by design the opposite of TCP and often you'll find the two protocols compared. UDP is often referred to as "fire and forget" (Kempf et al., 2011) this is a short way of describing how UDP deals with packets. When a packet is to be sent UDP sends the packet and moves onto the next. However, this means aside from the checksum in the UDP header (A checksum can be used to check that the UDP data has been transferred correctly) there is zero error checking for the UDP protocol and results in UDP being considered unreliable in comparison to TCP.

This lack of error-checking overhead, however, greatly improves the speed of UDP protocols. UDP is not used in situations where communication is vital, but does find itself implemented in examples like voice or video chat because errors in these services are not catastrophic to operation and these imperfections can even go unnoticed.

As the very small UDP header shows, it is a very simple protocol and is effectively a thin wrapper on top of the IP layer. UDP allows for very quick transfers but with the cost to reliability. Packet loss therefore, effects UDP massively where this overall effect will be fully discussed later.

3. Background

3.1.1.3. Address Resolution Protocol (ARP)

The tool will be deployed either by having it run on a small custom computer set up to act as a router for the synthetic test network or by jumping between a user and an actual router on a live network. The program will achieve this jumping between a user and the router by utilising a long term vulnerability in the Address Resolution Protocol (ARP) (Whalen, 2001), before this is explained further it is first important to understand the relevant parts of a local network.

Each computer on a Local Area Network (LAN) has an IP assigned to it, this is the identifier on the local network. Each computer's wireless card has a MAC address, a MAC address is a unique identifier for a computer and is used to link up this non-unique local address (IP) to a real computer on a network. The combination of IP address and MAC address allow for the reuse of local IP numbers over many LANs. The ARP protocol is therefore used to find out what MAC address is associated with a local IP address and visa versa. This is achieved via an entire network broadcast where computers that do not match ignore the ARP request, and the one being requested sends back a reply containing its MAC address. Each computer maintains its own version of the ARP data known as its 'ARP cache'.

An ARP packet is a very simple message format that contains an operation code that can either can be a request (1 opcode) or a response (2 opcode) where the packet also contains 4 addresses; two pairs of hardware and protocol addresses for the sender and target.

The ARP protocol contains no form of authentication thus, allowing a malicious computer to send out faked custom ARP requests to change specific values in the routers ARP table. The results in any computer running the software appearing as the router to the victim, while also appearing as the victim to the router, thus having all the traffic between the two routed through the attacking computer.

3. Background

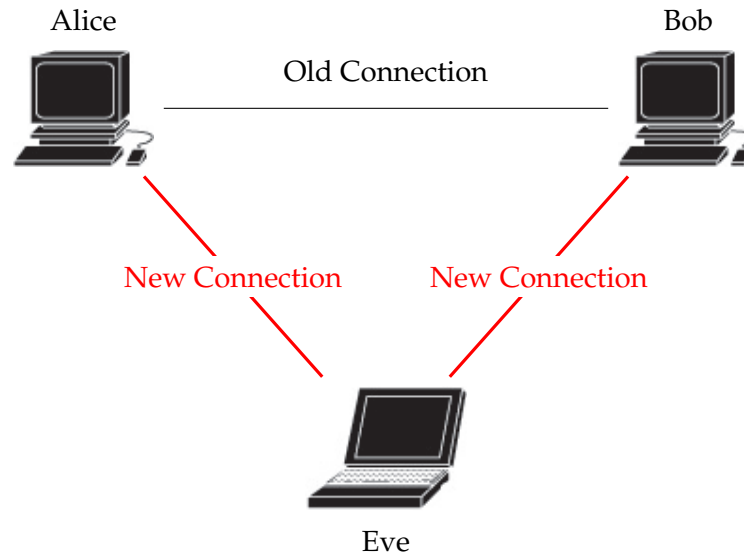


Figure 3.4.: Photo depicting a MITM attack

3.1.1.4. Custom Router - Raspberry Pi

As mentioned previously the effects will be performed on a custom computer acting as the router, because the network will be set-up to pass all traffic through this router, it will be able to simulate degradation on the entire networks traffic. The Raspberry Pi (Upton & Halfacree, 2014) has been decided as the choice for the small custom computer. Originally designed to teach children to code on an inexpensive (less than £30) computer, the Pi has now found itself involved in a multitude of uses ranging from NAS (Network Attached Server) to robot controllers. The Pi's versatility is useful in this project due to its ease to set-up as a router, this is because of the easy access of low level aspects of the operating system due to it running a Linux based OS. The Raspberry Pi will be run with its default Raspbian OS (Pi, 2014) that is built upon Debian (Murdock, 1994). This is a optimised OS for the internal ARM CPU that allows the Raspberry Pi to run modern stripped down programs on its very limited hardware, where however, it will have plenty of power to easily handle traffic as a router.

3. Background

3.1.1.5. Router alternatives

MinnowBoard

MinnowBoard ¹ is a company specialising in open source custom made processing units designed to remain compact while still being able to perform relatively heavy computation. Even their basic options have more RAM and processing speeds compared to that of the Raspberry Pi, but this comes at a higher cost, the basic model retails in at £110, more than 3 times to price of the Raspberry Pi. The increased processing speed would mean the total bandwidth capacity of the router could increase and would even allow the use of the GUI as an alternative to the command line, but this extra cost and increased processing speed does not add any extra value to the goal of the project and therefore the option was discarded.

Commercial router

A commercial out-of-the-box router could be used with a custom version of OpenWrt ². OpenWrt is a custom version of Linux that is designed for embedded devices. To install this version of Linux on a router would require 'flashing' the software. Flashing is the process of overwriting the embedded EEPROM or other type of embedded memory with the firmware of your choice, this process is lengthy and has no guarantee of working. This process however would provide the project with a device that could run the degradation script that is fully designed to function within a network, this could possibly mean faster processing speeds and a smoother experience. However, due to the uncertain set-up time compared to that of the Raspberry Pi this option was decided against. However, the option could be an aspect to implement in the long term improvements, this will be further discussed later in the report.

¹<https://minnowboard.org/>

²<https://openwrt.org/>

3. Background

3.2. Comparison of Technologies

3.2.1. Protocol Comparison

3.2.1.1. TCP Based Protocols

HTTP

TCP has various protocols built on top of it. One of the most widely used protocols is HTTP/1.1 (Fielding et al., 1999). HTTP is a structured language used to transfer data worldwide. HTTP's main use is the movement of web based data, this has been chosen as one of the protocols to be used in the test network to simulate live traffic. Its use is due to the protocols heavy use in the real world. HTTP will be utilised by a web browser that attempts to access a web page, the degradation will therefore be visible through how quickly the web page loads and how responsive the connection seems.

FTP

Another protocol based on top of TCP is FTP (File Transfer Protocol) (Postel & Reynolds, 1985), and as the broken down acronym suggests, this protocol is used to transfer files across a network. FTP has been selected also due to its integral use in a live network. It also has a quite intuitive way of representing the speed of a network through a download speed that is easy to understand for most users. The protocol will be tested with dummy files ranging from 1MB to 5GB. This size range was chosen to fully cover the normal spectrum of file size where it roughly ranges from a Word document to a high quality video.

Other TCP Considerations

SFTP

SFTP (SSH File Transfer Protocol) (Galbraith et al., 2006) was a consideration as a protocol to be used for the demonstration purposes. SFTP effectively is a secure version of FTP (but not to be mixed up with the Simple File Transfer Protocol). This was decided not to be used because of its added features such as authentication does not fit within the scope of the project, and does not relate to network degradation.

Mail based protocols

Another widely used set of protocols are the email based family of protocols. SMTP (Simple Mail Transfer Protocol) is involved in the sending of mail, while examples like

3. Background

IMAP/POP are used for receiving mail. These protocols use TCP as their transport layer protocols and are fundamental to email functioning on the internet, however, visualisation into their degradation would be no different to that of bare TCP. The implementation of an email client therefore, would add unnecessary complexity while offering almost zero added value to the demonstrative purposes of this project.

3.2.1.2. UDP Based Protocols

UDP is hugely affected by packet loss due to its inherent unreliability. The effects on UDP have decided to be visualised by a program that simulates image transfer by assigning an individual UDP packet a number that corresponds to a single pixel, this packet is then sent. Once the server receives the packet, it reads the pixel number stored in the packet and changes that specific pixel to green. This creates a matrix of red (lost) and green (received) pixels. This therefore, gives a simple but effective visualisation of packet loss.

Other effects can also be performed on UDP to simulate degradation. Latency and the incorrect arrival order of the packets are big signs of hostile transfer conditions and therefore, monitoring of the latency and order can be performed by the interface and could provide a much more detailed overview of the current health of the connection.

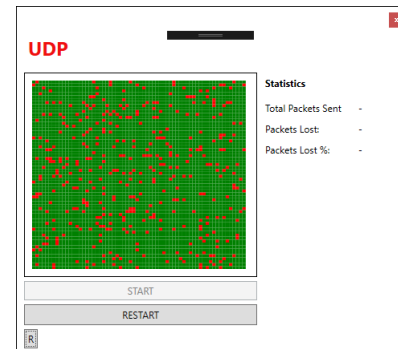


Figure 3.5.: Initial draft of the UDP user interface

Other UDP based protocols

UDP is used for situations where speed is required but reliability is not. Media streaming is a common example, lots of data needs to arrive to the client quickly where a small number of missed segments can go unnoticed. However, this kind of implementation would be a huge overhead. Even if 3rd party application was used it would not provide much more insight into what is happening to the protocol when the degradation is active. This is why UDP had been left as bare boned as possible and the protocol has been left relatively exposed to allow an effective way to visualise the effects.

3.2.1.3. Other protocols

QUIC

Quick UDP Internet Connections (Hamilton et al., 2016) is an early stage transport layer protocol developed by Google where it was announced in 2013. QUIC is built upon UDP

3. Background

and is designed to provide faster transfer speed in web applications that utilise TCP. It has a multitude of extra features, involving bandwidth estimation to reduce congestion, movement of congestion control into the application space rather than having the control embedded into the kernel meaning applications can adapt quicker and provide more bespoke decisions. QUIC also attempts to reduce latency by the simple inclusion of performing less transfers from that of an equivalent TCP connection. QUIC also includes techniques to attempt the handling of packet loss better than that of TCP. Techniques like: proactive speculative retransmissions ('Important' data is sent more than once).

There were initially concerns about UDP connectivity and the possibility that many users would be behind systems that block UDP traffic. This was mitigated by Chromium connectivity experiments (Roskind, 2013) where 91-94% of Google users could make a UDP connection outbound.

QUIC was not chosen as an addition to this project due to it being in its infancy, where its adoption is still within the early stages. There are however, available open source client and server pairs available and the inclusion of this transport protocol would be a good addition in potential future work.

SPDY

SPDY (Belshe & Peon, 2012) is a Google created transport protocol designed to transfer web content. Its name is not an acronym and is pronounced "Speedy". SPDY works by manipulating HTTP traffic with the goals of reducing web load, latency and improving web security. However, in 2015 the protocol was deprecated by Google (Blog, 2015), this was mainly due to the timing of the release of HTTP/2 where the improvements of SPDY were present in the upcoming version of HTTP. Full support for SPDY was removed in 2016. Due to SPDYs depreciation it was discarded as an option for this project.

3.2.2. Operating Systems

3.2.2.1. Linux

Linux is the broad definition of a family of computer operating systems. The defining characteristics of a "Linux" OS is the free and open source approach and the use of the Linux Kernel. Android, for example, has the largest market share in the mobile OS market (Share, 2015), Android utilises the Linux kernel as the centre for its operating system.

Linux was chosen for this project for its open source element that allows easy access to low-level features in the kernel. The section of the kernel that deals with the filtration of packets for uses like firewalls and packet sniffers is referred to as the "NetFilter". This acts as an API of sorts where packets can be routed to the net-filter queue to await a

3. Background

verdict (accept or drop) before being moved on. This is the basis for the functionality of the program. The program will run on the Raspberry Pi and route all packets that enter the box to the queue where the effects can be applied.

3.2.2.2. Windows

Linux was chosen over the other possible alternative; Windows. The Windows OS is distributed as closed source software under proprietary licences meaning to understand what is happening under the hood is far more difficult. Windows does provide a solution to the packet filtering in the same way that NetFilter does, it is called Windows Filtering Platform (WFP) and it is an API that allows access to the systems services in order to create packet filtering applications.

Windows, however, was not used because of the size of the Windows OS. Figure 3.6 contains the minimum system requirements for Windows 10³

| | |
|-----------------|---|
| Processor | 1 gigahertz (GHz) or faster processor or SoC |
| RAM | 1 gigabyte (GB) for 32-bit or 2 GB for 64-bit |
| Hard Disk Space | 16 GB for 32-bit OS 20 GB for 64-bit OS |

Figure 3.6.: Windows System Requirements

This compared with the full Debian⁴ OS that the stripped down version of the Raspberry Pi's OS is based off:

The Raspberry Pi in this instance will be run with no desktop.

As you can see from Figure 3.7 the Linux based OS has a much smaller set of requirements. RAM usage for example, is 8 times smaller. Therefore, due to Linux's transparency and overall efficiency the OS is far more suited for use in conjunction with the Raspberry Pi.

| No Desktop | |
|-----------------|----------------|
| Processor | 1GHZ Pentium 4 |
| RAM | 128 MB |
| Hard Disk Space | 2 GB |

| With Desktop | |
|-----------------|----------------|
| Processor | 1GHZ Pentium 4 |
| RAM | 512 MB |
| Hard Disk Space | 10 GB |

Figure 3.7.: Debian system requirements

³<https://www.microsoft.com/en-US/windows/windows-10-specifications>

⁴<https://www.debian.org/releases/stable/i386/ch03s04.html.en>

3. Background

3.3. Alternative Solutions

3.3.1. clumsy

A more modern application of network degradation is clumsy 0.2⁵. It provides various options (that can run in parallel) that affect network conditions.

The tool provides a way to affect network conditions. It solves issues involved with simulating harsh network conditions. It does allow for filtering and has a couple of filter pre-sets, it however, does not provide options to simulate live traffic and controls to capture or visualise traffic is not part of the main package.

