

MODERNISATION AND EXTENSION OF INETVIS: A NETWORK SECURITY DATA VISUALISATION TOOL

Submitted in partial fulfilment
of the requirements for the degree of

MASTER OF SCIENCE

of Rhodes University

By YESTIN JOHNSON

Grahamstown, South Africa

October 2018

Abstract

This research undertook an investigation in digital archaeology, modernisation, and revitalisation of the InetVis software application, developed at Rhodes University in 2007. InetVis allows users to visualise network traffic in an interactive 3D scatter plot. This software is based on the idea of the *Spinning Cube of Potential Doom*, introduced by Stephen Lau. The original InetVis research project aimed to extend this concept and implementation, specifically for use in analysing network telescope traffic.

The InetVis source code was examined and ported to run on modern operating systems. The porting process involved updating the UI framework, Qt, from version 3 to 5, as well as adding support for 64-bit compilation. This research extended its usefulness with the implementation of new, high-value, features and improvements.

The most notable new features include the addition of a general settings framework, improved screenshot generation, automated visualisation modes, new keyboard shortcuts, and support for building and running InetVis on macOS.

Additional features and improvements were identified for future work. These consist of support for a plug-in architecture and an extended heads-up display. A user survey was then conducted, determining that respondents found InetVis to be easy to use and useful. The user survey also allowed the identification of new and proposed features that the respondents found to be most useful. At this point, no other tool offers the simplicity and user-friendliness of InetVis when it comes to the analysis of network packet captures, especially those from network telescopes.

Keywords: Security Visualisation, Network Security, Data Visualisation

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System¹ (2012 version, valid through 2017):

- **Software and its engineering Integrated and visual development environments**
- *Human-centered computing Visualization systems and tools*
- Networks Protocol testing and verification

¹<http://www.acm.org/about/class/2012/>

Acknowledgements

I would like to start off by thanking the three companies for which I have worked over the last two years for their financial support, their flexibility, and for the study leave. These are: **Impact Radius**², **VOSS Solutions**³, and **SecureData Europe**⁴/**SensePost**⁵.

I would also like to thank the following:

Barry Irwin: My supervisor, for all of his support and patience over the past year while I worked on this research. His encouragement helped a great deal during the hard times when completion of this research seemed out of reach. He kept me motivated throughout the year, providing countless ideas, draft reviews, and general encouragement. Barry, I could not have done this without your support. Thank you!

Ronel Johnson: My loving wife. We were engaged when I started this degree, and I am glad to say that things went well, and we got married a short time before I completed this work. I could not have done this without your patience, understanding, and encouragement. It has been a really hard two years, and I hope to make up for all the time we could not spend together this year. Thank you for putting up with me at my worst and for having faith in me.

Survey Respondents: Thank you to all participants who participated in the user survey as part of this research. Your input was invaluable.

Liam Smit: For all the time and effort that you put into testing InetVis and finding and reporting bugs.

Ross Simpson: For your time, patience, and interest in helping test and improve InetVis.

Dane Goodwin: For your enthusiastic usage of InetVis and your well thought out and useful ideas.

Damjan Jovanovic: For your time and effort in getting the initial port up and running on your FreeBSD system.

Friends and Family: To everyone else who I have not mentioned, but who played a part in the last two years. I really appreciate all of your patience and support.

²<https://www.impactradius.com/>

³<https://www.voss-solutions.com/>

⁴<https://www.secddata.com/>

⁵<https://sensepost.com/>

Contents

List of Figures	viii
List of Tables	ix
List of Source Code Listings	x
1 Introduction	1
1.1 Background	1
1.2 Problem Statement	3
1.3 Research Goals	3
1.4 Scope	3
1.5 Document Conventions	5
1.6 Document Structure	5
2 Literature Review	7
2.1 Introduction	7
2.2 Network Security	8

2.2.1	Network Scans	8
2.2.2	Packet Captures	11
2.2.3	Network Telescopes	13
2.3	Data Visualisation	15
2.4	Network Security Data Visualisation	18
2.5	Related Tools	20
2.5.1	The GPL Cube of Potential Doom	20
2.5.2	IDS RainStorm	21
2.5.3	RUMINT	22
2.5.4	SIFT	24
2.5.5	AfterGlow	25
2.5.6	PortVis	26
2.5.7	Cyberspace Visualisation Tool	28
2.5.8	TNV	29
2.5.9	VisualFirewall	30
2.5.10	Mapper	31
2.5.11	FlowTag	32
2.5.12	P3D	34
2.5.13	DAVIX	35
2.6	Related Tool Discussion	36
2.7	Summary	38

3	Design and Implementation	40
3.1	Introduction	40
3.2	The Original Version	41
3.2.1	Architecture	41
3.2.2	Application UI	42
3.2.3	Source Code Structure	48
3.2.4	The DataProcessor Class	50
3.2.5	The GLVisWidget Class	50
3.2.6	Logging Framework	51
3.2.7	Original Updates	51
3.2.8	Summary	53
3.3	The Port	54
3.3.1	Methodology	56
3.3.2	Laying the Foundation	57
3.3.3	Target Environment	59
3.3.4	Version Control	59
3.3.5	Qt 3 to Qt 4	60
3.3.6	Qt 4 to Qt 5	66
3.4	Architectural Changes	68
3.5	Release and Peer Review	68
3.6	Challenges	70
3.7	Caveats	73
3.8	Summary	74

4	New Features and Improvements	76
4.1	Introduction	76
4.2	New Features	77
4.3	General Settings Framework	78
4.3.1	Design	79
4.3.2	GUI Component	79
4.3.3	Settings Persistence	84
4.3.4	Discussion	88
4.4	Optimised Screenshot Format	89
4.5	Specify Local Network Interface	91
4.6	Harlem Shake Mode	92
4.7	Visualisation Pane Autorotation	93
4.8	New Keyboard Shortcuts	93
4.9	Improvements and Bug Fixes	95
4.9.1	Application UI and Position/Size Persistence	96
4.9.2	Global Standard Out and Standard Error Log Files	96
4.9.3	MacOS Support	98
4.9.4	Support for HighDPI Displays	99
4.9.5	Opening PCAP Files	100
4.9.6	Improvements in Setting Home Network Range	100
4.9.7	About Dialog Modularity	101
4.9.8	Documentation Improvements	102
4.9.9	Source Code and Spelling Consistency	102
4.10	Summary	103

5	Unimplemented Features and Survey Discussion	105
5.1	Introduction	105
5.2	Unimplemented Features and Improvements	106
5.2.1	Plug-in/Module Functionality	106
5.2.2	Heads-up Display Implementation	107
5.2.3	Mouse-Over Support for Network Events	108
5.2.4	Application UI Modernisation	109
5.2.5	Font Size Adjustments	110
5.2.6	Native Video Output	110
5.2.7	Support for Time-Series Compressed Files	111
5.2.8	Event colouring by Netblock/AS/BPF Filter	112
5.2.9	Y-axis Source/Destination Port Toggle	113
5.2.10	Rewind/Play Backwards Option	113
5.2.11	Native Compressed File Support	114
5.2.12	Setting to toggle LogUI functionality	114
5.2.13	Improved Handling of Malformed PCAP Files	115
5.2.14	Re-Enable Recording of PCAPS for Monitoring Local Host	115
5.2.15	Joystick/Gamepad Support	116
5.3	User Survey	116
5.3.1	Survey Questions	117
5.3.2	Survey Responses and Discussion	118
5.4	Recommendations	121
5.5	Summary	122

6	Conclusion	124
6.1	Introduction	124
6.2	Research Summary	124
6.3	Research Evaluation	126
6.3.1	Modernise InetVis to Run on Modern Operating Systems	126
6.3.2	Explore Additional Features and Implement Improvements to InetVis	126
6.4	Significance of Research	127
6.5	Future Work	128
	Appendices	138
A	Code Listings	139
A.1	prepare_release.sh	139
A.2	install_inetvis.sh	140
A.3	uninstall_inetvis.sh	142
B	InetVis Screenshots	144
C	InetVis User Survey	146
D	InetVis Videos	156

List of Figures

2.1	InetVis showing a network scan and a port scan.	9
2.2	Example of a <i>Shodan.io</i> search for port 22 in South Africa.	11
2.3	The <i>libpcap</i> file format (Harris, 2015).	12
2.4	Sparklines tracking the price of foreign exchange (Tufte, 2006).	18
2.5	The GPL Cube of Potential Doom (Dragorn@kismetwireless.net, 2011). . .	21
2.6	The main IDS RainStorm display window (Abdullah <i>et al.</i> , 2005).	23
2.7	The IDS RainStorm zoom window (Abdullah <i>et al.</i> , 2005).	23
2.8	The galaxy, small, and machine view of NVisionIP (Yurcik, 2005).	25
2.9	The main VisFlowConnect-IP display (Yurcik, 2005).	26
2.10	The Afterglow display window (Marty, 2013).	27
2.11	The Portvis overview display (Marty, 2013).	28
2.12	The Cyberspace Visualisation Tool display (Bass <i>et al.</i> , 2017).	29
2.13	The TNV display (Goodall, 2009).	30
2.14	The Visual Firewall Visual Signature View (Lee <i>et al.</i> , 2005).	32
2.15	The Mapper display of scan detection (Coudriau <i>et al.</i> , 2016).	33

2.16	The FlowTag display (Lee, 2013).	34
2.17	The P3D display (Nunnally <i>et al.</i> , 2013).	35
3.1	Overview of the original InetVis UI hierarchy.	43
3.2	The InetVis visualisation pane.	44
3.3	The InetVis control panel.	45
3.4	The InetVis plotter settings dialog.	46
3.5	The InetVis reference frame settings dialog.	47
3.6	The InetVis documentation dialog.	47
3.7	The InetVis about dialog.	48
3.8	DataProcessor classes and logic flow.	50
3.9	Feature branch workflow.	60
4.1	The updated InetVis General Settings dialog.	81
4.2	Qt Creator, Signals and Slots, Configure Connection.	83
4.3	Qt Creator, Signals and Slots Editor Dialog.	83
4.4	Qt Creator, Signals and Slots, Creating a Connection.	83
4.5	Updated Plotter Settings dialog with Load button.	101
B.1	The updated InetVis About dialog.	145

List of Tables

2.1	Comparison of GPL Cube of Potential Doom and InetVis.	21
2.2	Visualisation tools available in DAVIX.	36
3.1	Overview of source code structure.	48
3.2	Overview of UI file structure.	49
3.3	Comparison of the major points between Qt 3, Qt 4, and Qt 5.	55
3.4	Original version Ubuntu package dependencies.	58
3.5	Signal changes between Qt 3 and 4.	65
3.6	Updated function names from Qt 3 to 4.	65
3.7	Lingering Qt 3 support class includes and their solutions.	65
3.8	InetVis software releases.	70
4.1	Cube autorotation keyboard shortcuts.	93
5.1	Rank table of most requested features by survey participants.	120
D.1	Overview of source code structure.	156

List of Source Code Listings

3.1	<i>DataProcessor::renderDataDynamic()</i> event plotting logic.	43
3.2	Exporting QT environmental variable.	58
3.3	Updates to <i>QString</i> usage in <i>sprintf()</i> method calls.	64
4.1	Header definition of new slots (generalsettingsdialog.h).	84
4.2	Source code definition of new slots (generalsettingsdialog.cpp).	84
4.3	Definition of a settings key (dataproc.h).	86
4.4	Standard method header file definitions (dataproc.h).	86
4.5	Standard method source file implementation (dataproc.cpp).	86
4.6	Use of settings in source code (dataproc.cpp).	86
4.7	Implementation logic of UI methods (generalsettingsdialog.cpp).	87
4.8	Default value definition (dataproc.h).	87
4.9	Default value initialisation (main.cpp).	87
4.10	Current implementation of screenshots (glviswidget.cpp).	90
4.11	Snippet of use of public Log API (glviswidget.cpp).	97
4.12	macOS Build Support (dataproc.h).	98
4.13	Support for HighDPI monitors (glviswidget.cpp).	99
5.1	Command line to generate video output from PNG.	111
A.1	Shell script to prepare a release.	140
A.2	Shell script to install InetVis.	142
A.3	Shell script to uninstall InetVis.	143

CHAPTER 1

Introduction

1.1 Background

The area of network security analysis has long been a complex one, with new threats emerging on the Internet every day (Lau, 2003). Traditionally, most of this analysis was done by network administrators. This work relied on network packet captures, firewall, and intrusion detection logs. The analysts would review a large amount of information in near real-time in order to determine the existence of malicious activity. With the ever increasing dimensions of Internet penetration, link speed, and use of the Internet for daily activities, the job performed by security analysts has become increasingly difficult. This trend is only set to accelerate in the future (Zhuang *et al.*, 2013).

Enter the use of information and data visualisation tools and techniques in the area of computer and network security. For more than a decade researchers, system administrators, and analysts have sought out solutions to this problem. Researchers have combined the areas of data visualisation and network security and created the combined area of security visualisation. This new area aims at improving the effectiveness of network security analysis by providing visualisation tools optimised to take advantage of decades of information and data visualisation research. These tools aim to take advantage of the visual

processing and pattern recognition capabilities of humans in order to augment traditional text-based network security analysis methods.

Visualisation tools have been developed that focus on various individual areas within the realm of network security. Certain tools focus on the effective analysis of *NetFlow* data, while others focus on the analysis of packet captures and the monitoring and visualisation of live network interfaces. Other tools specifically focus on the analysis of data from services such as firewalls and intrusion detection systems. Yet other tools offer a more generic approach of visualising network event data from packet captures.

This research is based on the *Internet Visualisation* (InetVis) tool, and its related research conducted at Rhodes University between 2005 and 2007 by van Riel and Irwin (van Riel, 2005; Irwin and van Riel, 2008). This tool was originally designed and implemented based on another tool - the *Spinning Cube of Potential Doom* - introduced by Stephen Lau (2003) a few years earlier. *Lau's Cube* was created in order to visualise the threats present on the Internet every day, for people at a conference, who were not necessarily in the computer science field. This tool aimed to raise awareness in the general public of the constant and ongoing network scans being performed by malicious attackers and automated tools on the Internet.

The creators of InetVis took Lau's concept and extended it, implementing new features and improvements such as support for *pcap*-style packet captures rather than IDS logs, and support for the UDP and ICMP protocols. These features specifically aimed at making use of InetVis in the area of network telescope traffic analysis. Network telescopes are blocks of IP address space which do not provide any services or actively communicate out to the Internet (Irwin, 2011). Any traffic arriving at a network telescope is unsolicited. Network telescope traffic is often captured by researchers and analysed at a later date. It is not uncommon for months, or years, worth of traffic to be analysed at once (Irwin, 2011). The use of visual analysis tools such as InetVis is highly useful in this area as they allow for more expedient and effective analysis of large network traffic packet captures.

InetVis offers a simple and effective UI and visualisation which are easily understandable by most users. The tool fills a niche in the market for a simple security visualisation tool which happens to be particularly suited for analysing network telescope traffic.

1.2 Problem Statement

The problem of ever increasing data volume for analysis in the area of network security is only getting worse, and new and improved visualisation tools are becoming increasingly important for effective network security analysis. Only so much can be done with textual analysis by automated computer systems, and the human element is needed for our improved visual processing capabilities.

InetVis proved useful to researchers upon its creation, as evidenced by its inclusion in the first release of the *Data Analysis and Visualisation Linux* (DAVIX) (Monsch, 2008). Years later, it continues to prove useful and is still being included in *DAVIX* (Marty, 2015). Researchers are still making active use of InetVis, as can be seen by reviewing the use of the `#inetvis` hashtag on Twitter¹. However, InetVis was last updated at the end of 2007 and since then much has changed in the domains of computer hardware, operating systems, and networks. As it stood at the beginning of this research, this tool would not easily run or compile on modern operating systems. Given the tool's use for network telescope, and other, analysis and for the reasons already mentioned regarding the need for improved security visualisation tools, it is evident that this tool is worthy of further research, analysis, and modernisation in order for it to be useful once more.

1.3 Research Goals

This research aims to further improve the situation within the area of network security analysis by reviving and modernising InetVis. The research question is thus: **How can InetVis be modernised and updated for enhanced use.** The research goals identified in order to answer this question are as follows:

1. Modernise InetVis to run on modern operating systems.
2. Explore additional features and implement improvements to InetVis.

1.4 Scope

Working in the area of security visualisation, and in particular, the software development work required in order to produce a useful tool is a major undertaking. There are many

¹<https://twitter.com/search?q=%23inetvis&src=typd>

aspects to consider and for this reason the scope of this research is limited from the start. Only the most pertinent aspects will be focused on.

The scope of this research is limited to:

1. An overview of related security visualisation tools.
2. Code archaeology of the original InetVis source code in order to understand the tool.
3. Porting InetVis to build and run on modern operating systems.
4. Updating and modernising InetVis as much as possible in the allocated time.
5. The identification of other high-value features for InetVis.
6. Determining the usefulness of InetVis to users.

Items that are specifically out of scope for this research include the full design and implementation of *all* of the features and improvements that were identified during this research. While it is desirable to identify and implement as many useful features as possible, many of the features identified were complex in nature, requiring substantial design and implementation work to be done. Thus, most of the more complex and high-value are out of scope for this research. They are discussed in detail in Chapter 5, in order to provide a good base for future work on this research and InetVis.

Other items which are out of scope include the use of the newly updated version of InetVis for the analysis of network telescope packet captures. While it would have been useful to make use of the updated tool to analyse packet captures, this activity was not undertaken as part of this research as it does not contribute to the core of this research project. Analysis of network packet captures was done in the original research (van Riel and Irwin, 2006a), and none of the core functionality has been modified since the initial analysis.

The user survey conducted in this research as part of the analysis of InetVis is also limited in scope in that it does not aim to perform a full user experience study. It is out of the scope of this research due to the fact that it does not contribute enough to the core research ideas. Given the half-thesis nature of this research there would not be time to design, administer, and implement the feedback from an involved user experience study.

1.5 Document Conventions

In this document certain stylistic and linguistic conventions are followed. These will now be described in order for the reader to better understand this document.

Italics: Use is made of *italicised text* in order to place emphasis on a word or phrase. Italics are always used for source code references in the text body, such as references to a filename or class name. Italics are also used when referencing specific parts of the InetVis UI. Further, italics are also used to highlight the names of related tools and pieces of software.

Bold: Use is made of **bold-faced text** sparingly throughout this document. However, it is used in places where it is necessary to group the text within a paragraph or two to a specific concept. In this case a subsection is not appropriate and thus bold text is used in order to emphasize the topic of the sentence or paragraph following.

Footnotes: Footnotes are used in this document to provide the reader with additional information, such as the URL of a website. Footnotes are also used to provide definitions for certain terms used within the document.

Source code listing environments: Source code listings are used and referred to within the body of the text when a small amount of code is useful in order to describe a certain concept. These listings make use of listing-specific styles and font sizes for better readability. Any larger pieces of code are provided in the appendices or the source code on Github is referenced.

1.6 Document Structure

The remainder of this document is structured as follows:

In **Chapter 2** the literature review is presented, which examines the foundational concepts of data visualisation and network security. A review of related tools is then presented. **Chapter 3** discusses the design, architecture, and implementation of the original version of InetVis. Following this, the work done to port the application to support modern operating systems is presented. **Chapter 4** then discusses the design and implementation of the new features and improvements added to InetVis as part of this research. **Chapter 5** goes on to discuss some of the features that were planned, but not

implemented, as part of this research. This chapter also provides a discussion on the user survey that was performed. Finally, this thesis ends with **Chapter 6** providing a review of the research goals, how they were met, and recommendations for future work.

CHAPTER 2

Literature Review

2.1 Introduction

This chapter presents a review of the background literature and related work required in order to fully appreciate the purpose of this research. The software application forming the focus of this research, InetVis (van Riel, 2007), has its roots in the network security and data visualisation domains. Given this, these two topics will be discussed in detail, with the relevant background provided on all pertinent concepts. Related research and similar tools will then be discussed and compared. A final discussion focusing on the common threads, advantages, and disadvantages of each tool is then presented.

This chapter begins in Section 2.2 with an introduction to the network security concepts that are necessary for the understanding of InetVis. This includes topics such as network scans (Section 2.2.1), packet captures (Section 2.2.2), and concludes with a discussion on network telescopes (Section 2.2.3). Data visualisation is then discussed in Section 2.3, with an emphasis on the data visualisation constructs relevant to this research. Section 2.4 presents a discussion on the intersection of these two areas, and looks at the eminent literature, concepts, and ideas in this hybrid field. A review of a subset of the related research and tools is then presented in Section 2.5. A discussion on the common themes present across all related tools is then presented.

This chapter concludes in Section 2.7 with a brief overview of the content presented here, highlighting the most relevant concepts, tools, and related research.

2.2 Network Security

In this section the most relevant network security concepts that form the foundation of this research are presented. The first topic to be discussed is an introduction and discussion of network scans (Section 2.2.1). Given the nature of the original research and InetVis, the concept of a network scan and the knowledge of common types of network scans is necessary in order to fully understand the usefulness of the application. The understanding of what a packet capture (Section 2.2.2) is, is another important concept in understanding how InetVis functions. Finally, the concept of a network telescope (Section 2.2.3), is presented and described, as it is important in the understanding of how analysis using InetVis works, and why it is useful.

2.2.1 Network Scans

In the domains of computer science and computer networking the term *network scan* refers to a common reconnaissance activity that is undertaken by a network administrator, or nefarious actor, in order to gather information regarding the services and hosts that are active within a certain range of IP address space (Barnett and Irwin, 2008). For example, a network administrator may use this technique in order to detect any unexpected computer systems on their network. On the other hand, a nefarious actor may use this technique in order to determine which computer systems are active and which are vulnerable. In this section the term network scan is used in the high-level sense in order to refer to all types of scans. A few of the most common types of scans will now be discussed.

The most accurate usage of the term network scan is a scan which targets a whole network or range of IP addresses (Mansmann, 2008). In general the scan will specifically probe one, or only a few, ports across the entire network range. This type of scan is most commonly used by nefarious actors in order to quickly and easily identify hosts on a network which are vulnerable to a specific exploit. This is one of the most common types of scans on the Internet. This type of scan is also commonly known as a *horizontal* scan, with the horizontal qualifier referring to the fact that the scan progresses horizontally across all

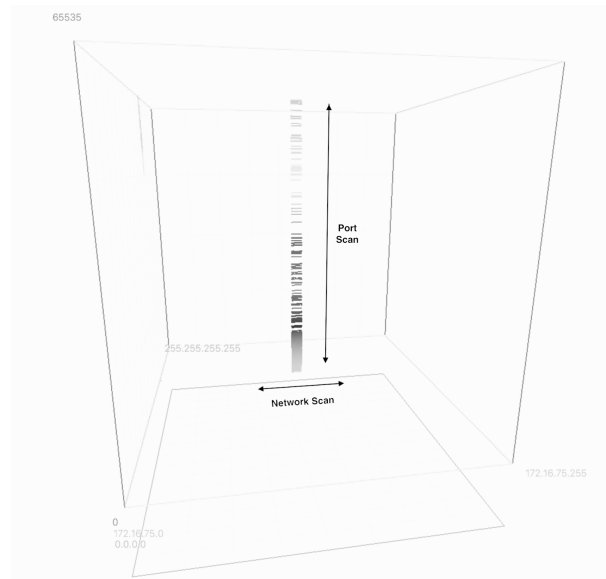


Figure 2.1: InetVis showing a network scan and a port scan.

targets (Barnett and Irwin, 2008). The port remains constant while the target host of the scan varies.

The next most common type of scan is the *port scan* in which the initiator of the scan targets several ports on a single host (Mansmann, 2008). By performing a port scan it is possible for an attacker to determine which services are active on a specific host, and furthermore, to determine the version of the service, the operating system, and various other pieces of information. This type of scan is also referred to as a *vertical scan*, with the vertical quantifier referring to the targeting of all of the *ports* on the host system. The target host of the scan remains constant, while the port varies. A nefarious actor may use this type of scan when targeting a specific host and attempting to find the easiest way in. This type of scan is also used by network administrators, and other legitimate users, in order to validate that only the ports which should be accessible to the Internet actually are. The online service *Shields Up*¹ exemplifies this concept and allows anyone to port scan their own computer network from the Internet.

In Figure 2.1 an example is shown of a combined network and port scan.

While the network scan and port scan are the two most common types of scans, many variations on these scans have been observed. An attempt at developing a taxonomy of network scanning techniques is presented in Barnett and Irwin (2008), with the highlights presented here. Coordinated and distributed scans are variations on the aforementioned

¹<https://www.grc.com/x/ne.dll?bh0bkyd2>

scans which add complexity by performing a scan from multiple distinct hosts and by targeting multiple hosts and ports. These types of scans are more difficult to detect when considering hosts in isolation as the target hosts and destination ports may be randomised across a large set of hosts. These scans may be further improved by making use of additional stealth techniques, such as scanning a host very slowly to evade traditional intrusion detection systems. These scanning techniques may be detected by the use of visual analysis software such as InetVis. By adjusting the various application parameters it is possible to easily detect these types of scans in a short period of time.

There are yet other scanning techniques which are used, i.e. anomalous diagonals and creepy crawly scans, which are detectable by InetVis and which are described in more detail in Barnett and Irwin (2008). Further variations on network scans consist of the specific scanning of TCP-only ports, UDP-ports, and IP protocols such as ICMP. The art of scanning is vast and has evolved much over the years. There are now many tools available in order to perform scans with the most well known being *nmap*². This tool is highly customisable and allows a user to control a large number of aspects regarding a scan, as well as providing additional intelligence on operating system and service detection, and providing support for scripts which further enhance its capabilities. Another commonly used network scanning tool is *masscan*³, which focuses on being the fastest Internet port scanner. It claims to be able to scan the entire Internet in under six minutes. A further contender in this space is *ZMap*⁴ which also aims to be one of the world's fastest single packet network scanners. It claims to be able to scan the entire IPv4 address space in only five minutes.

A popular online service worth mentioning is that of *Shodan*⁵, which allows users to easily analyse the entire Internet in seconds. *Shodan* has servers running which continually scan the entire Internet, the results of which are stored, and then query-able by users of the service. An example of a typical *Shodan* search query and its results can be seen in Figure 2.2.

Nowadays, those in the industry are aware of the fact that the entire Internet is continually being scanned. This fact was emphasised by Lau (2003) with his research and demonstration of the *Spinning Cube of Potential Doom*, aimed at showing normal people the dangers always present while on the Internet. Network scans may come from individual users, with a specific target, or from those scanning the entire Internet manually

²<https://nmap.org/>

³<https://github.com/robertdavidgraham/masscan>

⁴<https://zmap.io/>

⁵<https://www.shodan.io/>

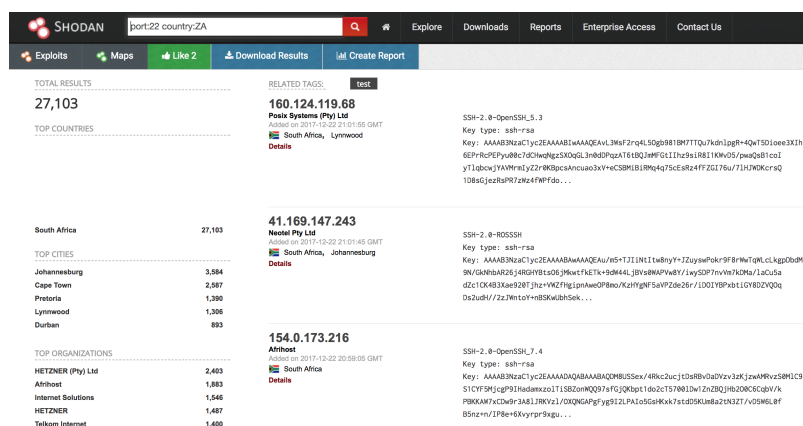


Figure 2.2: Example of a *Shodan.io* search for port 22 in South Africa.

looking for easily exploitable systems (low-hanging fruit). It is also commonplace for hosts infected with malware, such as *Conficker*⁶ or *Mirai*⁷, to continually scan the Internet in order to propagate and infect additional hosts. When analysing Internet scans it is often difficult to tell which scans are manual and which originate from malware. However, this sort of classification is possible, as shown in research by Irwin (2011).

2.2.2 Packet Captures

In the area of network administration it is often necessary to be able to view the packets that are being transmitted from a host or from an application. This is commonly used as a debugging mechanism in order to diagnose network connectivity issues. It is also used when developing a new application and a need arises to analyse the data received from a third-party service. A *packet capture* is used in this instance and is defined in two ways. Firstly, a packet capture is the act of intercepting data, typically network packets, transmitted through a network or a specific host. Secondly, a *packet capture* in the noun form, refers to the file, or files, on disk where packets are actually stored after they are intercepted and captured (Marty, 2009, p. 27-30). For example, a user would capture packets on a network and those packets would be stored in a packet capture file for further analysis.

There are various different types of packet capture formats, and tools, used in order to capture and analyse these files. These are discussed briefly in this section with more information available in Irwin (2011, p. 55-57) and Mansmann (2008, p. 17-18). The term

⁶<http://malware.wikia.com/wiki/Conficker>

⁷<http://malware.wikia.com/wiki/Mirai>

Global Header	Packet Header	Packet Data	Packet Header	Packet Data	Packet Header	Packet Data	...
---------------	---------------	-------------	---------------	-------------	---------------	-------------	-----

Figure 2.3: The *libpcap* file format (Harris, 2015).

*pcap*⁸ is often used as shorthand for packet capture and is also typically the file extension used for packet capture files.

One of the most common file formats that is used for packet capture files is *libpcap*. This is the standard format used in tools such as *Tcpdump* and *snort*, and is supported by the *Wireshark* and *tshark* family of tools (Harris, 2015). The *libpcap* format is open-source and is basic in nature, not storing a lot of additional information with the packets themselves. This file format can be seen in Figure 2.3. Due to this format's simplicity and open-source nature it is the most widely supported and adopted format for packet captures. The latest version of this format, as implemented in the *Wireshark* project, can be viewed in the *wiretap/libpcap.h* and *libpcap.c* files which can be found at Wireshark (2017).

The *libpcap* format was originally designed within the *Tcpdump* project which can be found at *Tcpdump* (2017). This version was specifically created for Unix-based operating systems, and *Winpcap*⁹ was introduced as its analogue on Windows-based systems. There are wrappers for most programming languages which support reading and writing the *libpcap* file format, such as in Java, Perl, and Python (Harris, 2015). There are some drawbacks to this format due to its simplicity, such as missing certain information which could be useful in packet capture analysis. The newer next generation *pcap* (*pcapng*) file format has been proposed by Harris (2015) and is already supported by some of the most commonly used tools, such as *Wireshark*. The *pcapng* Github repository¹⁰ contains the work that has been done thus far on this format and describes how it is implemented and how to make use of it.

While a few of the packet capturing and analysis tools have already been mentioned, there are many more, such as *Wireshark*, *tshark*, *tcpdump*, and *multicap*¹¹. These are described in more detail in the *Tools*¹² page of the *Wireshark* website. One of the most powerful packet interception and manipulation libraries is *scapy*¹³, which is written in Python and

⁸<https://en.wikipedia.org/wiki/Pcap>

⁹<http://www.winpcap.org/>

¹⁰<https://github.com/pcapng/pcapng>

¹¹<http://src.carnivore.it/multicap/about/>

¹²<https://wiki.wireshark.org/Tools>

¹³<http://www.secdev.org/projects/scapy/>

can easily be used within Python scripts in order to capture, manipulate, and decode all types of network packets.

There are yet other proprietary packet capture formats and mechanisms which are utilised by vendors such as Cisco and Nokia. These are not described here as they are not relevant to packet captures as they are used in this research. More information about these, and many other packet capture formats, can be found in the list of supported input file formats in the Wireshark documentation¹⁴.

In the world of network scanning and telescopes the use of packet captures is essential as the packets flowing into the telescope cannot be effectively analysed in real time. The incoming packets are stored in packet capture files and are sometimes kept for years before they are analysed. It is often simplest and most efficient to store these files in the libpcap format given their small size and wide support. It is then possible to analyse these files at a later date in order to gain an understanding of malware and worm propagation across the Internet, as is described in some detail in Irwin (2011, p. 172-206).

InetVis was originally designed and implemented to make use of the libpcap format, and furthermore, to interface directly with the *libpcap1* user-land API in order to capture, process, and write out packets within the application. This format was chosen due to the ubiquity of its support and use, making it easy for researchers and users to capture their own packets using other tools, and then to visualise them in InetVis.

2.2.3 Network Telescopes

A network telescope is defined as a passive sensor system which is used to monitor activity taking place on the Internet (Irwin, 2011). A network telescope can consist of anything from a single IP address, to an entire subnet of network address space, which is allocated and routed to a system or systems configured in order to passively accept and log all the traffic they receive. By operating in this manner it is possible to monitor mass-activity on the Internet and detect and analyse activities such as worm propagation and distributed denial-of-service (DDoS) attacks (Irwin, 2012). Network telescopes are also sometimes referred to as *darknets*, and while this term has become associated with other illicit and illegal activities in recent years, it is still used in much of the literature to refer to a network telescope (Fachkha and Debbabi, 2016). The passive nature of a darknet is one

¹⁴https://www.wireshark.org/docs/wsug_html_chunked/ChIOOpenSection.html#ChIOInputFormatsSection

of its most defining attributes. It does not initiate any outbound communications nor does it respond to any incoming communications.

Network telescopes have proven themselves useful for many reasons over the years as they provide a way for researchers to gain a unique insight into what activity is currently ongoing on the Internet. It is possible for researchers to view the propagation of malware over time, view the effects of DDoS attacks, and even get a gauge on the measure of Internet Background Radiation (IBR)¹⁵. Researchers are able to do this analysis in retrospect. For example, if a new piece of malware is found with a unique propagation signature, it is possible to look back and determine when such propagation activity was first detected.

The Security and Networks Research Group¹⁶ (SNRG) at Rhodes University has been running a network telescope for many years and has analysed many packet captures in order to produce papers on interesting Internet activities that have been detected by the telescope. Some early research and analysis performed by Rhodes University researchers is presented in van Riel and Irwin (2006a), van Riel and Irwin (2006b), Irwin and van Riel (2008), Schwagele and Irwin (2010), and Cowie and Irwin (2010). An in-depth analysis and description of the telescope is described in detail in Irwin (2011) and Irwin (2012). Much of the network telescope analysis work done since the creation of InetVis has utilised it in order to more effectively analyse the captured data. This is evident when considering the aforementioned references as many of them explicitly use InetVis.

Research in the area of network telescopes has also been performed by researchers at other universities throughout the world. Work performed by Pemberton (2007) specifically considers an empirical analysis of IBR and how best to sample network telescopes at scale. This work considers different network telescope deployment methodologies, whether larger or smaller network address ranges make a difference, and if IBR is uniformly distributed across a given range.

The area of network telescope research is actively pursued at Rhodes University within the SNRG, with research and papers regularly being produced on various aspects, such as, a look at a network telescope dashboard with data aggregation by Hunter (2010). Internationally, the research community tends to prefer the term darknet, and new research is being published every year in which network telescope analysis still refers to these as darknets. For example, a paper proposing a taxonomy for darknet traffic by Liu and

¹⁵ *You know, I coined the acronym "IBR" for Internet Background Radiation. It's just like static on the Net* - Steve Gibson, Security Now #003

¹⁶ <https://www.ru.ac.za/computerscience/research/securitynetworks/>

Fukuda (2014), and recent research by Fachkha and Debbabi (2016) looking at darknet traffic as a source of cyber intelligence.

Research is also being done specifically concerning the visualisation of darknet traffic (Coudriau *et al.*, 2016). In this paper the researchers look at applying an algorithm from the field of Topological Data Analysis in order to analyse a new visualisation method. This method allows the researchers to analyse a large number of packets visually much more quickly than they would have been able to using traditional methods. This analysis lead to the discovery of easily observable patterns in darknet data, which were missed by the Suricata¹⁷ intrusion detection system (IDS) (Coudriau *et al.*, 2016). This discovery is reminiscent of that made by the original InetVis researcher when developing the application, and being able to detect patterns of malicious activity that were not detected by the *Snort* and *Bro* intrusion detection systems (van Riel and Irwin, 2006a; Irwin and van Riel, 2008).

2.3 Data Visualisation

Data visualisation is part of the larger area of information visualisation, with specific focus on the visual representation of data (Barrera, 2009, p. 6-10). The superset of information visualisation adds a specific focus on conveying concepts clearly, and aims to take advantage of the visual processing capabilities of humans (Goodall, 2008), as well as other aspects, which help to convey all types of information clearly and effectively. Data visualisation as a science is used across the globe in almost every area of research. It aids researchers in the process of decision making and enables them to communicate complicated and abstract ideas more effectively to their peers, the larger scientific community, and the public at large (Barrera, 2009, p. 6-10). The visual capabilities of humans allow us to design and consume effective visualisation constructs, which further aid in the exploration and discovery of new information, by making use of our pattern recognition capabilities. As humans, we are able to look at a large amount of visualised data and from that we are able to detect patterns which are not readily identified by even today's most sophisticated software (Jin, 2008, p. 20-27).

While information and data visualisation techniques are extremely powerful, they do not completely replace text-based tools which are commonly used in conjunction with visual data representations (Barrera, 2009, p. 6-10). For example, even the best network traffic

¹⁷<https://suricata-ids.org/>

visualisation software will lack the granularity required in certain circumstances. In order to deeply analyse the data, the raw or processed textual base of the visualisation will need to be reviewed. While most effective data visualisation tools and constructs use the information systems mantra “Overview First, Zoom and Filter, then Details on Demand” described in Shneiderman (1996), which aims to promote the idea of providing details on demand, it is impossible to provide all the information that any potential user of the software might need. It is also the case that computer-based processing systems are much more effective than humans at large-scale, pre-defined computational tasks, and for this reason, these systems and modern information and data visualisation systems are most effective when used in conjunction with one another (Jin, 2008, p. 20-27). Humans are more easily able to explore data and identify patterns, while computers are more effective at performing large-scale computational tasks.

In data visualisation, each aspect of visualisation has been carefully considered over the years, in order to gain greater understanding into the most appropriate graphical elements to use, which scenarios to use them in, and how best to convey certain types of information. When data is visualised there are many aspects which can be used to differentiate data points from one another. For example, colour, shape, position, and size. By using these attributes it is possible to effectively visualise a large amount of data while optimising the visualisation for the fast visual processing and pattern recognition in our brains (Barrera, 2009, p. 6-10). This type of optimisation is done by taking advantage of the specifics of our visual capabilities, most notable of which is pre-attentive processing. As humans we have the built-in ability to detect patterns, outliers, and other meaningful pieces of visual information without having to actually search them out in a visual scene; this happens automatically in our brains (Barrera, 2009, p. 6-10). Our brains process visual information in parallel through a visual pipeline, as opposed to the serial processing that is used when processing text-based information (Goodall, 2008). This allows us to quickly analyse a visual scene and detect anything that stands out to us, without having to actually look at and process every aspect of the scene.

Pre-attentive processing takes place before attention is paid to a visual scene. It does not require any conscious effort on our part for the processing to take place and the speed of this processing is not affected by the number of visual elements (Jin, 2008, p. 20-27). This is further reason why taking advantage of our visual capabilities has become a focus in the area of information visualisation. Effectively exploiting our visual capabilities leads to more efficient pattern recognition. The *Gestalt* laws provide empirical guidelines for exploiting pre-attentive capabilities in order to enhance our cognitive abilities (Jin, 2008, p. 20-27). These laws take advantage of how our pre-attentive abilities cluster graphical

elements. For example, objects that are close together, the same shape, or vertically symmetrical, are considered to form a group. By designing data visualisation constructs to take advantage of this it is possible to further improve the efficiency of our visual capabilities and pre-attentive processing.

While powerful, information and data visualisation is not without its limitations and drawbacks in certain scenarios. Designers of graphical tools and graphs need to be aware of the common pitfalls and problems that arise when visualising data. For example, *occlusion* occurs when data points overlap adjacent points, thus making neither element particularly visible (Barrera, 2009, p. 6-10). This is often a problem with 3D visualisations and there exist various ways to combat this, namely allowing for interaction with the visualisation, so that the user can better view the occluded data.

The Awesome Dataviz¹⁸ Github repository provides a curated list of popular data visualisation frameworks, libraries, and software. Visualisation frameworks and libraries are presented for all of the major programming languages, as well as other resources being provided such as a link to a book by Tufte (1990, 1997, 2001). This resource provides a good overview of the data visualisation landscape.

For a deeper introduction and a more thorough look at the data visualisation domain the reader is directed to the many books written by Tufte (1990, 1997, 2001) on the subject. The work by Munzner (2014) is also particularly useful and provides a more up-to-date resource on modern data visualisation research. There is also substantial information provided online at *The Data Visualisation Catalogue*'s website, in which all of the different types of data visualisations are available, with descriptions, images, when to use them, and examples (Ribeca, 2017). The scatter plot¹⁹ visualisation used by InetVis is described in the *Data Visualisation Catalog* in some detail.

An in-depth discussion on security metrics is presented by Jaquith (2007), in which he explains and makes the case for the use of sparklines in data visualisation for security analysis. As described in Meharia (2012, p.46-48), sparklines are defined as small graphs, around the same size as the text or table that they are either embedded in, or close to. Sparklines are especially useful in showing the trend in value of a single metric over time. For example, see Figure 2.4. The concept of including widgets making use of sparklines in a HUD is discussed in Section 5.2.2.

This research makes use of the 3D scatter plot visual graph construct in InetVis in order to visualise network packet captures. While the use of a 3D scatter plot is based on the

¹⁸<https://github.com/fasouto/awesome-dataviz>

¹⁹<https://datavizcatalogue.com/methods/scatterplot.html>

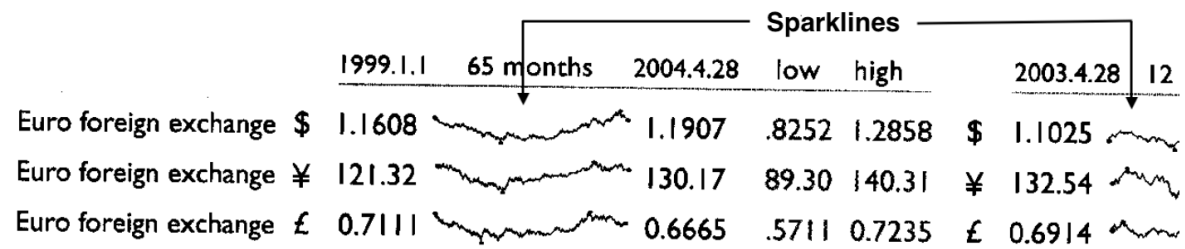


Figure 2.4: Sparklines tracking the price of foreign exchange (Tufte, 2006).

original research, the *Spinning Cube of Potential Doom* by Lau (2003), it is the most appropriate visualisation choice for this use case as it is good at highlighting outliers and clusters within data (Jin, 2008, p. 20-27). The tool also makes appropriate use of counters to the traditional issues found in 3D scatter plots. The effects of occlusion are combated by allowing user interaction with the 3D cube, and the point size is adjustable. The point colour is configurable, based on various attributes and filtering options are provided in order to remove extraneous points from the graph.

2.4 Network Security Data Visualisation

The combined research area of network security data visualisation combines the fields of information and data visualisation with the field of computer network security. This area is commonly referred to as security visualisation for short and aims to provide effective visualisations for common network and security data, such as packet captures and log entries, that allow analysts to perform their jobs more easily (Barrera, 2009, p. 40). This is precisely the aim of InetVis, making it a perfect fit for this category.

Exploration began in this area over a decade ago, with some of the early work laying the foundation for the visualisation techniques and choices that are commonplace in tools and frameworks today. One such piece of literature is the book by Conti (2007) which explores this area with specific focus on graphical techniques for network analysis. The interested reader should consider reading the relevant chapter in his book, Chapter 10: *Creating a Security Visualisation System*. This research borrows ideas from this book. Such as the concept of overcoming occlusion through zooming (Conti, 2007, p. 46). Other ideas were also incorporated from Conti (2007, p. 191-193) where optimal interface design is discussed, specifically the overview first, zoom and filter, details on demand rule mentioned earlier, and the VCR-like visual mechanism used for playing back packet capture files.

Another early book written in the area of security visualisation is *Applied Security Visualisation* by Marty (2009). In this book, Marty specifically focuses on the application of information and data visualisation ideas and foundations in the area of computer security. The reader is directed to this book for additional foundational reading in this area. Marty specifically calls out the fact that most visualisation tools are written by security experts who have no understanding of the information visualisation or human-computer interaction (HCI) fields, or by visualisation experts who do not have any real-world experience in the world of computer security (Marty, 2009, p. 7). He refers to this phenomenon as the dichotomy of security visualization. In this respect InetVis and its related research fall into the first category. However, by identifying this weakness it is possible to correct the missing visualisation and HCI elements of the tool.

Marty covers further topics which were also used in this research, namely the correct selection of data types, colour, size, shape, orientation, and the axes of the graph used by effective visualisation tools (Marty, 2009, p. 65-70). While most of these concepts are inherited from the parent field of data visualisation, Marty places specific focus on where to use which visualisation paradigm, for which type of data, and to enhance which analysis. An in-depth discussion on the use of scatter plots (Marty, 2009, p.82-84), and 3D scatter plots (Marty, 2009, p.101-103) is presented which highlights many of the strengths and weaknesses in the use of this visual system within InetVis. Most notable, is the difficulty often faced on identifying specific points when the graph becomes too cluttered, which is partially remedied in InetVis by the use of interactive controls such as zooming, panning, tilting, and rotation. The topics of visualisation interaction, as well as animation, are discussed in (Marty, 2009, p.104-108).

Another useful reference in this area is that of Barrera (2009), who performed research on the classification and selection of appropriate visualisation techniques. In this research the author considers common types of network attacks, finally concluding with recommendations and selection criteria for the aforementioned classes of attack. The author describes InetVis as one of the previous approaches to network scan detection (Barrera, 2009, p.77-80), and while he describes a TCP histogram approach as being superior, the author does conclude with some recommendations for the use of scatter plots due to their tried and true usage history. One of the recommendations, extended time windows, is implemented in InetVis, however the other two being time as one of the axes and remote host aggregation are absent.

There are other pieces of research in this area which are more introductory in nature, typically in the form of conference papers, such as those by Goodall (2008). Furthermore,

there is also research performed in this area at the Masters and Doctoral level, respectively by Jin (2008), Mansmann (2008), and more recently by Hunter (2010) and Nottingham (2016). Much of this research covers the basics of information visualisation theory, the visual capabilities of humans, and then proceeds to extend into the computer security area with topics such as detecting malicious network activity, and traffic monitoring, detection, and interpretation.

2.5 Related Tools

In this section a selection of tools will be introduced and discussed. In Section 2.6 a high-level discussion is presented, highlighting the common themes, features, the good, and the bad across the tools discussed here. This section does not aim to provide a full accounting of all security visualisation tools; only those that are related to InetVis. The goal of the discussion is to highlight commonality between tools, and to identify potential new features and improvements which could be incorporated into InetVis as part of this research and in future work.

2.5.1 The GPL Cube of Potential Doom

The *GPL Cube of Potential Doom* was created by Dragorn, the main project maintainer for the Kismet Wireless²⁰ project. As with InetVis, this tool was based off of the *Spinning Cube of Potential Doom* by Lau (2003), and as such it is similar to InetVis. This tool's executable, *doomcube*, accepts input via *stdin*. It then visualises this input using a 3D scatter plot in the same way as InetVis. A sample visualisation which depicts a port scan can be seen in Figure 2.5. The tool also supports command line arguments for setting the source and destination range, the number of points displayed, as well as the number of divisions in the background grid. When running, the application responds to mouse events in order to rotate the cube, as well as supporting keyboard shortcuts (Dragorn@kismetwireless.net, 2011). This tool and InetVis both suffer from the same problem, namely that they have not been updated since 2011.

While most tools will not be directly compared to InetVis, this one in particular is worth considering. Table 2.1 shows a comparison between the two tools, comparing various aspects of functionality. In summary, the GPL Cube of Potential Doom offers a fully

²⁰<https://www.kismetwireless.net/>

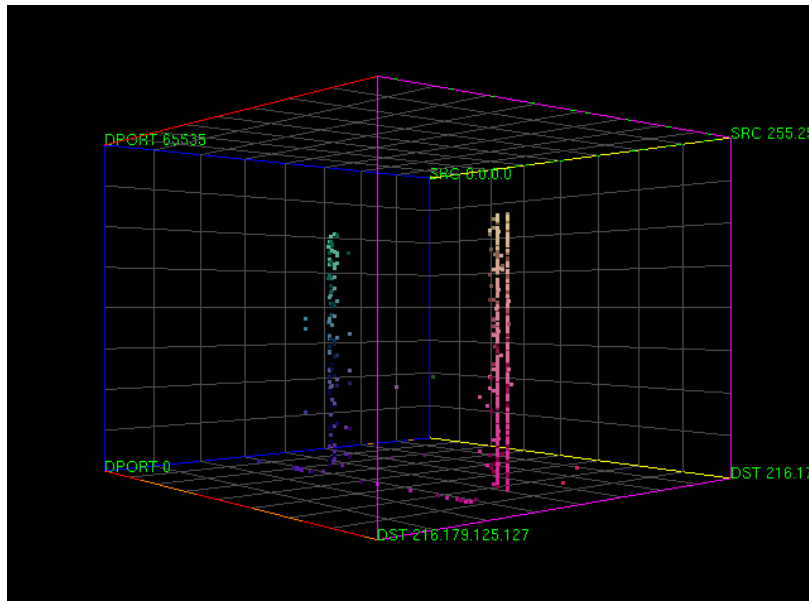


Figure 2.5: The GPL Cube of Potential Doom (Dragorn@kismetwireless.net, 2011).

Table 2.1: Comparison of GPL Cube of Potential Doom and InetVis.

	InetVis	GPL Cube
User Interface	Graphical	Command line interface
Keyboard Shortcuts	Numerous	Only a few
Input	libpcap capture files	Custom triple format
Monitoring	Supported	Not supported
VCR-like Operation	Supported	Not supported
Historic View	Full support	Partial support
Output	Various options	Not supported
Filtering	BPF-style filters	No support
3D Cube	Very flexible	Basic support

fledged implementation of Lau’s Cube, but does not extend it into some of the areas that InetVis does. InetVis provides a robust graphical interface with many useful options that aid in analysis, as well as supporting visualising pcap files natively, and with support for monitoring a local network interface.

2.5.2 IDS RainStorm

*IDS RainStorm*²¹ was designed and implemented in order to effectively deal with the large amounts of log data generated from intrusion detection systems (Abdullah *et al.*, 2005). It achieves this goal by presenting log data in such a way that it is easy for analysts to

²¹<https://github.com/chrislee35/ids-rainstorm>

detect anomalies. It does this by combining the scatter plot and the parallel coordinate plot visualisations (Jin, 2008). An overview screen is provided, and the user is able to drill-down into the data by making use of zoom functionality. The main user interface can be seen in Figure 2.6 and the zoom display can be seen in Figure 2.7.

On the main display, the x -axis is used to denote the time dimension, while the multiple y -axes represent the local IP addresses where alarms occur on the network. These y -axes are broken up into two-and-a-half class B networks, with bold horizontal lines denoting the start of a new class B network (Abdullah *et al.*, 2005). The zoom display window can be brought up by selecting a portion of the main window to investigate further. This presents a new display with additional information about the alerts in question. Legends are shown, explaining what the colouring of the points mean, and the local IP address is shown on the right-hand side, with the Internet IP address shown on the right. Lines are drawn connecting source and destination IP addresses in order to indicate on which host an alarm was triggered, and which foreign host triggered it.

IDS RainStorm accomplishes its goal of providing a new visualisation mechanism to enable analysts and administrators to work more effectively. It provides a high-level overview of all IDS alarms within a given network, taking into account large networks by using multiple y -axes, and further provides a detailed zoom view where the user can drill-down and determine further information about points of interest on the main display. The reader may consult the research by Abdullah *et al.* (2005) and Conti *et al.* (2006) for additional information.

2.5.3 RUMINT

*RUMINT*²² is closely related to *IDS RainStorm* as it was written by the same group of researchers, as well as being first introduced in the same piece of research by (Conti *et al.*, 2006). The goal of this research was to provide tools in order to help security analysts deal with the ever-increasing amounts of data that they have to analyse on a regular basis. Both *IDS RainStorm* and *RUMINT* are visualisation tools with the aim of making the life of security analysts easier.

RUMINT was designed in order to provide a detailed visual representation of packet level network data (Conti *et al.*, 2006). It was designed in this manner in order to improve on the status quo in the security visualisation field, where commonly only a few packet headers

²²<http://www.rumint.org/>

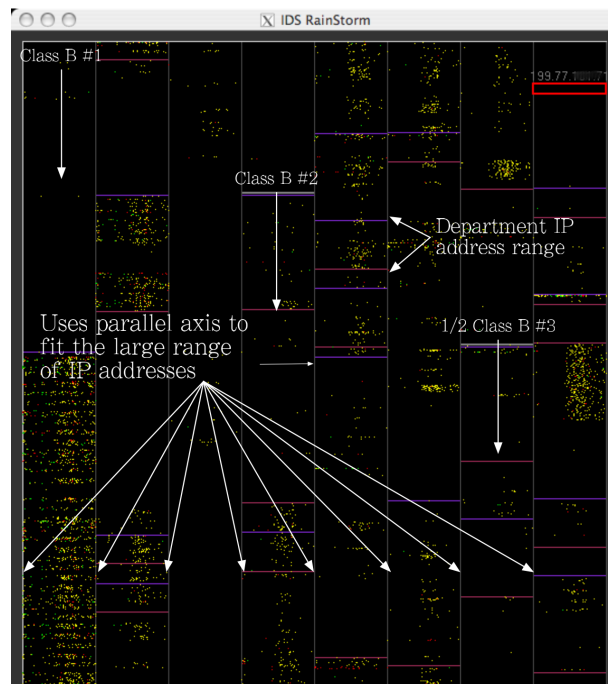


Figure 2.6: The main IDS RainStorm display window (Abdullah *et al.*, 2005).

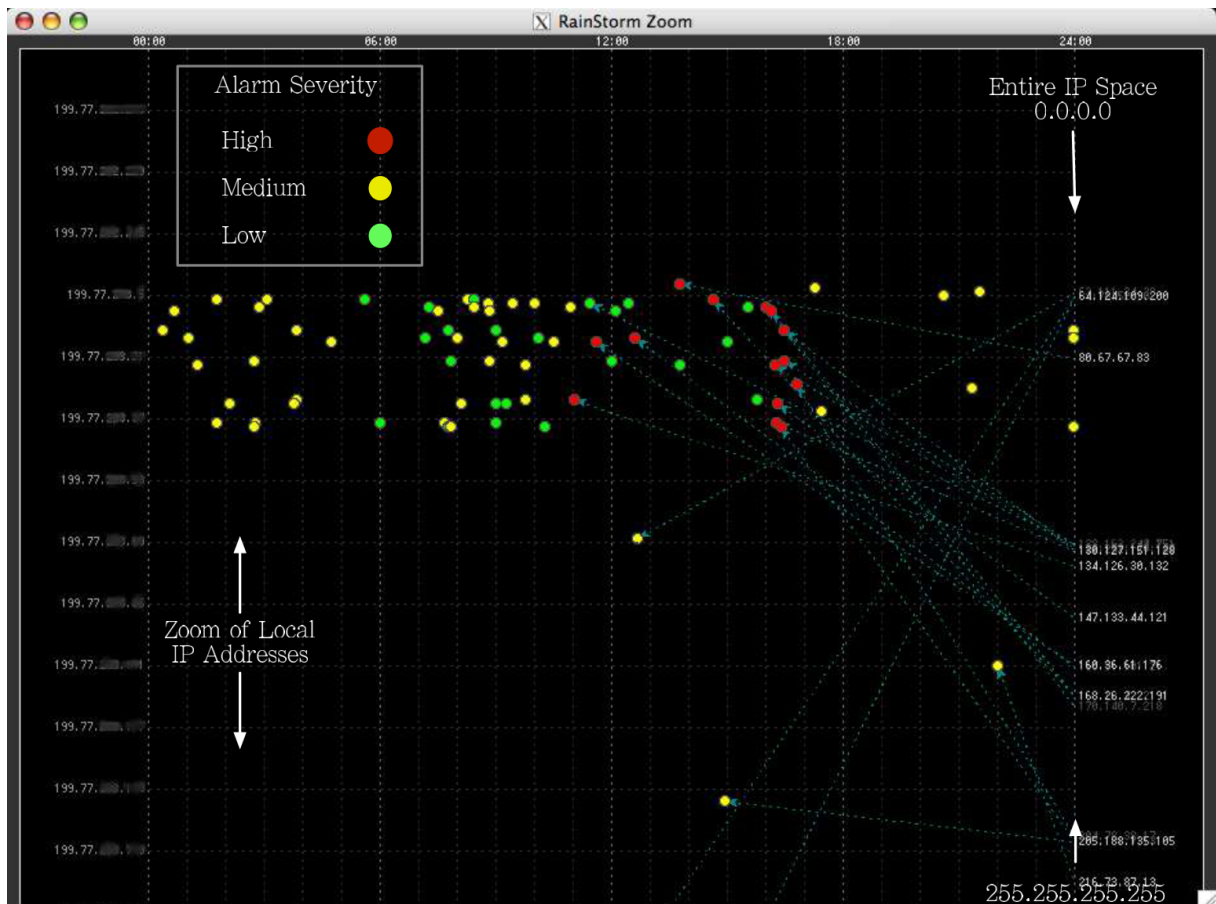


Figure 2.7: The IDS RainStorm zoom window (Abdullah *et al.*, 2005).

are used, and many of the high level attributes are ignored. *RUMINT* incorporates up to nineteen header fields and also provides video-player like functionality, together with a binary rainfall style visualisation (Conti *et al.*, 2006). It also complements *IDS RainStorm* in this way as it allows for an even deeper review on the data than does *IDS RainStorm*'s zoom view.

RUMINT makes use of eight distinct application visualisation windows in order to achieve its goal. It also provides a control panel interface with playback controls, similar to *InetVis* (Conti *et al.*, 2006).

As can be seen here, *RUMINT* is a highly detailed and complex piece of software and this review only scratches the surface on its capabilities. It makes use of many novel visualisations and allows the user a high degree of flexibility in deciding which information to visualise and how to visualise it. For more information the original research by Conti *et al.* (2006) may be consulted.

2.5.4 SIFT

The *SIFT project* follows a similar strategy to that of Conti *et al.* (2006) with *IDS RainStorm* and *RUMINT*, by designing and developing two tools within this framework in order to deal with different aspects of network traffic analysis. As with the other tools already discussed, the *SIFT framework* aims to make the life of the security analyst or administrator easier by employing visualisation techniques. The two tools produced, *NVisionIP* and *VisFlowConnect-IP*, are complementary, and are aimed to be used together for effective analysis (Jin, 2008, p. 8-9). Both of these tools are designed with *NetFlow*²³ data in mind, and only support the analysis of this type of data.

The first tool, *NVisionIP*, provides three views to the user with the aim of providing a high-level overview of the data, as well as providing the ability for users to drill-down into the data for further analysis. The galaxy view consists of a scatter plot where the *x* and *y*-axes correspond to the source and destination addresses respectively, and where the colour or shape of the point is configurable to some *NetFlow* property (Jin, 2008). The remaining two views make use of histograms in order to show relevant statistical information about configurable *NetFlow* properties. These views can be seen in Figure 2.8.

²³<https://en.wikipedia.org/wiki/NetFlow>

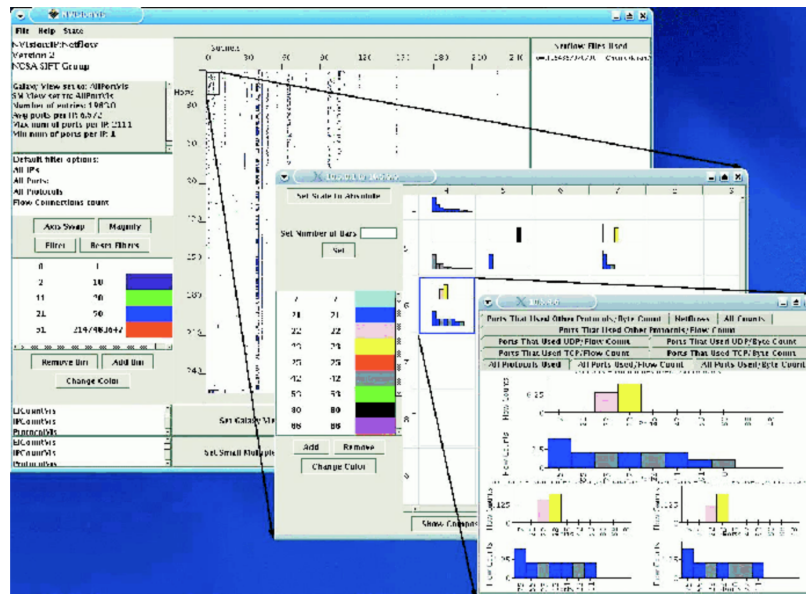


Figure 2.8: The galaxy, small, and machine view of NVisionIP (Yurcik, 2005).

The second tool, *VisFlowConnect-IP*, makes use of a simple parallel coordinate system in order to effectively represent the connections between IP addresses in *NetFlow* data. As can be seen in Figure 2.9, this tool makes use of three vertical axes, with the middle axis representing the source IP address, and the left and right axes representing the destination IP address, or subnet, depending on the level of detail the user is using (Jin, 2008, p. 8-9). This tool also offers two views which provide a more detailed display and allow the user to specifically focus on either the internal or the external view, focusing on incoming and outgoing connections separately. Furthermore, *VisFlowConnect-IP* attempts to assist the user in determining which connections are present on the network, who initiated them, and who they are with (Yurcik, 2005).

2.5.5 AfterGlow

AfterGlow consists of a collection of scripts which allow the user to generically visualise the relationship between entities in a data set (Jin, 2008, p. 11). The goal of the tool is to visualise the relationship between entities in a data set in order to make analysis more efficient. The tool makes use of two distinct visualisations, namely the treemap and the network graph. The tool can be applied to a wide-range of disparate data sets. For example, packet captures, email/firewall/web logs, intrusion detection system logs, and operating system logs (Jin, 2008, p. 11). *AfterGlow* makes use of the *GraphViz*²⁴

²⁴<http://www.graphviz.org/>

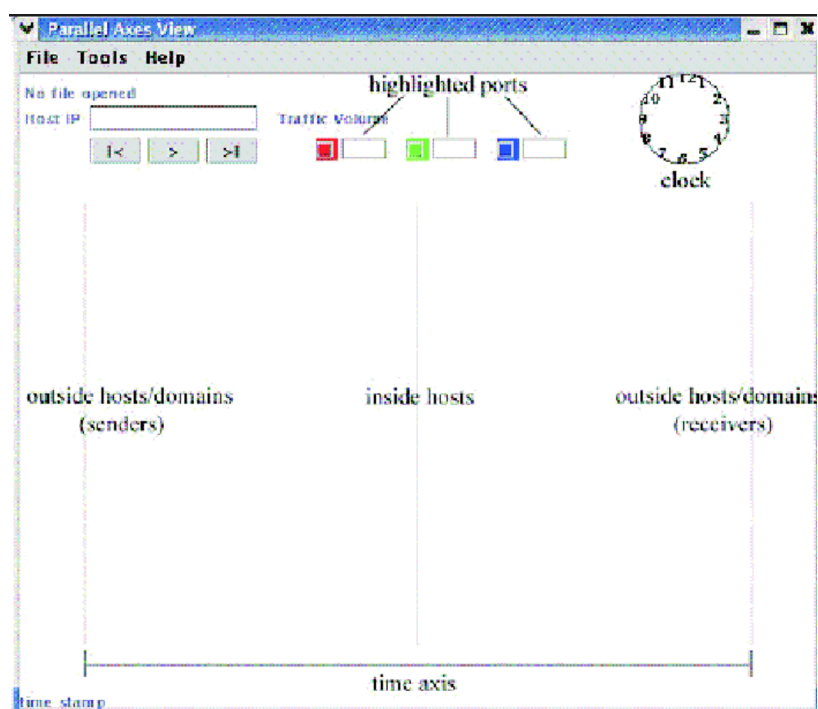


Figure 2.9: The main VisFlowConnect-IP display (Yurcik, 2005).

visualisation library due to its flexibility. However, this does mean that it lacks any real user interface.

Scripts are provided by the author for a few of the common use cases, such as visualising network packet captures and email logs. However, users are able to use whatever data they like as long as it is provided to *AfterGlow* in the CSV format (Marty, 2013). The main display window can be seen in Figure 2.10, which shows the visualisation of firewall data in the network graph style.

For more detailed information on this tool please see the official home page located at <http://afterglow.sourceforge.net/>.

2.5.6 PortVis

PortVis was introduced in research by McPherson *et al.* (2004) in order to support the visual analysis of coarse data sets. A coarse data set is defined as one in which most of the important details of the data are missing, sometimes even including the IP address. This type of data set may be encountered in an organisation where security concerns, or technical limitations, limit the amount of data that can be shared with outside security analysts (McPherson *et al.*, 2004). This tool offers a solution for the visual analysis of

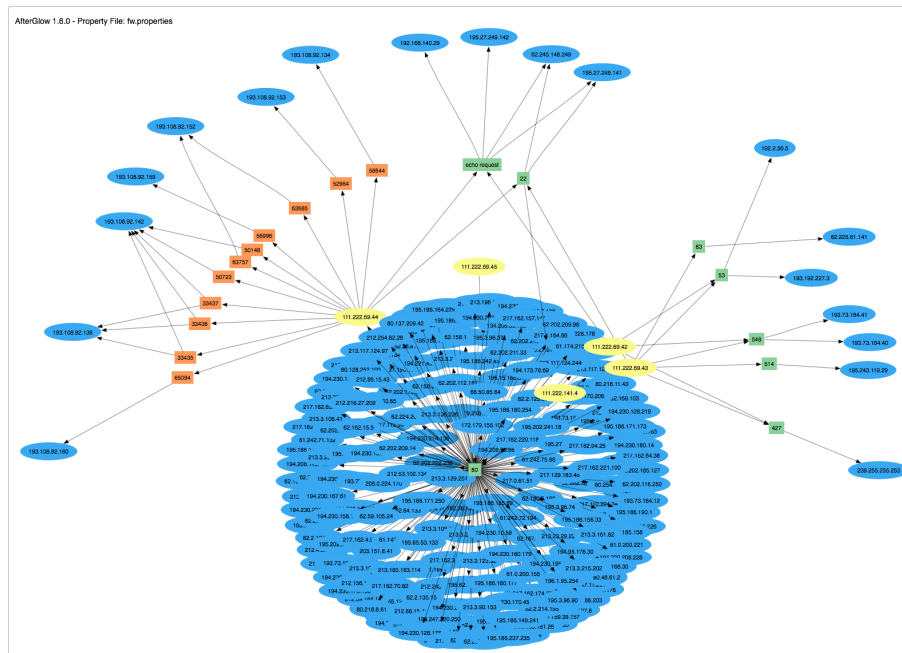


Figure 2.10: The Afterglow display window (Marty, 2013).

these data sets and allows analysts to discover security events without the usual amount of information. This tool has proven itself particularly useful in the detection of network and port scans (McPherson, 2004).

The main *PortVis* UI can be seen in Figure 2.11. The UI consists of multiple panes, all showing different information. *PortVis* makes use of the *high level overview with details on demand* paradigm and this can be seen in the UI. The far left vertical pane offers the overview, and is known as the *timeline* (McPherson *et al.*, 2004). It shows the entire visualised dataset, with time progressing forwards from top to bottom. The largest pane shown, in the centre of the application, is the *main* visualisation, which represents an hour of time, as selected in the *timeline*. The *main* visualisation allows the user to select a point to further drill-down into, as shown by the circle. The top right visualisation is the *detail* view, and shows detail about the point selected on the *main* visualisation. Further, the visualisation below it, shows the port details for whichever port is selected in the *detail* view. The bottom row of the display window consists of options to control the appearance of the visualisations, followed by settings to control the source data and the colouring used for the gradients within the visualisations (McPherson *et al.*, 2004).

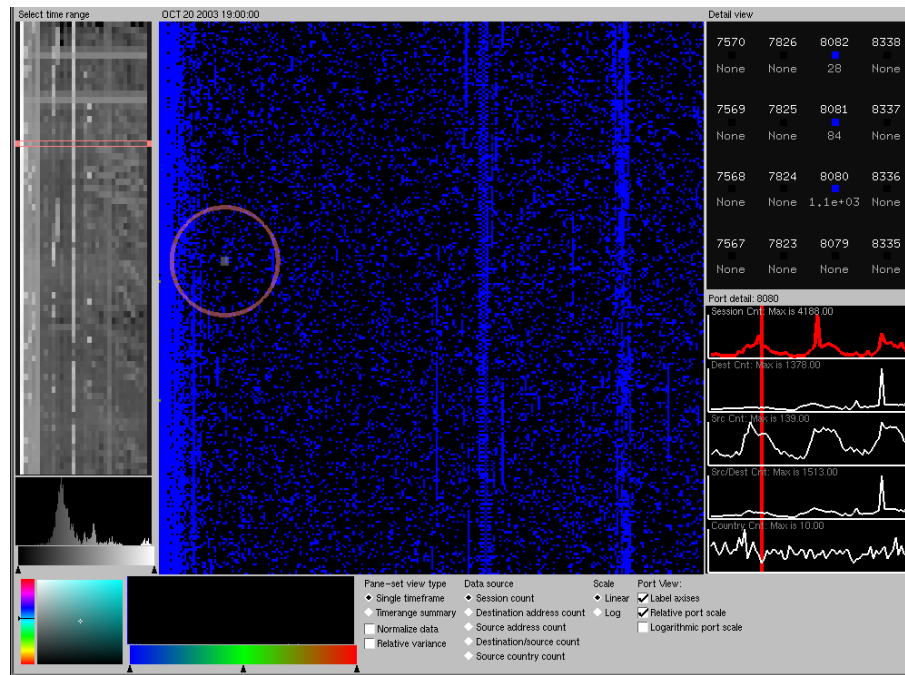


Figure 2.11: The Portvis overview display (Marty, 2013).

2.5.7 Cyberspace Visualisation Tool

The *Cyberspace Visualisation Tool* is not your typical security visualisation tool, as it differs significantly from those already discussed, and it is relatively novel. One of the goals of this tool is to find an effective way in which to visualise cyberspace activities in three dimensions, with the end result being that security analysts are more easily able to identify malicious traffic and threats (Lindemann, 2017). The tool developed by Bass's team combines data from multiple sensors in order to achieve the visualisation and to provide a cyberspace situational awareness overview to analysts (Bass *et al.*, 2017).

The interface of this visualisation tool is much like that of a computer game, the user is able to travel through the visualisation as if they were piloting a spacecraft transiting outer space. The tool was designed this way in order to make the normally dull analysis process much more exciting, and in order to offer a new perspective on the constantly evolving Internet traffic (Lindemann, 2017). The creators of the tool also make use of the visual capabilities of humans, particularly those of pattern recognition and cognitive processing. Visualisation tools, including InetVis, have been exploiting these capabilities for over a decade.

A sample of the UI is presented in Figure 2.12, which depicts a spacecraft and various visual objects. These objects are generated from sensors, turned into objects for the



Figure 2.12: The Cyberspace Visualisation Tool display (Bass *et al.*, 2017).

display, and likened to planets, stars, and other objects, in order to create the game like experience. The reader is directed to the ResearchGate project website for more information on this project²⁵.