This tool provides an easy to use and simple tool that requires no extra download and can be run straight from the .exe, however, this only works on Windows machines and it uses the WFP API that was mentioned in Section 3.2.2.2. It also only affects the traffic on the local machine, this is good for testing tools in the development stage of an application but cannot easily be used to simulate degradation across a live network.

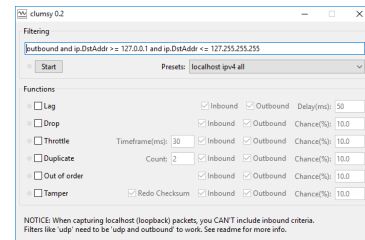


Figure 3.8.: UI of clumsy

3.3.2. TMNetSim

TMNetSim⁶ is another tool that covers aspects of this project. Unlike clumsy, TMNetSim is designed to provide degradation that expands over the network. However, to achieve this it has to be set-up on every machine, this is clunky and time consuming. The degradation script can negate this lengthy set-up with the intended inclusion of the ARP Spoofing (See Section 4.4.3) that will allow rapid deployment of the script between target and router. There is further criticism relating to the ease of use of this application, it has a large number of controls on its interface and will involve a larger learning curve to master compared to other simpler tools, this however might be the bi-product of more functionality; increased complexity.

This tool includes very complex ways of simulating realistic degradation with “Gaussian” and “Normal” delay modes, that attempt to provide more accurate representations of real-world delays. This is a great feature that could drastically improve the accuracy in simulating degradation and will need to be considered as an extra inclusion into this project.

⁵<http://jagt.github.io/clumsy/index.html>

⁶<http://www.tnmurgent.com/appv/en/87-tools/performance-tools/177-tmnetstim-quick-and-easy-network-simulation-tool>

3. Background

3.3.3. Comcast

Comcast⁷ is a tool designed to test systems in non-critical ways. It is only a thin wrapper around tools such as 'iptables', 'tc', 'ipfw' and 'pfctl'. It makes it very easy to add latency, packet loss and error to a data stream. The script is written in Go and runs terminal instances for each command.

The script allows very basic degradation to be performed but, because it is strongly coupled to other tools it means it is not very dynamic and new effects cannot be quickly added to the program.

This solution is very useful in applying quick and easy degradation to a single client but because of its lack of malleability the tool would be hard to apply to many situations.

⁷<https://github.com/tylertreat/comcast>

4. Technical Development

4.1. Tool Architecture

Below is the system architecture for the degradation tool (See Section 4.4.1 for NFQUEUE):

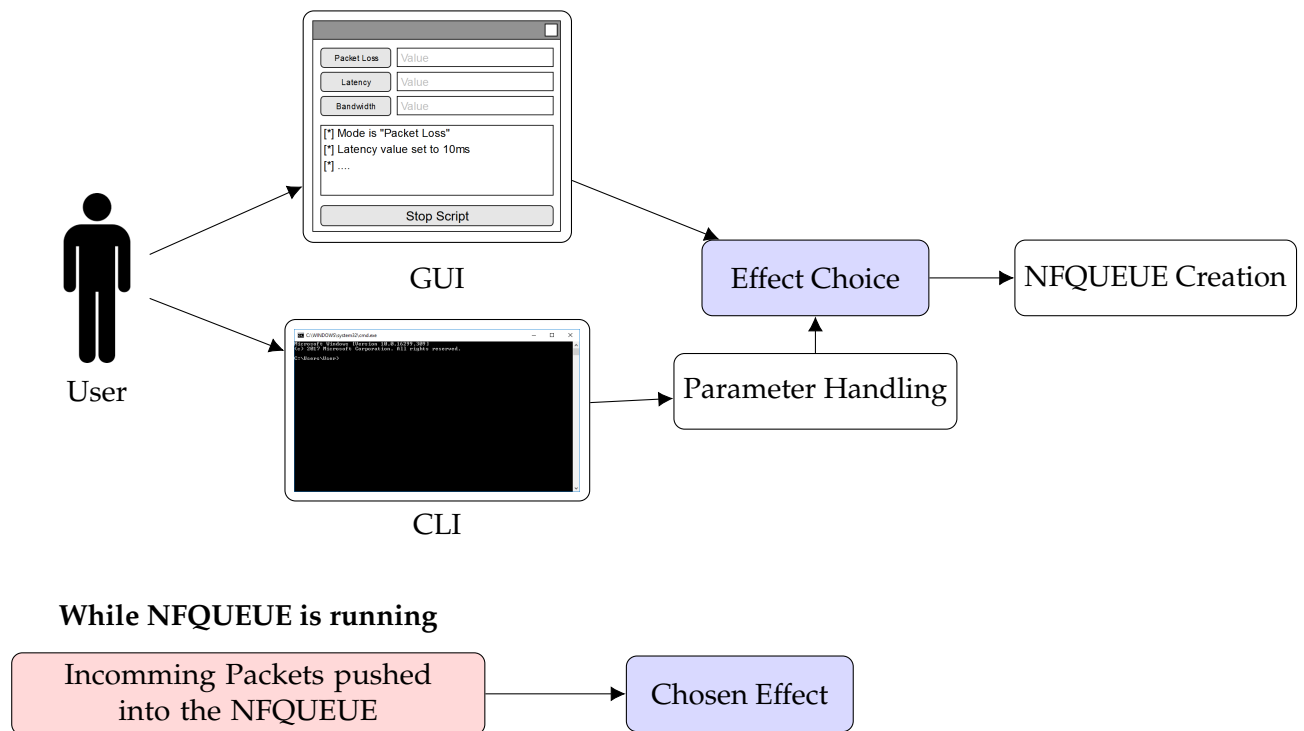


Figure 4.1.: Architecture of the overall degradation tool

It was decided early on that the script was to be controlled by two forms of interfaces; Graphical User Interface (GUI) and a text based Command Line Interface (CLI). This was to allow the tool to be run on the router (CLI) and also run on separate machines (GUI or CLI), making the tool versatile to many situations.

The system design is split into two stages:

- **NFQUEUE Setup**

This section involves the creation of the NFQUEUE object, the binding of effect method to the queue and initialisation of variables dictating preferences.

4. Technical Development

- **NFQUEUE Running**

Each packet that is placed into the queue triggers this section, the handler is called that assigns a task to run the selected effect on the packet to one of the worker threads in a previously created thread pool. This allows custom effects simulation with filtering dynamically applied to all the packets.

4.1.1. Parameter Handling

Parameter handling is only required on the CLI due to the unrestricted input of the interface. Below is the activity diagram for the parameter handling process:

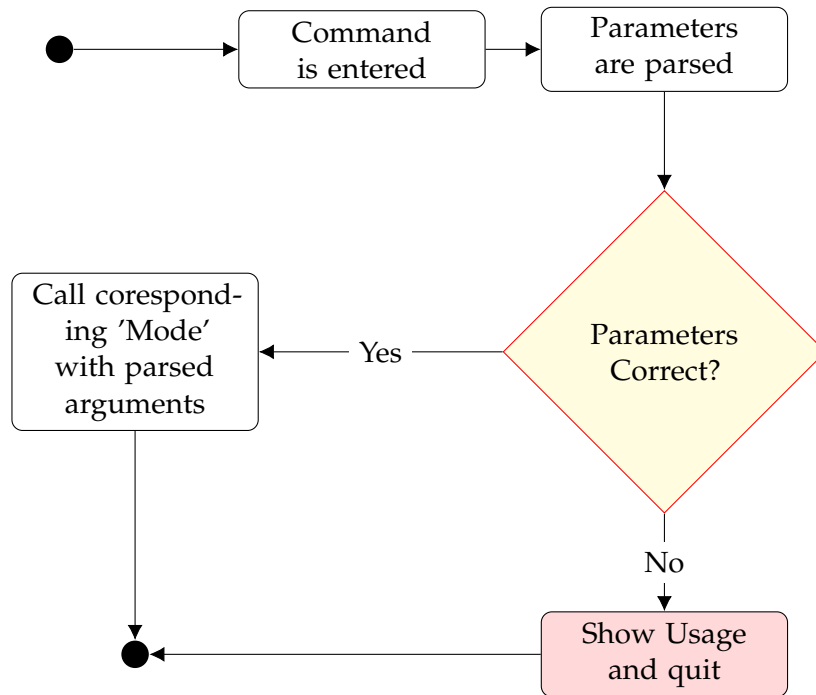


Figure 4.2.: UML Activity Diagram for parsing and handling parameters

The parameter handling as stated previously, is only required if the command line interface is used, this adds a little overhead in processing when using the command line but it is an unavoidable process. The error section when the 'Usage' is printed will stop the entire program from continuing and will print a structured help message with the required format of the command. For most errors, human readable output will be produced to aid in debugging.

4. Technical Development

4.1.2. Effect Choice & NQUEUE Creation

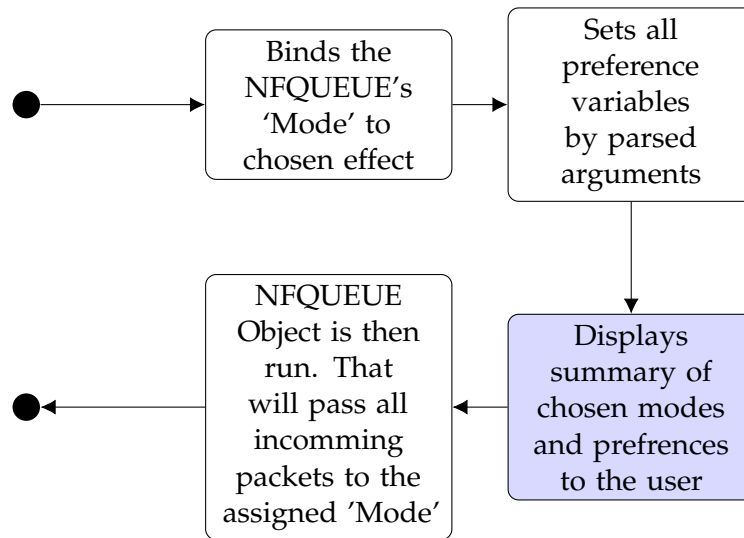


Figure 4.3.: UML Activity Diagram for assigning the choice of effect

Choosing a mode for the program gives the ability to switch between degradation effect and arrangement of effects. This process is exactly the same for GUI and CLI. This is due to the shared code between the different user interfaces, all the CLI does is parse arguments and calls the correct methods and the GUI just directly calls these methods when buttons are clicked.

4. Technical Development

4.1.3. Incoming Packets

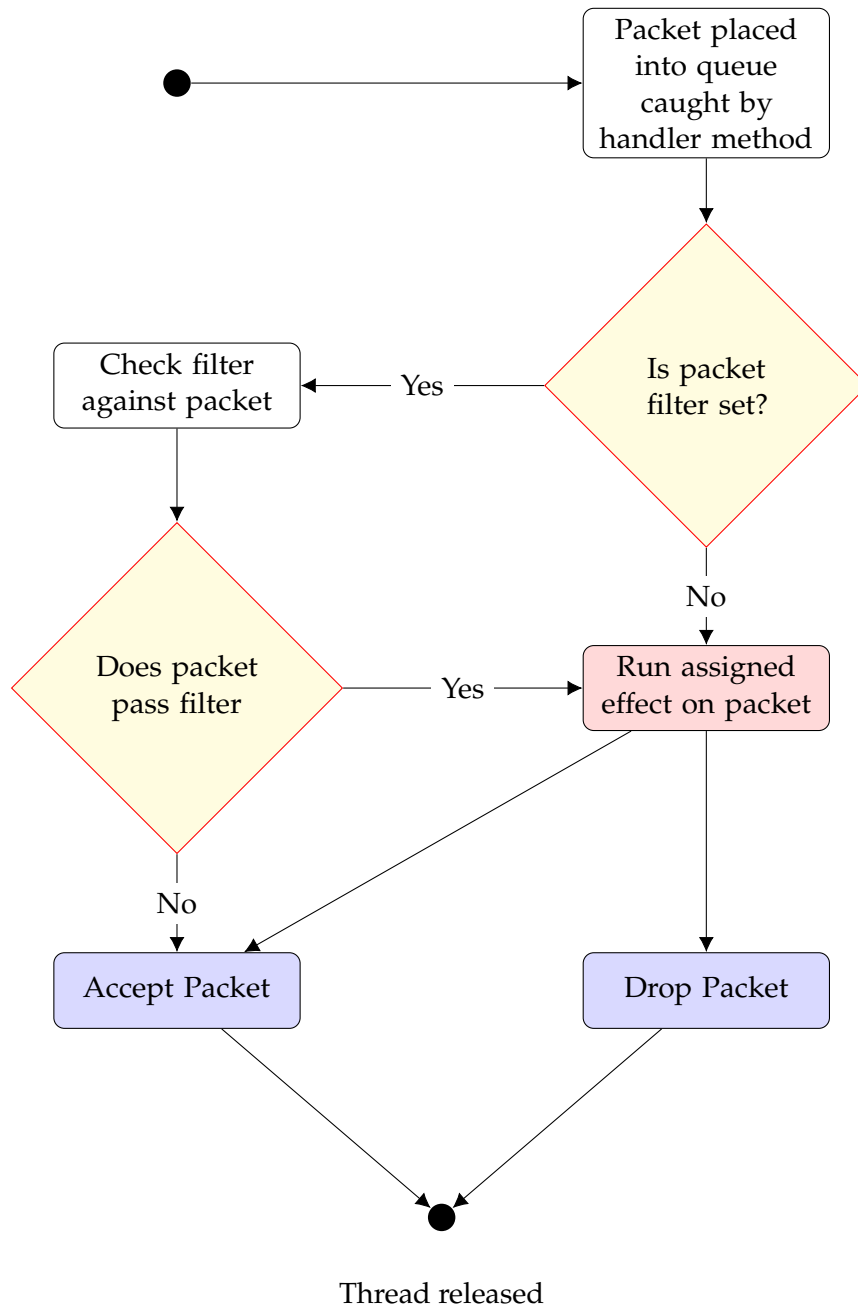


Figure 4.4.: Activity diagram for running an effect on a single packet

Figure 4.4 shows the activity diagram for the process of running an effect on a single packet. The script automatically effects every packet that is placed in the NFQUEUE,

4. Technical Development

this means protocol support does not need to be managed and no code manipulating or monitoring sockets is required.