2.5.8 TNV

TNV, which is short for either *The Network Visualiser* or *Time-based Network Visualiser*, according to the project's authors, is specifically aimed at the area of network packet capture analysis (Goodall, 2009). *TNV* was designed by looking at the work practices of security analysts, and aiming to implement a visualisation system which would allow them to gain an understanding of baseline network activity, easily highlight and explore the details of security events, and potentially even assist with network troubleshooting (Goodall, 2009). The tool has support for opening packet capture files in the libpcap format, as well as allowing for monitoring of a local network interface.

The main application visualisation window can be seen in Figure 2.13. The display is reasonably detailed and complex. The largest visualisation area is the main visualisation matrix, which combines the source IP address, destination IP address, and timestamp of network connections. These connections are represented as lines between communicating hosts, with colour, and other attributes, denoting the number of packets, and various

²⁵<https://www.researchgate.net/project/Cyberspace-Situational-Awareness>

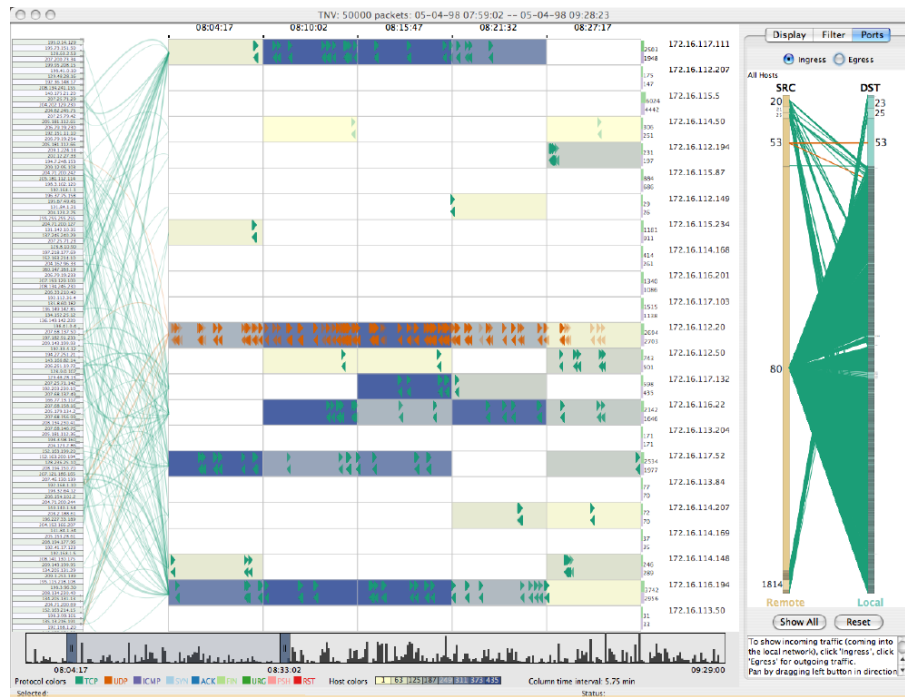


Figure 2.13: The TNV display (Goodall, 2009).

other pieces of information (Goodall *et al.*, 2005). The histogram visualisation below the host matrix provides an overview of the entire packet capture, and is user selectable and scrollable. Below this, a legend is shown indicating what each of the colours means throughout the matrix. The visualisation pane on the right of the main matrix shows the port activity for a selected pair of hosts.

Upon initial usage and review by McRee (2008), it was determined that while *TNV* is one of the slowest tools he tested, it provides interesting opportunities for visual analysis and manipulation. Menu options which allow the user to view packet details, as well as the usefulness of the time slider at the bottom of the display window, are also highly useful.

2.5.9 VisualFirewall

VisualFirewall was created by Lee *et al.* (2005) in order to provide analysts with an effective way to visually monitor their networks, and specifically, to aid in the configuration of firewalls. The tool provides the user with four distinct visualisations, namely: the real-time traffic display; the visual signature display; the statistics display; and the IDS alarm visualisation.

The visual signature view displays TCP and UDP connections as lines between local ports

and foreign hosts (Lee *et al.*, 2005), and can be seen in Figure 2.14. New connections are brighter at first, and older connection lines fade-out after a preset time. This view has a similar purpose to InetVis, and it proves useful in determining when incoming port scans and sweeps are occurring.

The real-time visualisation display is shown on the right, in the centre, in Figure 2.14, and shows how packets move from foreign hosts on the Internet (right-hand y -axis) to specific ports on the firewall of a network (on the left-hand y -axis). The user is able to select this visualisation and make it expand to fill the main visualisation area. Glyphs are used to represent incoming packets and motion is used in order to indicate the transit direction of packets (Lee *et al.*, 2005). This view is especially useful for auditing firewall configuration, as if a packet is rejected from a foreign host it is visualised as a reflection, while accepted packets continue straight to the left-hand side of the visualisation.

The statistics view can be seen at the bottom right of Figure 2.14 and it depicts the overall throughput of the network over time (Lee *et al.*, 2005). The x -axis represents time, and the y -axis represents throughput, with three lines being plotted for each time interval showing the total, incoming, and outgoing throughput rates. The IDS alarm view can be seen at the top right of the figure, this view represents IDS alarms triggered by remote hosts. The left-hand y -axis now represents groupings of IDS rules, which trigger alarms, and lines are drawn between foreign hosts and the relevant alarm types. The x -axis again represents time and this visualisation shows a day's worth of IDS alarms at one time.

The design of this tool is more complicated than most discussed here, as it uses an event-based architecture, where IDS alerts and firewall events are continually received, which are then processed and passed further along in the pipeline until the events are visualised.

2.5.10 Mapper

Mapper is a visualisation tool developed by Coudriau *et al.* (2016) which makes use of a novel mapping algorithm, also named *Mapper*, from the field of *Topological Data Analysis*. This tool was designed in order to effectively visualise and analyse a large number of network packets at once, and to make malicious activity easily observable by security analysts (Coudriau *et al.*, 2016). This tool was tested with network telescope traffic, and observable patterns pointing to malicious activities were found, that were not identified by current state-of-the-art intrusion detection systems. The goal behind the development of this tool, as well as the outcome, are reminiscent of the initial research

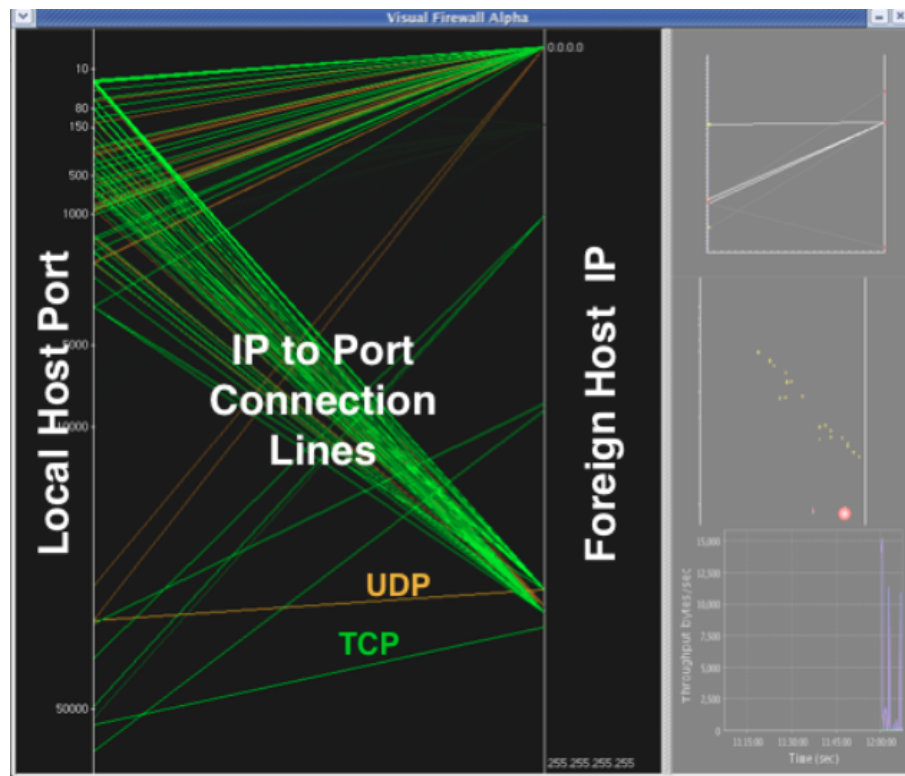


Figure 2.14: The Visual Firewall Visual Signature View (Lee *et al.*, 2005).

done on InetVis in 2005. The original researchers were able to effectively identify malicious activity which was missed by a well known IDS at the time. While InetVis leveraged off of *Lau's Cube* and used a basic 3D scatter plot, *Mapper* makes use of a more novel and complex visualisation and analysis algorithm.

The method used by this tool is complex and is not described in detail here, the interested reader is directed to the original research by Coudriau *et al.* (2016). Figure 2.15 shows a visualisation from the tool, where scans are detected and visualised using a data set of 8,000 network packets. Plane 1 represents the source IP address and port of a packet, while plane 2 represents the destination IP and port. Coloured lines are drawn between the two planes in order to indicate a network connection and the line thickness is used to denote the number of packets transiting a connection (Coudriau *et al.*, 2016). Given this, the scans in Figure 2.15 are easily recognisable.

2.5.11 FlowTag

FlowTag was introduced by Lee and Copeland (2006) as an interactive network trace viewer. This tool was developed in order to assist forensic and security analysts with

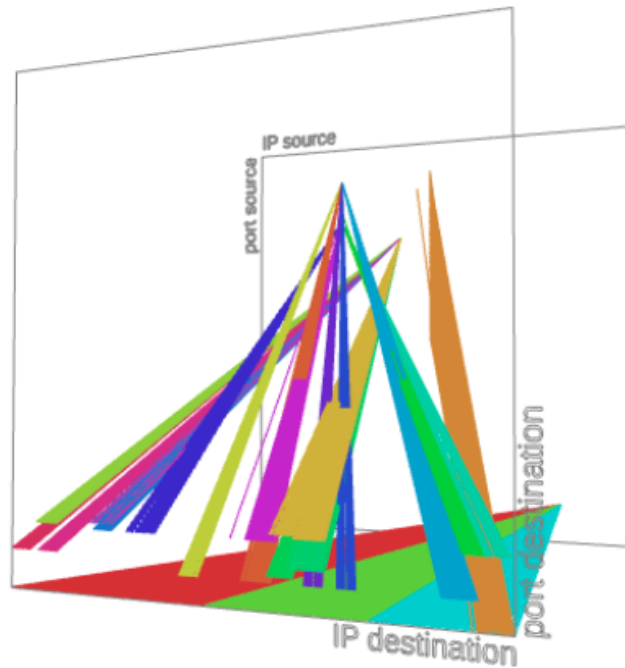


Figure 2.15: The Mapper display of scan detection (Coudriau *et al.*, 2016).

their work, by enabling them to apply visual filtering techniques, and to tag network flows in a meaningful manner. The tool also emphasizes collaboration, and makes it easy for researchers to share their tagged network flows with one another (Lee and Copeland, 2006).

FlowTag is able to operate on network packet capture files, it produces a database of network flows based on the input. After this, the results are visualised, as can be seen in Figure 2.16 (Lee, 2013). Once visualised, users of the tool can then proceed to tag flows with relevant keywords, filter out uninteresting flows, and view the payload of network events should they require extra detail.

Figure 2.16 shows the full interface of *FlowTag*, which consists of six distinct elements. Most of these views are self-explanatory and are described in detail in Lee (2013). The connection visualisation view however represents the only real *visualisation* within the tool. The left axis in this view maps to the local TCP port numbers on a host, while the right axis maps to the foreign (source) IP address initiating the network connection. The foreign IP addresses are shown in the order in which they appear in the packet capture, from top to bottom (Lee, 2013). The user is able to select a flow from the *flow table*, then tag it using the *flow tags* section, and then filter the content visualisation by tag, by

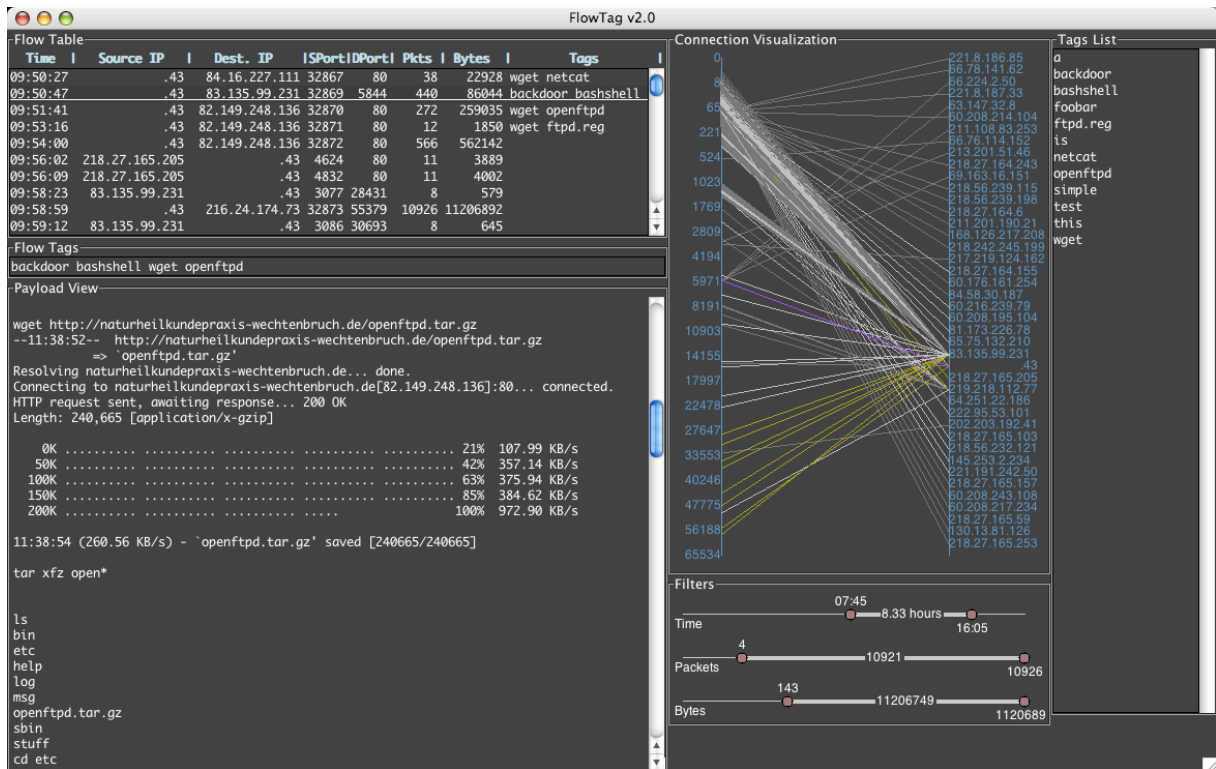


Figure 2.16: The FlowTag display (Lee, 2013).

using the *tag list* on the right of the connection visualisation.

FlowTag proves useful by providing a quick overview of all network flows to the user and providing the facility to get more details when necessary. The tagging system enables the analyst to more efficiently investigate network packet captures by reducing the number of network flows being visualised (Lee and Copeland, 2006).

2.5.12 P3D

P3D utilises a parallel 3D coordinate visualisation for use in the analysis of advanced network scans (Nunnally *et al.*, 2013). It uses the novel approach of using stereoscopic 3D parallel visualisation mechanisms in order to visualise and allow the effective analysis of network scans. The tool also aims to mitigate some of the attacks that may be used in order to thwart traditional visualisation tools. The main class of attacks focused on by this tool, and the related research, is that of occlusion-based attacks, which aim to confuse network security analysts (Nunnally *et al.*, 2013). Another novel idea used in this tool is that of an awareness region, which the tool uses to bring potentially interesting scans to the attention of the user.

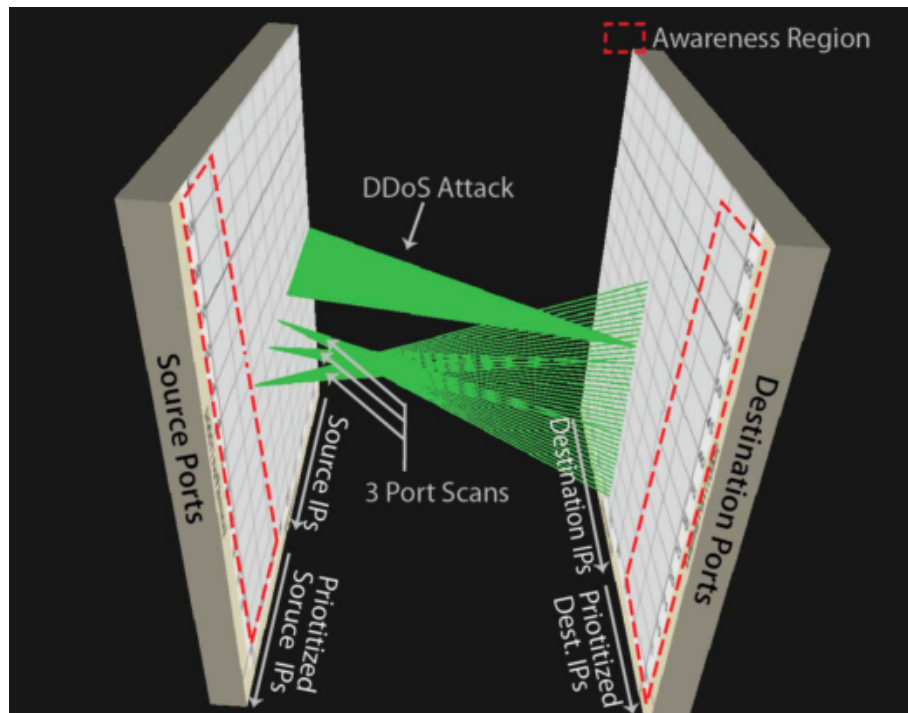


Figure 2.17: The P3D display (Nunnally *et al.*, 2013).

In Figure 2.17 a visualisation produced by *P3D* can be seen, consisting of two parallel 3D planes with lines drawn to indicate connections. The left plane consists of an x -axis representing the source IP address, and a y -axis representing the source port. The right plane similarly has the dimensions of destination IP address and destination port. By looking at the figure, the DDoS attack, and three port scans highlighted, can easily be seen and interpreted correctly even by inexperienced users. The DDoS attack shows visually how many source IP addresses are connecting to a single port on a single destination IP address, i.e. many hosts on the Internet are targeting a certain service on a specific host. The three port scans show the opposite scenario where three distinct source IP addresses each make connections to a wide range of ports across only one, or a few, destination IP addresses. This indicates a few hosts on the Internet are port scanning IP addresses within the internal network.

2.5.13 DAVIX

The *DAVIX* project²⁶, short for *Data Analysis and Visualisation Linux*, was released in August 2008 at the Blackhat/DEFCON conference week (Monsch, 2008). *DAVIX* is not

²⁶<http://www.secviz.org/node/89>

Table 2.2: Visualisation tools available in DAVIX.

AfterGlow	GGobi	GraphViz	Parvis	Shoki
Cytoscape	GUESS	InetVis	PicViz	TNV
Dotty & Ineato	Gephi	LGL	Ploticus	TimeSearcher
ELK Stack	Gitail	MRTG/RRD	R Project	TreeMap
EtherApe	GnuPlot	Mondrian	RT3DG	Tulip
FlowTag	Google Earth	NVisionIP	Rumint	Walrus

actually a visualisation tool, it is a live CD to be used for data analysis and security visualisation. It is a customised version of the Ubuntu 14.04 Linux distribution that has been preloaded with a large number of existing tools for use in data analysis, data visualisation, and related activities. It is only available as a VMWare image file at this time and the latest release was made in October of 2014 (Marty, 2015). The project also has a Github repository where all of the tools, installation scripts, and customisations are kept. It can be found at <https://github.com/secviz/davix/>. This repository is more up-to-date and allows users to make use of the latest scripts themselves, in order to create their own DAVIX installation, without the use of the outdated VMWare image.

DAVIX contains many of the tools that have already been discussed in this section, as well as a few that were not discussed. Table 2.2 shows all of the tools included in *DAVIX* at this time. This list is based on the official project website and not on the more up-to-date Github repository. Of particular note here is the fact that InetVis was included in the original version of *DAVIX*, and is still being included in the latest version.

This project has been described as highly useful, especially for beginners in the field, as finding and correctly installing the myriad of tools out there is not a simple task. *DAVIX* aims to integrate all of these existing tools into a single Linux distribution and take all of the tedium out of finding and installing the tools manually. Conti made this remark in a security visualisation tools survey paper by McRee (2008).

2.6 Related Tool Discussion

In this section the overarching principles, goals, and visualisation choices made in the tools described in Section 2.5 will be presented. This is followed by a discussion on the findings and is concluded with comparisons of the tools.

By examining the aforementioned tools trends emerge regarding what the tools were designed to accomplish, how they accomplished it, and what sort of visualisations they

used. One of the most notable points here is that almost all of the visualisation tools presented relatively complex user interfaces or visualisations when compared to the 3D scatter plot visualisation used by InetVis. Only the *GPL Cube of Potential Doom* (Section 2.5) was made up of one visualisation and was simple to understand. Likewise, *P3D* (Section 2.17) presents a clean and easy to understand UI even while making use of complex stereoscopic 3D methods. While this does not represent a shortcoming in any of the more complex tools it simply shows the emphasis the researchers and creators placed on providing a holistic visualisation tool. Given the basis for InetVis it is only natural for it to be simplistic and easy to understand. However, there is certainly room for more detail-oriented visualisation elements in InetVis.

Another common theme which was discovered amongst the tools is that the majority of them supported the pcap file format. Other tools made use of *NetFlow* data, and one or two notable exceptions made use of custom data formats or specialised event feeds. A few of the tools supported IDS and firewall alarm and event data in order to provide visualisations specifically tailored at the IDS and firewall security analysts. While yet others supported monitoring a local network interface allowing for real-time analysis. InetVis is similar to the majority of applications in this respect, with its support for pcap packet captures and live interface monitoring. The addition of *NetFlow* support may prove useful in InetVis as the entire packet contents stored in pcap files is not used at this time.

Almost every tool in this review had the goal of making the analyst's life easier by providing a visual means for the efficient analysis of network packet captures, firewall logs, and IDS logs. This is a noble goal and underpins the reason for the existence of the security visualisation field. If analysts were able to do their jobs effectively by merely monitoring textual logs, then this field would not exist. All visualisation tools exist in order to provide a more effective way of analysing a certain aspect of computer network security.

A few of the newer tools such as the *Cyberspace Visualisation Tool*, *Mapper* and *P3D* offer novel approaches to visualisation and data classification. These tools stray from the visualisation methods used a decade ago and make use of more complicated 3D representations of network events and hosts. These tools provide a unique perspective on network data and help to overcome some of the shortcomings of the older visualisation tools. *P3D* in particular aims to combat many of the occlusion class attacks that may be used to evade analysts. InetVis does not require this level of visualisation complexity given its primary function is to provide a light-weight and efficient solution for the analysis of

network telescope data.

Many of the visualisation tools discussed here also make use of more than one visualisation in conjunction with the main visualisation. For example, histograms of traffic throughput and detailed packet body data. These are features which could be highly useful if incorporated into InetVis. The implementation of a heads-up display with the capability of displaying multiple distinct visual widgets is discussed in Section 5.2.2. A further improvement which is common among related tools which could be implemented is support for mouse-over of event points (Section 5.2.3).

2.7 Summary

In this chapter the areas of network security, data visualisation, and network security data visualisation were presented and discussed. Typical forms of network scans were presented and discussed, notably the horizontal network scan and the vertical port scan. Following this the concept of a network packet capture was introduced and described. The concept of a network telescope was presented and its importance to this research was described.

The pre-eminent resources and research in the field of network security visualisation were then presented and discussed, with emphasis on those concepts and constructs used in this research. The books written by Conti (2007) and Marty (2009) on this topic represent the foundations of this field and were consulted in this research for effective extensions to the software.

Related research in this field, specifically concerning the tools and techniques that have come about within the past decade were then presented and discussed. Following this a review of the common themes across all tools and how they relate to this research was presented. A few novel visualisation techniques and tools have been developed in recent years, all of which do a good job at advancing the state of the field. Most tools were more advanced from a visual and analysis perspective than InetVis. Various aspects were identified which could be incorporated into InetVis, namely a heads-up display and improvements to the user interface interaction support.

InetVis is an efficient and effective tool for the visualisation and analysis of network traffic, particularly in the realm of network telescope packet analysis. The modernisation work

and improvements made in this research only serve to further strengthen and improve the usefulness of InetVis.

With the foundation presented in this chapter it is now possible to go into further detail on the work undertaken in this research. This work consists of a review of the original application design and source code, presented in Section 3.2, followed by a discussion on the work done to port the application to work on modern operating systems, in Section 3.3.

CHAPTER 3

Design and Implementation

3.1 Introduction

This chapter addresses two main topics: firstly, the design and implementation of the original InetVis software; and secondly, the work completed to port the application to function on modern operating systems. The overview of the original version provides a base for the state of the software at the commencement of this research, and is presented in Section 3.2.

The porting process is discussed in Section 3.3, and begins with a description of the rationale behind the port. Following this, the initial plan for the port is presented, outlining the architecture, operating system, environment, version control, and Qt framework. Section 3.4 describes the planned architectural changes, and Section 3.5 discusses how the software is released and peer-reviewed.

Section 3.6 discusses the challenges encountered during porting, and Section 3.7 describes caveats of the porting process.

Finally, the chapter concludes in Section 3.8 with a summary of the major points, emphasizing why the port was done and its value.

3.2 The Original Version

InetVis was designed and implemented by van Riel (2005) towards his Bachelor of Science with Honours in Computer Science degree. At the time the original research was undertaken, the *Spinning Cube of Doom* had just been released by Lau, but the source code was not made public. InetVis was written to provide a quick and easy way for researchers to visualise and analyse large packet captures in a short period of time. InetVis improves on the *Spinning Cube of Doom* further by enhancing its visualisation capabilities and making it more useful to network telescope researchers (van Riel and Irwin, 2006a,b, 2007; Irwin and van Riel, 2008).

The secondary goal of the original research was to determine how effective the tool is at detecting network traffic anomalies, which may or may not indicate intrusive activities. The original research was continued in a Masters research project, leading to papers being published exploring the usefulness of the tool (van Riel and Irwin, 2006a,b, 2007; Irwin and van Riel, 2008). In these papers the design of the tool, as well as evaluations of its usefulness are described. The tool proved useful at detecting intrusive activities that were undetectable by traditional signature-based intrusion detection systems.

The original implementations of both the *Spinning Cube of Potential Doom* and InetVis were constrained in their usefulness with respect to the hardware that was available at the time. In order to visualise any reasonably complex dataset a computer with a powerful processor and a discrete, high-performance, graphics card was required.

In this section an overview of the original design and implementation of the tool is described. The discussion begins with a review of the overall architecture of the application in Section 3.2.1, then continues to discuss how the application UI and source code structure are designed in Sections 3.2.2 and 3.2.3. In Section 3.2.4 a closer look is taken at the main data extraction and processing classes, and in Section 3.2.5 the OpenGL visualisation code is examined. A succinct discussion of the updates made within the scope of this research is then presented in Section 3.2.7, and a summary is given in Section 3.2.8.

3.2.1 Architecture

Secondary goals of the original research included being flexible enough to allow the support of extensions in the future, i.e. being modular in nature, and embracing the use of free and open-source software and libraries, allowing the tool to be used freely on any operating

system. In order to accomplish these goals the following design and architectural decisions were made.

The C++ programming language was chosen because of its support for object orientation, its support for third-party libraries, and due to its proven reputation for producing high performance applications (van Riel, 2005). For the graphics framework, the Qt framework was chosen as it provided one of the easiest ways to create graphical C++ applications, and due to its built-in support for the OpenGL graphics library.

The OpenGL graphics library was chosen due to its high performance 3D accelerated hardware rendering, which allowed for InetVis to be developed with interactive and responsive UI elements (van Riel, 2005). The use of this library aids in the realisation of the primary goal of this research, i.e. the development of a high performance visualisation tool.

In order to support the capturing of network traffic and to allow for straightforward processing of the pcap file format, the *LibPCap* library¹ was chosen (van Riel, 2005). This ensured that tried and tested code was being used to capture network traffic and to parse packet captures, leading to increased performance and reliability.

Given the cross-platform and open-source development philosophy of the chosen architectural components the secondary goals of this work were able to be met. This makes it possible to distribute this software freely on Linux, Windows, and macOS.

3.2.2 Application UI

The user interface (UI) of InetVis is the primary way in which the user will interact with the tool, therefore it must be easy to use and understand. In other words, it must offer a good user experience (UX). The user interface of InetVis was designed in a modular fashion in order to best make use of the screen real estate available, and to take advantage of multi-monitor configurations. The UI was split-up into logical components and use was made of appropriate visual elements in order to enhance the UX of the user. An overview of the UI elements is given in Figure 3.1 and is described further below.

When InetVis is opened the two primary widgets of the application are automatically displayed. These widgets are: 1) the visualisation pane (Figure 3.2), and 2) the control panel (Figure 3.3).

¹<http://www.tcpdump.org/>

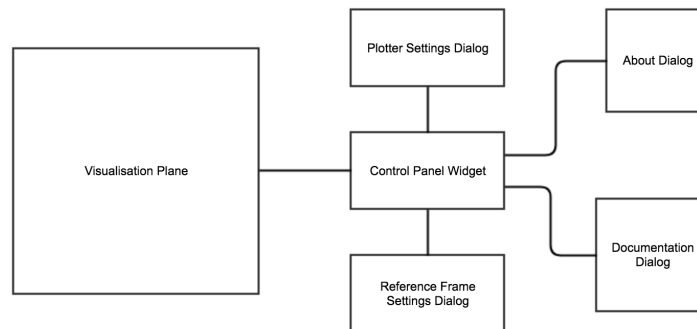


Figure 3.1: Overview of the original InetVis UI hierarchy.

The *visualisation pane* is the centre point of the application, it is where the network traffic is visualised on a 3D scatter plot. It allows the user to interact with the cube by using the mouse and the keyboard to rotate, zoom, and adjust various visual features. This allows the user to gain improved insight into the visualised data. The *visualisation pane*, or cube as it is sometimes referred to, can be seen in Figure 3.2. The pane consists of a 3D cube made up of coloured axes, with a 2D plane underneath it. The blue x -axis represents the user's home network IP range, which is referred to as the destination network when considering the packets that are visualised. The green y -axis represents the destination port range of the packet being visualised and by default covers the entire 16-bit port space. The red z -axis represents the Internet network, or more correctly the source network when referring to the visualisation of the underlying packets.

The *Data Processor* class is responsible for processing and plotting the network events on the 3D scatter plot. The majority of this logic is implemented in the *renderDataDynamic()* method. Listing 3.1 shows a snippet of the code used to draw the points in the visualisation pane when processing a packet capture or visualising live network traffic. The determination of the point's colour and coordinates is done at processing time, the points are then added into a queue from which the network events visualised.

Source Code Listing 3.1: *DataProcessor::renderDataDynamic()* event plotting logic.

```

3404 //set points colour
3405 glColor4fv((GLfloat*)itr->point.colour);
3406 //render point
3407 glVertex3fv((GLfloat*)itr->point.coord);
  
```

A packet is thus visualised by plotting a point on the graph, with the source IP address on the z -axis, the destination IP address on the x -axis, and the destination port number on the y -axis. The plane below the cube is used to visualise ICMP traffic, such as the

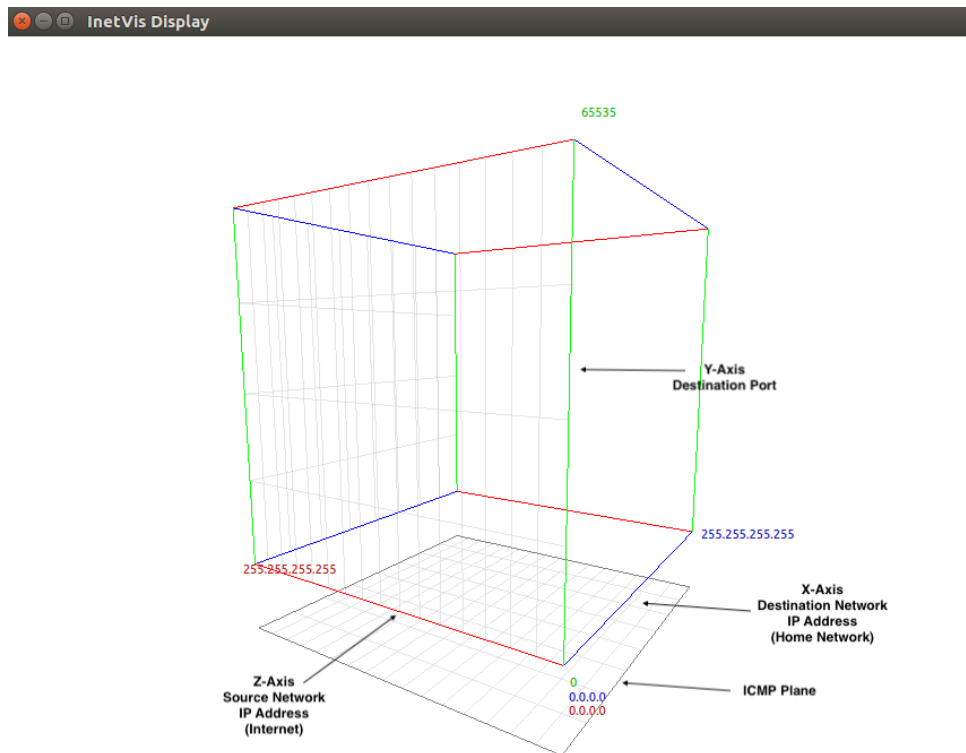


Figure 3.2: The InetVis visualisation pane.

ICMP type 8 echo request². The x and z -axes on the plane correspond respectively to the destination and source network address, as used on the cube.

The *control panel widget*, shown in Figure 3.3, provides the primary interface for the user to interact with. Internally the *DataProcessing* object is used, which is responsible for processing the incoming network packets. This class will be discussed in more detail in Section 3.2.4. By utilising the control panel controls, such as pausing playback, increasing the playback speed, and so on, the relevant functions are called within the *DataProcessor* object which in turn affects how the data is visualised.

It is also from the control panel's menu bar that the user is able to switch between live and replay mode and open other UI dialog boxes that contain further settings to customise the visualisation pane. The control panel consists of specific sections to allow: 1) the replay position of a packet capture file to be adjusted and viewed when replaying; 2) a play/pause control, a time scale factor to speed up or slow down the replay, as well as options to record a packet capture, screenshot, or frames of the visualisation; 3) The historical view group allows the time windows of how long events should be displayed on

²<http://www.networksorcery.com/enp/protocol/icmp/msg8.htm>

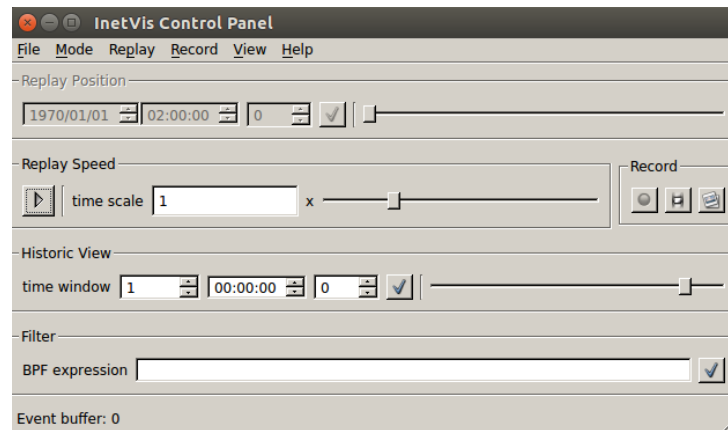


Figure 3.3: The InetVis control panel.

the pane to be adjusted; and 4) the filter section allows the user to filter the displayed events by means of a *Berkeley Packet Filter* (BPF)³ expression.

The *plotter settings dialog* allows the user to view and manipulate settings that are specific to how the visual plotter draws UI elements on the visualisation pane. This is one of the most commonly used dialogs as it allows the user to manipulate settings such as the home and Internet network ranges, the destination port range, and a few other point-specific settings.