4.2. Tool Class Design

4.2.1. Effect Class Diagram

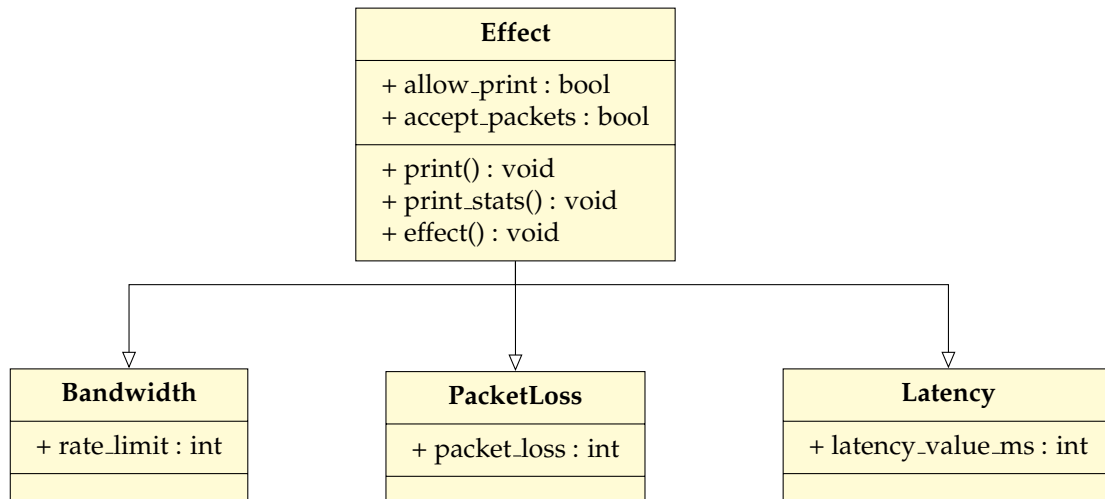


Figure 4.5.: UML Class diagram for producing degradation effects

The above design is for the various effects that will be implemented into the program. They will be self contained within separate modules encased in an object. This was chosen so each effect is modular and self contained from one another. The modularity will improve the potential readability and maintainability of the code base, it will also make the code base much more scalable where effects can be added with high speed because of the lack of repeated code. Each effect will inherit from the base class "Effect", where it will obtain methods and properties used for functionality and boolean variables for allowing printing and accepting packets. These boolean variables are necessary when 'chaining' effects together, where a packet can only be accepted once and only a single print will be required.

4.2.2. Tool Activity Diagram

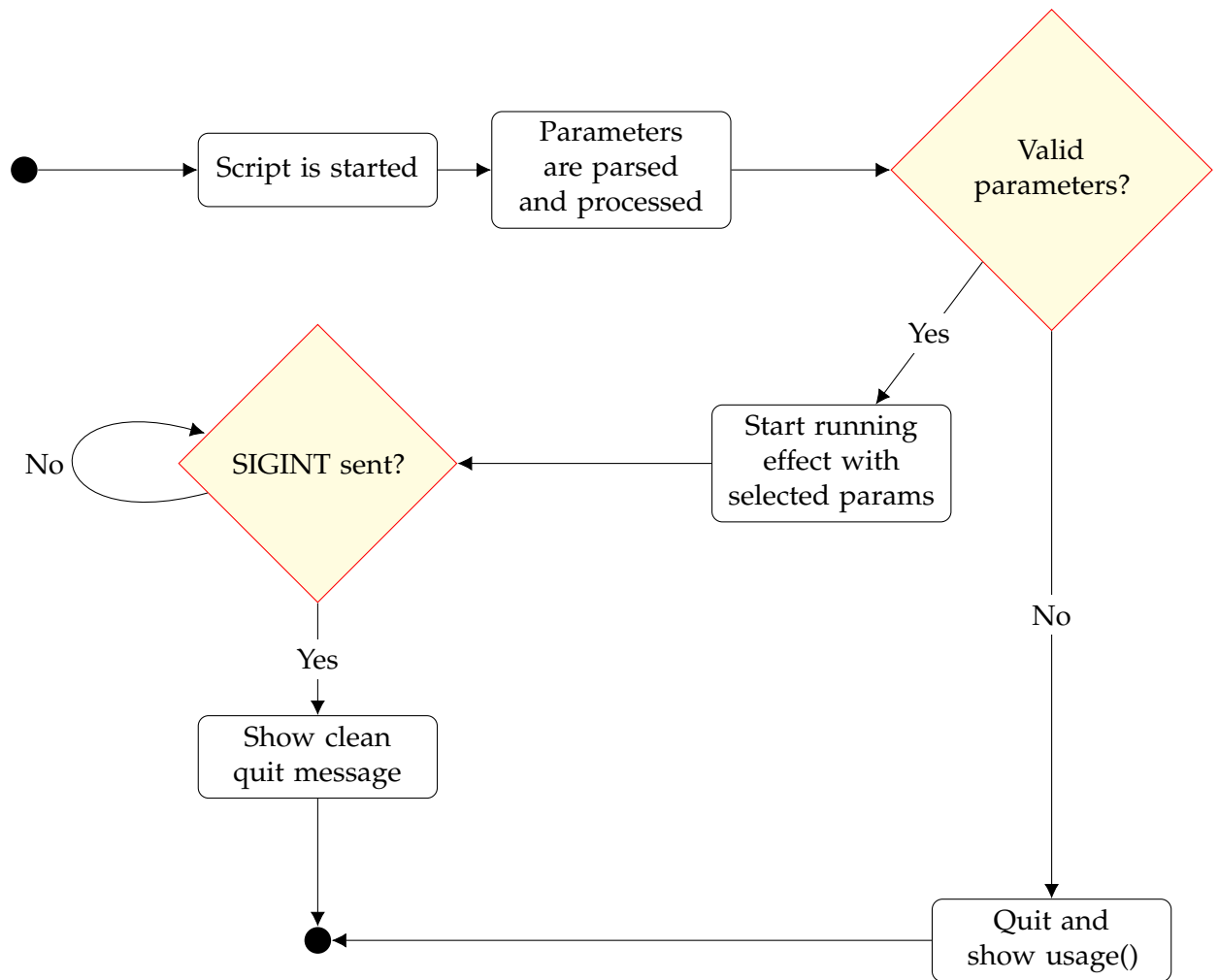


Figure 4.6.: Activity Diagram for the degradation tool

Figure 4.6 above is the activity diagram for the tool that will implement the effect classes above. This tool will be utilised directly from the command line or run by the GUI. The decision of having a single script that is controlled by multiple approaches allows for one central point of functionality and means code can be shared throughout the project. Figure 4.6 shows the design choices for the direct control on the script. SIGINT is mentioned in the centre of the diagram, SIGINT is a built in signal in the Linux operating system that represents an interrupt signal, where it is triggered by pressing “Control + C”, this is the most common/clean way of stopping a script. The other

4. Technical Development

alternatives to closing the script “Control + Z” (SIGTSTP) and “Control + /” (SIGQUIT) will be remapped to send a SIGINT signal and perform the same role, this will mean the program has one tidy point of closure.

4.3. Tool UI Design

The UI is one of the ways to control the degradation. The UI needs to have a way to easily include new controls that link up to parameters in the script, thus making the interface scalable and easy to maintain. A text box is required to display the same output as the terminal window, this can be achieved by ‘piping’ the stdout to a custom section of the user interface. The ‘stdout’ is the data stream that links up to a Linux terminal window, if the stdout points to somewhere else it will display where it is needed. Figure 4.7 shows the initial drafted design for the user interface.

Figure 4.8 contains the actual working user interface for the script. The design has been followed to some extent with only changes occurring in the layout of the controls on the left side designed to make it much easier to include new effects in the future. The inclusion of a window and output text box designed to allow ARP spoofing to occur has also been included in the working user interface.

4.4. Tool Implementation

The tool performs the degradation effect on the computer it is running on, the script is written in Python 3.6 (Pranskevichus & Selivanov) and can be run on a computer running a Linux based OS with the required version of the Python interpreter installed.

4.4.1. NFQUEUE

The script utilises a section in the Linux kernel referred to as the “NFQUEUE”. This is as the name suggests a queue that is stored in kernel memory that will store up packets until the user provides one of the two verdicts: ‘Drop’ or ‘Accept’. The packets are pushed into this queue by the use of ‘iptables’ rules. Iptables is a tool designed to filter

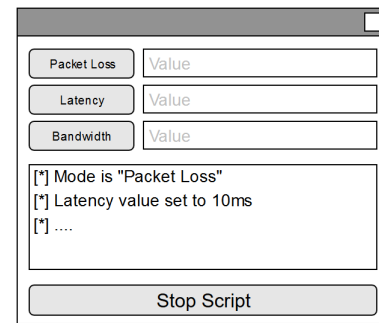


Figure 4.7.: Initial user interface design for the Degradation GUI

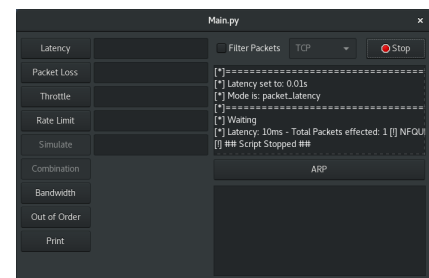


Figure 4.8.: Actual working Packet user interface

4. Technical Development

packets by criteria. Figure 4.9 contains a flow diagram for the path a packet might take through a system, each table can have filtering rules embedded into to it:

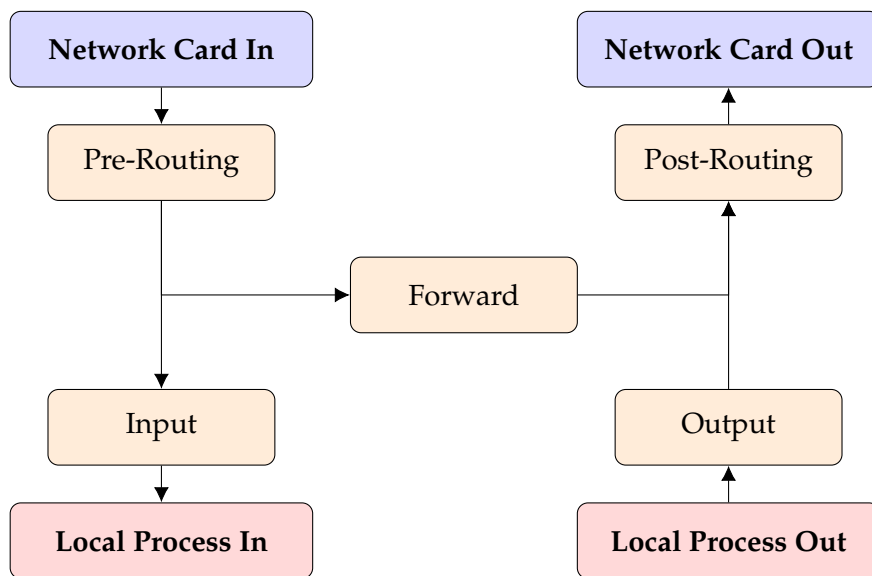


Figure 4.9.: Diagram showing the iptables map

If there was a case where all packets entering the machine need filtering, an iptable rule would be added to the "Pre-Routing" section of the table to catch all packets entering the machine.

To move all packets entering from the network card into the NFQUEUE the iptable rule would be added like so:

```
iptables -A INPUT -j NFQUEUE
```

Where '-A' tells iptables to append a rule onto the 'INPUT' table and '-j' is the rule that affects the packet, in this case pushing it into the default queue.

This forms the main component of the functionality of the script. A series of iptable rules that filter packets into the NFQUEUE and allow to script to perform verdicts on each packet separately. This per packet verdict allows for effects to be easily applied to packets entering and leaving the machine.

Below is code for the creation of the NFQUEUE object along side the iptable rules:

4. Technical Development

```
1 # iptables
2 os.system("iptables -A INPUT -j NFQUEUE")
3 os.system("iptables -t nat -A POSTROUTING -j NFQUEUE")
4
5 # Setup for the NQUEUE
6 nfqueue = NetfilterQueue()
7
8 try:
9     nfqueue.bind(0, mode) # 0 is the default NFQUEUE
10 except OSError:
11     print_force("[!] Queue already created")
12
13 nfqueue.run()
```

Figure 4.10.: Programmatically creating an NFQUEUE

In the code above, the variable `mode` contains a method that performs effects on the packets. Figure 4.11 shows what `mode` would equal if the effect chosen was latency.

The two iptable rules were chosen due to their coverage; they cover all incoming packets to the router (INPUT) and all packets being routed and coming directly from the router (POST-ROUTING). These rule also don't cause the same packet to be pushed into the queue more than once.

```
1 def packet_latency(packet):
2     """This function is used to incur latency on packets"""
3     if affect_packet(packet):
4         assign_thread(latency_obj.effect, [[packet, time.time()]])
5     else:
6         packet.accept()
```

Figure 4.11.: Example of a potential structure of a `Mode`

There are a couple of aspects to note in the above code listing: `affect_packet` is the method that checks the packet against the active filters. `assign_thread` is the method that assigns the object effect method (`latency_obj.effect`) to an idle thread in the pool and `packet.accept()` and `packet.drop()` are the ways the packet is assigned a verdict. This verdict can be assigned at any point in the code and becomes instantly active.

4. Technical Development

4.4.2. Degradation Effects

The script contains the functionality to simulate a plethora of effects. Each effects functionality will be described below. It is of note that any of these effects can be chained together in any order.

4.4.2.1. Latency

Latency as described in the background section, is the delay in initiating a task and seeing its results. Latency is simulated by using a timing mechanism where the arrival time of the packet is saved. The packet is then pushed into the NFQUEUE that triggers a single thread that will hold the packet for a set amount of time. The holding time is calculated by subtracting the amount of time the packet has already been in the script away from the target time. The packet is then marked as 'ACCEPTED' and pushed out of the queue.

```
1 def custom_effect(self, packet):
2     """Thread functionality"""
3
4     # # Dynamic time mode
5     # Parameters contained within a single object
6     if type(packet) is list:
7         packetObj = packet[0]
8         startTime = packet[1]
9
10        # Works out the time difference between
11        # thread conception and now
12        elapsed = time.time() - startTime
13
14        # Take the elapsed time off the target
15        wait_time = self.latency_value - elapsed
16
17        if wait_time < 0:
18            # No waiting
19            pass
20        else:
21            time.sleep(wait_time)
22
23        self.accept(packetObj)
24
25    # # Static time mode
26    else:
27        time.sleep(self.latency_value)
28        self.accept(packet)
```

Figure 4.12.: Code that simulates latency on a connection

4. Technical Development

4.4.2.2. Packet Loss

Packet loss is simulated by first assigning a target value, lets say for this example, 10%. For each packet the script randomly generates a number between 1 and 100, if that value is less than the target value the packet is dropped, and if the value is larger the packet is accepted. This therefore, creates the effect of packet loss. It does, however, require a fair amount of packets to balance out statistically and reach the percentage target.

```
1 def custom_effect(self, packet):
2     """This function will issue packet loss,
3     a percentage is defined where lower
4     values are dropped and higher values are accepted"""
5
6     if self.packet_loss_percentage != 0:
7
8         # random value from 0 to 100
9         random_value = random.uniform(0, 100)
10
11        if self.packet_loss_percentage > random_value:
12            self.dropped_packets += 1
13            packet.drop()
14
15        else:
16            self.accept(packet)
17    else:
18        self.accept(packet)
```

Figure 4.13.: Code that causes packet loss on all incoming traffic

4.4.2.3. Bandwidth

There are two modes created for bandwidth; rate limiting and a simple display. Rate limiting allows the script to limit the rate of bandwidth flowing through the machine and calculates the rate transferred over a period of 5 seconds, if the rate is higher it waits until the rate drops below the target. This gives it the ability to adjust quickly and allows the script to be run for a long period of time without the overall average of the bandwidth affecting future changes. Displaying the bandwidth works exactly the same but without the limit check, this is useful when checking the current download speeds or the max transfer rate.

Limiting bandwidth is displayed in Figure 4.14 below:

4. Technical Development

```
1 def custom_effect(self, packet):
2     """Used to limit the bandwidth"""
3
4     # Adds packet to the backlog
5     self.packet_backlog.append(packet)
6
7     # The algorithm will send until the backlog is empty or
8     # the limit is exceeded
9     while self.rate < self.bandwidth and len(self.packet_backlog) > 0:
10         self.send_packet(self.packet_backlog[0])
11
12         # Packet is removed from the list
13         del self.packet_backlog[0]
14
15         self.calculate_rate_overall_avg()
```

Figure 4.14.: Code that is used to limit the bandwidth of a connection

4.4.2.4. Out-of-order

This effect changes the order of packets coming into the script, this can be used to check its effect on UDP and TCP and can test how quickly these issue can be rectified. It works by queuing up every packet into a list, then a single thread randomly picks an index of that list, accepts the packet and allows it to leave. This, therefore, means at its most extreme the order can be last in, first out.

4.4.2.5. Connection Simulation

Connection simulation isn't quite a degradation effect but its intent is to simulate the effect of degradation on a common connection like 'WiFi' or a 3G connection to see how these are affected by the aforementioned degradation effects. This can be useful in some situations where, for example, testing a mobile applications performance over 3G with heavy latency, the mobile would connect to the script and the script would effect any traffic entering or leaving the handset.

The types of connections were simulated using common values for latency and packet loss. 3G, 4G and WiFi values were obtained from Ofcom studies (Ofcom, 2017) (Ofcom, 2015).

4. Technical Development

4.4.2.6. Jitter

Jitter mode clumps and separates packets in their transmission. Therefore, causing variation in inter packet timing. This is required test how the protocol deals with jitter in the connection. Jitter as explained in the background is the intra-packet latency i.e. the difference in time between arrivals of packets.

4.4.3. ARP Spoofing

The script also supports an ARP spoofing mode that allows it to sit between a gateway and a specified target. This means the script can perform all its functions on a single target with a very quick deployment time. As touched on in the background (See Section 3.1.1.3) the ARP protocol performs no authentication on any changes to the ARP cache meaning that you can perform a man in the middle attack and route traffic through a machine of your choice.

The script begins by grabbing the MAC addresses connected to the provided gateway and target IP addresses. Two ARP reply packets (denoted by the '2' opcode) are sent out. One packet telling the victim that the current machines MAC address maps to the gateway, and the other packet telling the gateway that the victims IP maps to the current machines MAC address.

Below is the process visualised:

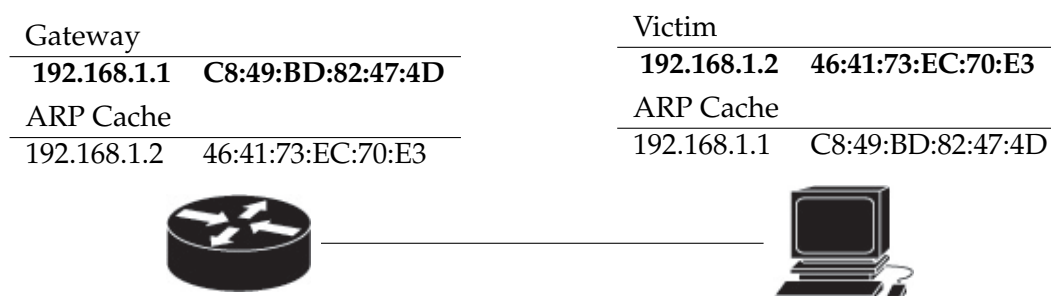


Figure 4.15.: Diagram displaying ARP Cache of a small network

Figure 4.15 shows both connected ends have gone through the process to resolve the IP addresses to MAC addresses, you can see above each device their corresponding IP, MAC and ARP Cache. The ARP cache, as mentioned previously, is a record linking IP addresses to MAC addresses.

4. Technical Development

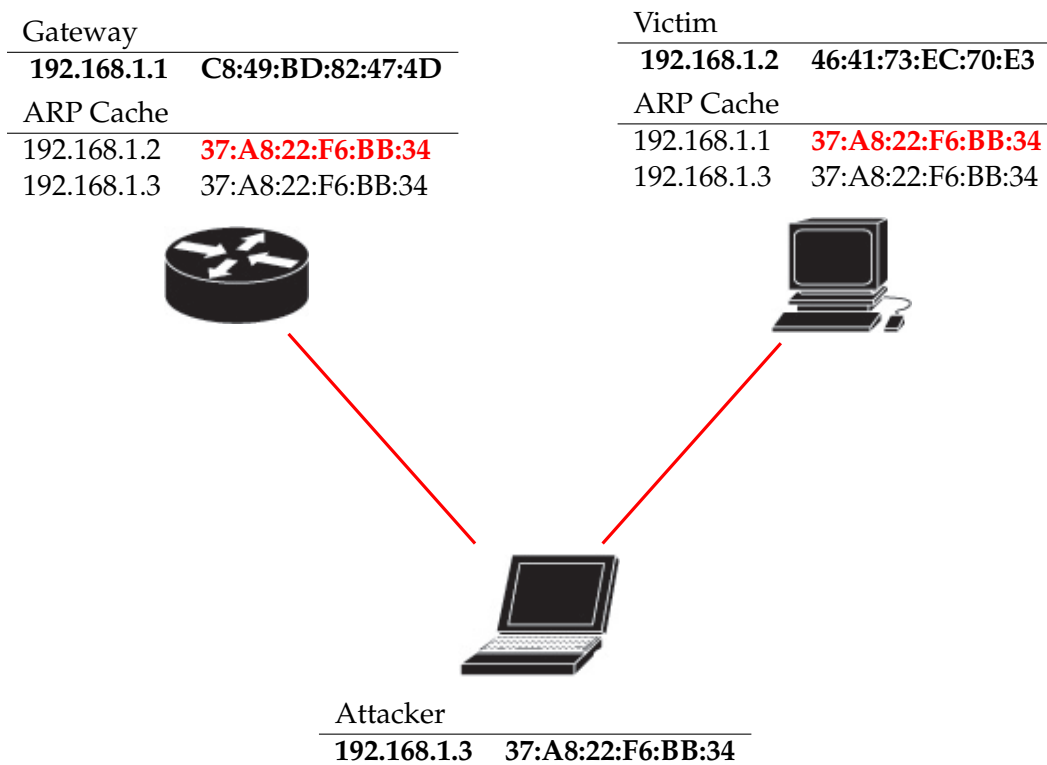


Figure 4.16.: Diagram visualising the ARP Tables after a spoofing attack

After the 3rd device connects to the network, both devices gain a record of its IP and MAC address in their ARP cache.

After both spoofed packets have been sent out, the ARP cache for each receiving end gets updated. The text in red of Figure 4.16 shows the changes caused by the spoofed ARP packets. Now when either end wants to talk to each other it will resolve the MAC address and send it with the new MAC address in the table, this will therefore, get routed through the attacking PC and will allow the script to receive all the traffic between the two parties.

4.4.4. Network Attacks

Two network attacks were implemented into the script, these served the purpose of providing very heavy artificial traffic designed to create heavy loads or interfere with integral protocols. Not many attacks were added as they do not provide much purpose apart from stress testing and they reside on the edge of the scope of this project. There will also be a brief discussion into possible techniques to mitigate the effects.

4. Technical Development

4.4.4.1. UDP Flooding

UDP flooding is a very simple attack, lots of UDP packets are created and sent over a network stream very quickly. This attack is designed to 'flood' the buffers in the receiving machine and slow down an internet connection. The script creates 10 or so threads that create and send packets concurrently, this amount of threads easily reaches the max throughput of the test machines NIC card and massively reduces internet connection on the receiving end.

The attack could be detected and prevented by a script that monitors the network transfers rates, this attack will create a huge spike in the transfer rate that can be detected and all traffic would be blocked from the sending party.

```
1  def effect(self):
2      """The main body of the attack"""
3
4      # Destination
5      port = random.randint(0, 65535)
6
7      # Creates a UDP packet with 1024 bytes of random data
8      # Scapy is used to construct the packet
9      pkt = bytes(IP(dst=self.target_ip) / UDP(sport=52, dport=port) /
10                  Raw(load=random._urandom(1024)))
11
12      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
13
14      while self.running:
15
16          # Uses the previously created socket to send
17          sock.sendto(pkt, (self.target_ip, port))
18
```

Figure 4.17.: Code displaying the sending of UDP packets

Figure 4.17 above is a section of code from the UDP flooding attack script. As mentioned previously around 10 threads will run the code at the same time in an attempt to maximise the throughput of UDP packets. The effects of the attack are visible on the target end and causes a visual degradation in the connection speed.

4.4.4.2. ARP Spamming

This attack works by once again exploiting the lack of authentication of the ARP protocol. In the entire network mode the script starts by detecting all active hosts on a network, it then starts sending out falsified ARP reply packets with randomly generated MAC addresses inside. This results in all computers on the networking having incorrect ARP

4. Technical Development

cache values and resulting in computers incorrectly resolving IP and MAC address combinations and traffic being blocked.

```
1 def start(self):
2     while self.running:
3
4         # Talks to every machine and changes the ARP value
5         # The list of hosts is obtained by an NMAP scan
6         for host_dst in self.active_hosts:
7             for host_src in self.active_hosts:
8                 if host_dst != host_src:
9
10                    # Send the ARP packet - rnd_mac_addresses() creates
11                    # a fully random MAC address
12                    self.send_arp_packet(host_dst,
13                                         host_src,
14                                         self.rnd_mac_addresses[count])
15
16
17 def send_arp_packet(self, dst, src, mac, op_code=2):
18
19     # Construction of the ARP Packet
20     pkt = ARP(op=2,
21               pdst=dst,
22               psrc=src,
23               hwdst="ff:ff:ff:ff:ff:ff",
24               hwsrc=mac)
25
26     send(pkt, verbose=0)
27
```

Figure 4.18.: Code from the ArpSpamming.py script

Figure 4.18 above shows the attack effecting the entire network. The scan finds all active hosts and returns them as a list that is then used as source and destination IP addresses that ultimately end up affecting the entire local networks ARP cache.

This attack is relatively easy to execute but can simply be prevented by issuing static ARP tables for each machine, with a few authenticated machines allowed to perform changes. This static table prevents unsolicited ARP replies from changing IP and MAC address combinations and also prevents ARP Spoofing from occurring on that network.

4.4.5. Router Design

The router is required to create an access point and allow access to the internet. Traffic will be routed through the router that is destined for other location, this will mean

4. Technical Development

traffic will flow through the "Forward" section of the iptables map (Refer to Figure 4.9). Therefore, a rule is required to catch any traffic from flowing into the machine.

The router also needs a couple of tools to manage the assigning of IPs (DHCP server) and creation of an access point (Hostapd ¹). The exact implementation of these tools will be explained in the next section.

4.4.6. Router Implementation

Hostapd has been used very simply to create a single protected access point, this however requires a WiFi dongle that supports Access Point Mode - the WiFi dongle used for this project was the TP-LINK TL-WN722N and was strong enough to provide ample range.

Figure 4.19 contains the configuration script used:

```
1 interface=wlan0
2 driver=rtl871xdrv
3 ssid=NETWORK_NAME
4 hw_mode=g
5 channel=3
6 wpa=2
7 wpa_passphrase=PASSWORD
8 wpa_key_mgmt=WPA-PSK
```

Figure 4.19.: Hostapd script used to create the access point

This script is used to define the password, SSID and any other configuration regarding the WiFi access point.

The router also needs to assign IP addresses to newly connected devices, it does this by utilising DHCP. DHCP stands for Dynamic Host Configuration Protocol and as the name suggests configures connected hosts. A DHCP server is created on the Raspberry Pi that will automatically assign a local IP to that device and allow it to communicate with other devices on the LAN and the internet.

¹<https://wl.fi/hostapd/>

4. Technical Development

```
1 subnet 192.168.10.0 netmask 255.255.255.0 {
2     range 192.168.10.10 192.168.10.20;
3     option broadcast-address 192.168.10.255;
4     option routers 192.168.10.1;
5     default-lease-time 600;
6     max-lease-time 7200;
7     option domain-name "local-network";
8     option domain-name-servers 8.8.8.8, 8.8.4.4;
9 }
```

Figure 4.20.: Configuration file used to create the DHCP Server

In Figure 4.20 above are the commands to define the subnet, DNS servers (Google's are used) and range of devices allowed on the network. This DHCP-Server allocates IP addresses ranging from 192.168.10.10 - 192.168.10.20, with the router being allocated 192.168.10.1.