It is common for a user to open the plotter settings dialog to optimise the home (destination) network range when first opening a packet capture or switching into monitor mode. In order to provide a good user experience, functionality is provided that allows the application to guess the most suitable home network address for any given packet capture file. By clicking the *Guess* button in this dialog box, the application will parse the loaded packet capture file and do its best to determine a suitable network address for it. It does this by considering the range of destination IP addresses found in every packet in the packet capture file. In the case of monitoring a network interface the application will use the libpcap library in order to determine the network settings of the selected network interface. This is done automatically for the user.

The original design was specifically focused on the destination network being the home network under consideration and as such the destination port number was chosen as the *y*-axis dimension. This was done as the software was originally planned to be used to visualise packet captures from network telescopes. In this use case, it made the most sense to consider the destination network to be the home network, as all incoming traffic was

³<http://biot.com/capstats/bpf.html>

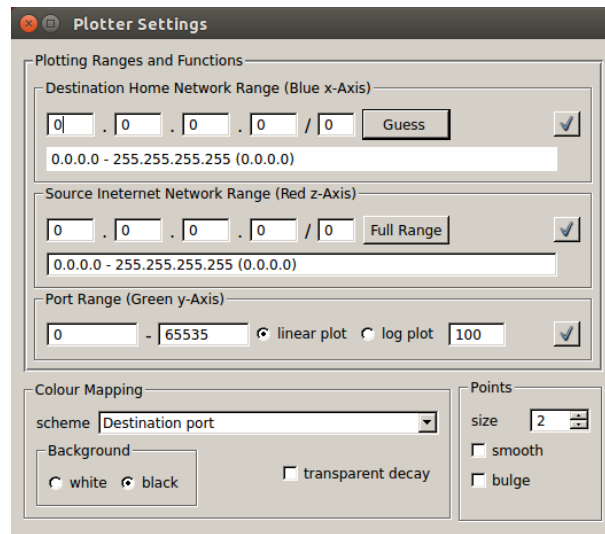


Figure 3.4: The InetVis plotter settings dialog.

coming from the Internet. In this case it did not make sense to consider the source port number, as this would not be useful. However, this short-coming does make the software less useful when visualising packet captures of network traffic exiting the destination network, i.e. when analysing malware propagation from a host. In this case, the source and destination are effectively switched around from standard InetVis usage, and as such source/destination confusion occurs, and the y -axis dimension of destination port proves less than useful. This phenomenon is described in more detail in Marty (2009, p. 26).

The plotter settings dialog can be seen in Figure 3.4.

The *reference frame settings dialog* allows the user to view and manipulate options relating to the reference frame of the graph in the visualisation plane. It allows the user to modify settings that govern the look and feel of the graph and the planes themselves. Options present in this dialog include whether to use perspective or orthographic projection mode, which reference frame markers to display, and so on. The grid partitions can also be controlled and the size adjusted for each of the axes. Furthermore, text labels can be enabled or disabled for the axes, including date and time, and the current rendering frame rate. By adjusting these settings the user is able to customise the visualisation pane however they like, which will aid in the analysis of visualised network traffic. The reference frame settings dialog can be seen in Figure 3.5.

The *documentation dialog* can be accessed through the *About* section on the control panel's menu bar. This dialog simply shows the documentation for the application. The documentation is loaded from an external file, *inetvisdoc.html*, which is located within the *doc* directory under the main InetVis directory root. The application relies on this file being

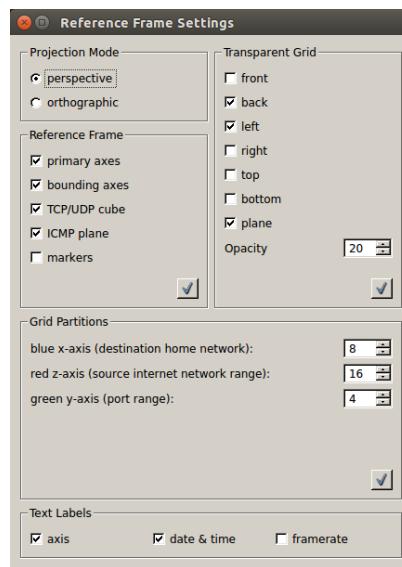


Figure 3.5: The InetVis reference frame settings dialog.

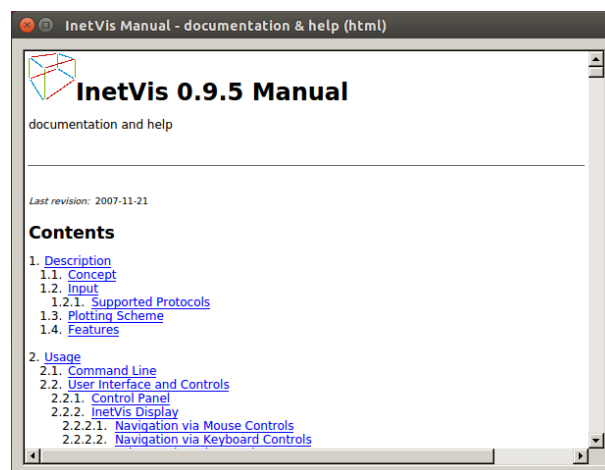


Figure 3.6: The InetVis documentation dialog.

present. If it is not present or cannot be opened an error will be displayed. Upon opening the documentation dialog the user is presented with an image such as that in Figure 3.6.

The final UI window is the *About dialog*, which is also accessible via the *Help* section of the menu bar. The function of this dialog is straightforward, as it shows the usual about information of the application. An image of this dialog can be seen in Figure 3.7. The screenshot shows the version number as *0.9.5-qt4*. This version represents the initial Qt 4 port which was done at the end of the original research. This port depended on the *Qt3Support* module which was removed in Qt 5, as is described in more detail in Section 3.2.7.

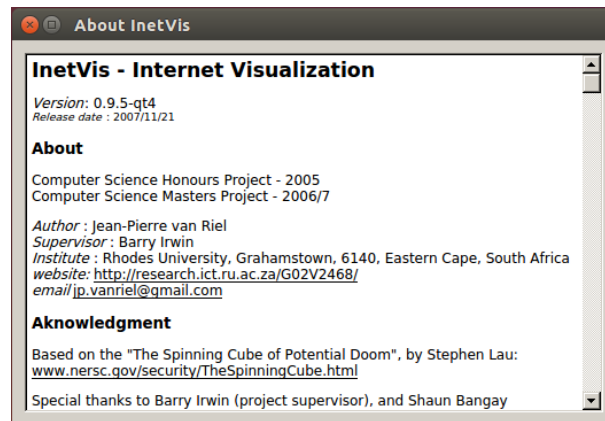


Figure 3.7: The InetVis about dialog.

Table 3.1: Overview of source code structure.

Source files	Header Files
dataproc.cpp	dataproc.h
glviswidget.cpp	glviswidget.h
graphicelement.cpp	graphicelement.h
logui.cpp	logui.h
packetevent.cpp	packetevent.h
packetheaders.cpp	packetheaders.h
plotter.cpp	plotter.h
timeutil.cpp	timeutil.h
main.cpp	

3.2.3 Source Code Structure

In this section the structure of the application will be described. Specifically, the source code, header, and other files that work in the background to process incoming packets, read packet capture files, visualise network events, and support the UI components of the application. The implementation can be divided into four areas, namely: 1) the UI forms and their implementation details, 2) the data processing class, called *DataProcessor*, 3) the OpenGL visualisation class, called *GLVisWidget*, and 4) the utility, logging and support files.

In Table 3.1 an overview of the source code files making up the application is provided, and in Table 3.2 an overview of the UI forms and header files is provided.

This section discusses the UI forms, header, and source code files, as well as the utility and other support files in use by InetVis. The other areas, namely those classes responsible for the data processing and extraction, the underlying OpenGL visualisation, and the logging

Table 3.2: Overview of UI file structure.

UI Forms	UI Header Files
controlpanel.ui	controlpanel.ui.h
helpdocumentationdialog.ui	helpdocumentationdialog.ui.h
plottersettingsdialog.ui	plottersettingsdialog.ui.h
referenceframesettingsdialog.ui	referenceframesettingsdialog.ui.h
visdisplay.ui	visdisplay.ui.h
aboutdialog.ui	

framework, are described in more detail in Sections 3.2.4 to 3.2.6.

The UI forms and the framework to support them are implemented as a combination of a Qt UI form (*.ui*), and a UI header file (*.ui.h*) where complex implementation logic is defined. The UI widgets and dialogs consist of the *control panel* widget, the *visualisation pane* widget, the *plotter settings* dialog, the *reference frame settings* dialog, the *documentation* dialog, and the *about* dialog. Each of these forms exists as Qt *ui* files, and most, excluding the *about* dialog, are backed by a UI header file, which extends the *ui* files with more complex implementation logic, specifically the custom implementation of Qt slots and the declaration of emitted Qt signals. Signals and slots are simply a means for communication between objects, that is used by the Qt framework⁴.

The remaining utility and support files exist to support the main components of the application. The *GraphicElement* data structure exists to represent a graphical element for display in the visualisation pane, by the *GLVisWidget*. The *PacketEvent* and *Packet-Headers* data structures exist to support the underlying processing and visualisation of network packets. Finally, the *Plotter* class exists as a static utility class that is used in various places through the application where drawing to the UI is necessary.

The *TimeUtil* class is another static utility class that is used in order to provide time conversion and manipulation functionality. The *main.cpp* file of the project is the entry point into the application. It is responsible for setting up any logging functionality, for starting up the main Qt application instance, instantiating all of the relevant UI forms and data processing classes, and for defining the connections between the signals emitted by certain objects with the slots defined by others.

The Qt project file, *inetvis.pro*, is used in order to pull together all of the elements of the application. It specifies what configuration to use when building and running, which UI forms, header files, and source code files to include, as well as what images to import,

⁴<https://doc.qt.io/archives/3.3/signalsandslots.html>

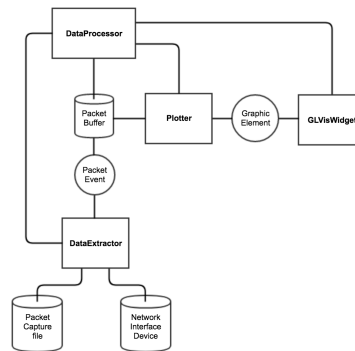


Figure 3.8: DataProcessor classes and logic flow.

and any Qt or platform specific configuration settings (The Qt Company Ltd, 2017b). It is possible to define different debug and release configuration options, as well as distinct source code includes, configuration, and behaviour for different operating systems.

3.2.4 The DataProcessor Class

The *DataProcessor* class is responsible for all of the background network packet processing that is done in InetVis. This class has two noteworthy subclasses, namely *Plotter* and *DataExtractor*. These are used in conjunction to enable the background processing work of the application. The *DataExtractor* is responsible for extracting the underlying packet data from either a local file or a network interface. This object then constructs *PacketEvent* objects and places them in a buffer which is then consumed by the *DataProcessor*. The *Plotter* class is used in order to transform a *PacketEvent* into a *GraphicElement* which is then used in plotting to the visualisation pane.

The *DataProcessor* is also responsible for signalling the *GLVisWidget* regarding what to display on the visualisation pane. The relationship between these classes can be seen in Figure 3.8.

3.2.5 The GLVisWidget Class

This class is one of the main UI components of the application. It is defined through a header and a source code file, and then injected into the underlying *VisDisplayWidget* UI widget where it becomes the logic behind the visualisation and behaviour of the visualisation pane. Thus, this class extends the *QGLWidget* class. The *GLVisWidget* class has

methods corresponding to the initialisation, resizing, and painting of graphical events. It also has support for handling user generated events such as mouse clicks and key presses.

When the user interacts with the visualisation pane directly these interactions are handled by this class. Internally the *DataProcessor* is responsible for making direct use of this widget in order to trigger the drawing of events and graphical elements on the plane. To exemplify this, when the user presses the play button in the control panel, this triggers the *DataProcessor*, which utilises the *DataExtractor* and *Plotter*, and finally the *GLVisWidget* is signalled to actually display the network events.

3.2.6 Logging Framework

A robust logging and debugging framework was present in the original version of InetVis. This provides great benefit to future researchers, developers, and power users of the tool. This framework is made up of two distinct functions, the first being the dedicated UI logging class, and the second being the logging and debug directives that are present in the header files of the application.

The UI portion of the logging framework is implemented by the *LogUI* header file and C++ source code file. This class provides functionality that allows the researcher to record aspects of the user's interaction with the application and store it in a file on the system. This proves useful to researchers and developers who are extending the software and trying to determine the source of bugs. It also provides a function which could be used to gather further data in a usability study. Various other classes within the application that are concerned with handling UI events make use of the *LogUI* class in order to log an event every time a user interaction takes place.

The debug and logging directives that exist in the header files for the more complicated classes provide the researcher with a lot of control over what information is recorded during some of the most complex tasks that the application performs. Some examples of what types of debug logging can be switched on are: debugging the data processor, debugging the data extractor, and debugging the packets that are processed.

3.2.7 Original Updates

During the three years (2005 - 2007) that InetVis was under initial development updates were made to the base software in order to enhance it. The first update worth discussing

is that of the Qt 4 support that was implemented using the *Qt3Support* module (The Qt Company Ltd, 2016c).

InetVis was originally written in Qt 3. However, by the time the development was well under way Qt 4 was released and considered mature enough for use. Given the major architectural changes that took place between Qt 3 and 4, official tools and porting guides were made available in order to help developers port their existing Qt 3 code to Qt 4. One such tool is the aptly named *qt3to4* porting tool (The Qt Company Ltd, 2016d).

Given the initial state of the InetVis source code it is apparent that the original author did make use of this tool in order to update the code to compile under Qt 4. Extensive use was made of the Qt 3 support module, *Qt3Support*, which was added to Qt 4 and allowed the use of original Qt 3 classes in Qt 4 applications. This initial work made it easy to identify which Qt 3 legacy classes were still in use upon the complete removal of this module in Qt 5. See Section 3.3 for further details.

The original code was published on the author's website⁵, with the Qt 4 work being introduced in version 0.9.5. As part of this research the source code was moved to Github, and can now be found at *yestinj/inetvis*⁶.

Another major update that was performed by the original author was that of adding support for Microsoft Windows. This involved updating the Qt project file to specify certain directives for each platform, as well implementing code modifications to adapt to the new platform. Finally, the inclusion of some standard libraries which were Windows specific was necessary. By providing support for Microsoft Windows the author ensured that InetVis could reach new users on this platform and allowed researchers who may not have had the opportunity to use the tool to do so.

As described in the manual⁷ of version 0.9.5, the Windows port was "*quick and dirty*". This version was expected to be buggy as it did not undergo proper testing. It also did not have an installer of any kind. It was also determined that the OpenGL performance under Windows was not up to par with that of the Linux version.

⁵<http://www.cs.ru.ac.za/research/g02v2468/inetvis.html>

⁶<https://github.com/yestinj/inetvis/tree/cfe7a8be34fe78b2e0f63bbb09c98cf299a69fd2>

⁷<http://www.cs.ru.ac.za/research/g02v2468/inetvis/0.9.5/doc/inetvisdoc.html#4>.

3.2.8 Summary

In this section a closer look at the design and implementation details of the original version of InetVis was taken. An introduction was given as to why the tool was originally written and its primary objectives. The tool aimed to implement and extend the *Spinning Cube of Potential Doom* in order to determine the effectiveness of this visualisation construct in detecting network traffic anomalies.

The architecture of the tool was then presented, specifically focusing on the choice of C++ as the implementation language, Qt as the UI framework of choice, and the use of third party libraries such as OpenGL and LibPcap.

The UI components of the tool were then discussed, namely the Qt UI forms implemented to create the *ControlPanelWidget*, *VisualisationPlane*, *PlotterSettingsDialog*, *ReferenceFrameSettingsDialog*, *AboutDialog*, and *DocumentationDialog*. These components were all described in detail, including how they relate to each other, and their purpose.

Next, the source code structure of the application was discussed and the major areas were defined. The UI helper classes were explained and the utility and support classes were discussed and introduced. Next, detail was provided on the *DataProcessor* and *GLVisWidget* classes which are responsible for the background processing and visualisation of packet events.

The logging framework was then discussed, the *LogUI* class, allowing the recording of all useful user interface events, and the use of debug and log definitions in the *DataProcessor* class.

A closer look was then taken at the updates that were made before initial development ceased. These updates included two important steps towards ensuring the application's usefulness and longevity going forward, namely compilation under Qt 4, and support for the Windows platform. These updates paved the way for the port to Qt 5 which is discussed in Section 3.3.

As a final note, while the original version of InetVis was complete and functional in all of the most important areas, there was still room for improvement. This reason, together with the work required to get InetVis working properly under Qt 5, form the basis for this research and the reason for starting with the port. The porting work is discussed next in Section 3.3 and the additional features and improvements made once the port was complete are discussed in Chapter 4.

3.3 The Port

The original version of InetVis proved useful to network telescope researchers at Rhodes University, as described in Section 3.2. Unfortunately, as often happens with software, the world moved forwards while the software did not. InetVis was not maintained after it was initially written and eventually it became difficult to use on modern operating systems. InetVis was originally written for Ubuntu 4.10 using the Qt 3 framework and compiled to a 32-bit binary. In order to port the application to work on modern systems the underlying Qt 3 code needs to be updated in a way that is consistent with the current version of Qt 5.

Given the ten years worth of progress since the original version was written there have been numerous hardware and software improvements. This software was written in 2005, and was written and tested on a Pentium 4 3.0Ghz 32-bit system with 1GB of RAM and a 128MB NVidia GPU. In contrast, for this research the software was run and developed within a virtual machine on an Apple Macbook Pro with an Intel Core i7 2.2Ghz processor, 16GB of DDR3 memory, and an Intel Iris Pro graphics card.

In the last ten years processors have matured with 64-bit processors and operating systems becoming ubiquitous and with multi-core hyper-threaded processors being commonplace. System memory has increased in speed and capacity to a stage where 8GB of DDR3 memory is considered standard for most users. Graphics cards have likely seen the most improvements in architecture and performance and are now routinely the most expensive component in a computer system.

The Steam hardware and software survey from November 2017⁸ highlights some of these specifications, albeit only for PC gamers using the Steam software. In this survey of Steam users it was determined that a 64-bit operating system was the most common, together with 8GB of memory, a CPU with 4-cores, and a reasonably powerful mid-range graphics card.

The Tom's Hardware article entitled "*How to Build a \$500 Gaming PC*"⁹ from May 2017 describes the most performant PC that could be built on a relatively low budget, which consisted of a modern 3.9 Ghz multi-core CPU, 8GB of DDR4 memory, and a mid-range graphics card.

⁸<http://store.steampowered.com/hwsurvey/>

⁹<http://www.tomshardware.com/reviews/how-to-build-a-500-dollar-gaming-pc,5274.html>

Table 3.3: Comparison of the major points between Qt 3, Qt 4, and Qt 5.

	Qt 3	Qt 4	Qt 4.5	Qt 5
Date	2001	2005	2009	2012
Owner	Trolltech	Nokia (2008)	Nokia	Digia (2011)
License	GPL	GPL	LGPL	LGPL
OS	Linux	Windows	Windows	Android, iOS
	macOS	Symbian	Symbian	Windows Phone

Another point in favour of performing this research, and porting InetVis to modern operating systems, is that by updating the project to compile to a 64-bit binary, it will be possible to address a much larger memory space than is currently possible. Using 32-bit pointers the application is able to address at most 4GB of memory, while by using 64-bit pointers it will be able to address up to 16EB of memory. This will allow users of the software to open ever larger packet captures. This will allow researchers and users to keep up with the ever increasing growth of the Internet, with respect to increased bandwidth capacity and data usage. This will help the tool to scale to modern network and data rates.

The improvements between Qt 3 and Qt 4, and then Qt 5, represent a dramatic change in the entire Qt ecosystem. From the license that it may be used under, to the architecture and availability of the framework. This makes the use of Qt more amenable for every day development and use. With version 3, Qt was on its way to become free and user-friendly software. Qt 3 was owned by Trolltech and was licensed under the GPL¹⁰. It supported both Linux and macOS. Eventually, when Qt 4 was released in 2005 it became free to use, with a proprietary paid for version available¹¹. After this, Qt went through many more changes, such as being bought by Nokia in 2008, which added support for the Symbian S60 platform. In 2009, Qt 4.5 was released with the source code now available under the LGPL¹², which made it easier to work with in closed applications (Riley, 2008). Later, in 2011, Qt was acquired from Nokia by Digia, who released Qt 5 near the end of 2012 with a focus on bringing the Qt framework to the Android, iOS, and Windows Phone platforms (Goddard, 2012).

Qt has gone from strength to strength gathering support for more operating systems and increasing licensing openness. By updating InetVis from the original Qt 3 to the latest version of Qt 5, it will gain these new features and improvements and pave the way for

¹⁰<https://www.gnu.org/licenses/gpl-3.0.en.html>

¹¹<https://dot.kde.org/2005/06/28/trolltech-releases-qt-40>

¹²<https://www.gnu.org/licenses/lgpl-3.0.en.html>

future InetVis development.

Given these reasons, porting InetVis to work on modern operating systems was determined to be a task worth undertaking. The port is the first step in this research and it will provide a base on which additional functionality can be implemented. The end result is a version of InetVis that has been ported to work on modern operating systems and that has been extended to become more useful to users. These enhancements are discussed in detail in Chapter 4.

Next, in Section 3.3.1 an overview of the porting methodology will be presented.

3.3.1 Methodology

In order to successfully port the original version of InetVis a solid methodology and plan-of-action was required. The initial version of InetVis was reviewed, in Section 3.2, in order to gain an understanding of its architecture. Papers by van Riel and Irwin (2006a), as well as the project's website (van Riel, 2007), proved valuable resources. The frameworks that were used will be identified, and the further research performed will be examined, as these items are required in order to complete the port.

The next step in the process involved getting InetVis functional once more. This was done in order for its existing functionality and user interface to be fully explored and understood, while at the same time identifying dependencies and limitations. This process is further described in Section 3.3.2. Upon completion of this task, the target environment, both hardware and software, were considered and decided upon. This is discussed in further detail in Section 3.3.3, and includes considerations such as the target operating system, architecture, and frameworks.

The choice of version control and the reasoning behind this choice will be discussed in Section 3.3.4.

Following this, a methodological approach will be presented in order to thoroughly explore each aspect of the application in order to complete the port. The process of porting the code from Qt 3 to Qt 5 is discussed in Sections 3.3.5 and 3.3.6.

Architectural changes not relating to the Qt graphical framework are then explored and discussed. This will consider aspects such as the goal of producing a 64-bit binary of the application. This is discussed in detail in Section 3.4. Following this, the port work will

be concluded with a summation of the work performed, the implications of the changes, and their effect on the software as a whole.

3.3.2 Laying the Foundation

The first step of the process to get a functional port of InetVis on a modern operating system is to properly understand the design of the application and to get it up and running again with minimal source code modification.

In order to get InetVis running on a reasonably modern operating system and hardware the following process was followed:

1. The best-match operating system was selected.
2. A virtual machine was created with the selected operating system.
3. The original source code and binary file were downloaded.
4. The dependencies mentioned in the user manual were installed.
5. Any missing dependencies were fetched manually and installed.
6. The binary file was run and the functionality tested.
7. The source code was compiled and the built binary was run and tested.

Based on this process, the steps that were carried out are now described in greater detail.

Ubuntu 14.04 LTS 32-bit was chosen as the operating system to start on, as it is still being updated by Canonical, but is not the latest version of Ubuntu available. It would also require the least effort to get InetVis running in this environment.

A virtual machine was created for the selected operating system. Use was made of VMware Fusion on macOS Sierra. The virtual machine was created with the default recommended settings for Ubuntu 14.04 32-bit.

The distribution package from the projects website (van Riel, 2007) was then downloaded and extracted. Following this, the original user manual was explored and the relevant build dependencies, listed in Table 3.4, were installed. The latest versions provided by the Ubuntu repository were installed.

Table 3.4: Original version Ubuntu package dependencies.

Package	Version
build-essential	11.6ubuntu6
g++	4:4.8.2-1ubuntu6
libc6	2.19-0ubuntu6.13
libc6-dev	2.19-0ubuntu6.13
libstdc++6	4.8.4-2ubuntu1 14.04.3
make	3.81-8.2ubuntu3
libpcap-dev	1.5.3-2
libgl1-mesa-dev	10.1.3-0ubuntu0.6
libqt4-dev	4:4.8.5+git192-g085f851+dfsg-2ubuntu4.1
qt4-dev-tools	4:4.8.5+git192-g085f851+dfsg-2ubuntu4.1
libqt4-qt3support	4:4.8.5+git192-g085f851+dfsg-2ubuntu4.1

It was now possible to open InetVis successfully and to receive a functional user interface. It was possible to open a packet capture file, to replay it, and adjust various options such as the playback speed and cube parameters. Monitoring the local network interface for incoming packets required the binary to be run as the root and to allow the virtual machine instance the permission to monitor the network device it was attached to.

The final step in this phase was to get the InetVis source code compiling. A few minor issues were encountered at first, which needed to be fixed before the compilation succeeded. The first of these was that there was a missing include statement for *stdio.h* in *packetheaders.h*. The second was the selection of the version of Qt to use for compilation, which could be defined either by a command line argument, or through an environment variable. The environment variable method was chosen and the command can be seen in Listing 3.2.

Source Code Listing 3.2: Exporting QT environmental variable.

```
export QT_SELECT=qt4
```

Finally, there were compiler warnings when compiling InetVis and some initial work was done to remove the most serious of the warnings. A few signed/unsigned integer declarations were improved and empty if-statements and other unused code commented out or removed completely. It was now possible to open the newly compiled version of InetVis and to perform the same functions and tests as before. These changes were committed to the Github repository, as described in Section 3.3.4, and they constituted the first release of the project, version 0.9.5.2¹³, under this research.

¹³<https://github.com/yestinj/inetvis/releases/tag/v0.9.5.2>

3.3.3 Target Environment

The choice of target environment for the port was important because it has an effect on how future-proof the application would be once ported. It is important to optimise the choice of target environment in order to ensure that InetVis will remain usable for the foreseeable future.

The first aspect of the target environment to consider is that of the underlying operating system. Given that InetVis was initially developed on Ubuntu and that the operating system chosen to get the software up-and-running again was also Ubuntu, it was deemed appropriate to select the most recent LTS release of Ubuntu¹⁴ to ensure compatibility and support. At the time of this research the most recent LTS version of Ubuntu was version 16.04.

The choice of target architecture is a straightforward one. While InetVis was originally written and compiled to a 32-bit binary, it is appropriate to target a 64-bit environment. Compiling InetVis to a 64-bit binary increases compatibility with modern systems and further future-proofs InetVis. Given this, the 64-bit version of Ubuntu 16.04 LTS is the target operating system for the port.

The bulk of the development and testing for this port was done in an Ubuntu 16.04 64-bit LTS VMware virtual machine running on macOS. This was done for practicality as the researcher's primary workstation was a Macbook Pro running macOS. Further testing and development was done on bare metal systems by other interested parties. This was done to ensure that any specific quirks or bugs were caught and fixed in a timely manner. The robustness of the software was improved by doing this, as there is no substitute for testing software on different hardware and operating systems.

As with the original version of InetVis, software development was performed using the *QtCreator*¹⁵ development environment.

3.3.4 Version Control

Version Control Systems (VCS) have existed for many years and in recent years they have become increasingly important for the implementation and management of complex

¹⁴<http://releases.ubuntu.com/16.04/>

¹⁵https://wiki.qt.io/Qt_Creator

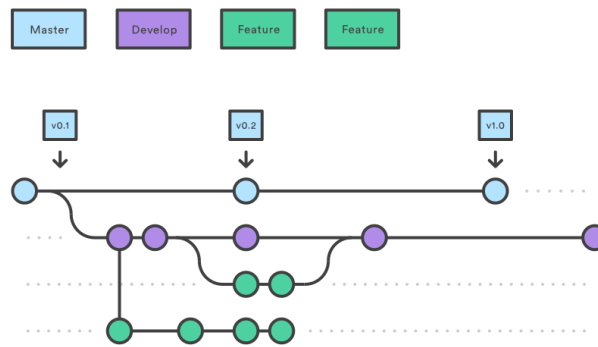


Figure 3.9: Feature branch workflow.

software applications. To this end, it was decided that InetVis should be put under a version control system for better management and maintainability. Git¹⁶ was chosen for this purpose due to its popularity and ease-of-use.

The feature branch workflow¹⁷ was used together with Git in order to properly manage the project. This workflow consists of one master, one develop, and numerous feature branches. The master branch should always be production-ready, the develop branch should contain development code which may not be ready for production, and all feature work should be done in dedicated feature branches. Feature branches are branched off of develop and merged back when the work is complete. When releases are made develop will be merged into master and a tag created to track the version number of the release. An overview of this process can be seen in Figure 3.9¹⁸.

This workflow lends itself to distributed development teams where multiple contributors may be working on the same functionality at the same time. While this was not the case during this research, it is conceivable that this could be the case in the future. For the extra effort up-front it was possible to gain much greater flexibility in the future and to promote maintainability.

3.3.5 Qt 3 to Qt 4

The most complex and time-consuming aspect of the port was the update from Qt 3 to Qt 4. The GUI of InetVis was written in Qt 3.3 which was released in 2004. It was later updated to Qt 4, which was released in 2005¹⁹.

¹⁶<https://git-scm.com/>

¹⁷<https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>

¹⁸<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

¹⁹https://wiki.qt.io/Qt_History

Many architectural changes took place in the Qt framework between versions 3 and 4, all aimed at making it easier to use and more robust, as described in the *Porting to Qt 4* guide by The Qt Company Ltd (2016a). Unfortunately, this meant that a large part of the framework changed between these versions and that updating to version 4 required a substantial amount of work. Fortunately, given the popularity of Qt and the time since Qt 4 was released there were many official references online detailing how to port Qt 3 code to Qt 4. The aforementioned guide by The Qt Company Ltd (2016a) describes the process of porting from Qt 3 to 4 in great detail and links to many other useful guides which concern themselves with specific parts of the process.

In order to ensure a successful port the basics of Qt 3 were researched in order to properly understand the implementation of the existing application. C++ Programming with Qt 3, by Blanchette and Summerfield (2004), was used as a reference and starting point for this research. Once a good understanding was gained the Qt 3 to 4 porting documents were reviewed in order to determine what to expect. Based on this review it was possible to break the porting process up into the following steps:

1. Reviewing the *qt3to4* tool (The Qt Company Ltd, 2016d).
2. Removal of the *qt3support* module (The Qt Company Ltd, 2016c).
3. Porting the UI files to the Qt 4 format (The Qt Company Ltd, 2016b).
4. Fixing any errors that came up.
5. Reviewing the source code for instances of *Q3* or *Qt3* strings to be updated.
6. Manually moving *.ui.h* file logic into C++ wrapper files.

Each of these steps will now be described in greater detail.

The *qt3to4* tool: This tool is provided by the Qt Foundation to assist developers in moving their projects from Qt 3 to Qt 4. As much of the process as possible has been automated by the tool. After further investigation it was determined that this tool operates mainly by renaming classes that are now part of the Qt 3 support library.

Removing Qt3Support: This library was aimed at easing the transition between Qt 3 and Qt 4 projects. However, since its removal from Qt 5 it will also have to be removed from the project. This was one of the first steps taken in modifying the code for the

Qt 4 port. It involved removing the *QT += qt3support* statement from the application's project file. Upon building the project after this change there were numerous compile time errors. For example, all of the *Q3* prefixed classes no longer existed. The most prevalent of the errors encountered and how they were resolved is described in the following sections.

Porting the UI files: In order to port the UI files use was made of the *wic3* tool. This tool allowed the generation of header and source code files from the original Qt 3 UI file, in order to allow implementation of any custom signals and slots. It also allowed the generation of a Qt 4 style UI file. These two functions were used as a starting point in the process of converting the forms to Qt 4 style and in order to maintain functionality within the application itself.

Each UI form went through the following process:

1. Generate the headers and source code files: *wic3 [formname].ui > [formname].h*, and *wic3 -impl [formname].h [<formname].ui > [formname].cpp*.
2. Generate the Qt 4 style UI file: *wic3 -convert [formname_qt3].ui > [formname_qt4].ui*.
3. Removal of the older Qt 3 UI file from the *FORMS3* entry in the project file.
4. Addition of the newer Qt 4 style UI file to the *FORMS* definition in the project file.

The challenges encountered in the porting process are discussed in Section 3.6.

During the conversion process the *wic3* tool would output warning messages when there were logic or objects that could not be automatically ported to Qt 4. This occurred with the more complex forms such as the control panel. The simpler UI forms converted without issue, although a few did have additional logic in *.ui.h* files which would need to be ported later on. The majority of the warnings produced were due to object attributes no longer being available in Qt 4.

Another important change to consider and account for is how the Qt 4 version of the *wic* tool (used to turn *.ui* files in to *.cpp* and *.h* files at compile time), produces a different output structure than the Qt 3 version of this command. In Qt 3, the tool would produce a header file and an implementation class which inherited from the relevant Qt widget. This allowed the developer to use these generated files in their code, creating instances of them, and to work directly with UI files as they would with traditional, C++ source code files.

The Qt 4 version of *uic* uses an intermediary class in the header file definition and uses in-line functions to remove the need for a separate C++ source code file. Furthermore, it does not subclass any of the Qt widgets, or even the base *QObject*. These changes necessitated additional work to be done in order to keep as much of the implementation logic of the application the same as possible.

The second of the two methods presented in the Qt 4 designer porting documentation (The Qt Company Ltd, 2016b) was chosen in order to accomplish this task, namely a new class (with a header and an implementation file) was introduced for every UI form. Each of these would inherit from the parent UI file in the *Ui* namespace, i.e. *Ui::ControlPanelWidget*, as well as subclassing the base *QWidget* class or one of its subclasses. This produced a wrapper class with the same interface as that of the files generated by the Qt 3 version of *uic*. This solution made for a drop-in replacement of the older style classes.

For each UI form the following procedure was followed:

1. Create the header file extending both the relevant *QWidget* and the UI form.
2. Create the corresponding implementation file, calling the *setupUi()* method of the UI form in the constructor.
3. Update the main source code file to import and reference the new wrapper class, instead of the actual UI file.
4. Search the code base for any other instances of direct UI file use and replace them with wrapper class use.

At this stage the majority of the UI form porting work had been completed, at least, enough so that the application was able to locate and refer to all of the UI forms, but with errors due to removed or missing functionality in the UI wrapper classes.

Fixing compile time errors: Upon compilation of the project under Qt 4 and with all of the UI files converted to the Qt 4 style, and wrapper classes for them implemented, there were still many warnings and errors. This is due to the fact that while the UI code had been ported over the *uic3* tool was not able to port everything, i.e. where functionality was completely removed or method interfaces changed. This step explores the common issues that were encountered and fixed.

First, the warnings produced by the *uic3* tool were examined and remedied. These consisted of member functions or objects that had no analogue in Qt 4, such as *BelowIcon*

in *QToolButton*, *lineWidth* in *QLineEdit*, and *setName* in *QAction*, and *QActionGroup*. These instances were removed, where no obvious analogue existed in Qt 4, and where functionality was not changed to a significant degree. When this occurred a note was made to revisit it in the future if necessary. Other warnings occurred due to changes in method signatures. These could usually be fixed by updating the method call to provide the new arguments that were required.

Once these warnings were fixed the actual compilation errors were considered. One of the more common errors encountered was the use of a *QString* in a *sprintf()* call, where the behaviour had changed, and the *QString* now had to first be converted to an acceptable standard string type. This was done using the *toAscii()* method as can be seen in Listing 3.3²⁰. Another error which was encountered in the *GLVisWidget* class is that of the handling of keyboard and mouse events as the *state()* method had been removed from the *QEvent* class. The newer *buttons()* method was used to achieve the same functionality for the mouse events and *modifiers()* was used for the keyboard events. The majority of these changes were implemented in commit *e7208a19*²¹.

Source Code Listing 3.3: Updates to *QString* usage in *sprintf()* method calls.

```

@@ -358,9 +358,8 @@ void GLVisWidget::paintGL()
- sprintf(&frameFileName[frameDirCharIndex], QString('/', +
-     currentTime->toString("/yyyyMMdd-hhmmsszzz") + "_snap%02d.ppm"),
-     snapshotCount);
+ QString fname_part = "/" + currentTime->toString("/yyyyMMdd-hhmmsszzz") + "
+     _snap%02d.ppm";
+ sprintf(&frameFileName[frameDirCharIndex], fname_part.toAscii(), snapshotCount
+ );

```

There were numerous errors that were encountered due to changed or removed functionality in the UI. For example, setting the minimum size of a *QVBoxLayout*. This used to be done with *Minimum()* and is now done with *SetMinimum()*. There were many other related errors that were encountered, generally leading to UI file or source code updates. At times the UI files had to be edited via a text editor and not *Qt Designer/Creator* as the UI did not expose the older entries.

A few more minor changes were made throughout the application where in Qt 4 the names of methods had changed to be more verbose and useful. For example, in *QFile* the older *setName()* method was replaced with the *setFileName()* method.

Icon related errors were also encountered as the way this functionality works was changed dramatically in Qt 4. These were remedied by using the new mechanism of a Qt *resource*

²⁰<https://github.com/yestinj/inetvis/commit/f6b994>

²¹<https://github.com/yestinj/inetvis/commit/e7208a19>

Table 3.5: Signal changes between Qt 3 and 4.

Class	Old Signal	New Signal(s)
QDateTimeEdit	valueChanged	dateTimeChanged

Table 3.6: Updated function names from Qt 3 to 4.

Class	Old Function	New Function
QRadioButton	isOn()	isChecked() dateChanged timeChanged

*file*²². This file was created, the aforementioned icons added to it, and a reference to the file added to the project file under the *RESOURCES* configuration option. Finally, all references in the UI and source code were updated to refer to the new virtual path to the icons within the resource file.

A few of the signals emitted by standard Qt classes had also changed. For example, see Table 3.5²³.

Furthermore, buttons throughout the application that were modified from within the code, i.e. *QRadioButton*, had to be updated to use newer, more expressive functions. An example of this can be seen in Table 3.6.

Removal of remaining Qt 3 functionality: Once all of the other compile errors had been fixed there were still a few lingering instances of old-style Qt 3 classes remaining. The code was searched for the relevant strings and instances were removed or renamed where possible. Some examples of this can be seen in Table 3.7.

.ui.h file logic: The final part of this process consisted of ensuring that the signals and slots defined by the UI forms were actually implemented. In Qt 3 the *.ui.h* file was used to hold static class members and other implementation logic that did not fit into the main

²²<https://doc.qt.io/qt-4.8/resources.html>

²³<https://doc.qt.io/qt-4.8/qdatetimeedit.html#signals>

Table 3.7: Lingering Qt 3 support class includes and their solutions.

Class	Old Include	New Include
LogUI	q3textstream.h	QTextStream
PlotterSettingsDialog	Q3GroupBox	QGroupBox
PlotterSettingsDialog	Q3ButtonGroup	QGroupBox

UI file. This logic was not ported during the use of the *uic3* tool and thus had to be handled separately.

For every *.ui.h* file the implementation logic was replicated in the new UI wrapper classes header and source code files, and the old *.ui.h* file was removed. This ensured that when the UI form was instantiated the custom code used to initialise it was run and that the implementation details were in place for the public slots that the object exposed. In order for this to work each UI form's *.ui* file had to be updated to declare all of the objects slots. This then worked in conjunction with the slots defined in the wrapper classes header file to provide the public interface of the form.

At this stage it became apparent that not all of the implementation logic was in the *.ui.h* files and the original Qt 3 style UI files were then reviewed in a text editor. It was found that many variables were defined in the original UI files, that were not in the *.ui.h* file, and that were not converted by the *uic3* tool. Given this, all of the older UI files were reviewed and any variables, or other missing implementation logic, was added to the wrapper class's header and implementation files.

A summary of this process is described in the following list:

1. Create UI file header and implementation source code files.
2. Replicate implementation logic in existing *.ui.h* file.
3. Update the *.ui* file to list the public slots of the class, defined in the wrapper class.
4. Replicate definitions and implementation logic found in the original *.ui* file.

3.3.6 Qt 4 to Qt 5

The first step in the Qt 4 to Qt 5 porting process was to review the existing literature on the topic. There are various official and unofficial references on the Qt wiki²⁴, such as the *Transition from Qt 4.x to Qt 5 guide* (The Qt Company Ltd, 2016e), *The C++ API changes* document (The Qt Company Ltd, 2017a), the *Qt 5 porting guide* (The Qt Company Ltd, 2017c), *Porting Desktop Applications from Qt 4 to Qt 5* (Tranter, 2013), *Porting from Qt 4 to Qt 5* (Ire, 2012b), and *Automated Porting from Qt 4 to Qt 5* (Ire, 2012a).

²⁴<https://wiki.qt.io>

Once the relevant literature had been reviewed, the Ubuntu Qt selection was updated to *qt5*, and the Qt Creator build settings were updated to specify building under Qt 5. Once this was complete it was possible to compile the code under Qt 5 and proceed to examine the build failures, while looking out for specific items mentioned in the porting documentation.

The changes in the Qt framework between versions 4 and 5, described in The Qt Company Ltd (2016e), are not nearly as onerous as those between versions 3 and 4. While changes were made to improve the modularisation of the Qt framework's code base, this was done carefully, to not significantly affect Qt 4 projects when upgrading to the latest version of Qt 5. The version of Qt 5 provided by the Ubuntu package repository²⁵ was used as the target of the porting work. One of the major changes in this area that required work be performed was the renaming of all instances of the *QtGui* header include to the updated *QtWidgets* definition. Together with this, the project file was updated to include the widgets entity in the *QT* variable definition.

The next major change that was required was the deprecation of the *toAscii()* and *fromAscii()* methods, in favour of the newer *fromLatin1()* and *toLatin1()* methods. There were numerous instances of these methods throughout the code base. All such instances were updated to the new method of conversion to ASCII-style strings.

A further change that was required, and which was common throughout the code base, was the deprecation of the *QAction::activate()* signal. This signal was being used in various places in order to display a new widget, or activate functionality, when a menu item was clicked. The new way of doing this in Qt 5 is by using the *QAction::triggered()* signal instead (Qt Centre Forum, 2007b). While this reference states that this signal was actually removed with the removal of the *Qt3Support* module, it appears that it only caused compilation and functional issues when compiling under Qt 5.

Another signal that was deprecated in Qt 5 is that of *QLineEdit::lostFocus()*²⁶, which was replaced by the *QLineEdit::editingFinished()* signal (Qt Centre Forum, 2007a). This change was made where the deprecated signal was used in the code base.

As can be seen, none of these changes were significant from an architectural point of view, and they were generally accomplished through simple refactoring once the root cause of the compile-time failure had been determined.

²⁵<https://packages.ubuntu.com/xenial/qt5-default>

²⁶<http://doc.trolltech.com/4.3/qlineedit-qt3.html>

A few unexpected, new pieces of behaviour, and bugs, were picked up once the project was compiling and running successfully under Qt 5. Some of these were fixed, such as a bug with the initial size of the *GLVisWidget* window being inconsistent in Qt 5, this is tracked in Github by issue #26²⁷ and was rectified in commit *39c4253f*²⁸. Other issues were identified, however none were considered critical at the time, and these are discussed in more detail in Section 3.7.

3.4 Architectural Changes

Considering the aspect of application architecture the goal of this phase of the research discouraged any significant changes to be made. The least amount of work was to be done in order to get the application up-and-running on a modern operating system. The only changes made to the architecture of the application were those necessitated by the differences between Qt versions.

The first major architectural change is described in detail in Sections 3.3.5 and 3.6. In this section only the additional architectural changes that were required are discussed. Given the changes in Qt 4, UI elements were no longer subclasses of the *QObject* class, meaning that they could not be used in the same way that they were used in the application under Qt 3. For this reason header and source code wrapper classes were introduced for each UI form.

Another architectural change that took place during the port was the change from compilation of a 32-bit binary, to that of a 64-bit binary. In order to accomplish the goal of 64-bit compilation, the *-m32* linker and compiler flags were removed, and a few other minor changes were also made to these flags, in order to optimise the build process when compiling the application.

3.5 Release and Peer Review

A private Github repository was created in order to ensure that the port was completed to a sufficient degree before any sort of public release took place. Once the port was deemed to be in a stable and working state, i.e. when all of the major goals of the port

²⁷<https://github.com/yestinj/inetvis/issues/26>

²⁸<https://github.com/yestinj/inetvis/commit/39c4253f>

had been met, the repository was made public and version 2.0.1 was officially released. This repository is available under *yestinj/inetvis*²⁹ on Github, with v2.0.1 being available under the releases³⁰ page. This was a limited, soft release, not highly publicised, and was mainly used by a group of information security professionals interested in the functionality provided by the software.

The group of reviewers tested the release in vastly different environments, and reported bugs which were fixed in subsequent point releases of v2.0.1. These bugs were logged either through private channels, or via the use of GitHub's issue tracking system³¹.

In order to facilitate the release process and agile development three bash scripts were written. The *prepare_release.sh* script found in Appendix A served the purpose of packaging the binary file, help documentation, image and other assets, and installation and removal scripts, into a tarball for distribution. The installation and removal scripts, *install_inetvis.sh* and *uninstall_inetvis.sh* were created in order to install InetVis into the desktop menu system, and PATH, of an Ubuntu system, and to remove them again, if required. These scripts can be found in Appendix A.

When all of the development work for a release was complete, the following procedure was followed:

1. Merge the feature branch into develop.
2. Merge develop into master.
3. Fetch the latest updates to master on the development VM.
4. Run *make clean*, then *qmake*, and finally *make* to clear out all old assets, and to build a new binary file from master.
5. Run *prepare_release.sh* specifying the version number, i.e. *2.0.1* to create the release tarball.
6. Extract the release tarball that was just created and run the installation script.
7. Ensure that the script ran successfully and that the new version was installed as expected.

²⁹<https://github.com/yestinj/inetvis>

³⁰<https://github.com/yestinj/inetvis/releases>

³¹<https://github.com/yestinj/inetvis/issues>

Table 3.8: InetVis software releases.

Version	Date Released	Details
0.9.5.2	1 March 2017	Base release.
0.9.6-beta	17 April 2017	Beta release of Qt 4 port.
2.0.0-alpha	24 May 2017	Alpha release of completed port work.
2.0.1	13 June 2017	First official release of complete port.
2.0.2	14 June 2017	Bug fix release.
2.0.3	15 June 2017	Bug fix release.
2.0.4	19 June 2017	Bug fix release.
2.1.0	21 September 2017	New feature release.
2.1.1	10 October 2017	Final bug fix release.

8. Run InetVis from the path, or menu system, and test all of the basic functionality.
9. Create a new release in the Github project, specifying the version for the tag, and uploading the release tarball file.

By making use of this release system, Github, and the Github Slack plug-in, it was possible to easily create new releases which had undergone some testing, and to inform interested parties either following the project on Github, or making use of the *#inetvis* discussion channel on the *ZATech* Slack community³². This enabled peer review and discussion of all releases and of the workflow itself.

Table 3.8 provides a list and description of each of the versions of InetVis that were released as part of this research.

3.6 Challenges

One of the major challenges that was encountered during the process of porting InetVis was the initial work on porting to Qt 4, without the use of the *Qt3Support* module. This was compounded with the author's unfamiliarity with the Qt framework, and the C++ language in general. For these reasons, the learning curve on getting the port started was steep.

The author's inexperience with C++ and the Qt framework meant that the basic tasks of getting familiar with the code base, the tool chain, and development environment in-use

³²<https://zatech.github.io/site/>

proved to be time consuming. Eventually, after research and trial-and-error, all of the most important aspects of the system were understood. For example, how to configure which Qt version to build with, how to make effective use of the Qt Creator IDE, and how the options and configuration in the Qt project file affected the compilation and run-time behaviour of the application. A similar procedure had to be followed with the C++ language when performing the first porting updates. For example, the proper use of the header and source code files, and how to properly extend and initialise widget wrapper classes. Given the architectural changes that were required for the upgrade to Qt 4, extensive use of wrapper classes, and understanding of how C++ classes are defined, and how they inherit from parent UI widgets, was essential.

Given the architectural changes between Qt version 3 and 4, there was additional work and investigation that needed to be done in order to successfully update the project to compile under Qt 4 without the *Qt3Support* module. One of the primary challenges in this area was the change in UI form format from Qt 3 to Qt 4. This is outlined in detail in the Qt documentation³³. However, the main points are summarised as follows: Qt Designer changed from being an IDE, to being a robust form builder in Qt 4. This allowed the form editing capabilities to be embedded in other development environments. On top of this, Qt Designer 4 could no longer edit code, only UI forms. It was also unable to open Qt 3 UI forms due to the changes in the format of these files. Resource files and improved name-spacing were also introduced with Qt 4.

The changes to the UI file format, as well as the deprecation of *.ui.h* files proved to be the most troublesome of architectural changes to adapt to. While the *uic3*³⁴ tool was available, which generated Qt 4 header, source, and Qt 4 compatible UI files, from the original Qt 3 UI files. This tool has many shortcomings. For example, it is unable to copy across any of the implementation logic in the *.ui.h* files, and the generated *UI* files had to be inspected manually, in a code editor, making compatibility changes where necessary, as the tool was not able to translate every single Qt UI item from version 3 to 4.

In Qt 3 Designer, it was possible to add the implementation logic for an object's slots. This logic was typically implemented in the *.ui.h* files, and this behaviour was changed in Qt 4. The architecture was updated such that no implementation logic was allowed in UI files, hence it had to be implemented elsewhere. This was accomplished by first defining the slots in the UI file graphically. For example, connecting the *clicked()* action of the *Play* button in the *control panel* widget to the *play()* slot in the same widget. The

³³<https://doc.qt.io/qt-4.8/porting4-designer.html>

³⁴<https://doc.qt.io/qt-4.8/porting4-designer.html#working-with-uic3>

slot was then defined in the widget's wrapper class's header file, and the implementation details were moved over from the old *.ui.h* file, to the relevant wrapper source code file.

The following general procedure was devised and followed when it came to porting the UI files to the Qt 4 format:

1. Use *uic3* to generate the header and source code files from the old-style widget.
2. Use *uic3* to generate a UI file in the newer format for Qt 4.
3. Review the generated header and source code files manually, making any required compatibility updates.
4. Add the new UI file to the Qt project file, and remove the older UI file.
5. Open and review the generated UI file to ensure that it looks as expected.
6. Review the old-style *.ui.h* file (if one exists), and copy any implementation logic to the new header and source code files.
7. Complete steps 1 through 6 for all UI files in the project.
8. Update the *main.cpp* file, and anywhere else relevant, to refer to, import, and instantiate the new UI file wrapper classes.
9. Clean and build the project using *make clean*, *qmake* and *make*.
10. For each widget, examine and fix the errors that are generated at compile time.
11. Once all errors are fixed a resource file should be introduced, containing icons for use in the widgets.
12. Each widget then needs to be updated to reference the new resource file, and objects using a given icon updated to reference the icon's new location.

Errors: Many of the errors encountered were due to classes, methods, or fields being removed, renamed, or deprecated in Qt 4. Examining the relevant class on the Qt 4 wiki³⁵ helped a great deal in finding the cause of the error. For each object, the older Qt 3 classes and methods are listed, with their current state, as well as new methods to use in the place of the old, deprecated, ones when this is the case.

³⁵<https://doc.qt.io/qt-4.8/>

Some of the errors did not have a straightforward solution, i.e. if functionality was removed entirely, or architecture was changed in some other significant way. These cases should be dealt with individually, and in general the simplest solution should be chosen, and a note made to revisit it if functionality was changed (or removed) in any considerable way. Other common errors found were artifacts in the generated UI header files that are produced during build time. For example, errant Qt 3 classes, or methods, were still referenced in the UI file, but not in a way which was visible in the Qt Creator Designer view. To resolve these type of errors, the offending entity in the generated header file (i.e. *.ui/ui_controlpanel.h*) would need to be tracked down, then the relevant entity located in the UI file (i.e. *controlpanelwidget.ui*) using a text editor, and finally the offending entity would need to be updated to something Qt 4 compatible, if possible, or removed entirely and a note made to revisit it in the future should any problems arise.

By following this procedure it was possible to successfully update all of the UI files to make them Qt 4 compatible, and to ensure that the original functionality was maintained as much as possible. While other work was required for a successful port, this particular area was the most challenging and time-consuming due to the necessity of the *big bang* approach, i.e. all of the UI files had to be converted, updated, and the full process followed, before it would be possible to successfully compile the application. It was not possible to test the compilation part way through the work.

3.7 Caveats

The process of porting InetVis from the Qt 3 to Qt 5 went smoothly for the most part, and resulted in a working and functional application, capable of building and running on modern operating systems. However, certain bugs and unwanted behaviours were discovered, but not fixed during the process of the porting. They were deemed to be either too complex, not high enough priority, or a combination of the two.

The only caveats worth mentioning were discovered during the process of porting Qt 4 to Qt 5. While Qt 5 did bring with it improvements to the modularity of the Qt framework, the issues encountered were unexpected, and the root cause was not easily determinable.

The first caveat that was encountered was that of the *control panel* and *display window* overlapping completely on start-up, intermittently, once the project was compiling and running successfully under Qt 5. This caveat has been partially resolved by the introduction of a persistent settings file, and the addition of implementation logic to save and

restore the position and size of these two widgets upon application start-up. See Section 4.3 for more details.

The second caveat worth mentioning is that of the issue of random, intermittent, segmentation faults occurring when attempting to record a packet capture, when monitoring a local network device. This is, unfortunately, a more serious bug, and at this stage it represents a regression in the feature set of the original application.

Originally, even up to Qt 4, it was possible to make use of InetVis to monitor local network traffic, and then to record that traffic to a packet capture file. For some unexplained reason in Qt 5 sometimes when localhost is being monitored, and the record button is clicked, the application will crash with a segmentation fault (signal *SIGSEGV*) being triggered. This does not happen every time, and thus far the cause has not been determined. Time was spent attempting to debug this issue, by enabling debugging symbols, and making use of *gdb*³⁶ and *Valgrind*³⁷, without success.

Given that the original use of this software tool was to open network telescope packet captures and to be able to visualise and analyse vast amounts of data, it was decided by the researcher and research supervisor that this feature was not integral to the application, and could be disabled for now, until time was available to properly diagnose and correct the issue. There are many other network packet capturing tools available. For example, *tcpdump*³⁸, which are capable of recording the packet activity on a local network interface, which can then be replayed and visualised within InetVis.

3.8 Summary

In this chapter the design and implementation of the original version of InetVis was thoroughly explored. This was done as InetVis forms the basis for this research. This chapter then went on to explore the work that was undertaken in order to port the ten year old application to make use of the latest frameworks, and to run efficiently on modern operating systems and hardware.

The original version was designed and implemented to be modular in nature, and consisted of UI forms, data extraction and processing logic, OpenGL visualisation logic, and a robust logging, debugging, and support framework.

³⁶<https://www.gnu.org/software/gdb/>

³⁷<http://valgrind.org/>

³⁸http://www.tcpdump.org/tcpdump_man.html

Given that the last update to the original software occurred in 2007, InetVis was no longer able to run on modern operating systems without extensive work being done. In the years since the original research this tool has proved useful to network telescope researchers at Rhodes University, and even internationally, as shown by its inclusion in the *DAVIX* live data visualisation CD³⁹, curated by Marty. Given this, and the fact that it did not appear as if any other similarly useful tool existed for this purpose, this research was proposed in order to port InetVis to modern operating systems, and then to further expand on the port by making improvements and introducing new features.

By performing the port, a large part of which involved updating the Qt graphical framework from version 3 to 5, and enabling 64-bit compilation, InetVis was able to be useful to researchers once more. The newer version of Qt allows for much improved design, functionality, and flexibility of distribution.

The port itself presented many challenges that arose due to the extensive changes between Qt 3 and Qt 4, consisting of major changes to the UI forms format used, and the major re-architecture of the framework. This required extensive research and trial-and-error work to be done in order to obtain an application that compiled and ran successfully.

At the end of the porting process the application code base was made available via Github, allowing other developers and users to easily obtain, modify, and study the underlying code base. This paves the way forward as it will now be far easier for other developers to extend the application to suit their own needs. The onus no longer needs to rest with the original, or current, developer thereby breathing a new life into the tool as an open-source, collaborative project.

The next chapter, Chapter 4, goes on to discuss the new features and improvements which were built on top of the work performed in this chapter.

³⁹<http://www.secviz.org/node/89>

4.1 Introduction

In this chapter the features and improvements that were added to InetVis as part of this research are introduced and discussed. The basis for this feature work is the first release of the fully ported application, as described at the end of Chapter 3. This is version 2.0.1 of InetVis, which contains all of the work done in porting InetVis to work on modern operating systems, as discussed in Chapter 3.

The features and improvements which are discussed in this chapter help to take InetVis beyond its original feature set and target audience. They allow for users to have improved control over their user experience, as well as improving the usability of the application with various other small improvements. Furthermore, with preliminary support for macOS and 64-bit operating systems, InetVis has become a much more useful application.

In Chapter 2, a number of potential new features and improvements which could be incorporated into InetVis were identified while reviewing the background literature and related tools in this area. The selection of which features and improvements would be included in this research, and implemented in InetVis, was done so based partially on

this information, but more so on the feedback received from early alpha-testers of the software through personal communications channels, as well as potential features and improvements identified by the original author of InetVis. Early testers of the updated software helped to identify fatal bugs on different operating systems and architectures, but also provided feedback on their usage of the software and their experience with the InetVis UI.

While this research was being performed, two members of the Rhodes University Computer Science Department, namely Irwin and Herbert, forked the InetVis GitHub repository in order to contribute their own minor improvements to the application. It must be noted that the primary researcher did not have any involvement in these improvements, apart from review and inclusion in the original repository, and that they were done at the individuals sole discretion. Some of these improvements are included in this section, as they help to highlight the fact that InetVis was written and improved in such a way that other people, besides the researcher, were able to get the software up and running, and even make small, but useful, contributions to it. These improvements consist of *Harlem Shake Mode* (Section 4.6), *Visualisation Pane Autorotation* (Section 4.7), *About Dialog Modularity* (Section 4.9.7), and *Documentation Improvements* (Section 4.9.8).

While the information obtained in Chapter 2 is invaluable for the evolution of InetVis, it was decided that priority would be given to bugs, features, and improvements identified by early users of the software, as this provided a valuable insight into what users of the software would actually find useful and beneficial should they wish to continue using the software in their everyday work. The same logic was applied for certain improvements which were implemented in order to make the developer's, and future developers and researchers, lives easier while working on InetVis

This chapter is broken up into two major sections, new features are presented in Section 4.2 and improvements and bug fixes in Section 4.9. The former describes all of the major new features that were added to InetVis during this research, while the latter focuses primarily on smaller improvements and bug fixes. This chapter is concluded with a summary of the most relevant points, describing the usefulness of the new features added to InetVis.

4.2 New Features

In the following sections the conception, design, and implementation of features introduced to InetVis during this research will be described. The rationale behind each of these

features will then be discussed.

While there are varying definitions for the term “feature” in software development, as described in Classen *et al.* (2008), in this research a feature is considered a non-trivial piece of work, which introduces new functionality into the application, is user-facing, and can be regarded as a new innovation in the software.

By introducing new features into InetVis it is possible to further improve its usefulness to researchers, and to improve its feature parity with similar tools. The new features introduced during this research also lay the foundation for future improvements, and specifically attempt to enhance the original capabilities of the tool, making it useful to a wider audience of users and researchers (see Section 2.2.3).

The new features that will be discussed are presented in Sections 4.3 to 4.8.

For each of the new features that is user-facing, an explicit goal of a modernised user interface, with an improved user experience was taken into account. This can be seen in the general settings framework (Section 4.3) where a tabbed dialog box is used instead of the traditional approach of one dialog box per type of setting.

For each of the new features, an introduction to the concept is given, followed by rationale of why the feature would be useful in InetVis. Following this, a discussion on the initial design and implementation is provided. Each new feature discussion will be concluded with a statement on how the feature should prove useful to researchers and users of InetVis.

4.3 General Settings Framework

The original implementation of settings in InetVis lacked two important characteristics:

Item 1: Application level settings, allowing the user to control certain aspects of the application, not specifically related to the application’s current mode of operation or state. Settings of this nature do in fact exist in the original source code, however, they are not easily accessible to regular users. An example of one of these settings is the directories in which packet captures and screenshots are stored.

Another important example of this is the existence of various debugging flags and settings defined within the source code. The original author implemented an extensive set of debug

flags which controlled debug output and debug code. While not all of these settings may be useful to every user, having the option to enable, disable, and adjust certain debug settings while the application is running could prove useful to researchers and security analysts.

Item 2: The persistence of application level settings is a prime candidate for implementation. If the user sets the default record directory they will expect this to persist across application sessions. The same is true for debug settings, default home network range, and screenshot type and quality settings.

The combination of these two items also improves the flexibility and extendibility of the application. By introducing a general settings framework that provides this functionality, it becomes possible to quickly and easily add new application level settings. The goals of this framework are as follows:

1. Persistence of arbitrary application level settings regardless of operating system.
2. An easily extensible UI construct in which to view, modify, and save application level settings.

4.3.1 Design

The design of the general settings framework was two-fold. The first aspect to consider is that of the GUI component, i.e. how the user will view and modify the settings. While the second aspect concerns that of settings persistence, namely how the settings are stored persistently on disk.

4.3.2 GUI Component

The GUI component of this feature was designed to seamlessly fit in with the rest of the application and to make use of modern GUI programming standards. Keeping this in mind, a new QT UI form was introduced, namely *GeneralSettings*. This form extended the *QDialog*¹ QT base form. This was done in order to be consistent with the implementation of other UI objects throughout the application.

¹<http://doc.qt.io/qt-5/qdialog.html>

It was important that the UI dialogue be designed and implemented in such a way as to meet the design requirements, but still remain simple to understand, navigate, and use. This was accomplished by the use of the *QTabWidget*² element which allows the grouping of UI elements across multiple separate tabs within the same dialogue window. This UI element was used to full effect by creating tabs for each type of general setting to be supported at the time of creation, namely:

1. Recording settings (Figure 4.1a).
2. Home network settings (Figure 4.1b).
3. Log file settings (Figure 4.1c).
4. Snapshot settings (Figure 4.1d).

Each of the aforementioned general settings tabs can be seen in the relevant figures. These screenshots were taken of InetVis v2.1.1 running under macOS 10.13. With this addition it is possible for new settings to easily be added to the application with minimal additional UI work.

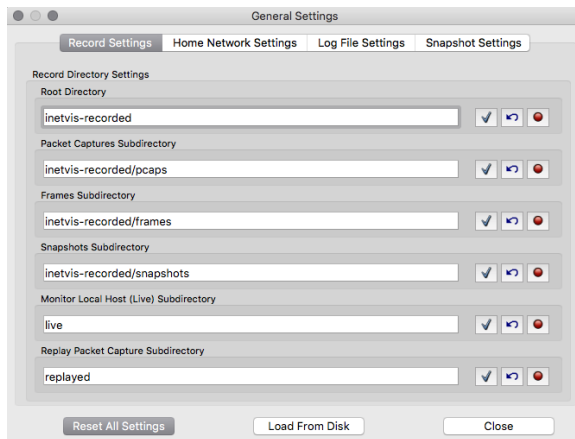
Record Settings tab: This tab contains the settings related to the directories in which assets generated using the recording functions in the application are stored. The user is also able to view the default values for each of these directories, as this was not always obvious to new users of the application.

Home Network Settings tab: This tab contains settings related to the default home network. The settings present are: *Default home network*, *Show home network not set warning*, and *Default live monitoring interface*. These settings were specifically introduced in order to improve the application due to reports from original users of the software. These settings are described in greater detail in Sections 4.5 and 4.9.6.

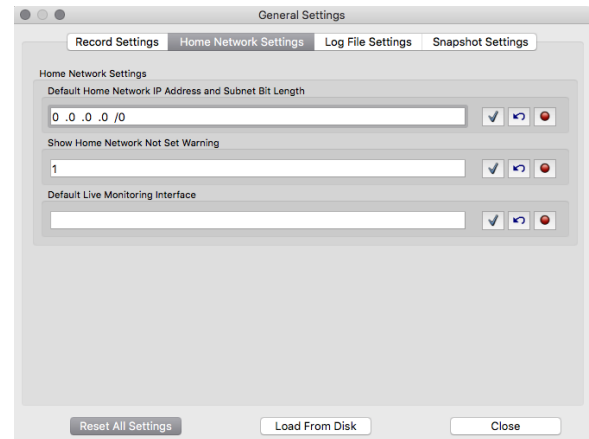
Log File Settings tab: This tab contains the settings related to the application log file directories and filenames. The user is given the option of changing the root log file directory, as well as the standard output (*stdout*), and the standard error (*stderr*)³ filenames. These settings were introduced in order to aid in the implementation of the global logging facility, described in Section 4.9.2.

²<http://doc.qt.io/qt-5/qtabwidget.html>

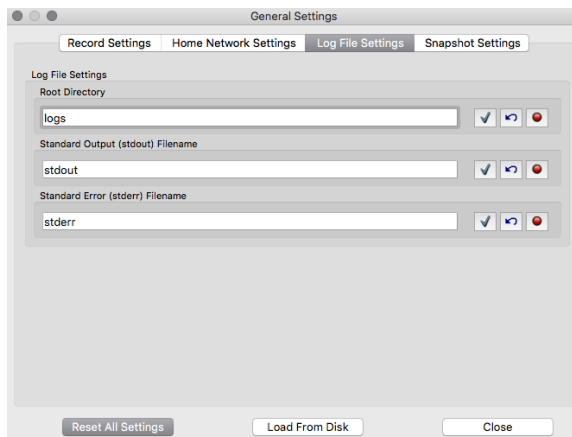
³https://en.wikipedia.org/wiki/Standard_streams



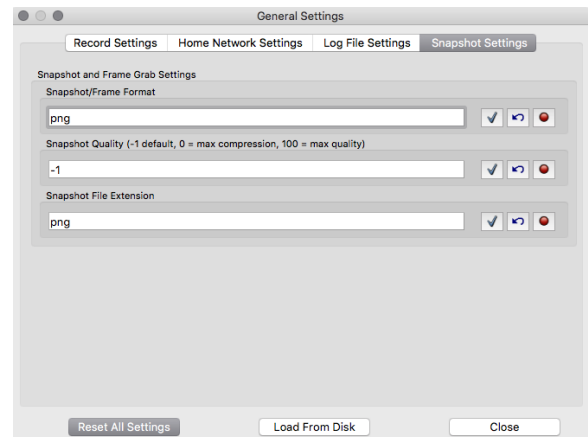
(a) The Record Settings tab.



(b) The Home Network Settings tab.



(c) The Log File Settings tab.



(d) The Snapshot Settings tab.

Figure 4.1: The updated InetVis General Settings dialog.

Snapshot Settings tab: This tab contains settings related to snapshots and frame captures. It is possible for the user to update the screenshot format, quality, and file extension used when saving screenshots or frame captures. This tab was introduced in order to improve the original inflexibility of the snapshots taken by the application, and is described in more detail in Section 4.4.

The general approach that was used to implement each of these settings is described below. This procedure also acts as a guide for the addition of new settings into the system under the general settings framework. It should be noted that the final step of this procedure, adding the implementation logic for the new slots, cannot be done before first adding the persistence functionality for the new setting, as is described in the next section.

1. Open the *GeneralSettings* UI form in Qt Creator.
2. If you are adding a new grouping of settings, add a new tab to the *QTabWidget*.
3. For each new setting add a *QGroupBox*.
4. Adjust the size of the dialog itself, or the main *QGroupBox* as necessary.
5. Add a *QLineEdit* object to the *QGroupBox*.
6. Update the fields of the new line edit field as appropriate.
7. Add *QToolButtons* for the save, load, and reset buttons.
8. Open the Signals and Slots editor.
9. Add new signals, from each of the tool buttons you added, to the main *QDialog* object.
10. Add a new slot to the *General Settings* dialog by clicking *Edit* in the *Configure Connection* window (Figure 4.2).
11. Add a slot of the appropriate name, use the existing slots as guidelines (Figure 4.3).
12. Once added connect the *clicked()* signal of the tool button to the new slot (Figure 4.4).
13. Add the new slot definitions to the general settings header file (Listing 4.1).
14. Add the slot implementations to the general settings source code file (Listing 4.1).

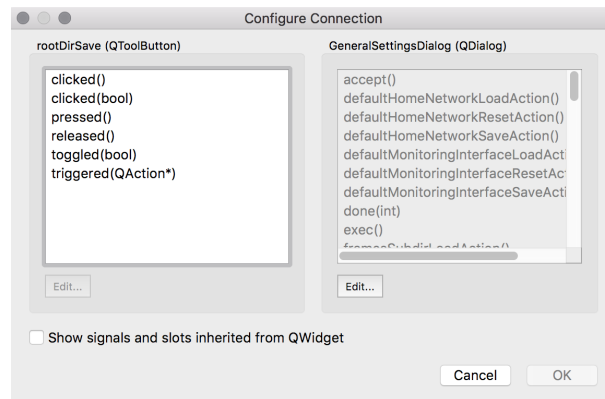


Figure 4.2: Qt Creator, Signals and Slots, Configure Connection.

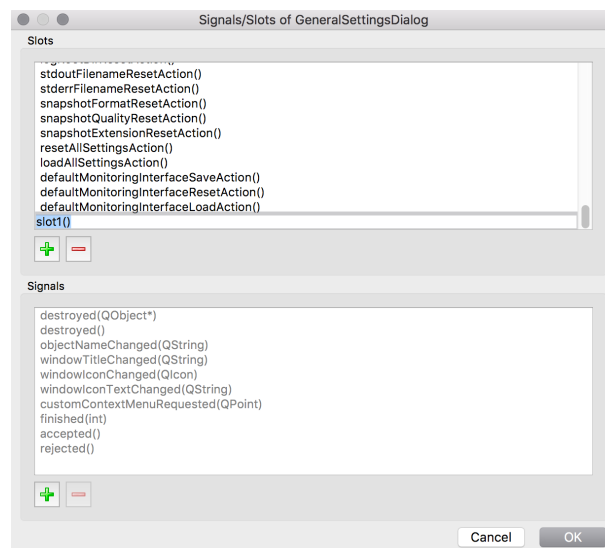


Figure 4.3: Qt Creator, Signals and Slots Editor Dialog.

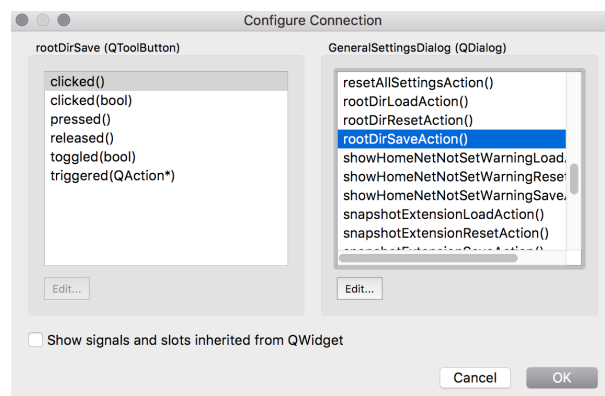


Figure 4.4: Qt Creator, Signals and Slots, Creating a Connection.