4.5. Testing

There are various sections of the project that require a separate testing plan:

- Traffic simulation programs
These are the client server programs that will simulate traffic over the synthetic net. These tests will need to make sure each client and server performs its role correctly.
- Packet script and effects
This is the script that will run on the custom router. Tests will be checking effects do their basic jobs and the script can be controlled effectively.

4.5.1. Traffic simulation programs

The test plan for this section will need to check all the intended functionality of each window. There are three aspects of each that will need testing:

- UI
Tests will be created that click buttons and check the user interface works effectively through automated testing.
- Business Logic
Code behind the UI will have the relevant methods testing with expected and actual results.

4. Technical Development

- Real world usage

The involvement of multiple windows and more complex functionality can be tested by using automated testing scripts.

Appendix A contains the test plan for the program, each separate program has had its user interface and business logic tested and programs that are to be used together have had “Live” tests created to check their interaction together. The automated tests have been achieved by using the CUIT (Coded User Interface Tests) ² that are built into Visual Studio 2017. These allow clicks and movements to be recorded and repeated.

4.5.2. Packet Script

It was decided that a separate test plan was needed for the script that will be run on the router. This was because its design is considerably different to that of the traffic simulation programs. Each individual effect needs a basic test that uses a loopback ping test to simulate incoming traffic where a criteria is looked for. For example, to test packet loss, the test pings the script until a packet is lost or a time-out is reached, this can perform a basic test on the functionality of the effect. Each effect will also require validation for the passed parameters, there will be a test created that will test various values inside and outside of the validation range.

4.5.3. Testing correctness

Figure F.1 in the appendix contains the correctness table for the UDP Client and Server. Each test was run and the outcome was compare to what was expected. Out of 25 tests 100% passed and produced the correct results. These tests are implemented into Visual Studio ³ and are run every time a change is implemented into the server or client. Figure 4.21 contains a screen shot of the test window after all tests have completed.

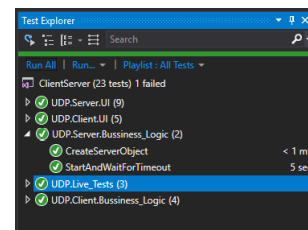


Figure 4.21.: Test explorer

The degradation tool also required testing for the various effects. The effects all have a basic test that checks for a successful start up and a more in-depth test that checks the effects functionality against ping packets sent over the localhost. The two attacks also needed testing, these however do not include a test on their functionality as it would be very difficult and time consuming to create a test that tests the effectiveness of the attack. Figure F.2 in the appendix contains the testing correctness table for the script tools; all tests passed.

²<https://msdn.microsoft.com/en-us/library/dd286726.aspx>

³<https://www.visualstudio.com/>

4. Technical Development

4.5.4. Testing methodology considerations

Initially the methodology chosen to start development of the traffic simulation programs was “Test Driven Development” (Beck, 2003), this was a tight methodology that increased development time and gave the added benefit of showing that new changes have not broken older functionality. This methodology was good to start off the project and allowed for a tight structure to be created where no time was wasted debugging previously working code, but as mentioned this methodology slowed development time down, this was chosen to be abandoned after 3-4 weeks to an ad-hoc approach. This change was due to time restraints on the project and the less-important role that the traffic simulation programs had on the project compared to the degradation simulation script. The degradation script was developed in the ad-hoc approach. Ad-hoc was chosen due to issues stated previously about the unknown aspects of the script and how it would operate and it was decided that the time invested to write up test scripts would assume too much and would risk them having to be completely rewritten. Tests therefore, were added alongside new functionality.

4.6. Experiential design

4.6.1. Protocol Degradation

The tool was tested against the various protocols and rough results were obtained to validate the correct function of the degradation tool.

4.6.1.1. HTTP Experiment

HTTP traffic was tested through an automated script. The degradation tool was activated for packet loss and latency and the website was downloaded using the *wget* command. The website chosen was the Wikipedia page for the University of Hull ⁴. This page was chosen due to it being a good example of a simple web page.

Appendix B contains the results for the HTTP testing. Figure B.1 contains the latency results. The page load time value is the average taken over a series of 5 repeat test. The results were as expected; the increase in load times is linear with the increase of latency.

4.6.1.2. FTP Experiment

FTP traffic was tested with two hosts connected to the Raspberry Pi router; one host the client and the other the server. Transfer speed of a 5MB file were tracked and recorded.

⁴https://en.wikipedia.org/wiki/University_of_Hull

4. Technical Development

Appendix C contains the results for the experiment. As expected as the degradation becomes more violent the transfer time increases. As latency increased, transfer speed of the file decreased in a linear fashion.

4.6.1.3. UDP Experiment

UDP was tested using the custom server client created for visualising the packet loss, therefore, only packet loss was tested for UDP. In a similar way to FTP a server and client both connected to the router running the tool. The state of the user interface was recorded as significant packet loss values.

Appendix D contains the progression of the interface as the packet loss value becomes more extreme. The clear representation of arrived and lost packets makes it immediately apparent of the effect of packet loss on a UDP connection. UDP does not resend when packets are lost, so in situations with abnormally large packet loss values an application relying on UDP can be completely incapacitated.

4.6.2. Latency Accuracy

In the initial stages of the project there were experiments that tested the effects of the degradation on the network. The tests were performed on the loopback (Internal network 127.0.0.1) with ICMP ping packets.

This test was designed to show the accuracy of different ways of simulating latency, the first way being a static timer that just causes the thread to sleep for a set amount of time or a dynamic timer that calculates the elapsed time since the creation of the thread and the command telling the thread to sleep, the dynamic timer takes this elapsed time off the target latency value.

Figure 4.22 visualises the results in a single graph. The relatively high percentage error in the small ranges of latency values is due to the error proportion being a much larger chunk of the overall target latency and therefore being a higher percentage error.

4. Technical Development

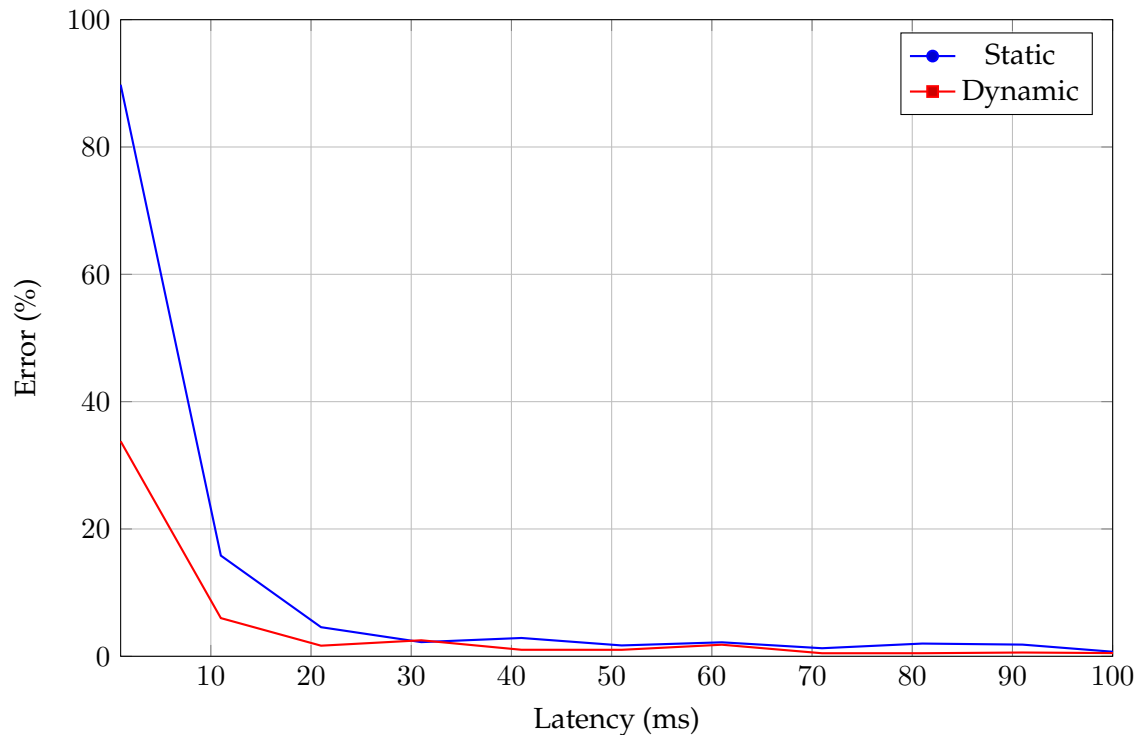


Figure 4.22.: Error percentage in simulating latency

Figure 4.22 shows the dynamic form of issuing latency is overall more accurate in simulation. However, it is still not perfect and there seems to be a small overall error present. This performance is however, more than suitable for the scope of this project.

4.6.3. Visualising effects

The most effective way to visualise real world connections were running tests on SpeedTest.net⁵. This is very useful to quickly visualise a certain effect on the network.

As you can see from both images, the 100ms has evidently been applied effectively and the speed of the connection has dropped by around 10Mbps. This means in real world terms the time taken to download a 1GB file would take 35 seconds longer to download. This is not a considerable reduction but the latency is having an obvious effect on the network quality.

⁵<http://beta.speedtest.net/>



Figure 4.23.: The initial connection speed



Figure 4.24.: Network speed with a latency of 100ms

4. Technical Development

4.6.4. UDP Flooding

To test the effect of the UDP flooding required experimentation. A single target was selected on a network just containing the attacker and victim and, like previous tests, SpeedTest.Net was used to obtain bandwidth results. Each test was run with the same computers on the same network. As you can see from the two images the effect of the UDP flooding is devastating on a connections speed, taking a fast internet connection and reverting it to a crawl. If more computers running the script were active the internet connection could easily be fully blocked.



Figure 4.25.: Initial speed without UDP flooding active



Figure 4.26.: Speed test with UDP Flooding active

5. Evaluation

5.1. Experimentation Results

5.1.1. Test Environment

The test environment used to create these experiments has been kept exactly the same for every experiment. TCP transfer has been simulated over the localhost using a tool called Iperf (Gates et al., 2003) that allows the creation of a client server pair where information such as Cwnd and RTT can be obtained from the output of the tool.

The specification of the computer used for these experiments is contained in Figure 5.1.

| | |
|-------|--------------------------|
| Model | Lenovo ThinkPad x240 |
| OS | Linux 4.14.13-1-ARCH |
| CPU | Intel i5-4300U @ 1.90GHz |
| NIC | Intel Wireless 7260 |
| RAM | 4GB |

Figure 5.1.: Test machine technical specifications

5.1.2. Effect of Packet Loss on Download Speeds

Figure 5.2 below shows the download speed measured in Mbps against the increasing rate of packet loss caused by the degradation tool, as you can see the download rate only reaches around 25%, this is due to the connection test failing at anything higher. The tests were performed on the localhost to reduce any residual degradation that may exist already in the current network.

5. Evaluation

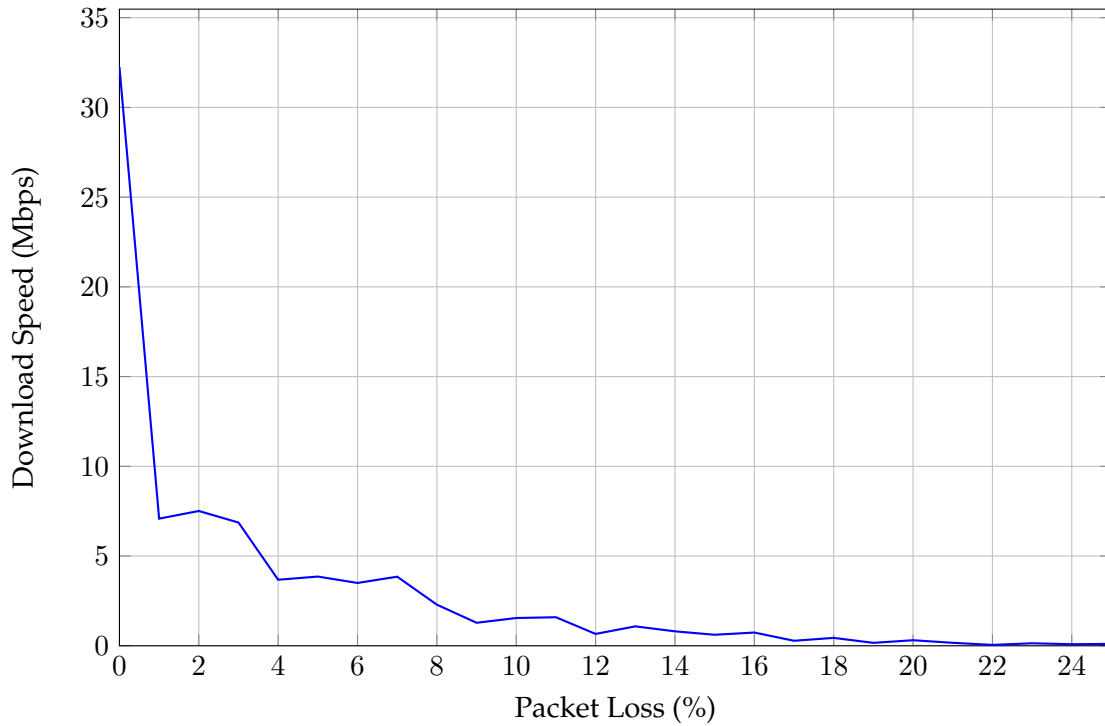


Figure 5.2.: Graph displaying Packet Loss (%) against Download Speed (Mbps)

My initial hypothesis for the sharp decreases in download rate when packet loss increases from 0% to 1% in Figure 5.2 is due to the congestion control mechanisms built into TCP. TCP as mentioned in the background section (See Section 3.1.1.1 Congestion Control) allows for multiple devices to co-exist on a network and needs to dynamically balance network resources for every client. It balances available network resources through congestion control algorithms where in this context packet loss is assumed as a sign of congestion. It adapts to signs of congestion by reducing its transfer rate. This adaptation means the transfer rate drastically reduced until it reaches a platform. The transfer rate is dictated by a few factors, one known as the congestion window (Cwnd) and the advertised size of the receiving window (Rwnd). The overall window size is the smaller of these two numbers, and the smaller the window size, the less packets can be sent in one time frame. The congestion window therefore is reduced every time packet loss is detected.