Source Code Listing 4.1: Header definition of new slots (generalsettingsdialog.h).

```

70     void logRootDirSaveAction();
71     void logRootDirLoadAction();
72     void logRootDirResetAction();

```

Source Code Listing 4.2: Source code definition of new slots (generalsettingsdialog.cpp).

```

378     void GeneralSettingsDialog::logRootDirSaveAction() {
379         if (LogUI::isEnabled()) {
380             LogUI::logEvent("[GS] Save log root directory button
pressed");
381         }
382         // Write the new value
383         QString newValue = ui->rootLogDirLineEdit->text();
384         Log::setLogRootDir(newValue);
385
386         QString loggingDefaultDir = Log::getLogRootDir();
387         ui->rootLogDirLineEdit->setText(loggingDefaultDir);
388     }

```

4.3.3 Settings Persistence

The second step in the realisation of the general settings framework is that of settings persistence. It was deemed prudent and efficient to make use of the *QSettings* class in order to implement this aspect of the feature. This Qt class is specifically designed to provide a mechanism for persistent and platform-independent application settings (The Qt Company Ltd, 2017d). *QSettings* provides a convenient key-value store which allows for the easy storage of various data types under specific keys. It internally handles where and how these settings are persisted to, and read from, disk.

In order to configure how the settings are persisted to disk use was made of a few core Qt methods to set the organisation name, organisation domain, and the application name. By setting these values upon start-up they would then be available to the *QSettings* object for use in determining the filename and location to store the settings. This method is described under the basic usage section of the *QSettings* documentation page (The Qt Company Ltd, 2017d).

Together with settings persistence comes the need for the definition of certain default values, should a settings file not yet exist. The application will need to generate and store reasonable defaults on first use. A new function was introduced in the *main.cpp* file to serve this purposes. The *initialiseQtSettings()* function was introduced, and can be seen in the *main.cpp* file, which can be found online in the Github repository⁴.

⁴<https://github.com/yestinj/inetvis/blob/master/src/main.cpp>

The next aspect to consider was where to store the settings keys that would be used when reading and writing settings, and where the support functions would be placed. It was decided that the relevant key definitions, and accessor and mutator methods would be defined in the existing classes. For example, settings relating to functionality implemented in the *DataProcessor* class would be implemented in the associated *dataproc.h* and *dataproc.cpp* files. Each setting required a settings key, a method to read the value of the setting from disk, write the value to disk, and to check whether or not the setting is currently defined.

The approach that was used in the implementation of the persistence of each setting is as follows:

1. Add a definition for the setting's key to the relevant header file (Listing 4.3).
2. Add a definition of the setting's default value to the relevant header file (Listing 4.8).
3. Add definitions for the *get*, *set*, and *isSet* methods to the header file of the relevant class (Listing 4.4).
4. Add implementations for the methods declared in the header file to the related source code file (Listing 4.5).
5. Make use of the setting retrieval to replace hard-coded values (Listing 4.6).
6. Implement the *reset*, *save*, and *load* GUI functions using the methods defined above (Listing 4.7).
7. Initialise the setting to a reasonable default when the application runs for the first time (Listing 4.9).

Source Code Listing 4.3: Definition of a settings key (dataproc.h).

```

163 // Settings key definitions
164 #define RECORD_DEFAULT_DIR_KEY "dataproc/recording/default_dir"
165 #define RECORD_PCAPS_SUBDIR_KEY "dataproc/recording/pcaps_subdir"
166 #define RECORD_FRAMES_SUBDIR_KEY "dataproc/recording/frames_subdir"
167 #define RECORD_SNAPSHOTS_SUBDIR_KEY "dataproc/recording/snapshots_subdir"
168 "
169 #define RECORD_LIVE_SUBDIR_KEY "dataproc/recording/live_subdir"
170 #define RECORD_REPLAY_SUBDIR_KEY "dataproc/recording/replay_subdir"
171 #define DEFAULT_HOME_NETWORK_KEY "dataproc/home_network/"
172 #define SHOW_HOME_NETWORK_NOT_SET_ERROR_KEY "dataproc/home_network/"
173 #define MONITOR_INTERFACE_KEY "dataproc/home_network/"
174 #define SCREENSHOT_FORMAT_KEY "dataproc/screenshot/screenshot_format"
175 #define SCREENSHOT_QUALITY_KEY "dataproc/screenshot/screenshot_quality"
176 #define SCREENSHOT_EXTENSION_KEY "dataproc/screenshot/"
177 #define SCREENSHOT_EXTENSION "screenshot_extension"

```

Source Code Listing 4.4: Standard method header file definitions (dataproc.h).

```

436 // configuration
437 static QString getRecordDir();
438 static void setRecordDir(QString recordDir);
439 static bool isRecordDirSet();

```

Source Code Listing 4.5: Standard method source file implementation (dataproc.cpp).

```

3547 QString DataProcessor::getRecordDir() {
3548     QSettings settings;
3549     return settings.value(RECORD_DEFAULT_DIR_KEY).toString();
3550 }
3551
3552 void DataProcessor::setRecordDir(QString recordDir) {
3553     QSettings settings;
3554     settings.setValue(RECORD_DEFAULT_DIR_KEY, recordDir);
3555 }
3556
3557 bool DataProcessor::isRecordDirSet() {
3558     QSettings settings;
3559     return settings.contains(RECORD_DEFAULT_DIR_KEY);
3560 }

```

Source Code Listing 4.6: Use of settings in source code (dataproc.cpp).

```

2031 //main record dir
2032 if(!dir.exists(getRecordDir())) {
2033     if (!dir.mkdir(getRecordDir())) {
2034         return false;
2035     }
2036 }

```

Source Code Listing 4.7: Implementation logic of UI methods (generalsettingsdialog.cpp).

```

82     void GeneralSettingsDialog::rootDirLoadAction() {
83         if (LogUI::isEnabled()) {
84             LogUI::logEvent("[GS]_Load_root_directory_button_pressed
85         ");
86         }
87         QString loggingDefaultDir = DataProcessor::getRecordDir();
88         ui->rootDirLineEdit->setText(loggingDefaultDir);
89     }
90
91     void GeneralSettingsDialog::rootDirSaveAction() {
92         if (LogUI::isEnabled()) {
93             LogUI::logEvent("[GS]_Save_root_directory_button_pressed
94         ");
95         }
96         // Write the new value
97         QString newValue = ui->rootDirLineEdit->text();
98         DataProcessor::setRecordDir(newValue);
99
100        // Read it out just to make sure it worked.
101        QString loggingDefaultDir = DataProcessor::getRecordDir();
102        ui->rootDirLineEdit->setText(loggingDefaultDir);
103    }

```

Source Code Listing 4.8: Default value defintion (dataproc.h).

```

177    // Default settings values
178    #define RECORD_DEFAULT_DIR_DEFAULT "inetvis-recorded"
179    #define RECORD_PCAPS_SUBDIR_DEFAULT "inetvis-recorded/pcaps"
180    #define RECORD_FRAMES_SUBDIR_DEFAULT "inetvis-recorded/frames"
181    #define RECORD_SNAPSHOTS_SUBDIR_DEFAULT "inetvis-recorded/snapshots"

```

Source Code Listing 4.9: Default value initialisation (main.cpp).

```

46    if (!DataProcessor::isRecordDirSet()) {
47        DataProcessor::setRecordDir(RECORD_DEFAULT_DIR_DEFAULT);
48    }

```

Keeping with good software development practices, two issues were created under the InetVis Github repository in order to track and record the implementation of this feature. The first issue, #54⁵, was created to cover the general settings framework, specifically the persistence aspect of it. The other issue, #62⁶, specifically covered the implementation of the GUI portion of the framework.

Two feature branches were created in order to work on this feature, which were merged back into master before the v2.1.0 milestone. Pull request #63⁷ was created in order to merge the feature/settings_file feature branch, which dealt with persistence and initialisation.

⁵<https://github.com/yestinj/inetvis/issues/54>

⁶<https://github.com/yestinj/inetvis/issues/62>

⁷<https://github.com/yestinj/inetvis/pull/63>

The second pull request, #70⁸, for the `feature/settings_ui` feature branch, dealt with the introduction of the general settings dialog. This feature branch was also used for work on the implementation of features and improvements which are discussed in the next section.

4.3.4 Discussion

During the development of this feature groundwork was laid for various other improvements. These new features and improvements are discussed in further detail in the upcoming sections, namely:

1. Improved control over screenshot format and quality (Section 4.4).
2. The ability to specify which local interface to monitor (Section 4.5).
3. Improvements in the setting of a default home network range (Section 4.9.6).
4. Specification of a log directory and filenames (Section 4.9.2).
5. Application UI position/size persistence (Section 4.9.1).

By making use of the general settings framework it was possible to implement many other features and improvements in a greatly simplified manner. Given the implementation choices this feature is robust enough to support all currently supported Qt platforms, i.e. Linux, macOS, and Microsoft Windows. The choice of persistence mechanism also allows for viewing and editing of the settings file, manually, by the user should they desire.

In the future this implementation could be improved by using input fields tailored to the setting in question, i.e. combo boxes and spin boxes. Furthermore, validation on new settings values could be added, as only rudimentary input-mask style validation is done at this time.

This feature provides a much improved user experience and customizability to the application, and lays the foundation for future development of the InetVis software.

⁸<https://github.com/yestinj/inetvis/pull/70/files>

4.4 Optimised Screenshot Format

In the original version of InetVis there are two recording options which allow the visualisation pane to be saved to disk. These two functions are frame capture and image snapshot. By utilising the output of a frame capture it is possible to produce a video file of the visualisation under investigation using a tool such as *ffmpeg*⁹.

This functionality was used by the original researcher as a way of generating high quality screenshots of the application for use in presentations and papers. It also provided the input for the creation of video files demonstrating the functionality of the tool. The original implementation of this feature was done by making use of standard C++ functionality, in order to ensure efficiency. It did not make use of any high-level functionality from the Qt framework.

The screenshots were saved in the Portable Pixel Map (PPM) format¹⁰, which is considered one of the lowest common denominator image formats, making it perfect for interoperability. However, this format is inefficient and produces large file sizes due to the fact that it does not support compression. For these reasons, it is straightforward to implement in C++, as it does not rely on any complex implementation logic.

In the decade that has passed since the original implementation of this function much has changed, and the Qt framework has evolved to include standard functionality for saving images in multiple formats using the *QImage*¹¹ class. Furthermore, the use of image file formats such as Portable Network Graphics (PNG)¹² is ubiquitous, and is more convenient to work with given the smaller file size (Iwaya, 2014).

For these reasons the implementation of PNG screenshots was chosen as one of the early features to be implemented. This feature also helps to improve the user experience as the generated screenshots and frame captures would be smaller than the older style PPM files. At a few megabytes per PPM file, the disk space requirements can quickly add up. By using the more efficient PNG format, users will more easily be able to use InetVis on virtual machines and in other disk space constrained environments.

In order to implement this feature use was made of the general settings framework in order to allow the user to specify the format in which to save screenshots, the screenshot

⁹<https://www.ffmpeg.org/>

¹⁰<http://netpbm.sourceforge.net/doc/ppm.html>

¹¹<http://doc.qt.io/qt-5/qimage.html>

¹²https://en.wikipedia.org/wiki/Portable_Network_Graphics

quality, as well as the file extension. These screenshot specific settings can be seen in the relevant tab of the general settings dialog, shown in Figure 4.1d.

At first, only the PNG screenshot option was implemented, however, support was later added for the PPM and BMP¹³ formats. This was done as it was determined that performing frame captures while using the PNG file format was significantly slower. This resulted in a slow down of the entire visualisation to less than ten frames per second, from a previous average of twenty five. The re-introduction of the PPM format, and the addition of the BMP format, gives the user of the application the choice of whether they prefer smaller screenshots with more computational complexity, or larger but more computationally efficient ones. One of the items discussed under the future work, Section 6.5, is the possibility of adding support for screenshots to be processed and written to disk by a separate thread in an asynchronous manner, so as to not affect the performance of the application.

By making use of the *QImage* class, and the general settings framework, it was possible to implement the required functionality with minimal code. This can be seen in Listing 4.10, which shows how PNG screenshots are implemented in InetVis at present.

Source Code Listing 4.10: Current implementation of screenshots (glviswidget.cpp).

```

416 // Handle each screenshot format differently as they may require some unique
    tweaking..
417 // The null check is to act as a failsafe incase config is not set for some
    reason.
418 if (screenshotFormat.toString() == "png" || screenshotFormat == "") {
419     // Read the GL screen pixels into the captureBuffer
420     glReadPixels (0, 0, width, height, GL_RGBA_EXT, GL_UNSIGNED_BYTE,
captureBuffer);
421
422     // Create and write out a PNG image screenshot.
423     QImage image((const unsigned char*)captureBuffer, width, height, QImage::
Format_RGBA32);
424     // For some reason it's flipped, need to fix that here before outputting.
425     image = image.mirrored();
426     int screenshotQuality = DataProcessor::getScreenshotQuality();
427     bool success = image.save(frameFileName, "PNG", screenshotQuality);
428
429     if (!success) {
430         Log::logError(QString("ERROR: Failed to output frame capture").
append(frameFileName));
431     }

```

Two issues were created under the Github repository in order to track the work on this feature, they were divided into: 1) implementing the PNG screenshot functionality (#49¹⁴), and 2) adding persistence and UI elements to allow adjusting snapshot settings in the UI (#61¹⁵). These issues were implemented in the feature/png_screenshot¹⁶ and fea-

¹³https://en.wikipedia.org/wiki/BMP_file_format

¹⁴<https://github.com/yestinj/inetvis/issues/49>

¹⁵<https://github.com/yestinj/inetvis/issues/61>

¹⁶<https://github.com/yestinj/inetvis/pull/55>

ture/snapshot_settings¹⁷ feature branches, and merged into main via pull requests. The re-addition of the PPM screenshot format, and the introduction of the BMP format, was added into the code later, via pull request #83¹⁸.

4.5 Specify Local Network Interface

One of the observations made by the researcher upon beginning this research was that there was no option to specify which local network interface to monitor when switching the application into monitor local host mode. It was not obvious which network interface was selected in the case of multiple network interfaces being present on the system. This observation was echoed by early testers of the port of the application, and issue #31¹⁹ was created in Github.

The original implementation of the monitor local host mode makes use of the *libpcap* library in order to perform this function. The crux of this implementation rests on the *pcap_lookupdev()* function which is described in detail in the official man page²⁰. This function is designed to find the default device on which to capture, and is in fact deprecated. The newer *pcap_findalldevs* function should be used, which returns a list of all available devices, where the first item in the list is the default device.

Given the uncertainty by users and researchers concerning which network interface is used for monitoring, the addition of a setting to allow the user to select which interface to monitor was chosen for implementation.

By examining the existing code flow, from when a user engages this mode, it was possible to determine that there was already support for defining a network interface to monitor within the source code. When this mode is activated a signal is emitted from the control panel, specifying the selection of a network interface. This signal is then consumed by the data processor, where it is actioned appropriately based on the network interface string, *netInterface*, passed in by the signal.

Use was made of the general settings framework in order to implement this functionality. A new element was added to the home network settings tab, named *Default Live Monitoring Interface*, which allows the user to enter the name of a specific network interface to

¹⁷<https://github.com/yestinj/inetvis/pull/66>

¹⁸<https://github.com/yestinj/inetvis/pull/83>

¹⁹<https://github.com/yestinj/inetvis/issues/31>


²⁰http://www.tcpdump.org/manpages/pcap_lookupdev.3pcap.html


monitor, and have that specific device used in future by the application when entering this mode. This field can be seen in Figure 4.1b.

The majority of the source code that was modified to implement this feature was the generic general settings framework implementation, except for the actual use of the user defined interface when switching to monitor local host mode. The introduction of this feature was done as part of the general settings framework implementation, using the `feature/settings_ui` feature branch and pull request #70²¹ (as described in Section 4.3).

The introduction of this feature represents a small, but important, step forward in the development of InetVis. In order to make the application appeal to a wider audience and range of use cases, it is important that monitor local host mode be improved. However, the implementation of this feature could still be improved further. A combo box could be used in place of the *QLineEdit* object in the UI, and be populated with valid choices for which network interface to set. An alternative option exists, where the user can be prompted to select an interface to monitor when they attempt to activate this mode. This option is consistent with the user experience of similar applications.

4.6 Harlem Shake Mode

Harlem shake mode () is one of two new features added to InetVis entirely by the efforts of Herbert²² and Irwin²³. This feature causes the cube in the visualisation pane to jitter, in a motion reminiscent of the Harlem Shake Internet meme²⁴.

While introducing a bit of fun to the application, this feature represents the first entirely automated movement mode supported by InetVis. A user of the application may spend many hours watching network traffic be visualised, the introduction of this mode helps to keep things interesting, and offers the user a new perspective on the data. This mode can be activated by using the  keyboard shortcut.

²¹<https://github.com/yestinj/inetvis/pull/70/commits/5c10c5>

²²<https://www.ru.ac.za/computerscience/people/academicstaff/mralanherbert/>

²³<https://www.ru.ac.za/computerscience/people/academicstaff/profbarryirwin/>

²⁴[https://en.wikipedia.org/wiki/Harlem_Shake_\(meme\)](https://en.wikipedia.org/wiki/Harlem_Shake_(meme))

Table 4.1: Cube autorotation keyboard shortcuts.

Key	Description
①	Toggle cube autorotation.
②	Decrease rotation speed.
③	Increase rotation speed.

4.7 Visualisation Pane Autorotation

The visualisation pane cube autorotation feature is the second of two new features contributed to the research entirely by Herbert and Irwin. This feature provides a new mode to the application that, when enabled, will cause the cube to rotate around the x -axis automatically. The user is given the option of enabling or disabling autorotation, and speeding up and slowing down the speed of rotation by means of keyboard shortcuts. See Table 4.1 for a listing of these shortcuts.



This feature allows the user to gain a new perspective on the data being visualized. The constant rotation of the cube provides the user with an easy way of viewing the visualisation from different angles without having to rotate the cube manually.


4.8 New Keyboard Shortcuts


The original version of InetVis made good use keyboard shortcuts in its initial implementation. These shortcuts enabled the user to perform actions such as rotating the cube, zooming in/out, toggling fullscreen mode, opening each of the settings dialogs, and hiding or showing the home network axis labels. Power users often prefer to interact with software applications using the keyboard, and not rely on the mouse. Users are able to perform most of the useful functions one would expect in order to interact with the visualisation cube.


During the work done to port InetVis, the researcher, and early testers, noticed that a few useful functions of the application were not available as keyboard shortcuts. A list of the most useful keyboard shortcuts was compiled and implemented. These new keyboard shortcuts allow the user to have greater control over the application without having to use the mouse. A list of the keyboard shortcuts now supported by InetVis can be seen in the About dialog box, in Appendix B, Figure B.1b







A few of the new, and more interesting, keyboard shortcuts and their associated functionality will now be discussed in more detail:

Screenshot (): One of the earliest new keyboard shortcuts was introduced by Irwin in order to extend the usefulness of the screenshot recording functionality. A new keyboard shortcut, the letter  , was added to the visualisation pane that would allow the user to take a screenshot simply by pressing this key. This shortcut was implemented in pull request #73²⁵.

Toggle play/pause (): The option to toggle the playback state of a packet capture file in replay mode by using the spacebar was introduced in commit *1658be2*²⁶. This shortcut allows the user to remain focused on the visualisation pane, and to quickly pause and resume playback whenever they like. This keyboard shortcut was implemented in feature branch *feature/improved_keyboard_shortcuts*, and merged into the code base as part of pull request #76²⁷.

Reset Display (): Another option that was added as a keyboard shortcut is that of resetting the display cube. When the keyboard shortcut key, zero (0), is pressed on the visualisation pane, all of the events currently on screen will be cleared, and only new events will be plotted. This proves particularly useful when analysing large network packet captures. This shortcut was implemented as part of the bulk of the keyboard shortcut work, in feature branch *feature/improved_keyboard_shortcuts*, in commit *1658be2*.

Toggle FPS (): The addition of the option to display the FPS counter in the lower right of the visualisation pane, as a keyboard shortcut helps to improve the user experience. By pressing nine (9) on the visualisation pane, it is now possible to toggle the display of the FPS text on the pane. This shortcut was implemented in the *feature/fps_shortcut* feature branch, and was merged in in pull request #84²⁸.

Quit application ( + ): Closing the control panel, or the visualisation pane, alone will not close the entire application, and each window must be closed before InetVis will exit. This problem has been solved by the introduction of the  +  shortcut key (or  +  on macOS). This shortcut functions on the visualisation pane, as this is the most likely origin of a need to quit the application. The application itself can function without the control panel, or any other window being open. This was implemented as part





²⁵<https://github.com/yestinj/inetvis/pull/73>

²⁶<https://github.com/yestinj/inetvis/pull/76/commits/1658be2>

²⁷<https://github.com/yestinj/inetvis/pull/76>

²⁸<https://github.com/yestinj/inetvis/pull/84>

of the feature/improved_keyboard_shortcuts feature branch, merged in, in pull request #76, and specifically implemented in commit *97cfdff*²⁹. A new signal was added which is emitted when the shortcut keys are pressed, and connections added to connect this signal to the *reject()* and *close()* slots of the main application widgets and dialogs.

Additional new keyboard shortcuts were specifically added for Harlem Shake mode (), and the cube autorotation mode ( ,  , ), and are described in Sections 4.6 and 4.7 respectively.

While many useful keyboard shortcuts have been added to the application, there still exist more which could be added to improve the user experience of the application. The goal is to be able to operate the application fully from the keyboard, without having to use the mouse. Notable missing functionality in this regard includes moving back and forth through an open packet capture file by seeking, jumping to a certain position, and adjusting the replay speed and historic view parameters of the visualisation.

4.9 Improvements and Bug Fixes

This section specifically discusses the improvements and bug fixes that were introduced to InetVis as part of this research.

A primary focus of many of these improvements is to improve and modernise the GUI layout, implementation, and associated functionality of the application. The original GUI is dated by today's standards and could certainly do with rejuvenation and improvement. The notable improvements in this area consist of the overall improvement in the application UI with size and position persistence (Section 4.9.1), support for HighDPI displays, such as on macOS (Section 4.9.4), direct support for opening *.pcap* files from the UI (Section 4.9.5), improvements in setting the default home network address (Section 4.9.6), and the modularity introduced to the about dialog (Section 4.9.7).

Other improvements have also been made, such as the capability to build and run the application under macOS (Section 4.9.3). This helps a great deal with rapid development and use of the application on Apple operating systems. The introduction of a logging framework catering for standard output and standard error, makes a start at unifying the disparate logging that takes place in various locations within the application (Section 4.9.2).

²⁹<https://github.com/yestinj/inetvis/pull/76/commits/97cfdff>

Finally, a few smaller improvements are made to improve the overall spelling, grammar, and English language use within InetVis (Section 4.9.9), and improving the format and content of the documentation (Section 4.9.8).

4.9.1 Application UI and Position/Size Persistence

In the original version of InetVis the application opened both the control panel and the visualisation pane on start-up. These windows were positioned next to each other on the user's screen. Unfortunately, this behaviour was not consistent, and on occasion the control panel would be positioned on top of the visualisation pane. This was described in Section 3.7 as a caveat to the original work on porting the application to Qt 5. With the upgrade Qt 5 this behaviour became more frequent.

As part of the introduction to the usage of the *QSettings* class, an example is given on how to persist application window size and position (The Qt Company Ltd, 2017d). This example turned out to be the perfect solution to this problem, as it dealt with the initial setting of the size and position of an application window, as well as allowing the user to customise the layout to suit their needs, and have their preference persist across application sessions.

This improvement was tracked in issue #59³⁰, and implemented in commits *f86d285c*³¹ and *1dea27a3*³². When the control panel is closed, the settings are now saved using *QSettings* and the previous position and size are read and applied when the respective windows are opened for the first time on application launch.

4.9.2 Global Standard Out and Standard Error Log Files

The original version of InetVis makes use of various logging constructs such as the UI log functionality and the general debug logging scattered throughout the application. The large number of debug flags, and the debug code placed in all core functions throughout the application go a long way in assisting future developers in their attempts to improve the application.

³⁰<https://github.com/yestinj/inetvis/issues/59>

³¹<https://github.com/yestinj/inetvis/commit/f86d28>

³²<https://github.com/yestinj/inetvis/commit/1dea27>

The main issue encountered with the implementation of this debugging code is that all output is written to either standard output or standard error. The use of these output streams is common in applications, especially when an application is still in active development. However, these logs are not persistent. Should specific logs be needed in the future this would have to be anticipated in advance. Furthermore, this method of logging only works if the application is run from a console. If the application is run from a GUI context, such as through Ubuntu's menu launcher, then no logs are written. For these reasons the introduction of an improved logging framework was chosen for implementation.

Issue #50³³ was created in order to track this improvement, which was then implemented in feature branch `feature/application_logfile`, and merged in, in pull request #65³⁴. A new class was introduced, *Log*, which consisted of a header file and a source code file. The usage of this class's public API can be seen in Listing 4.11, where use is made of the new logging framework in order to output debug information.

Source Code Listing 4.11: Snippet of use of public Log API (`glviswidget.cpp`).

```
429     if (!success) {  
430         Log::logError(QString("ERROR: Failed to output frame capture")).  
        append(frameFileName);  
431     }
```

Users of the application are now able to specify the logging root directory, the standard output and standard error filenames from the log file settings tab in the general settings dialog. The implementation of these settings is consistent with the other settings under this framework, as discussed in Section 4.3. This tab and the aforementioned settings can be seen in Figure 4.1c.

It should be noted that this logging framework is only made use of in recently introduced locations in the application. Going forward, the logging framework should be used for new debug and application output. Over time other debug statements can be updated as they are encountered.

With the introduction of this feature future users and researchers will easily be able to track down application output, debug, and error information by examining the relevant log files. This is the standard approach used by well designed applications as it allows the examination of the application logs when errors occur. This also allows developers and researchers to delve more deeply into the inner workings of the application.

³³<https://github.com/yestinj/inetvis/issues/50>

³⁴<https://github.com/yestinj/inetvis/pull/65>

4.9.3 MacOS Support

The original version of InetVis was developed for Linux-based operating systems, and in particular, Debian derivatives, such as Ubuntu. Since then, the software development world has seen a shift towards the use of high-end Macbook Pros for software development work in startups and larger corporates³⁵ due to their increased reliability and lower burden on IT staff (Bort, 2016). Given the similarities between macOS and Linux, due to their common ancestor, it was feasible that InetVis would be able to run on the BSD-based macOS operating system without a significant amount of work being required.

The first step was to build the application under macOS and to fix any errors that were encountered. This was done using *Qt Creator* version 5.9.1. In contrast version 5.5.1 was used for compilation under Ubuntu as it was the latest version provided by the default Ubuntu software repositories.

In order to track the work on this improvement issue #56³⁶ was created, with the required changes being implemented in feature branch `feature/macos_build_support`, and merged in through pull request #57³⁷. The crux of the work came down to removing an old, and unsupported, Qmake CFLAGS entry, namely `-macaccumulate-outgoing-args`. The addition of specific macOS logic for the importing of the OpenGL header files was also required. See Listing 4.12 for how this was implemented.

Source Code Listing 4.12: macOS Build Support (`dataproc.h`).

```

60     #ifdef MAC
61     #include <OpenGL/gl.h>
62     #endif
63
64     #ifdef LINUX
65     #include <GL/gl.h>
66     #endif

```

Once this improvement was implemented it was tested, and it was determined that most functionality worked as expected. The only feature that did not work was monitor local host mode. This is likely due to the way in which the *libpcap* headers are imported differs on macOS. As this mode of operation was not the primary focus of this research a solution was not explored. When necessary, the Linux version was used in order to utilize this functionality.

A further caveat with this improvement is that the *inetvis.app* file built by *Qt Creator* could not be run in a standalone manner. This issue was briefly explored, however the

³⁵<https://www.quora.com/Why-do-most-professional-programmers-prefer-Macs>

³⁶<https://github.com/yestinj/inetvis/issues/56>

³⁷<https://github.com/yestinj/inetvis/pull/57>

solution deemed too complex, as it required the inclusion and referencing of the *OpenGL* and *libpcap* libraries. At this time InetVis can only be run successfully on macOS when built and run through *Qt Creator*.

Regardless of these caveats, the ability to run InetVis greatly improved the speed at which new features and improvements could be developed and tested.

4.9.4 Support for HighDPI Displays

The original version of InetVis was written at a time when the resolution and pixel density of monitors were consistent. There was not yet the notion of a pixel scaling factor, such as is common today in monitors supporting resolutions above full HD (FHD) (Shultz, 2011). This scaling factor is necessary because without it, everything on display on ultra-high resolution monitors would be too small to see comfortably (Coppock, 2017).

After the initial support for macOS was added (Section 4.9.3), it was discovered that the timestamp and frames per second display elements, usually in the bottom left and bottom right of the visualisation pane respectively, were missing. This issue was investigated and it was determined that this only occurred when the application was opened on the main Retina display monitor of the Macbook Pro, and not when it was opened on an external FHD monitor. This issue was originally tracked under issue #68³⁸, and later under issue #94³⁹ which contained a more general description of the issue once the cause was known.

This issue was solved by the use of the *devicePixelRatio()* function provided by the OpenGL library, and is shown in Listing 4.13. This work implemented in commit 8f7737⁴⁰.

Source Code Listing 4.13: Support for HighDPI monitors (glviswidget.cpp).

```

360     int localDevicePixelRatio = devicePixelRatio();
361
362     //draw time label
363     if (dateTimeDisplayOn) { //glColor3f(1.0, 1.0, 1.0);
364         renderText(10, height/localDevicePixelRatio - 10, currentTime->
365 toString("yyyy/MM/dd_ _hh:mm:ss:zzz"), fontLabels);
366     }
367
368     //draw fps
369     if (fpsTextOn) { //glColor3f(1.0, 1.0, 1.0);
370         renderText(width/localDevicePixelRatio - 60, height/
371 localDevicePixelRatio - 10, fpsText, fontLabels);
372         fpsFrameCount++;
373     }

```

³⁸<https://github.com/yestinj/inetvis/issues/68>

³⁹<https://github.com/yestinj/inetvis/issues/94>

⁴⁰<https://github.com/yestinj/inetvis/commit/8f7737>

With this improvement the application will be able to continue functioning correctly on all modern computer monitors, no matter the display resolution or pixel ratio. This improvement helps to ensure that InetVis will remain useful and relevant for the foreseeable future.

4.9.5 Opening PCAP Files

In the original version of InetVis it was not possible to open a *.pcap* file directly by using the File -> Open / Replay Capture File file selection dialog. The only two file extensions that were supported were *.cap* and *.dump*.

In order to open a pcap file, the file in question would first need to be renamed with a supported file extension. This represented a small bug in the user experience of the application and it was pointed out by early testers. This was reported in issue #43⁴¹.

The implementation of this improvement involved the modification of a single line of code in the control panel. This was implemented in commit 8661226⁴².

By implementing this small, but useful, improvement to the application the user experience was improved for every user, by saving them time and allowing them to open commonly used pcap files directly without first having to rename them.

4.9.6 Improvements in Setting Home Network Range

In the original version of InetVis there is the option to have the application guess what the most appropriate home network range is. This functionality is useful, however, it does not work correctly on malformed or incomplete packet capture files. This issue was captured in Github issue #30⁴³. The underlying problem of improving the handling of malformed packet captures is considered in Section 5.2.13.

The general settings dialog was updated to include a new field which allows the user to define a custom home network range. The plotter settings dialog was also updated to include a *load* button with the purpose of loading the user-defined home network range from general settings, and then populating the existing fields in the dialog. The user

⁴¹<https://github.com/yestinj/inetvis/issues/43>

⁴²<https://github.com/yestinj/inetvis/commit/8661226>

⁴³<https://github.com/yestinj/inetvis/issues/30>

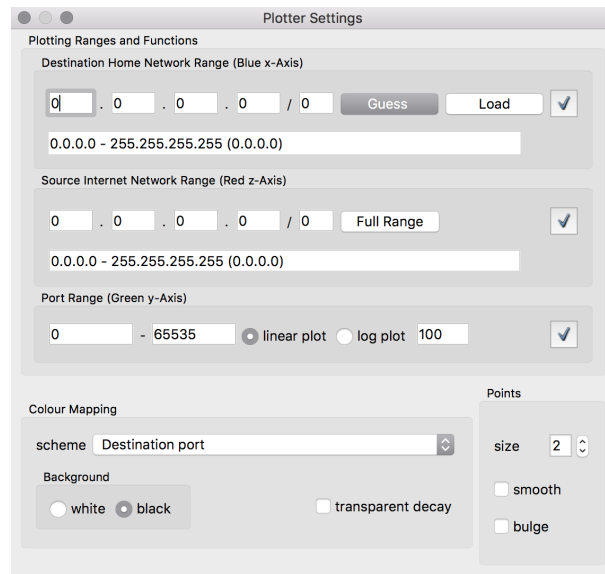


Figure 4.5: Updated Plotter Settings dialog with Load button.

is then able to review the loaded home network range, and click *Save* to have them set within the core of the application. When the user does this, the UI is updated to show only the specified home network range.

The user-defined home network range is *not* loaded into the application internals or UI automatically, as this would cause issues should the user open a packet capture file which uses a different home network range. The visualisation cube may be empty and it would not be obvious to the user what the problem is. In this case, the user would then need to open plotter settings and update the home network range appropriately. This setting is most useful to users who will be working with multiple packet captures with the same home network range, such as with packet captures coming from a network telescope.

The addition of the general settings framework implementation for this new setting was done in feature branch `feature/home_network_persistence`, and was merged in, in pull request #64⁴⁴. The UI demonstrating these changes can be seen under the home network settings tab in the general settings framework shown in Figure 4.1b. The *load* button that was added to plotter Settings can be seen in Figure 4.5.

4.9.7 About Dialog Modularity

The original about dialog box contained all of the information in a single, scrolling-box, window. This implementation is not optimal for many reasons, namely there is no quick

⁴⁴<https://github.com/yestinj/inetvis/pull/64>

and easy way to tell what topics are actually covered in the large amount of text, without first skimming through the entire contents of the dialog box.

Improvements to this part of the application were largely contributed by Irwin, in order to assist with improving the overall user experience and usability of the application. With the introduction of additional keyboard shortcuts (Section 4.8), there needed to be an obvious location where a user could find out which keyboard shortcuts were supported.

The about dialog was rewritten in a more modular format and was implemented in pull request #71⁴⁵. As with the general settings dialog, use was made of the *QTabWidget* object in order to split the information in the dialog up into tabs containing specific information. Within each tab use was made of the *QTextBrowser* object in order to display HTML formatted text. Screenshots of the updated about tabs can be seen in Appendix B.

4.9.8 Documentation Improvements

The original version of InetVis was shipped with an extensive manual on how to operate it. With the advent of this research the documentation required updates, and was improved by Irwin through contributions to the Github source repository. These contributions were merged in, in pull request #82⁴⁶. Separate, stand-alone, HTML files were introduced in this pull request, with the content of the about dialog tabs. New application screenshots were introduced as separate files for inclusion in the documentation, and the original HTML-style manual was deprecated in favour of a PDF created by the use of L^AT_EX and the *LyX* front-end environment.

4.9.9 Source Code and Spelling Consistency

During the initial work to port InetVis numerous spelling and grammatical errors were encountered in both the source code and in the application's UI. In order to improve the overall quality of the application it was important to fix all spelling and grammar issues. By doing this, the application should feel more polished and professional. Issue #45⁴⁷ was created in order to track and address this problem. Whenever these sort of errors

⁴⁵<https://github.com/yestinj/inetvis/pull/71>

⁴⁶<https://github.com/yestinj/inetvis/pull/82>

⁴⁷<https://github.com/yestinj/inetvis/issues/45>

were encountered in the application, while performing other work, they were fixed. Given this, no specific commit, or set of commits, was created for the correction of these errors. Also, it is possible that yet more spelling and grammar errors do still exist in rarely used parts of the application and source code.

A similar exercise was performed with the general source code quality and code conventions. While the work done throughout this research was performed, if code was added, modified, or removed from a particular file, the file would first be reviewed, and changes made where appropriate. These changes largely drew on the researcher's prior experience as a professional software developer. Most changes were small, and related to improving the consistency in the use of tabs, spacing, curly-braces, white space, comments, and variable naming. For example, control statements, such as the *if* statement, which were not making use of curly-braces for one-line conditional bodies were improved to make use of said curly-braces (Stack Exchange, 2011). Over the years, the best practices in this regard have evolved, leading to the omission of curly-braces in these situations being discouraged.