The test devised to prove this hypothesis will involve once again an iperf client/server where a packet is dropped after a time period while the congestion window size is tracked. Below is a graph showing the result of a test spanning 10 seconds where a single incoming packet was dropped every 1 second, the graph is measuring Cwnd sizes on the y axis and Time on the x axis.

5. Evaluation

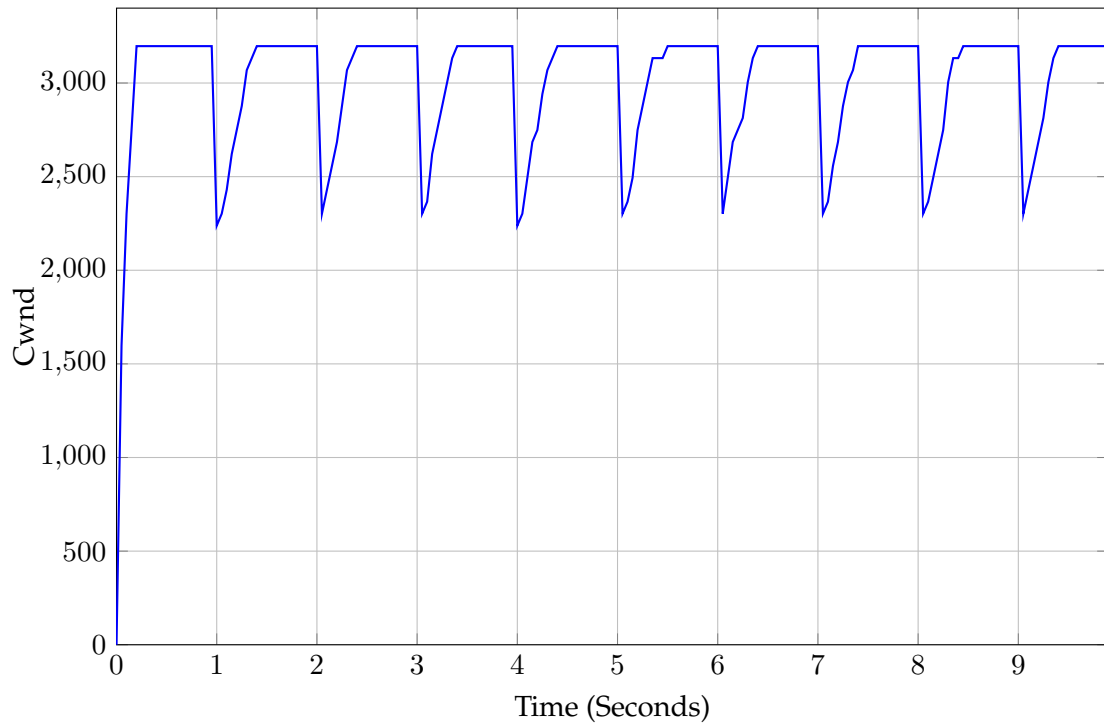


Figure 5.3.: Experimental designed to show how packet loss effects the Cwnd value

As you can see around the 1 second mark in Figure 5.3 there are almost identical drops in the Cwnd size. These controlled drops demonstrates that packet loss causes the congestion window to reduce in size and if packet loss is substantial and frequent enough it can cause a huge drop in transfer rates.

This is known in live networks as the 'sawtooth' pattern where the client is probing for more bandwidth. This is demonstrated in Figure 5.4.

5. Evaluation

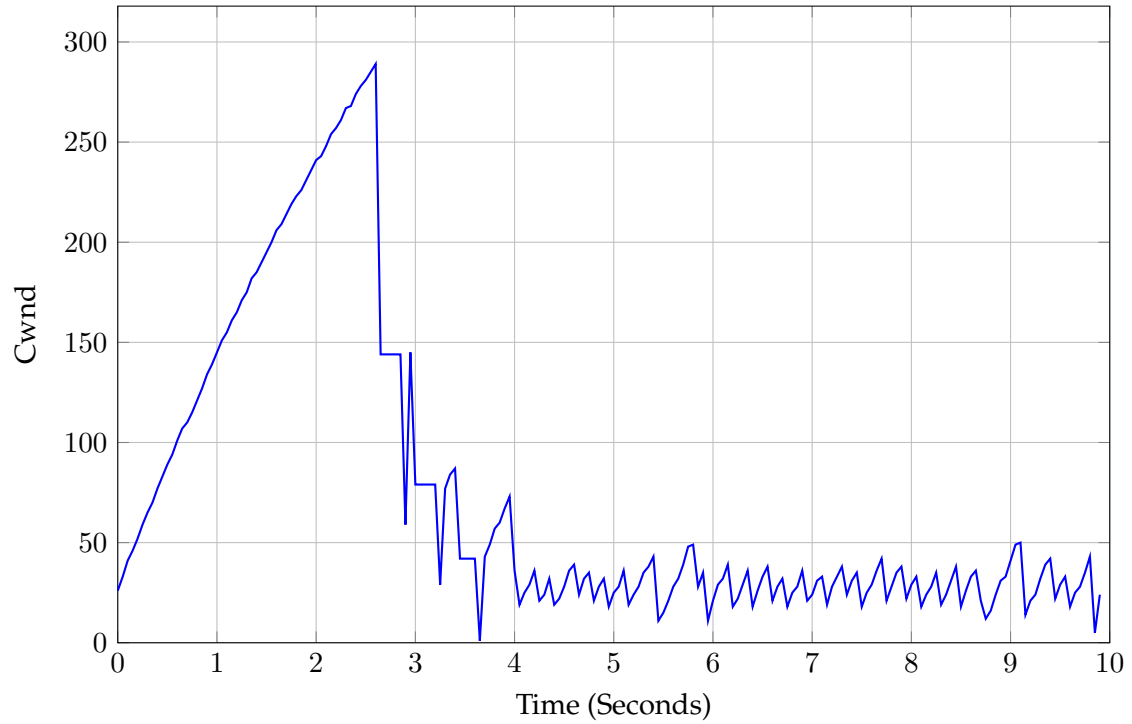


Figure 5.4.: Live Cwnd capture on a real-world connection

From the period of 3 seconds onwards is the main operation of the TCP protocol to continuously test the network for more bandwidth. The maximum cwnd of the connection can be worked out by drawing a line above the spiked sections.

5. Evaluation

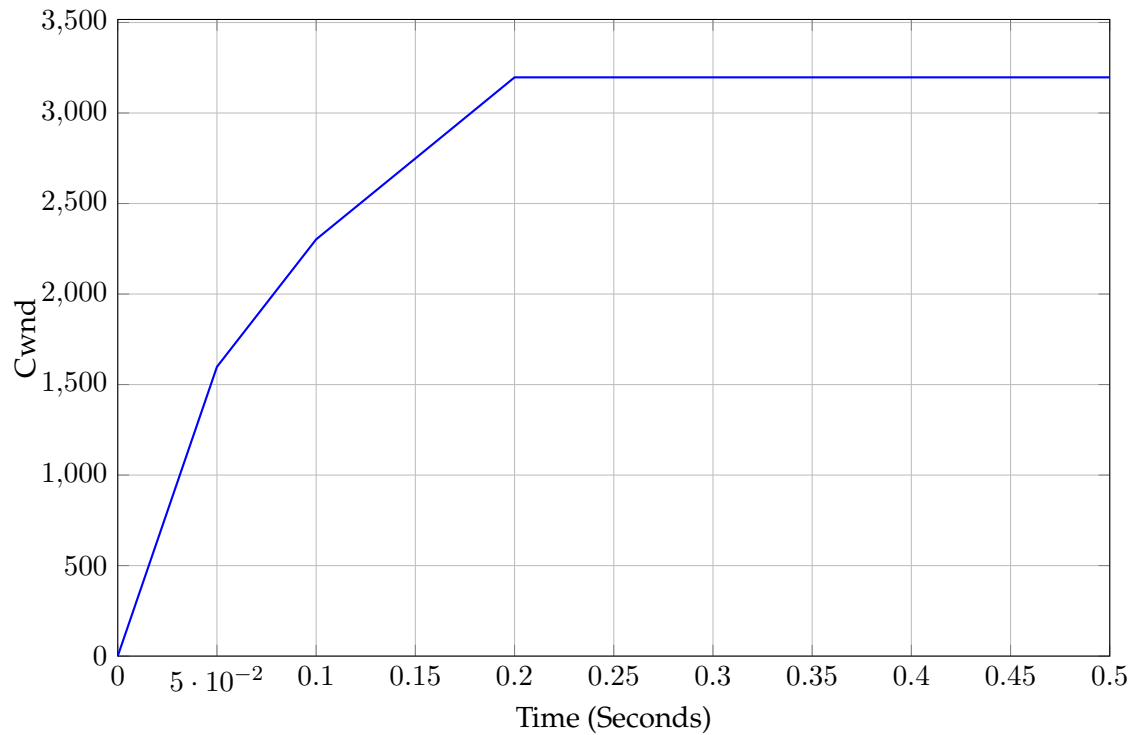


Figure 5.5.: Graph displaying TCP Slow Start

The initial increase in transfer rate in Figure 5.3, is displayed in Figure 5.5. This is known as the TCP Slow Start and this is used by the protocol to naturally align itself into the network eco-system. The protocol starts off with a modest transfer rate to prevent situations where an initial abnormally large transfer rate would fill buffers and cause network congestion. It adjusts by incorporating the techniques explained above to manually increases its transfer rate to fit naturally into the current network.

5. Evaluation

5.1.3. Effect of Latency on Download Speeds

Figure 5.6 shows the impact of latency on the connections throughput. The data was collated using an iperf server and client pair running over the host machine's localhost.

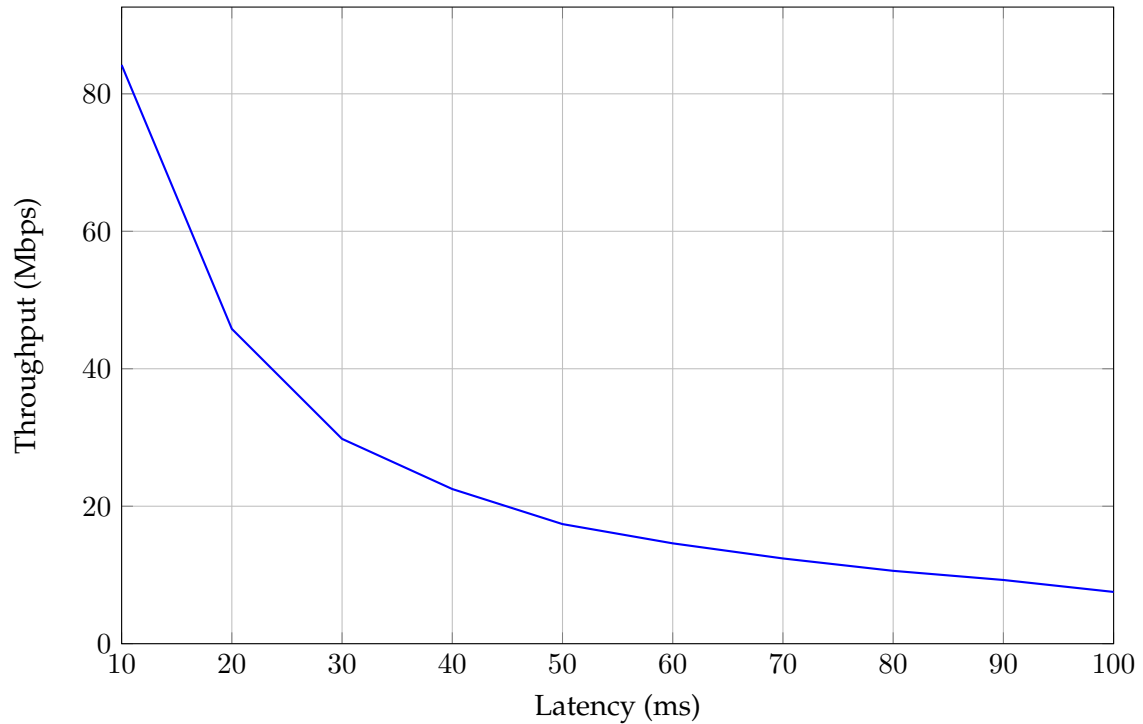


Figure 5.6.: Graph displaying Latency (ms) against Throughput (Mbps)

As it is evident, latency has an inverse relationship with that of the throughput. The reason for the reduction in throughput is due to the need to wait for acknowledgements. When TCP has sent a full window size (See 3.1.1.1 Sliding Window) it will wait until acknowledgements have been received before sending more packets. This requirement to wait therefore means higher latency values increase the idle time of a TCP connection where extended periods of time are spent waiting for acknowledgements.

5. Evaluation

Inconsistency in the latency value also has an adverse effect on the throughput of the connection due to packets being deemed as “lost” because they take longer than the time defined in the *RTO* value. The *RTO* variable defines the amount of time to wait before issuing a retransmission and deeming a packet as lost, this can be deemed as packet loss and the TCP protocol will act accordingly. RFC 6298 (Paxson et al., 2011) defines the calculation of *RTO* to involve the two state variables: *RTTVAR* (Round-trip time variation) and *SRTT* (Smoothed round-trip time). These values are updated each time there is a new RTT measurement.

$$RTO = SRTT + MAX(G, K * RTTVAR)$$

where:

K is 4

G is Time Granularity

Figure 5.7.: Equation for calculating *RTO*

Figure 5.7 contains the main equation for calculating the *RTO* value. This equation is used to take into the residual RTT of a network when calculating the expected retransmission time. This means large networks with large latency values are taken into account.

5.1.4. Latency effect on Slow Start

Slow start (See Section 3.1.1.1) is used to slowly ramp up the congestion window size until a suitable level is found. The most recent RFC 5681 (Allman et al., 2009) that defines the Slow Start mechanism defines the behaviour as:

During slow start, a TCP increments cwnd by at most SMSS bytes for each ACK received that cumulatively acknowledges new data.

Each ACKs frequency therefore is determined by the RTT, i.e. the time it takes for a segment to be sent plus the time required to acknowledge that segment.