4.10 Summary

In this chapter the features that were implemented as part of this research were presented and described. The most important new feature was the general settings framework (Section 4.3). Optimisations to the screenshot format and settings (Section 4.4) were also introduced, making PNG the default screenshot and frame capture format.

The other features that were included as part of this research include the option to specify which local network interface to monitor (Section 4.5), and a few new keyboard shortcuts (Section 4.8). Irwin and Herbert introduced two new automated visualisation modes, Harlem shake mode (Section 4.6), and autorotation (Section 4.7), showing that it is reasonably straightforward for other researchers to extend the application.

The second part of this chapter, (Section 4.9), focused on the improvements which were made to InetVis as part of this research. Most of these changes were reasonably small and self-contained, and consisted of items such as improvements to the application UI and window positioning (Section 4.9.1), a standard way of logging output to standard output and standard error (Section 4.9.2), preliminary support for macOS (Section 4.9.3), and support for HighDPI displays (Section 4.9.4).

The combination of these features and improvements implemented as part of this research represent the first concerted effort to bring InetVis up to date with modern standards and functionality in the last decade. With the addition of these improvements InetVis will prove more useful than ever to its original audience, and increase its appeal to a wider audience of analysts and researchers. These changes have also begun the process of modernising the InetVis UI and functionality to ensure that it remains user friendly and usable in the future.

There is still a way to go, and more features and improvements which could be implemented. The ideas explored in this realm, during this research, are presented next in Chapter 5. This chapter also goes on to describe the results of the brief user survey which was done as part of this research, in order to act as a guide for the priority of the implementation of new features and improvements.

Unimplemented Features and Survey Discussion

5.1 Introduction

In the previous chapter the features and improvements implemented as part of this research were introduced and discussed. In this chapter, additional features and improvements which were identified, but not implemented, are discussed. These features and improvements would be useful additions to InetVis.

One of the most comprehensively planned features is that of a plug-in architecture, which is discussed in Section 5.2.1. This would allow users of InetVis to easily extend the application by writing simple plug-ins.

There are various other features and improvements discussed in this chapter, such as a heads-up display (Section 5.2.2), application UI modernisation (Section 5.2.4), *y*-axis source/destination port toggle (Section 5.2.9), and support for joysticks and gamepads (Section 5.2.15).

This chapter then presents the results of a user survey which was created and administered as part of this research (Section 5.3). This survey was created and targeted at professionals in the field who have prior experience in network security analysis. The survey was not

used for statistical analysis. The aggregate results were used in order to justify the priority for the implementation of the planned features.

Recommendations are then presented in Section 5.4, based on the planned features and the user survey results, in order to better inform future work on InetVis. Finally, this chapter concludes in Section 5.5 with a brief summary of the contents of this chapter and the most relevant points that were presented.

5.2 Unimplemented Features and Improvements

This section takes a closer look at the features and improvements that were identified, but not implemented due to time constraints. The items presented in this section represent the combination of ten years worth of ideas, as well as ideas and suggestions contributed by the original researcher, users, the research supervisor, as well as the current researcher. This section should be the first port of call for any new researchers wishing to extend the functionality of InetVis.

The features and improvements presented in the following sections would represent a great leap forward in the functionality and usability of the application, if implemented. Furthermore, these features would lay a solid foundation for easier future development on InetVis.

5.2.1 Plug-in/Module Functionality

One of the features that was discussed in the original research (van Riel, 2005, p. 64-65) is that of a plug-in, or a module, framework for InetVis. This could allow other developers to extend the application in a straightforward manner. This was introduced to the Github repository as issue #13¹.

The essence of this feature is such that the application would require fundamental re-design and extension in a way that makes it straightforward to implement certain types of functionality through the use of stand-alone modules. It would be possible to add new functionality to the application simply by writing a new module to implement the desired functionality. This framework would require the addition of hooks to certain points in

¹<https://github.com/yestinj/inetvis/issues/13>

the source code, i.e. before an event is processed, before an event is drawn, and when the visualisation cube is initialised. The developer need only override certain methods and functions, which would in turn be called by the application at the appropriate time.

With the addition of a module framework it would be possible to implement certain types of application functionality more easily. One example considered thus far has been the implementation of a heads-up display (HUD) (Section 5.2.2). This functionality was identified by early testers of the application as being useful.

Other possibilities for this framework are support for additional base-plane layouts, besides for the standard 3D scatter plot, the post-processing of events before they are visualised, the support for reading of captures files in different formats, the support for controlling how packet captures are written to disk, and the adjustment of colouring options based on different packet metadata.

The following items would need to be introduced in order to complete this feature:

1. The introduction of a C++ stub class for modules to override the functionality of.
2. The introduction of a suitable data structure to hold lists of the stub classes.
3. The usage of the stub module class methods in various locations throughout the code.
4. The introduction of a means by which modules will be loaded into the application.

5.2.2 Heads-up Display Implementation

The general idea behind a heads-up display is that of a GUI that provides supplementary information in a way that provides the information at a glance. This idea first came about with pilots flying aircraft and is also a commonly used term today in the automotive industry². In the case of InetVis, there is a lot of information that is not displayed during the visualisation, and a HUD could be used to further enrich the analysis process. For example, HUD elements could be introduced containing information such as the incoming packet rate, a graph of the packet rate over time, summary information on unique source IP addresses and ports, and geolocation information on the visualised events. Certain metrics could be implemented using spark lines (Section 2.3). There is a large amount

²https://en.wikipedia.org/wiki/Head-up_display

of information contained within packet captures which is, at this time, not visualised, or processed in any way.

The original version of InetVis was designed in such a way as to have multiple windows, making it useful for multi-monitor setups, it was restricted by the screen resolutions and aspect ratios of the time (van Riel, 2005, p. 68). Since then, many improvements have taken place. For example, wide-screen aspect ratios have become commonplace, as did much larger computer monitors with far higher resolutions. The combination of these items resulted in more screen real estate being available for use, especially to the left and the right of the visualisation pane. The additional real estate could easily be used by permanent dialog windows displaying all sorts of useful information, or by temporary pop-up dialog boxes showing distinct pieces of information depending on the user's interaction with the cube.

In order to implement a HUD, it will be necessary to add the capability to easily define new dialog boxes to the application, which can be toggled, and will be shown on the screen at the same time as the visualisation pane. The information within the dialog boxes should be updated by a separate thread, so as to not affect the visualisation cube, nor the user input to the application. In Section 6.5, a discussion is presented on improving the concurrency support within InetVis, as the application is not thread-safe at this point.

5.2.3 Mouse-Over Support for Network Events

Given the simplicity of the original implementation of InetVis some of the more complex user interaction features supported by other tools were not implemented. One of the most sought after features was the ability to gain further insight into specific points on the visualisation pane. This could be done by a mouse-over or mouse-click on a point on the cube, which would then pop-up a new dialog box, or GUI element, which would display the source IP address, source port, and any other pertinent information. This feature ties in nicely with the HUD feature presented in Section 5.2.2, and the module framework in Section 5.2.1.

This feature was investigated during this research, however, it was discovered that the selection, or clicking, of points on a 3D cube was not straightforward. It required understanding of how 3D computer graphics work, and hence was not explored further. With this feature users would be able to easily view further information on any network event on the cube and not have to resort to opening the packet capture in a separate analyser to

pin-point this information. This would allow for much faster turnaround time in network traffic analysis.

This issue is tracked in Github under number #52³, in order to highlight it for future implementation. This feature would need to be implemented in the *GLVisWidget* class.

5.2.4 Application UI Modernisation

This section focuses on the UI as a whole, and discusses why improving and modernising it would be a useful step in the application's evolution. An in-depth discussion on the existing application UI is covered in Section 3.2.2.

The UI of the application is a user's first interaction with InetVis. Given this, the UI should be as understandable and welcoming as possible. While the original version's UI was well designed and the architectural choices were sound, further improvements can be made in the vein of modernising the application's GUI.

One of the most useful UI improvements considered thus far is the consolidation of all of the application's many dialog windows into a modern user interface element. This could be implemented by combining the most used functionality of the control panel into the visualisation pane, as widgets, along the side and/or top of the pane, and the control panel widget hidden and not required for normal use.

A further improvement which could be made towards this goal is the consolidation of the various settings dialogs into one dialog window making use of multiple tabs. This approach is similar to that taken in the design of the general settings dialog. Trade-offs will need to be considered in this design, as the user should not have to delve deep into a settings dialog to modify common settings.

Finally, all of the existing UI elements in use were already present in Qt 3. Some of these elements are now deprecated and should not be used in modern code. With Qt 5.9, there are options available for use in the UI, as described in the Qt 5 introduction page⁴. The application's UI objects should be reviewed, and any that are found to have more appropriate modern equivalents should be replaced.

³<https://github.com/yestinj/inetvis/issues/52>

⁴<http://doc.qt.io/qt-5/qt5-intro.html>

5.2.5 Font Size Adjustments

In the original version of InetVis, the font size of both the axis labels and the text displayed on the visualisation cube (timestamp and frames per second), was hard-coded within the application source code. Due to this, the font size, and style, of these two separate types of text could not be modified independently, and furthermore could not be adjusted at all without modifying the source code of the application. This approach and implementation worked initially, however, the need for the independent adjustment of these font sizes is becoming more important.

The necessity of this improvement was discovered while implementing the improvements to support HighDPI displays, as discussed in Section 4.9.4. The tweaks performed in the aforementioned improvement simply adjusted the text size and placement manually in order to improve the application UI. While prudent values were chosen, they will likely not suit all users and all display resolutions and types, thus the need for a user configurable setting to allow the adjustment of the font size manually. This improvement was created as issue #95⁵, in order to keep track of this improvement.

In order to implement this improvement the following approach could be used. The general settings framework can be used in order to allow the user to adjust the font size for both the axes' labels and the other visualisation text. The second aspect is that the code that controls the text-size of these elements is in one location, as described in Section 4.9.4, and is thus easily modified to make use of a value set in the general settings framework.

The implementation of this improvement, while small, forms part of the overall goal of modernising and improving the overall UI and usability of the application, thus enhancing the UX.

5.2.6 Native Video Output

In the original version of InetVis the frame record feature (Section 3.2.2) was used as a simple way of producing video output based on the frames generated by the application when visualising a packet capture. This feature allows the user to capture all frames rendered in the application as graphics files to disk. Based on this output it is possible

⁵<https://github.com/yestinj/inetvis/issues/95>

to use a tool such as *ffmpeg* in order to produce a video file that is the combination of all frame captures. This can be done in *ffmpeg*⁶ by using the command shown in Listing 5.1.

Source Code Listing 5.1: Command line to generate video output from PNG.

```
ffmpeg -r 20 -pattern_type glob -i '*.png' test.mp4
```

This feature was identified early on in the research based on feedback from the initial research and is recorded in issue #18⁷.

This feature could prove particularly useful as it would allow for the immediate output of a video file, and not require the intermediate step of first producing frame captures and then combining them. Users would more easily be able to record visualisations. Being able to show a video demonstrating some particular network scan is much more descriptive than only being able to show still images of said visualisations.

This potential improvement was not determined to be high priority due to the existing frame capture implementation, as well as the ease of performing screen recordings on modern operating systems. It is straightforward to do so on all major platforms and even within virtual machines, as can be seen in Lim (2017) where over forty free tools and techniques for screen recording are discussed. Another reason for this decision is that given the current single-threaded nature of the application, processing video output would not be feasible while still retaining a responsive UI and visualisation. More information can be found about the need for improved concurrency in InetVis in Section 6.5.

5.2.7 Support for Time-Series Compressed Files

One of the ideas for a potential new feature came about due to research done at Rhodes University into GPU accelerated protocol analysis for large traffic traces (Nottingham, 2016). In this research, Nottingham developed an intermediate format for the storage and indexing of packet captures by making use of a technique dubbed time-series compression. These intermediate files would be computed by using a high powered GPU, which would make quick work of the packet capture, and store it in a highly compressed format with an in-memory index allowing for quick lookup and seek times. More detail on this format and method can be found in Nottingham's PhD thesis (Nottingham, 2016, p. 122), with implementation details available in the associated Github repository (Nottingham, 2015).

⁶<https://www.ffmpeg.org/>

⁷<https://github.com/yestinj/inetvis/issues/18>

An issue to track this feature was added to the project's Github repository, issue #16⁸. This feature could prove useful due to the fact that packet captures have only gotten larger over time, as Internet bandwidth and capacity increase and as more people are using the Internet every day. This fact, together with the increased amount of malware on the Internet that does automated scanning, will lead to larger and larger packet captures needing to be analysed. While the current improvements made to InetVis certainly help in this regard it is always preferable to be more efficient and to not simply use as much processing power and memory as possible.

If support for reading time-series compressed packet captures was added to InetVis it would be possible to visualise much larger packet captures using far less processing power and memory. An intermediate step or utility would be required in order to produce the time-series and index files from packet captures.

5.2.8 Event colouring by Netblock/AS/BPF Filter

InetVis allows the user to adjust the colour of the points plotted based on various packet attributes, such as the destination port, the source port, the protocol, and the packet size. By default, the colour mapping scheme is set to use the destination port, which has the result that as points moves up the y -axis their colour changes. These basic colour mapping options are useful for additional analysis, however, they are not that flexible. There are many other attributes within the metadata of a packet which could be used for more effective colouring.

One example of a useful colour scheme which could be introduced is colouring points based on user specified BPF filters, or based on the originating network's netblock or AS assignment. This would allow the user to analyse these additional dimensions within the data with ease, with the flexibility of BPF filters, and be able to easily determine if there is any significance based on the netblock or AS number. BPF filters in particular could allow the user to colour the events on almost any criteria, as long as these criteria can be expressed as a set of BPF expressions. This feature could lead the way forward to more advanced analysis within InetVis, as BPF filter expressions could be derived from other sources. For example, based on research done on categorising darknet traffic by Liu and Fukuda (2014). By utilising BPF filters based on research such as this it could be possible to easily determine which traffic may be malicious in nature.

⁸<https://github.com/yestinj/inetvis/issues/16>

This feature was considered early on during this research and is recorded in Github under issue #15⁹. The feature would require the addition of user configurable settings fields in the UI, and likely the addition of a few related to the colour mapping scheme drop-down box within the plotter settings dialog. The implementation of the colouring scheme would need to be updated to account for the new options, and this would be done in the *setColourScheme()* method within the *DataProcessor* class.

5.2.9 Y-axis Source/Destination Port Toggle

As mentioned previously, one of the initial design decisions made was to base InetVis on the *Spinning Cube of Potential Doom*. As such, the axes of the 3D cube used for the scatter plot were fixed in nature, i.e. the *x*-axis is always the destination network range, the *y*-axis is always the destination port, and the *z*-axis is always the source network range. It is not possible to change the dimension plotted on each axis, as is sometimes the case in related tools (Section 2.5). For the initial research (van Riel, 2005), which was concerned with darknet traffic, this did not pose a problem. However, with the extension of InetVis to other use cases this fixed-axes system becomes problematic.

Improving InetVis by allowing the user to select the packet attribute shown on the *y*-axis, as a toggle between source and destination port will greatly improve the usefulness of the application in the case where it is used to analyse traffic that is not from a network telescope. This improvement will need to be implemented in the application source code under the *Plotter* class, the *setRange()*, and *plot()* methods in particular will need to be updated, as this is where the axes ranges are set and the network events are plotted. The code related to how the *y*-axis label is set will also need to be updated and this is located within the *DataProcessor* class.

5.2.10 Rewind/Play Backwards Option

InetVis was originally designed to be a 3D scatter plot for network event visualisation, as well as providing VCR-like functionality. Having the option to play a network capture file backwards could prove to be useful. The user would be able to pause when an interesting pattern is discovered and slowly play the visualisation in reverse. This could lead to further insights into the visualised data.

⁹<https://github.com/yestinj/inetvis/issues/15>

Implementing this feature will require some work as the code is built around the fact that the visualisation will progress in the normal direction. The code will need to be updated in order to remove items from the event buffer in a LIFO manner, i.e. the last item added to the buffer, the most recent, will be the first to be removed. Once removed the visualisation pane will be redrawn in order to remove the event point. The *packetEventBuffer* deque is found within the *DataProcessor* class, and most changes will take place there. Further changes to the UI and general settings will be necessary in order to add a toggle, or a different play button that will cause the application to play the packet capture file in reverse. This issue is tracked in Github under issue #98¹⁰.

5.2.11 Native Compressed File Support

Another useful improvement is native support for opening compressed packet capture files. Often times packet captures are large, especially those from network telescopes where it is not uncommon for many months worth of network traffic to be contained within a single network packet capture. Given their large file size, these files are usually compressed in some manner and need to be uncompressed before they can be used. While this is not a major shortcoming of the application, it is a small improvement which could improve the user friendliness and usability of the application.

Similar to Section 5.2.7, the code in the *openCaptureFile()* method will need to be updated, as well as the file open dialog box, in order to allow the user to open common compressed files with the associated extensions. This issue is tracked in Github under issue #51¹¹.

5.2.12 Setting to toggle LogUI functionality

The *LogUI* class (Section 3.2.6) allows the logging of all UI events to a log file. At this time, this can only be enabled or disabled by updating the source code. This requirement makes this functionality unusable to casual users.

An option to enable or disable this feature should be added to the application, potentially under the general settings dialog. This improvement is tracked by issue #96¹² in Github.

¹⁰<https://github.com/yestinj/inetvis/issues/98>

¹¹<https://github.com/yestinj/inetvis/issues/51>

¹²<https://github.com/yestinj/inetvis/issues/96>

Implementation wise, whether or not this feature is enabled is currently controlled within the *main* source code file, which calls *!LogUI::enable()*, this logic would need to be moved to within the logic handling of the general settings dialog, such that when the checkbox is checked, this method will be called. Likewise the *disable()* method will need to be called when the checkbox is unchecked.

5.2.13 Improved Handling of Malformed PCAP Files

When this research began the original version of InetVis was tested prior to the porting work. It was determined that the application was particular about the type of packet capture files which it accepted. It was possible to open any packet capture with InetVis, however, some of these initial tests caused the application to crash. Put simply, InetVis was not very error tolerant. It was determined that InetVis only supports well-formed packet captures. For example, if a packet capture contained packets that were not properly formed then InetVis could potentially crash.

The current solution to this problem is to make use of *tcpdump* in order to pre-process an unknown packet capture, and produce an output file containing only complete and well-formed packets, which are then easily consumed by InetVis.

This issue highlights the fact that the packet processing code in InetVis is not as robust as it could be. It is non-obvious when a malformed packet capture file is loaded, that this is the actual problem. The user might be led to believe that the application is faulty, when it is simply that the packet capture they tried to load was slightly malformed. The *readCaptureFile()* in the *DataExtractor* class is responsible for this logic, and makes extensive use of the *pcap_next_ex*¹³ function provided by *libpcap*. This function will need to be closely examined and updated to be made more robust. In the case of malformed packets, they should be handled appropriately and not halt the execution of the entire application. This issue was added as issue #29¹⁴.

5.2.14 Re-Enable Recording of PCAPS for Monitoring Local Host

During the work done to port the application to Qt 5 a problem was encountered in the record packet capture functionality, when used in monitor local host mode. Perform-

¹³http://www.tcpdump.org/manpages/pcap_next_ex.3pcap.html

¹⁴<https://github.com/yestinj/inetvis/issues/29>

ing this action would occasionally result in a segmentation fault being thrown by the application. This issue was described in detail in Section 3.7.

For safety, this functionality was disabled in the application when in monitor local host mode. This issue was not high priority as a user of the software could make use of *tcpdump* in order to capture network traffic to a capture file. However, this functionality should be re-enabled in the future once it has been investigated and fixed. This issue is tracked by issue #27¹⁵ in Github.

5.2.15 Joystick/Gamepad Support

One of the original ideas that was envisioned for InetVis was the support of a joystick or a gamepad. This would enable a user to more easily navigate the visualisation pane and allow them to have more fun while analysing packet captures. The additional buttons on the joystick or gamepad would function as keyboard shortcuts and would be mapped to commonly used functions within the application.

In order to implement this functionality, use could be made of the *QtGamePad* add-on library, for gamepads, and of the *QGameController*¹⁶ library for joysticks. By using these two classes in a similar way to how the mouse and keyboard events are handled in the *GLVisWidget* could be employed. Code that would react to events from the device could be added, and then the appropriate logic based on the type of event activated.

5.3 User Survey

As part of this research a user survey was created and administered to a group of volunteers. These volunteers were selected based on their extensive experience in the information security, network security, and software development fields. The goal of this user survey was to inform the prioritisation of future features and development work in InetVis. Potential features were identified in Section 5.2, as being useful to, or missing from, InetVis. The use of the user survey responses helped a great deal in determining which features would be useful to real-world users of the application in realistic use-cases. A secondary goal of this user survey was to inform on the current user interface and user

¹⁵<https://github.com/yestinj/inetvis/issues/27>

¹⁶<https://github.com/openforeveryone/QGameController>

experience of InetVis, in order to determine if there are any specific weaknesses or areas of confusion.

The user survey responses were not used in any statistical analysis and were only used in order to inform the future development of InetVis. A survey was chosen as the mechanism for gathering this information as it provided an efficient way of gathering input from a reasonably sized pool of subject matter experts, while encouraging them to try out InetVis and become familiar with how it works. By performing this survey it was possible to gain feedback from participants who could potentially use this tool in their every day work and who could comment and provide quality feedback on the application's user interface and functionality.

InetVis v2.1.1 was released prior to this user survey and was the version that the recordings of the visualisations were done with. The participants were expected to make use of this version of InetVis if they decided to use the tool for the tasks set out in the survey. The `thesis_base`¹⁷ tag was used in Git in order to keep a record of which version was used for the user survey, and which version of InetVis was released at the time of completion of this research.

The remainder of this section describes the user survey questions and provides an overview of the participant's responses. Ethical clearance for this user survey was obtained from Rhodes University with ethics clearance number CIS17-07.

5.3.1 Survey Questions

The user survey questions, and the survey addendum, can be seen in Appendix C. The user survey contained questions relating to the participant's current role, technical ability, and their familiarity with network traffic analysis and visualisation tools. The participant was required to either watch videos of InetVis in action or to make use of InetVis themselves in order to analyse the provided packet captures. The participants were then asked questions regarding how they found working with InetVis, and how easy the scans were to identify. The survey ended by asking the participants to rank a list of features in the order in which they deem most important for implementation, and to provide any that they believe are important, but which were not in the given list.

¹⁷https://github.com/yestinj/inetvis/releases/tag/thesis_base

5.3.2 Survey Responses and Discussion

Five complete and detailed responses were received as part of this user survey. All of the participants had a technical background and all participants had worked with packet captures before. Most participants considered themselves experienced, and either worked in security related roles or were software developers. Both of these roles require the person to be intimately familiar with certain software and network frameworks, which inevitably leads to the use of packet capture and analysis tools.

While all participants had used *Wireshark* for network traffic analysis in the past, most did not have much experience with other network traffic analysis tools. All participants were also somewhat familiar with textual network packet analysis using tools such as *TCPDump*. One participant did have some prior experience with *ZenMap*¹⁸, and found it to be more useful than InetVis.

The participants were given two images, one showing *TCPDump* output of a port scan, and the other showing the same port scan visualised using InetVis. All participants agreed that InetVis presented the information in a more easy to understand manner. Two participants did point out that the *TCPDump* output was more useful in finding detailed information, such as IP addresses and ports. This weakness has been identified and there are various ideas for remedying it. For example, the implementation of a heads-up display (Section 5.2.2), and mouse-over support for network events (Section 5.2.3). Other features presented in Section 5.2 and Section 6.5 also address this weakness.

Participants were given the option of running InetVis in order to visualise the provided packet captures, or to watch online videos (Appendix D) in which the packet captures were visualised, allowing them to simply focus on the interpretation of the visualised information. Three participants made use of the tool itself, while the remaining two participants watched the provided videos. One of the participants ran the tool and also watched all of the videos. Two introductory videos were also provided describing how to make use of InetVis. All participants watched these videos.

Of the three participants that ran the tool, one found it somewhat useful, while two found it very useful. All participants who made use of the tool said that they found it to be easy to use. Two of these participants reported that they had minor issues such as dependency version conflicts, however, they were able to overcome them. One participant reported to having no issues installing and running the tool. The two participants who

¹⁸<https://nmap.org/zenmap/>

did not use the tool, but watched the videos, did attempt to run it, however, they ran into trouble installing and running InetVis, and eventually gave up. More than one participant commented that the installation should be more straightforward, such as by making use of a pre-built *.deb* file. This was considered during this research and should be considered in future work (Section 6.5).

Four of the participants found the InetVis user interface easy to understand and to work with, however one participant indicated that they did not know where to start. The introductory videos provided with the survey were enough to help the participant to make effective use of InetVis. Three of the participants indicated that the *home network not set* warning dialog was particularly annoying and unnecessary. Two participants suggested automatically setting the home network for the user, and allowing them the opportunity to override it. Two improvements made in this research (Section 4.9.6) attempt to address this issue by providing the user a way to suppress this warning dialog and giving the user a way to save a default home network range. It is obvious that these settings were not useful enough and that an option should be provided to automatically select an appropriate home network range. One participant did have trouble with the way the mouse rotated the cube and suggested an *invert mouse* option be introduced for users who would prefer this. This idea is discussed briefly in Section 6.5.

All of the participants reported that it was very easy to identify each of the given network scans provided in the packet capture files, whether they had run the tool or watched the videos. This is a welcome result as it shows that the visualisation aspect of InetVis is indeed straightforward, even to users who have not made use of similar network visualisation software before.

The participants were then given a list of ten potential new features and improvements to InetVis, and asked to rank them from one to ten, with one being most useful, and ten being least useful. They were also given the option of marking any features they did not find useful. The rankings from all users were aggregated, with points being allocated to a potential feature in such a way that a ranking of *one* would score ten points, *two* would score nine points, and so on down to a single point for a ranking of ten. In the case where a participant marked a feature as not useful, it was given a score of zero. This led to the results presented in Table 5.1, showing the most requested to the least requested features. The highest score a feature could obtain using this system is fifty. These results were influenced by participants who marked a few features as not useful, dragging down their total scores. For example, the option to colour events based on BPF matches was ranked second most useful by two participants, and then third and fourth most useful

Table 5.1: Rank table of most requested features by survey participants.

Rank	Feature	Score
1	A module framework	43
2	Dump all events on screen to a packet capture file	40
3	A client/server architecture	37
4	Colour events by BPF filter match	33
5	A heads-up display	23
6	Support for video output	18
7	Updated version of OpenGL	17
8	Alternate input support	15
9	Support for time-series compressed files	10
10	Overhaul the UI	8

by two more participants. However, as one participant ranked this feature as not at all important, it ended up only ranking fourth in the list.

As can be seen in Table 5.1, the three most requested features are: a module framework (Section 5.2.1), the ability to dump all events on screen to a packet capture file, and the implementation of a client/server architecture. This outcome highlights that the participants were unaware that they could dump all events on screen to a packet capture file by using the packet recording feature. However, this does further motivate for the re-enabling of packet recording when monitoring localhost (Section 5.2.14). It also highlights how this is a good choice for future work, as it is reasonably straightforward to add a new button that allows just the recording of the current view to a packet capture file. This is briefly discussed in Section 6.5.

One participant also suggested an option that allows the user to incrementally step-through a packet capture file as it is possible to do with modern video playback systems. This would allow for more controlled and precise playback compared to the current option of playing back only by using the time dimension. This is discussed in brief in Section 6.5.

The participants were asked if they had any other suggestions for features that they would like to share. One participant suggested a few worthwhile ideas which are enumerated in the following list:

1. The introduction of a dashboard showing traffic only for a certain range of specified IP addresses.
2. The option to explicitly set which ports to show network events from.

3. A way to show more detail behind a point on the visualisation pane.

Another participant asked for a feature to select a specific set of source IP addresses, destination IP addresses, and ports. This is another function that could be performed by using the BPF filter expression currently supported in InetVis. This points to the fact that participants in this survey were not familiar enough with BPF expressions to know what they are capable of. Additional documentation and instruction should be created in order to fully explain BPF expressions to new users, see Section 6.5. Other participants mentioned that it was hard to know the specific port numbers of network events, especially when zoomed out, and suggested the inclusion of port numbers on the y -axis. This could be solved by increasing the number of ticks on the axis labels.

Participants were asked for any final comments on their experience using InetVis. The answers were a mix between participants who found the software easy to install and run, and those who found it more difficult to use, but got the hang of it after watching the introductory videos. Most participants agreed that InetVis was useful for its intended purpose and one participant mentioned how they would be using it more in the future.

The results of the user survey performed here provide a good deal of insight into what potential users of InetVis are looking for in a network security visualisation tool. Features that were of interest to all participants were discovered and are discussed in more detail in the context of the other unimplemented feature work in Section 5.4. Aspects that participants found particularly difficult were identified and potential solutions discussed in future work. Finally, participants also provided feedback on the usefulness of InetVis and it was determined that all participants found InetVis useful in some way. This provides motivation for this research and for further research being performed concerning InetVis.

5.4 Recommendations

Considering the prioritisation and planning of features discussed in Section 5.2, as well as the results of the user survey responses, a set of recommendations for future work is presented below. This specifically only discusses overlapping features mentioned in both Section 5.2, and the user survey responses in Section 5.3. Other suggestions for future work are presented in Section 6.5.

A plug-in/module framework: The introduction of a plug-in framework stood out as the most useful feature as voted for by participants of the user survey. This feature was one of the earliest discussed and considered in this research, described in Section 5.2.1. The addition of a plug-in architecture to InetVis adds numerous possibilities in terms of how users can extend InetVis for their own uses. This also facilitates the implementation of many other suggested and recommended features. For these reasons, this feature should be seriously considered for development and implementation in future research on InetVis.

Colouring events by BPF filter match: This feature was ranked as fourth most useful by participants of the survey and was ranked as second most useful by two participants (Section 5.3.2). The overall score was skewed by a participant who did not find this feature useful, however, given that the other participants all found this feature to be highly useful, it should not be discounted. This feature was also partially planned and discussed in Section 5.2.8, as it was considered useful. This feature would allow users to more easily filter out network events that were not of interest to them, by highlighting those events which they deem to be most useful for analysis. For these reasons this feature should be seriously considered in future work on InetVis.

Heads-up display with mouse-over support: This feature actually consists of two closely related elements, namely the introduction of a HUD (Section 5.2.2) and the introduction of mouse-over support for points on the visualisation pane (Section 5.2.3). It is difficult to envision a useful HUD, without the added support of mouse-over interactions. In the user survey responses, respondents ranked a HUD as the fifth most useful. However, the majority of the participants also mentioned the lack of detail being available in the InetVis interface, and how support for such things would go a long way. For these reasons, this combination of features would greatly improve the capabilities of InetVis to provide users with detailed information on the visualised network events and thus should be seriously considered in future research and development.

5.5 Summary

In this chapter two specific topics were discussed, namely planned features and improvements, and the user survey administered as part of this research. Both of these topics were discussed with the goal of motivating for new features and improvements to be made to InetVis in future research.

The features and improvements that were discussed in Section 5.2, were identified and planned as part of this research, some coming from future work recommendations from previous work and some coming from new use-cases for the tool. These features range from complex implementations such as a plug-in framework (Section 5.2.1), to improvements which are smaller in both scope and implementation complexity such as modernising the application UI (Section 5.2.4).

A brief discussion on the user survey and the participant responses was then presented in Section 5.3. The user survey asked the participants some questions about their familiarity with analysing network packet captures, and then proceeded to ask the participants to make use of InetVis to analyse a set of packet captures. The feedback from the user survey (Section 5.3.2) highlighted that the participants all found InetVis to be useful. Some participants found the application hard to install and run, leading to the conclusion that the application's installation process and robustness should be improved in the future.

Participants were also asked to rank a list of potential features, and the results were combined to produce a ranking table, listing the most to least useful features as voted on by the participants. The plug-in framework, the functionality to dump all events on the screen to a packet capture file, and the introduction of a client/server architecture were the three top features chosen.

Finally, recommendations were presented in Section 5.4 which suggested features to be implemented based on the planned, but unimplemented, work done in this research and on the responses to the user survey. A plug-in framework, colouring of events based on BPF matches, and support for a heads-up display with mouse-over support were selected as the three most desirable features for future work.

This chapter laid a foundation for future work on InetVis and network security data visualisation in general. It did this by providing a detailed set of useful features and by making use of user feedback from the survey to determine which features were most important and which areas of InetVis needed the most work.

6.1 Introduction

This research aimed to further the field of network security data visualisation by modernising and updating InetVis. This research is considered useful as the problem of network security analysis is only getting more difficult. The incorporation of a visual analysis system is of paramount importance for effective analysis.

This chapter begins with a summary of the work presented in this thesis in Section 6.2. This is followed by the evaluation of the research goals, laid out in Section 1.3, which is presented in Section 6.3. The significance of this research is then presented in Section 6.4. Finally, idea for future work are discussed in Section 6.5.

6.2 Research Summary

This thesis started out with an introduction (Chapter 1) into the problem space and presented a background, a problem statement, the research goals, the scope of work, and ended with documentation conventions and a discussion on the document structure. This

research set out to investigate, update, and modernise InetVis in order to allow it to once more be useful to practitioners in the security visualisation field. The more complex features and improvements to InetVis were out of scope in this research.

Chapter 2 presented a literature review, where the relevant concepts in network security, data visualisation, and security visualisation were explained and discussed. In particular, the usefulness of the 3D scatter plot visualisation method was presented. Tools related to InetVis were also presented and discussed, concluding with a discussion on the high-level themes and trends present across all of the presented tools. InetVis is a simple tool, however, it fills a niche in the field of network telescope traffic analysis. It could do with an improved feature set to enable it to be more useful to future researchers.

Chapter 3 presented a discussion on the design, architecture, and implementation of the original version of InetVis. The goals of the original tool were discussed, namely the modularity, and support for efficient analysis of very large packet captures. This chapter also discussed all of the work done to port InetVis to run on modern operating systems. The port was successfully completed, with the application now able to run on the latest version of Ubuntu, and being fully functional on virtual machine, as well as standard computer hardware. The work done in order to port the application from Qt 3 to Qt 5 was described in detail, as it represented the most complex part of the porting process.

Chapter 4 presented a discussion of the new features and improvements that were implemented in InetVis as part of this research. The general settings framework was one of the major improvements, and with its introduction it was possible to improve the application in many other ways with minimal effort. Support for running InetVis effectively on HighDPI displays, and on macOS, was also discussed. Lastly, the introduction of new keyboard shortcuts to improve the user experience was presented.

Chapter 5 discussed the plans for new features and improvements which were not implemented as part of this research. The heads-up display and support for a module framework are two of the most complex, and most useful, extensions which were identified. This chapter also discussed the user survey that was used in order to determine how users found InetVis, and which features they would most like to see in the tool. It was found that experienced participants found InetVis to be useful. However, lacking easy installation and the ability to drill-down into the details. Participants ranked potential features, and when reconciled with the planned features, the most useful items for future work were identified as the addition of a plug-in framework, colouring events by a BPF filter match, and a heads-up display with mouse-over support.

6.3 Research Evaluation

In order to evaluate the effectiveness of this research, the outcomes discussed in the body of this thesis are compared with the research goals that were introduced in Chapter 1. These goals are presented below, with a discussion on the degree to which each goal has been met. The research question of **How can InetVis be modernised and updated for enhanced use** is also answered below by the review of the research goals.

6.3.1 Modernise InetVis to Run on Modern Operating Systems

This goal was fully met as part of this research. The original application code was reviewed in detail, and presented in Section 3.2. This was used as a foundation for the work done to port InetVis to run on modern operating systems, which was presented in detail in Section 3.3. The major aspects of the port were the update from the Qt 3 to Qt 5 graphical framework, and the native support for modern 64-bit systems. At the end of this process, a fully functional version of InetVis was released for modern versions of Linux.

The usefulness of InetVis has also been extended with the introduction of support for macOS (Section 4.9.3), and partial support for Microsoft Windows added by Irwin¹.