A test has been devised to demonstrate the ability of the tool to simulate a certain RTT and therefore affect the slow start thus showing how the protocol functions.

5. Evaluation

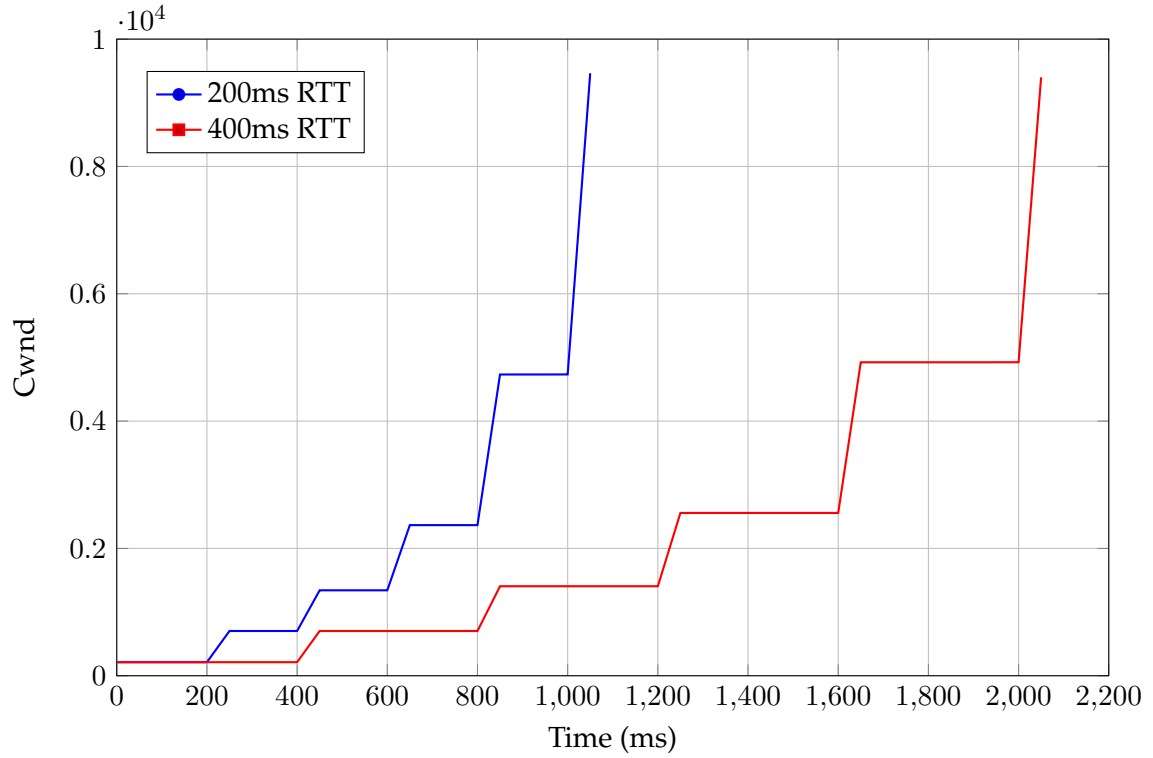


Figure 5.8.: Graph displaying Cwnd in the Slow Start period with two different RTT values

Figure 5.8 shows the result of the experiment. The RTT was simulated using the latency effect of the script where the latency is doubled when forming the RTT (Two equal distances). Therefore 200ms RTT was simulated using 100ms on the script.

As you can see from both lines the increase in Cwnd value occurs in intervals of the respective RTT values, this is as expected due to the statement contained in the RFC document. Doubling the value of the RTT increases the time taken to finish the slow start of a connection by double thus the increase is a linear relationship.

The overall error in simulating RTT was also recorded and the overall error was 1.258% and 1.151% respectively for each plot. The error was calculated using the average RTT.

5. Evaluation

5.2. Project Achievements

Initial Goal

Create a custom simulated network that can demonstrate and visualise network degradation and common DoS attacks, this is so network engineers can identify weak spots and points of strain

5.2.1. Objective One - Achieved

Develop a fully working client server pair that communicate using HTTP/1.1 and FTP over the TCP protocol alongside a client and server that use the UDP protocol (See Section 2.1)

This objective initially involved the creation of a custom client server solution for HTTP, FTP and UDP. This involved a lengthy development period where each client and server would require the implementation of required features for the project. It was decided after a period of 3.5 weeks that the simulated network having a custom solution created no added value to the initial goal of the project, this is why HTTP has been replaced by pre-existing browsers such as Google Chrome ¹ or Mozilla Firefox ² and and FTP solution replaced by FileZilla's ³ server and client pair. These are all projects that have had multiple years invested into their development and offer increased stability and selection of features. They have all the features required to meet the goal of this project. Therefore, were chosen as more suitable solutions. UDP on the other hand was left as a custom solution, for the sacrifice of realism the UDP client and server shows a clear visualisation of UDP packet loss.

However, even with the changes each sub point of the objective has been met.

5.2.2. Objective Two - Achieved

Create a program that runs on a Linux based OS that can be used to simulate degradation and attacks. This program will then be run on the router. (See Section 2.2)

In the aims and objectives section lists out the criteria required to meet this objective. The created solution meets all of these criteria. It provides the functionality to simulate packet loss, latency and bandwidth plus a selection of other effects. The other effects were chosen to provide more depth to the project and allow the possibility to chain effects together to simulate more sophisticated degradation. The script also allows UDP flooding and ARP spamming. It contains a mode to increment the TTL but this mode was discarded from the main advertised functionality due to its inherent difficulty in displaying its effect on a connection due to the hard task for forming loops in a networks structure. This script is also functional on the set-up Raspberry Pi Router.

¹<https://www.google.com/chrome/index.html>

²<https://www.mozilla.org/en-US/firefox/new/>

³<https://filezilla-project.org/>

5. Evaluation

5.2.3. Objective Three - Achieved

Create a working custom router that can be used to simulate degradation in conjunction with the program outlined in Objective 2 (See Section 2.3)

The project has met the requirements of this objective. The router can be used in place of a commercial router and can handle around 3 clients at a single time while still providing acceptable transfer speeds. A client can connect to the router by wireless or wired means.

5.3. Project Management

The implementation of the degradation tool required planning and structure to ease the process of developing the software. This project used GitHub ⁴ and its various tools available to provide structure to the project.

5.3.1. Issues

Issues was one of the main ways problems and enhancements were recorded and tracked, comments were placed on there with information regarding how to fix the issue and what the issue requires for completion. Appendix E contains a snapshot of an actual issue that had been closed over the development of the project in Figure E.1. An example of what was used for a description of the issue is displayed by Figure E.2.

This provided a nice overview of what needed completing or improving.

5.3.2. Commits

Commit messages were also carefully structured with informative messages that allow the progress of the project to be easily viewed. Figure 5.9 contains an example of commits used in the real project.

⁴<https://github.com/>

5. Evaluation



Figure 5.9.: Sample of some commit messages used in the GitHub project

5.4. Further work

The project could be continued and improved in the ways discussed below.

5.4.1. More protocols

In the background there was discussion into the inclusion of other transport protocols or other TCP and UDP based protocols. This would give this project a much more in-depth analysis of more types of network traffic. QUIC would be a great addition to future goals of the project. QUIC is an incredibly new protocol and with the backing of Google gives it a higher chance of being adopted as a standard transfer protocol.

Testing and experiments could be performed to detect the effects of degradation such as latency and packet loss of this young protocol and could even contribute to the infant knowledge base.

5.4.2. More platform support

The tool is currently only supported on Linux. Therefore, further work could be to create version that run on Windows or Mac OS. This would allow the tool to be used for many more situations and on many more systems. This however would involve a large amount of work due to the differing ways that each OS would deal with firewall based packet filtering.

5. *Evaluation*

5.4.3. **Larger test network**

Another further improvement would be to use a more powerful router to host the script. Increased speed would allow support for a larger synthetic test network with more clients. Therefore, would provide more possible feedback because of the further improved realism and size of the test network. This improvement is mainly limited by the processing power of the computer used to host the script.

This also touches on the possible improvement of the efficiency of the script, the use of a more embedded orientated language would provide a sizeable increase. A language similar to C or C++ would be suitable, this however, would be time consuming and library support would not be as comprehensive as Python's. Therefore, resulting in more time required to port the tool.

6. Conclusion

The project, as mentioned above has succeeded in achieving its goals. It was able to provide a test network where various forms of degradation were performed on the network using a script hosted on a small Raspberry Pi router. The script is supported by a Linux OS and performs most of its functionality utilising a section in the Linux Kernel known as the NFQUEUE. Although further improvements were discussed above the project was fully functional and the tool can be used to simulate degradation very effectively.

A. Test Plan

| Section | Test Code | Test Name | Description | Expected |
|-------------------|-----------|-----------------------|--|-------------------------------------|
| UDP Client | | | | |
| Logic | | | | |
| | UC-L1 | TestStartUp() | Creates a client object before each test | n/a |
| | UC-L2 | TestCleanUp() | No clean-up needed | n/a |
| | UC-L3 | Connect-Disconnect() | The client connects performs no action then disconnects | n/a |
| | UC-L4 | CreateClientObj() | A client object is created | n/a |
| | UC-L5 | SendPacket() | Sends a single UDP packet | n/a |
| | UC-L6 | CheckGridSend() | Checks the method sends out the correct number this depends on grid size | n/a |
| UI | | | | |
| | UC-U1 | TestStartUp() | Loads up a fresh window from the command line | n/a |
| | UC-U2 | TestCleanUp() | Clicks the X in the top right corner | n/a |
| | UC-U3 | ConnectClick() | Clicks the "Connect" button | n/a |
| | UC-U4 | RestartClick() | Clicks the "Connect" button then the "Restart" button | n/a |
| | UC-U5 | Connect-InvertCheck() | Clicks the "Connect" button and checks all the buttons invert their "Enabled" property correctly | Connect(Disabled), Restart(Enabled) |
| | UC-U6 | Restart-InvertCheck() | Clicks the "Connect" button, then the "Reset" button and checks the "Enabled" properties for all buttons are correct | Connect(Enabled), Restart(Disabled) |
| | UC-U7 | TextEntry() | Enters text into the text box and then "Connect" is clicked | n/a |
| UDP Server | | | | |
| Logic | | | | |

A. Test Plan

| | | | | |
|----|--------|-----------------------------|--|-----------------|
| | US-L1 | TestStartUp() | Creates the client object | n/a |
| | US-L2 | TestCleanUp() | n/a | n/a |
| | US-L3 | CreateObject() | Tests that the server object can be created successfully | n/a |
| | US-L4 | WaitForTimeout() | Starts the server and checks it closes cleanly after time-out | n/a |
| UI | | | | |
| | US-U1 | TestStartUp() | Loads up a fresh window from the CMD line | n/a |
| | US-U2 | TestCleanUp() | Clicks the X in the top right hand corner | n/a |
| | US-U3 | StartClick() | Click the "Start" button | n/a |
| | US-U4 | RandomiseClick() | Click the "Randomise" button | n/a |
| | US-U5 | Start-ResetClick() | Clicks the "Start" button then resets and checks it returns to it's default state | n/a |
| | US-U6 | Randomise-ResetClick() | Clicks the "Randomise" button, the resets and checks it returns to it's default state | n/a |
| | US-U7 | Start-CheckInvert() | "Start" button clicked and then the test checks for "Enabled" property is inverted | n/a |
| | US-U8 | RestartCheckInvert() | "Start" then "Restart" button clicked and the test then checks for the correct inverting of the buttons "Enabled" property Start = Enabled, Restart = Disabled | n/a |
| | US-U9 | Stat-PacketLossReset() | Checks when the restart button is clicked the statistics is returned to default | "-" |
| | US-U10 | Stat-TotalPacketLossReset() | Checks when the restart button is clicked the statistics is returned to default | "-" |
| | US-U11 | Stat-TotalPacketSentReset() | Checks when the restart button is clicked the statistics is returned to default | "-" |
| | US-U12 | Stat-CheckDeafult() | "Start" is clicked and the test then waits for a time-out, it then checks the default values of the statistics | PacketLoss(100) |

A. Test Plan

| | | | | |
|------------|-----|------------------------------|---|-----|
| Live Tests | | | | |
| | U-1 | TestStartUp() | Both windows were opened on the command line | n/a |
| | U-2 | TestCleanUp() | Both windows are closed by automated actions | n/a |
| | U-3 | SendAndRecieve-Valid() | The client sends packages to the server and the server accepts packets | n/a |
| | U-4 | SendAndRecieve-Valid-Twice() | The performs the same action at the above test but does the whole loop an extra time to check if the reset works correctly | n/a |
| | U-5 | SendAndRecieve-Invalid() | This test doesn't connect the client to the server but a random IP on the network, this is to check the server will act accordingly | n/a |

Packet Script Test Plan for the default effects.

| Section | Test No | Test Name | Description | Expected |
|----------------|---------|--------------------|---|---|
| Effects | | | | |
| PacketLoss | | | | |
| | P-1 | StartPacketLoss() | Starts the script with the expected parameter (e.g. -pl 10) | Script should start PacketLoss mode |
| | P-1 | PacketLossEffect() | Pings the script and checks the effect on the ping packets, the test pings until one packet is lost. Checking for exact percentages isn't reliable enough | Some packet loss |
| Latency | | | | |
| | L-1 | StartLatency() | Starts the script with valid parameters for latency (e.g. -l 10) | Script should start in Latency mode |
| | L-2 | LatencyEffect() | Pings the script and checks the latency value within a margin of error | Latency effect on ping packets |
| Bandwidth | | | | |
| | B-1 | StartBandwidth() | Starts the script with the expected parameters (e.g. -rl 100) | Script should start in Bandwidth limit mode |

A. Test Plan

| | | | | |
|--------------|-----|------------------------|---|--------------------|
| | B-2 | BandwidthEffect() | Starts bandwidth mode and starts pinging the localhost, after a certain amount of packets the rate is calculated and checked against the rate limit value | n/a |
| Out-Of-Order | | | | |
| | O-1 | StartOrder() | Starts the script and checks the effect initialises correctly | n/a |
| | O-2 | OutOfOrderEffect() | Starts the effect and checks using the localhost that the packet sequence numbers are not in sequential order when they return | n/a |
| Jitter | | | | |
| | J-1 | StartJitter() | Starts the script in the mode and checks all initialisation finishes correctly | n/a |
| | J-2 | JitterEffect() | Ping packets are sent on the localhost and a difference in packet latency is looked for | n/a |
| Validation | | | | |
| | V-1 | LatencyValidation() | Check that the validation works for the latency mode | Range = 1-1000ms |
| | V-2 | PacketLossValidation() | Check that the validation works for the packet loss mode | Range = 1-100% |
| | V-3 | BandwidthValidation() | Checks that validation works for bandwidth mode | Range = 1-10000B/s |
| | V-4 | JitterValidation | Checks that validation works for difference values | Range = 10ms-100ms |

B. HTTP Testing

| Effect Value (m/s) | Page Load Time (Seconds) |
|--------------------|--------------------------|
| 50 | 0.952 |
| 100 | 1.670 |
| 150 | 2.434 |
| 200 | 3.243 |
| 250 | 3.910 |
| 300 | 4.648 |
| 350 | 5.554 |
| 400 | 6.328 |
| 450 | 7.278 |
| 500 | 7.886 |
| 550 | 9.285 |
| 600 | 10.088 |
| 650 | 10.933 |
| 700 | 11.768 |
| 750 | 12.551 |
| 800 | 12.436 |
| 850 | 15.239 |
| 900 | 16.279 |
| 950 | 20.743 |
| 1000 | 19.162 |

Figure B.1.: Page loads times of https://en.wikipedia.org/wiki/University_of_Hull when latency is active

C. FTP Testing

| Description | Download Time (Minutes:Seconds) |
|-------------|---------------------------------|
| No script | 0:15 |
| 0m/s | 0:17 |
| 10 m/s | 0:34 |
| 50 m/s | 0:41 |
| 100 m/s | 1:29 |
| 150 m/s | 2:02 |
| 200 m/s | 2:46 |

Figure C.1.: Time to download a 5MB file when latency is being simulated by the tool

| Description | Download Time (Minutes:Seconds) |
|-------------|---------------------------------|
| 1% | 0:35 |
| 5% | 0:38 |
| 10% | 1:02 |
| 15% | 1:45 |
| 20% | 2:38 |
| 25% | 4:34 |

Figure C.2.: Time to download a 5MB file when packet loss is being simulated by the tool

D. UDP Testing

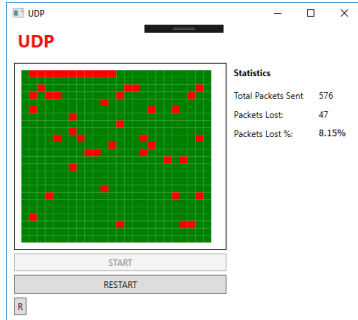


Figure D.1.: 5% packet loss

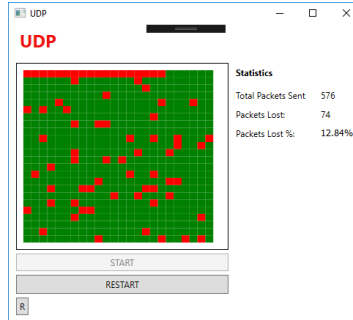


Figure D.2.: 10% packet loss

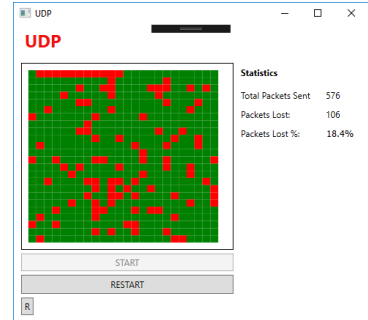


Figure D.3.: 15% packet loss

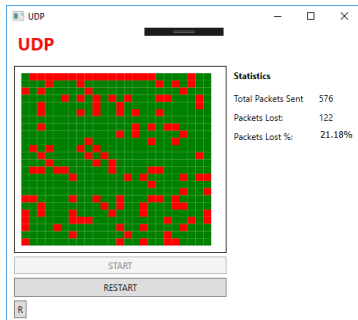


Figure D.4.: 20% packet loss

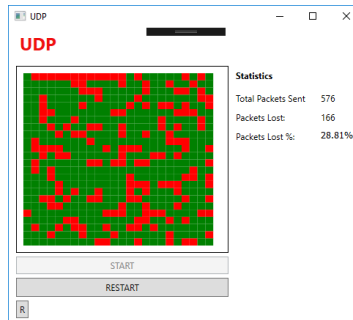


Figure D.5.: 25% packet loss

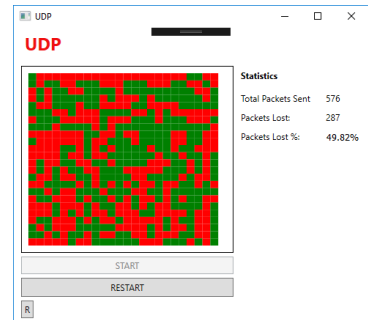


Figure D.6.: 50% packet loss

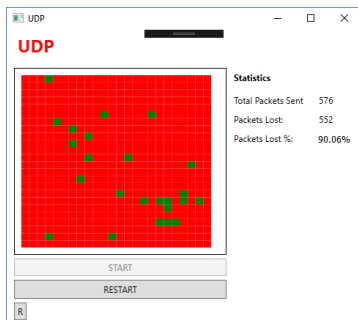


Figure D.7.: 95% packet loss

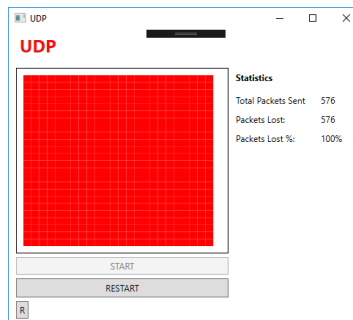



Figure D.8.: 100% packet loss


E. GitHub Issues

| | | | | | | | | |
|--------------------------|----------|---|-------------|--|------------|--------------|------------|--------|
| <input type="checkbox"/> | 🔔 2 Open | ✓ 63 Closed | Author ▾ | Labels ▾ | Projects ▾ | Milestones ▾ | Assignee ▾ | Sort ▾ |
| <input type="checkbox"/> | 🔔 | PacketCapture - Could add a new attack that changes all the packets window sizes to 0 | enhancement | #64 by AidanFray was closed 19 days ago | | | | |
| <input type="checkbox"/> | 🔔 | Tests - Download speed tests are not closing threads correctly when an exception is raised | bug | #62 by AidanFray was closed 25 days ago | | | | |
| <input type="checkbox"/> | 🔔 | Reading - Intrtesting TCP related info | | #61 by AidanFray was closed 19 days ago | | | | |
| <input type="checkbox"/> | 🔔 | Experimentation - Need to run all tests to collate data into a graph | report | #60 by AidanFray was closed 19 days ago 📊 7 of 7 | | | | |
| <input type="checkbox"/> | 🔔 | PacketCapture - Retransmission values for FTP transfer test seems too high | bug | #59 by AidanFray was closed on Feb 8 | | | | |
| <input type="checkbox"/> | 🔔 | Demonstration - Entire demo needs to be tested and run through | report | #58 by AidanFray was closed on Feb 14 | | | | |
| <input type="checkbox"/> | 🔔 | Experimentation - List of possible experiments and what their expected results could be | report | #57 by AidanFray was closed 19 days ago 📊 3 of 3 | | | | |
| <input type="checkbox"/> | 🔔 | PacketCapture - GUI - Clear the terminal window when the packet script is stopped | enhancement | #56 by AidanFray was closed 25 days ago | | | | |
| <input type="checkbox"/> | 🔔 | PacketCapture - Connection simulation isn't representing values as accurately as is needed | bug | #55 by AidanFray was closed on Jan 31 | | | | |

Figure E.1.: Snapshot of actual issues created in the project



PacketCapture - ARPSpoofing.py and Packet.py do not work together #14


 Closed AidanFray opened this issue on Nov 20, 2017 · 1 comment




AidanFray commented on Nov 20, 2017

This could possibly be due to conflicting iptables rules

 AidanFray added the  label on Nov 20, 2017

 AidanFray self-assigned this on Nov 20, 2017



AidanFray commented on Nov 20, 2017

The command below solved the issue:

```
iptables -A FORWARD -j NFQUEUE
```


 AidanFray closed this on Nov 20, 2017

Figure E.2.: Example of an issue with discussion

F. Testing Correctness

| | | |
|--------------|-----------------------------|------|
| UDP Client | | |
| Logic | | |
| UC-L3 | Connect-Disconnect() | Pass |
| UC-L4 | CreateClientObj() | Pass |
| UC-L5 | SendPacket() | Pass |
| UC-L6 | CheckGridSend() | Pass |
| UI | | |
| UC-U3 | ConnectClick() | Pass |
| UC-U4 | RestartClick() | Pass |
| UC-U5 | Connect-InvertCheck() | Pass |
| UC-U6 | Restart-InvertCheck() | Pass |
| UC-U7 | TextEntry() | Pass |
| UDP Server | | |
| Logic | | |
| US-L3 | CreateObject() | Pass |
| US-L4 | WaitForTimeout() | Pass |
| UI | | |
| US-U3 | StartClick() | Pass |
| US-U4 | RandomiseClick() | Pass |
| US-U5 | Start-ResetClick() | Pass |
| US-U6 | Randomise-ResetClick() | Pass |
| US-U7 | Start-CheckInvert() | Pass |
| US-U8 | RestartCheckInvert() | Pass |
| US-U9 | Stat-PacketLossReset() | Pass |
| US-U10 | Stat-TotalPacketLossReset() | Pass |
| US-U11 | Stat-TotalPacketSentReset() | Pass |
| US-U12 | Stat-CheckDeafult() | Pass |
| UDP Combined | | |
| Live | | |
| U-3 | SendAndRecieve-Valid() | Pass |
| U-4 | SendAndRecive-Valid-Twice() | Pass |
| U-5 | SendAndRecieve-Invalid() | Pass |

Figure F.1.: Testing correctness for the UDP Server and Client

F. Testing Correctness

| | | |
|--------------|------------------------|------|
| PacketLoss | | |
| P-1 | StartPacketLoss() | Pass |
| P-2 | PacketLossEffect() | Pass |
| Latency | | |
| L-1 | StartLatency() | Pass |
| L-2 | LatencyEffect() | Pass |
| Bandwidth | | |
| B-1 | StartBandwidth() | Pass |
| B-2 | BandwidthEffect() | Pass |
| Out-Of-Order | | |
| O-1 | StartOrder() | Pass |
| O-2 | OutOfOrderEffect() | Pass |
| Jitter | | |
| J-1 | StartJitter() | Pass |
| J-2 | JitterEffect() | Pass |
| Validation | | |
| V-1 | LatencyValidation() | Pass |
| V-2 | PacketLossValidation() | Pass |
| V-3 | BandwidthValidation() | Pass |
| V-4 | OutOfOrderValidation() | N/A |
| V-5 | JitterValidation() | Pass |

Figure F.2.: Testing correctness for the main script functions

Bibliography

- Allman, M., Paxson, V. & Blanton, E. (2009) Tcp congestion control. *Technical report*.
- Beck, K. (2003) *Test-driven development: by example*. Addison-Wesley Professional.
- Belshe, M. & Peon, R. (2012) Spdy protocol.
- Blog, C. (2015) Hello http/2, goodbye spdy. *Haettu*, 26, 2016.
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999) Hypertext transfer protocol-http/1.1. *Technical report*.
- Galbraith, J., Saarenmaa, O., Ylonen, T. & Lehtinen, S. (2006) Ssh file transfer protocol (sftp). *Technical report*, Internet Draft. <http://www.ietf.org/internet-drafts-ietf-secsh-fliexfer-12.txt>.
- Gates, M., Tirumala, A., Ferguson, J., Dugan, J., Qin, F., Gibbs, K. & Estabrook, J. (2003) Iperf. Website: <http://dast.nlanr.net/Projects/Iperf> (August 2006).
- Hamilton, R., Iyengar, J., Swett, I. & Wilk, A. (2016) Quic: A udp-based secure and reliable transport for http/2. *IETF, draft-tsvwg-quic-protocol-02*.
- Henderson, T., Floyd, S., Gurtov, A. & Nishida, Y. (2012) The newreno modification to tcp's fast recovery algorithm. *Technical report*.
- Huston, G. (2006) Gigabit tcp. In *IPJ*, Citeseer.
- Kempf, J., Arkko, J., Beheshti, N. & Yedavalli, K. (2011) Thoughts on reliability in the internet of things. In *Interconnecting smart objects with the Internet workshop*, volume 1, 1–4.
- Murdock, I. (1994) Overview of the debian gnu/linux system. *Linux Journal*, 1994(6es), 15.
- Ofcom, U. (2015) 4g and 3g network performance november 2015: Measuring mobile broadband performance in the uk.
- Ofcom, U. (2017) Broadband performance april 2017: The performance of fixed-line broadband delivered to uk residential consumers.
- Paxson, V., Allman, M., Chu, J. & Sargent, M. (2011) Computing tcp's retransmission timer. *Technical report*.

Bibliography

- Pi, R. (2014) Raspbian. *Dosegljivo*: <https://www.raspberrypi.org/downloads/raspbian/>. [Dostopano: 6. 12. 2015].
- Postel, J. (1980) Rfc 768: User datagram protocol. *Technical report*.
- Postel, J. & Reynolds, J. (1985) Ietf rfc 959 file transfer protocol: Ftp.
- Postel, J. et al. (1981) Rfc 793: Transmission control protocol. *Technical report*.
- Pranskevichus, E. & Selivanov, Y. () Python documentation. what's new in python 3.6.
- Roskind, J. (2013) Quic: Multiplexed stream transport over udp. *Google working design document*.
- Share, N. M. (2015) Desktop operating system market share. *Fonte* <http://www.netmarket-share.com/operating-systemmarket-share.aspx>.
- Upton, E. & Halfacree, G. (2014) *Raspberry Pi user guide*. John Wiley & Sons.
- Whalen, S. (2001) An introduction to arp spoofing. *Node99 [Online Document]*, April.
- Xiaoming, L., Sejdini, V. & Chowdhury, H. (2010) Denial of service (dos) attack with udp flood. *School of Computer Science, University of Windsor, Canada*.