6.3.2 Explore Additional Features and Implement Improvements to InetVis

The goal of exploring additional features and implementing feasible improvements to InetVis was fully met. The exploration of additional features and improvements took place in Chapters 2, 4, and 5. The implementation of feasible updates took place in Chapter 4, where detailed information was given on how features were chosen, designed, and implemented. The general settings framework (Section 4.3) was one of the highlights of this work, as it allowed many of the other features and improvements to be implemented more easily.

Some of the features which were identified, but were not implemented in this research, for various reasons, are presented and discussed in some detail in Chapter 5. This ensures

¹<https://github.com/barryirwin/inetvis/tree/Windows-port>

that future researchers will have a good starting point for new research and development on InetVis.

6.4 Significance of Research

This research is valuable and of significance due to the fact that it has fully achieved the research goals set out at the onset of this research. These research goals were aimed at producing research and improvements to the InetVis visualisation tool that would help to further the area of security data visualisation, specifically relating to the area of network telescope traffic visualisation and analysis.

Given the state of the network security analysis field, and the worsening situation with more data being collected for analysis every day, it is important for new tools and techniques to be developed in order to facilitate the effective analysis of all of this additional network traffic. InetVis has proven itself to be useful for network traffic analysis, particularly in the case of network telescope traffic. This research updated, modernised, and added new features to this tool in order to make it functional once again, and to make it even more useful to users than it was before. While this tool does not represent anything revolutionary in the area of security visualisation, it represents an incremental step forward in the the evolution of InetVis.

This research is also significant in that the porting process from Qt 3 to Qt 5 was complex and time consuming. The procedure followed, as well as useful notes, were highlighted in this research. This information would be useful to anyone, in any field, who has the need to port an old application to the latest version of Qt 5.

Finally, this research presents a discussion considering the overarching principles and aspects found when examining a sufficiently large number of network security visualisation tools. This review was not exhaustive, however, it represents a useful piece of survey work on related tools which could be useful to other tool developers and researchers in the field.

At the end of this research a functional and stable version of InetVis is available once again for Ubuntu Linux. InetVis now also runs on macOS, and is much more customisable and stable than before. This research has served to resurrect a tool which was used extensively by network telescope researchers at Rhodes University upon its creation, and is now available for use in the analysis of network traffic once again, and to a much wider audience than before.

6.5 Future Work

As has already been discussed, there is much work that could still be done to improve InetVis. The most useful additional features that were identified are discussed in detail in Chapter 5. The two most important features discussed are the introduction of a plug-in system to InetVis, and also the introduction of a heads-up display. These two items are discussed in detail in Sections 5.2.1 and 5.2.2.

While the work presented in Chapter 5, and specifically the recommendations (Section 5.4), is particularly useful for consideration in future research on this topic, there are yet other future work suggestions which are made in the remainder of this section.

A good first candidate for future work is that of performance testing and analysis. The original version of InetVis was written to be efficient and to fully utilise the hardware available at the time. At this point, a decade later, InetVis runs with acceptable speed in a virtual machine on a midrange laptop. While no specific performance testing work has been done, it would be useful to know exactly how InetVis performs in extreme circumstances, such as when working with very large packet captures. Work could be done in order to benchmark the application using various sizes of packet capture files, and using various hardware configurations, in order to determine what the limits of the application are.

Support for new and improved packet capture formats, such as *PcapNg*, discussed in Section 2.2.2, is another potentially small feature which could prove very useful to researchers in the future. Should this format gain further acceptance and support by the industry, it will be particularly useful for InetVis to be able to natively read this file format. The use of newer network monitoring and packet processing libraries, such as *libtrace*², could also be a useful addition to InetVis, so that it does not rely solely on *libpcap*.

In order for the application to become more useful to a wider audience it will be necessary for proper packaging to be implemented. In this research an installation script was developed for use on Ubuntu. However, this mechanism is not robust. The application should be packaged in a *.deb* file (or another distribution specific format), making it easier and more attainable for users of Linux operating systems. The same should be done for the macOS version, a complete *app* package file should be compiled to allow the application to install and run on a macOS system. The Microsoft Windows version of the application

²<https://research.wand.net.nz/software/libtrace.php>

has had some preliminary work performed on it as part of this research. However, it also requires a proper packaging and installation strategy.

Another item for consideration in future work is that of performing an in-depth use-case study. InetVis has the facility for logging of all user interface actions, which could prove highly useful in the analysis of a use-case study. In order to make the application more user-friendly and useful, it is necessary to discover how users use the application, where they have trouble, and where improvements can be made.

In Section 4.4, the idea of updating InetVis to make use of multiple threads was introduced, as it would be useful in order to offload the creation and output of screenshots to disk without affecting the rendering performance of the visualisation in InetVis. This improvement could be implemented by adding each screenshot task to a queue, which would be processed by another thread, and thus not affect the main thread running the application visualisation. However, InetVis was not written in a thread safe manner, which means that substantial work will need to be done to ensure that all data accesses and updates are thread safe. This is not a trivial undertaking, especially with concurrency libraries differing by operating system. However, it is an improvement which could greatly increase the speed and responsiveness of the application.

One final improvement which was mentioned in the user survey responses is that of the need for further documentation or video tutorials on how to effectively utilise InetVis. The basics are covered well by the application documentation at present. However, the more esoteric functions such as the BPF filter are not. More than one participant suggested a feature or use case which could be accomplished by making use of the BPF filtering mechanism already in the application. In the future, it would be particularly useful to record further tutorial videos, such as the ones created for this research and survey (appendix D), in order to discuss complex topics, such as BPF filtering.

References

- Abdullah, K., Lee, C., Conti, G., Copeland, J. A., and Stasko, J.** Ids rainstorm: visualizing ids alarms. In *IEEE Workshop on Visualization for Computer Security, 2005. (VizSEC 05)*, pages 1–10. IEEE, 2005. ISBN 0780394771. doi:10.1109/VIZSEC.2005.1532060.
- Barnett, R. J. and Irwin, B.** Towards a Taxonomy of Network Scanning Techniques. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pages 1–7. ACM, 2008. ISBN 9781605582863. doi:10.1145/1456659.1456660.
- Barrera, D.** Towards Classifying and Selecting Appropriate Security Visualisation Techniques. Masters thesis, Carleton University, 2009.
- Bass, T., Zuech, R., Gutzwiller, R. S., and Giacobe, N. A.** Cyberspace Situational Awareness. 2017. Last accessed: 2017-11-23.
URL <https://www.researchgate.net/project/Cyberspace-Situational-Awareness>
- Blanchette, J. and Summerfield, M.** C++ GUI Programming with Qt 3. Prentice Hall, 1st edition, 2004. ISBN 978-0131240728.
- Bort, J.** Macs are a third as expensive to own as Windows PCs, IBM’s IT guy says. 2016. Last accessed: 2017-12-29.

- URL <http://www.businessinsider.com/an-ibm-it-guy-macs-are-300-cheaper-to-own-than-windows-2016-10>
- Classen, A., Heymans, P., and Schobbens, P. Y.** What's in a feature: A requirements engineering perspective. *Fundamental Approaches to Software Engineering*, 4961 LNCS(April):16–30, 2008. ISSN 03029743. doi:10.1007/978-3-540-78743-3_2.
- Conti, G.** Security data visualization: graphical techniques for network analysis. No Starch Press, 2007. ISBN 1-59327-143-3.
- Conti, G., Abdullah, K., Grizzard, J., Stasko, J., Copeland, J. A., Ahamad, M., Owen, H. L., and Lee, C.** Countering security information overload through alert and packet visualization. *IEEE Computer Graphics and Applications*, 26(2):60–70, 2006. ISSN 02721716. doi:10.1109/MCG.2006.30.
- Coppock, M.** Blurry apps ruining your 4K monitor? Adjust high-DPI scaling in Windows 10. 2017. Last accessed: 2017-12-29.
URL <https://www.digitaltrends.com/computing/how-to-adjust-high-dpi-scaling-in-windows-10/>
- Coudriau, M., Lahmadi, A., Francois, J., Coudriau, M., Lahmadi, A., Francois, J., Analysis, T., Coudriau, M., and Lahmadi, A.** Topological analysis and visualisation of network monitoring data: Darknet case study. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2016. doi:10.1109/WIFS.2016.7823920.
- Cowie, B. and Irwin, B.** A Baseline Numeric Analysis of Network Telescope Data for Network Incident Discovery. In *Southern African Telecommunications and Applications Conference (SATNAC)*. 2010.
- Dragorn@kismetwireless.net.** The GPL Cube of Potential Doom. 2011. Last accessed: 2017-07-16.
URL <https://www.kismetwireless.net/doomcube/>
- Fachkha, C. and Debbabi, M.** Darknet as a Source of Cyber Intelligence: Survey, Taxonomy, and Characterization. In *IEEE Communications Surveys and Tutorials*, volume 18, pages 1197–1227. 2016. ISBN 1232550000. ISSN 1553877X. doi:10.1109/COMST.2015.2497690.
- Goddard, L.** Digia to pay Nokia 4 million Euros for Qt as framework heads towards cross-platform future. 2012. Last accessed: 2017-12-27.

- URL <https://www.theverge.com/2012/8/10/3233105/digia-nokia-qt-acquisition-4-million-euros>
- Goodall, J. R.** Introduction to visualization for computer security. In *Conference on Visualization Security (VizSec2007)*, pages 1–17. Springer, 2008. ISBN 978-3-540-78242-1. ISSN 2197666X. doi:10.1007/978-3-540-78243-8.
- Goodall, J. R.** Computer Network Traffic Visualization Tool. 2009. Last accessed: 2017-11-22.
URL <http://tnv.sourceforge.net/index.php>
- Goodall, J. R., Lutters, W. G., Rheingans, P., and Komlodi, A.** Preserving the Big Picture: Visual Network Traffic Analysis with TNV. In *IEEE Workshop on Visualization for Computer Security, 2005. (VizSEC 05)*, pages 47–54. 2005. ISBN 0-7803-9477-1.
- Harris, G.** Libpcap File Format. 2015. Last accessed: 2017-11-20.
URL <https://wiki.wireshark.org/Development/LibpcapFileFormat>
- Hunter, S. O.** A Network Telescope Information Visualisation Framework. Honours thesis, Rhodes University, 2010.
URL <http://www.cs.ru.ac.za/research/g07h3314/oldsite/resources/thesis.pdf>
- Ire, S.** Automated porting from Qt 4 to Qt 5. 2012a. Last accessed: 2017-12-27.
URL <https://www.kdab.com/automated-porting-from-qt-4-to-qt-5/>
- Ire, S.** Porting from Qt 4 to Qt 5. 2012b. Last accessed: 2017-12-27.
URL <https://www.kdab.com/porting-from-qt-4-to-qt-5/>
- Irwin, B.** Network Telescope Metrics. In *Southern African Telecommunications and Applications Conference (SATNAC)*. 2012.
- Irwin, B. V. W.** A framework for the application of network telescope sensors in a global IP network. Doctoral thesis, Rhodes University, 2011.
URL <http://nrfnexus.nrf.ac.za/handle/20.500.11892/21351>
- Irwin, B. V. W. and van Riel, J.-P.** InetVis: A Graphical Aid for the Detection and Visualisation of Network Scans. In *Conference on Visualization Security (VizSec2007)*, pages 255–273. Springer, 2008. ISBN 978-3-540-78243-8. ISSN 2197666X. doi:10.1007/978-3-540-78243-8.

- Iwaya, A.** Is the PNG Format Lossless Since it Has a Compression Parameter? 2014. Last accessed: 2017-12-28.
URL <https://www.howtogeek.com/203979/is-the-png-format-lossless-since-it-has-a-compression-parameter/>
- Jaquith, A.** Security Metrics: Replacing Fear, Uncertainty, and Doubt. Addison-Wesley Professional, 2007. ISBN 978-0321349989.
- Jin, Z.** Visualization of network traffic to detect malicious network activity. Master of science, Norwegian University of Science and Technology, 2008.
- Lau, S.** The Spinning Cube of Potential Doom. 2003. Last accessed: 2017-07-16.
URL <https://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2003/cube-of-doom/>
- Lee, C.** chrislee35/flowtag. 2013. Last accessed: 2017-11-22.
URL <https://github.com/chrislee35/flowtag>
- Lee, C. and Copeland, J.** Flowtag: A Collaborative Attack-Analysis, Reporting, and Sharing Tool for Security Researchers. In *Conference on Visualization Security (VizSec2006)*, pages 103–107. ACM, 2006. ISBN 1595935495. doi:10.1145/1179576.1179597.
- Lee, C. P., Trost, J., Gibbs, N., Beyah, R., and Copeland, J. A.** Visual firewall: Real-time network security monitor. In *Conference on Visualization Security (VizSec2005)*, pages 129–136. IEEE, 2005. ISBN 0780394771. doi:10.1109/VIZSEC.2005.1532075.
- Lim, H.** Screen Capture Tools: 40+ Free Tools and Techniques. 2017. Last accessed: 2018-01-08.
URL <https://www.hongkiat.com/blog/screen-capture-tools-40-free-tools-and-techniques/>
- Lindemann, K.** Researchers render cyberspace like a 3D video game to make identifying threats easier. 2017. Last accessed: 2017-12-18.
URL <https://www.researchgate.net/blog/post/researchers-render-cyberspace-in-3d-like-a-video-game-to-make-identifying-threats-easier>
- Liu, J. and Fukuda, K.** Towards a Taxonomy of Darknet Traffic. In *IWCMC 2014 - 10th International Wireless Communications and Mobile Computing Conference*, pages 37–43. IEEE, 2014. ISBN 9781479909599. doi:10.1109/IWCMC.2014.6906329.

- Mansmann, F.** Visual Analysis of Network Traffic: Interactive Monitoring, Detection, and Interpretation of Security Threats. Doctoral thesis, University of Konstanz, 2008.
URL http://kops.uni-konstanz.de/bitstream/handle/123456789/5658/Diss_Mansmann.pdf?sequence=1&isAllowed=y
- Marty, R.** Applied Security Visualization. Addison-Wesley, 2009. ISBN 9780321510105.
- Marty, R.** AfterGlow. 2013. Last accessed: 2017-11-22.
URL <http://afterglow.sourceforge.net/>
- Marty, R.** The DAVIX Live CD. 2015. Last accessed: 2017-11-21.
URL <http://www.secviz.org/node/89>
- McPherson, J.** Network Visualizations - About PortVis. 2004. Last accessed: 2017-11-22.
URL <http://vis.cs.ucdavis.edu/NetVis/PortVis/about.html>
- McPherson, J., Ma, K.-L., Krystosk, P., Bartoletti, T., and Christensen, M.** PortVis: A Tool for Port-Based Detection of Security Events. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 73–81. ACM, 2004. ISBN 1581139748. doi:10.1145/1029208.1029220.
- McRee, R.** Security Visualization: What you don’t see can hurt you. *Systems Security Association (ISSA) Journal*, 6(June):38–41, 2008.
URL <http://holisticinfosec.org/toolsmith/docs/june2008.pdf>
- Meharia, P.** Use of visualization in digital financial reporting: The effect of Sparkline. Doctoral thesis, University of Kentucky, 2012.
URL <http://search.proquest.com/docview/1498142202?accountid=17242>
- Monsch, J.** DAVIX 1.0.1 Released. 2008. Last accessed: 2017-12-21.
URL <http://www.secviz.org/content/davix-101-released>
- Munzner, T.** Visualization Analysis and Design. CRC press, 2014. ISBN 9781466508934.
- Nottingham, A.** PhD Research - GPF+. 2015. Last accessed: 2017-02-22.
URL <https://github.com/anottingham/PhdResearch>
- Nottingham, A.** GPU Accelerated protocol analysis for large and long-term traffic traces. Doctoral thesis, Rhodes University, 2016.
URL <http://hdl.handle.net/10962/910>

- Nunnally, T., Chi, P., Abdullah, K., Uluagac, A. S., Copeland, J. A., and Beyah, R.** P3D: A parallel 3D coordinate visualization for advanced network scans. In *IEEE International Conference on Communications*, pages 2052–2057. IEEE, 2013. ISBN 9781467331227. ISSN 15503607. doi:10.1109/ICC.2013.6654828.
- Pemberton, D.** An Empirical Study of Internet Background Radiation Arrival Density and Network Telescope Sampling Strategies. Msc thesis, Victoria University of Wellington, New Zealand, 2007.
URL https://ecs.victoria.ac.nz/foswiki/pub/Main/IanWelch/Dean_Pemberton_MSC_Thesis.pdf
- Qt Centre Forum.** QLineEdit - lostFocus signal problem. 2007a. Last accessed: 2017-12-27.
URL <http://www.qtcentre.org/threads/8785-QLineEdit-lostFocus-signal-problem>
- Qt Centre Forum.** What is the difference between the activated() and triggered() SIGNALs for a QAction? 2007b. Last accessed: 2017-12-27.
URL [http://www.qtcentre.org/threads/5862-What-is-the-difference-between-the-activated\(\)-and-triggered\(\)-SIGNALs-for-a-QAction](http://www.qtcentre.org/threads/5862-What-is-the-difference-between-the-activated()-and-triggered()-SIGNALs-for-a-QAction)
- Ribbecca, S.** The Data Visualisation Catalogue. 2017. Last accessed: 2017-05-31.
URL <http://www.datavizcatalogue.com/>
- Riley, D.** Nokia Acquires Trolltech For \$153 Million. 2008. Last accessed: 2017-12-27.
URL <https://techcrunch.com/2008/01/28/nokia-acquires-trolltech-for-153-million/>
- Schwagele, C. and Irwin, B.** A Literature Review of Network Monitoring Through Visualisation and the Inetvis Tool. 2010. Last accessed: 2018-01-07.
URL <http://www.cs.ru.ac.za/research/g07s3491/downloads/lit/litReview.pdf>
- Shneiderman, B.** The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages*, pages 336–343. 1996.
- Shultz, G.** Get a better view in Windows 7 by adjusting DPI scaling. 2011. Last accessed: 2017-12-29.
URL <https://www.techrepublic.com/blog/windows-and-office/get-a-better-view-in-windows-7-by-adjusting-dpi-scaling/>

- Stack Exchange.** Single statement if block - braces or no? [closed]. 2011. Last accessed: 2017-12-29.
URL <https://softwareengineering.stackexchange.com/questions/16528/single-statement-if-block-braces-or-no/16530>
- Tcpdump.** Tcpdump and Libpcap. 2017. Last accessed: 2017-11-20.
URL <http://www.tcpdump.org/>
- The Qt Company Ltd.** Porting to Qt 4. 2016a. Last accessed: 2017-12-27.
URL <https://doc.qt.io/archives/qt-4.8/porting4.html>
- The Qt Company Ltd.** Porting UI Files to Qt 4. 2016b. Last accessed: 2017-12-27.
URL <https://doc.qt.io/archives/qt-4.8/porting4-designer.html>
- The Qt Company Ltd.** Qt3Support Module. 2016c. Last accessed: 2017-12-27.
URL <http://doc.qt.io/qt-4.8/qt3support-module.html>
- The Qt Company Ltd.** qt3to4 - The Qt 3 to 4 Porting Tool. 2016d. Last accessed: 2017-10-15.
URL <https://doc.qt.io/qt-4.8/qt3to4.html>
- The Qt Company Ltd.** Transition from Qt 4.x to Qt 5. 2016e. Last accessed: 2017-12-27.
URL https://wiki.qt.io/Transition_from_Qt_4.x_to_Qt5
- The Qt Company Ltd.** C++ API changes. 2017a. Last accessed: 2017-12-27.
URL <https://doc.qt.io/qt-5/sourcebreaks.html>
- The Qt Company Ltd.** Creating Project Files. 2017b. Last accessed: 2017-10-15.
URL <http://doc.qt.io/qt-5/qmake-project-files.html>
- The Qt Company Ltd.** Porting Guide. 2017c. Last accessed: 2017-12-27.
URL <https://doc.qt.io/qt-5/portingguide.html>
- The Qt Company Ltd.** QSettings Class. 2017d. Last accessed: 2017-10-28.
URL <http://doc.qt.io/qt-5/qsettings.html>
- Tranter, J.** Porting Desktop Applications from Qt 4 to Qt 5. 2013. Last accessed: 2017-12-27.
URL <https://www.ics.com/blog/porting-desktop-applications-qt-4-qt-5>
- Tufte, E.** Envisioning Information. Graphics Press, may edition, 1990. ISBN 978-0961392116.

- Tufte, E.** Visual Explanations: Images and Quantities, Evidence and Narrative. Graphics Press, february edition, 1997. ISBN 978-1930824157.
- Tufte, E.** The Visual Display of Quantitative Information. Graphics Press, 2nd edition, 2001. ISBN 978-1930824133.
- Tufte, E.** Beautiful evidence. Graphics Press, 1st edition, 2006. ISBN 978-1930824164.
- van Riel, J.-P.** Multi-dimensional Visualization for the Analysis of Internet Traffic and the Identification of Intrusive Activity. *BSc Honours Thesis, Rhodes University*, 2005.
- van Riel, J.-P.** Network Security Visualisation. 2007. Last accessed: 2017-09-28.
URL <http://www.cs.ru.ac.za/research/g02v2468/inetvis.html>
- van Riel, J.-P. and Irwin, B.** Identifying and Investigating Intrusive Scanning Patterns by Visualizing Network Telescope Traffic in a 3-D Scatter-plot. In *Information Security for South Africa - Proceedings of the ISSA 2006 Conference*, pages 1–12. 2006a.
- van Riel, J.-P. and Irwin, B.** InetVis, a Visual Tool for Network Telescope Traffic Analysis. In *Proceedings of the 4th international conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa - Afrigraph '06*, pages 85–89. ACM, 2006b.
- van Riel, J.-P. and Irwin, B.** Toward Visualised Network Intrusion Detection. In *Southern African Telecommunications and Applications Conference (SATNAC)*, pages 3–6. 2007.
URL <http://www.cs.ru.ac.za/research/G02V2468/publications/vanRiel-SATNAC2006.pdf>
- Wireshark.** Code Review / wireshark.git / tree. 2017. Last accessed: 2017-11-20.
URL <https://code.wireshark.org/review/gitweb?p=wireshark.git;a=tree;f=wiretap;h=7f31dcf2134b501d83e512b58d44643859092381;hb=HEAD>
- Yurcik, W.** Visualizing NetFlows for Security at Line Speed: The SIFT Tool Suite. In *LISA*, pages 169—176. 2005.
- Zhuang, Y., Cappos, J., Rappaport, T. S., and Mcgeer, R.** Future Internet Bandwidth Trends: An Investigation on Current and Future Disruptive Technologies. Technical report, Tech. rep. tr-cse-2013-04, Polytechnic Institute of NYU, 2013.
URL <https://ssl.engineering.nyu.edu/papers/tr-cse-2013-04.pdf>

Appendices

APPENDIX A

Code Listings

In this appendix the full source code listings of the three shell scripts that were written in order to facilitate release packaging, installation, and removal of InetVis releases are introduced, discussed, and provided in full. These scripts are also referred to and briefly discussed in Section 3.5.

A.1 `prepare_release.sh`

This shell script was written in order to prepare and package a release of InetVis once it is ready. It assumes various facts, and various locations of certain assets, as can be seen by examining the listing. Most importantly, this script assumes that the *inetvis* binary has already been built, and that it is in its standard location within the *src* directory. This can be done before using this script by using first *qmake*, and then *make* in order to build the binary. This script takes in a version number when run. For example, `./prepare_release.sh 2.1.1`. The script first creates temporary directories based on the given version number, it then copies the binary, docs, license files, other resources such as the desktop and icon files, installation and removal scripts which have been customised based on the version number, and finally the readme file. The script completes by archiving the temporary directory into a gzipped tarball, i.e. *inetvis-2.1.1.tgz*, and removes the temporary directory.

Source Code Listing A.1: Shell script to prepare a release.

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: inetvis <version>"
    exit 1
fi

VER=$1
DOC_DIR="./doc"
SRC_DIR="./src"
LIC_DIR="./lic"
BIN_NAME="inetvis"
IMAGE_DIR="images"

TMP_DIR="inetvis-${VER}"
BIN_DIR="bin"
RES_DIR="resources"

echo "Preparing archive for ${TMP_DIR}..."

echo "Creating working directory structure..."
mkdir ${TMP_DIR}
mkdir ${TMP_DIR}/${BIN_DIR}
mkdir ${TMP_DIR}/${RES_DIR}

echo "Copying binary..."
cp ${SRC_DIR}/${BIN_NAME} ${TMP_DIR}/${BIN_DIR}

echo "Copying docs..."
cp -r ${DOC_DIR} ${TMP_DIR}

echo "Copying licenses..."
cp -r ${LIC_DIR} ${TMP_DIR}

echo "Copying resources..."
cp ${BIN_NAME}.desktop ${TMP_DIR}/${RES_DIR}
cp ${SRC_DIR}/${IMAGE_DIR}/${BIN_NAME}.png ${TMP_DIR}/${RES_DIR}

echo "Copying scripts and readme..."
sed "s/<ver>/${VER}/g" install_${BIN_NAME}.sh > ${TMP_DIR}/install_${BIN_NAME}.sh
chmod a+x ${TMP_DIR}/install_${BIN_NAME}.sh
sed "s/<ver>/${VER}/g" uninstall_${BIN_NAME}.sh > ${TMP_DIR}/uninstall_${BIN_NAME}.sh
chmod a+x ${TMP_DIR}/uninstall_${BIN_NAME}.sh
cp ../README.md ${TMP_DIR}

echo "Creating archive ${TMP_DIR}.tgz..."
tar -zcf ${TMP_DIR}.tgz ${TMP_DIR}
echo "Removing directory ${TMP_DIR}"
rm -rf ${TMP_DIR}
echo "Done"
```

A.2 `install_inetvis.sh`

This shell script was written in order to install a specific InetVis release on a given system. It was specifically built for Linux, and modern Ubuntu distributions in particular, and may not work correctly on others. It is simply invoked by running the script with no arguments, i.e. `./install_inetvis.sh`. The script will then create a directory in `/opt` for the release, i.e. for the latest version of InetVis it would create `/opt/inetvis-2.1.1`. The

script then creates a symlink of the newly created directory targeting */opt/inetvis*, in order to provide a consistent path for accessing the InetVis binary and resources address releases. The script then copies across the binary file, resources, documents, and licenses to the newly created directory under */opt*. A *.desktop* file is then copied to the relevant location to allow for inclusion in the Ubuntu menu system. A symlink is then created from */opt/inetvis/bin/inetvis* to the users local bin directory, i.e. */usr/local/bin* in order to have *inetvis* be in the *PATH* of the user's system. Finally, the script sets the *cap_net_raw*, and *cap_net_admin=eip* capabilities on the InetVis binary in order to allow it permission to capture local network traffic without running as root.

The script was written in such a way that a user is able to install each new version as it is released, and have the symlinks be updated as necessary, without affecting the installed older versions.

Source Code Listing A.2: Shell script to install InetVis.

```
#!/bin/bash

BIN_NAME=inetvis
VER=""

ICON_PATH=/usr/share/icons/hicolor/48x48/apps/
APPS_PATH=/usr/share/applications/
USR_BIN=/usr/local/bin

echo "Creating directories..."

OPT_DIR_VER=/opt/${BIN_NAME}-${VER}
if [[ ! -d "${OPT_DIR_VER}" ]]; then
    sudo mkdir ${OPT_DIR_VER}
else
    echo "Not creating ${OPT_DIR_VER} as it already exists"
fi

OPT_DIR_SYM=/opt/${BIN_NAME}
if [[ ! -L "${OPT_DIR_SYM}" ]]; then
    sudo ln -s /opt/${BIN_NAME}-${VER} /opt/${BIN_NAME}
else
    echo "Symlink for ${OPT_DIR_SYM} already exists, updating it to ${OPT_DIR_VER}."
    sudo rm /opt/${BIN_NAME}
    sudo ln -s /opt/${BIN_NAME}-${VER} /opt/${BIN_NAME}
fi

echo "Copying files..."
sudo cp -r bin/ /opt/${BIN_NAME}-${VER}
sudo cp resources/${BIN_NAME}.png ${ICON_PATH}
sudo cp -r doc/ /opt/${BIN_NAME}-${VER}
sudo cp -r lic/ /opt/${BIN_NAME}-${VER}

echo "Copying menu entry..."
sudo cp resources/${BIN_NAME}.desktop ${APPS_PATH}

echo "Creating symlink for inetvis..."

if [[ ! -L "${USR_BIN}/${BIN_NAME}" ]]; then
    sudo ln -s /opt/${BIN_NAME}/bin/${BIN_NAME} ${USR_BIN}
else
    echo "Symlink at ${USR_BIN}/${BIN_NAME} already exists."
fi

echo "Setting capabilities on binary to allow monitoring interfaces"
sudo setcap cap_net_raw,cap_net_admin=eip /opt/${BIN_NAME}/bin/${BIN_NAME}

echo "Done"
```

A.3 uninstall_inetvis.sh

This shell script was written in order to uninstall InetVis from a system. As with the installation script, this is tailored specifically toward modern Ubuntu Linux distributions. This script is relatively straight-forward, as it simply removes the *.desktop* file, application icon, the user local bin symlink, the */opt* symlink, and finally the specific releases directory under */opt*, i.e. */opt/inetvis-2.1.1*. Each uninstall script is tailored to remove the version that its related install script was created to install. The uninstall script only removes the aforementioned files if they exist, and prints errors if not.

Source Code Listing A.3: Shell script to uninstall InetVis.

```
#!/bin/bash

echo "Uninstalling_inetvis..."

BIN_NAME=inetvis
VER=""
ICON_PATH=/usr/share/icons/hicolor/48x48/apps
APPS_PATH=/usr/share/applications
USR_BIN=/usr/local/bin

if [ -f "${APPS_PATH}/${BIN_NAME}.desktop" ]; then
    echo "Removing_desktop_file..."
    sudo rm ${APPS_PATH}/${BIN_NAME}.desktop
else
    echo "Not_removing_desktop_file,_it_doesn't_exist"
fi

if [ -f "${ICON_PATH}/${BIN_NAME}.png" ]; then
    echo "Removing_icon..."
    sudo rm ${ICON_PATH}/${BIN_NAME}.png
else
    echo "Not_removing_icon,_it_doesn't_exist"
fi

if [ -L "${USR_BIN}/${BIN_NAME}" ]; then
    echo "Removing_${USR_BIN}_symlink..."
    sudo rm ${USR_BIN}/${BIN_NAME}
else
    echo "Not_removing_symlink_${USR_BIN}/${BIN_NAME},_it_doesn't_exist"
fi

if [ -L "/opt/${BIN_NAME}" ]; then
    echo "Removing_/opt_symlink..."
    sudo rm /opt/${BIN_NAME}
else
    echo "Not_removing_/opt/${BIN_NAME},_doesn't_exist"
fi

if [ -d "/opt/${BIN_NAME}-${VER}" ]; then
    echo "Removing_/opt/${BIN_NAME}-${VER}_directory..."
    sudo rm -rf /opt/${BIN_NAME}-${VER}
else
    echo "Not_removing_/opt/${BIN_NAME}-${VER},_doesn't_exist"
fi

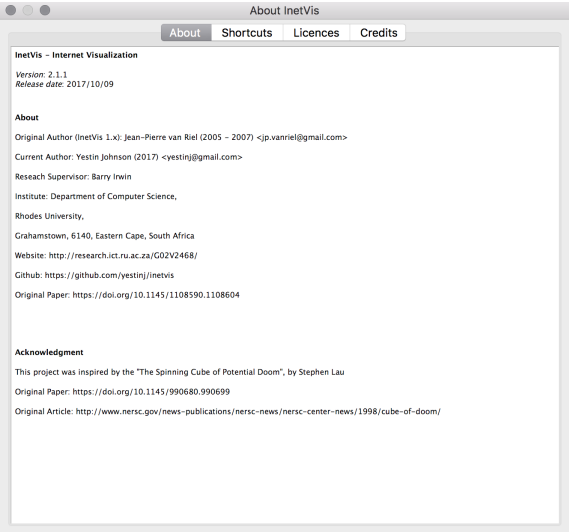
echo "Done!"
```

APPENDIX B

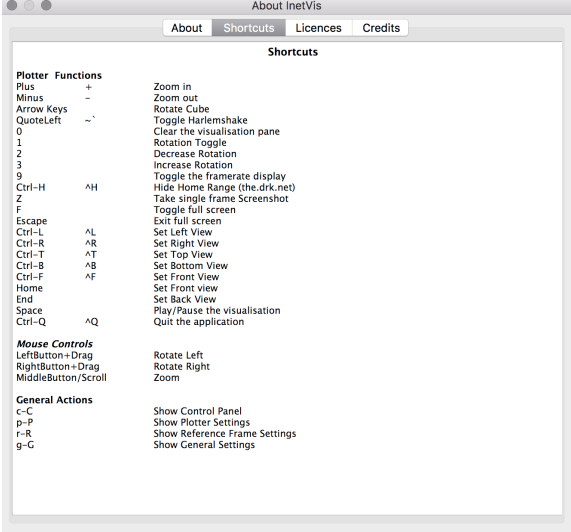
InetVis Screenshots

In this appendix various screenshots of the InetVis user interface are provided in order to provide additional context of how the About dialog box has been updated. The screenshots provided specifically show how the About dialog box has been split from one screen into a tabbed interface, with each tab containing specific information. This is discussed further in Section 4.9.7.

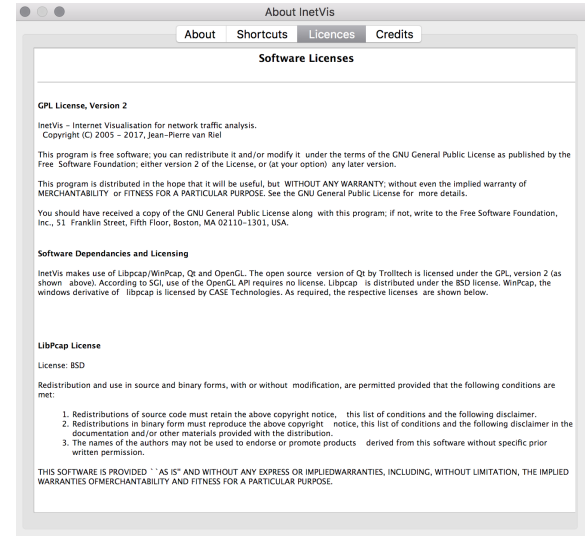
Figure B.1a shows the primary About dialog tab. This is the tab which the About dialog will open to. This figure provides basic information on the application, such as its authors and release information. Figure B.1b contains a listing of the keyboard shortcuts supported by InetVis. Figure B.1c displays the software licenses which InetVis, and its third party components, are available under. Figure B.1d is the credits tab, and consists of special thanks to contributing researchers.



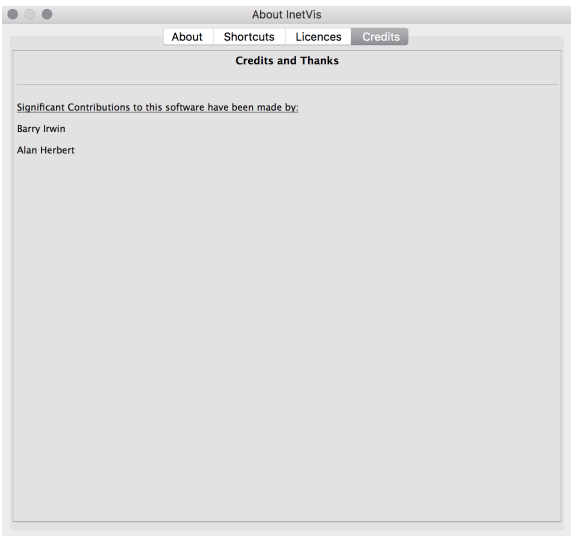
(a) Primary About dialog tab.



(b) Shortcuts About dialog tab.



(c) Licenses About dialog tab.



(d) Credits About dialog tab.

Figure B.1: The updated InetVis About dialog.

APPENDIX C

InetVis User Survey

In this appendix the InetVis user survey, and user survey appendix are included for reference purposes. This user survey was created and completed by participants in order to inform the future development of InetVis. The user survey was discussed in detail in Section 5.3.

InetVis User Survey

Respondent ID: _____

1 Introduction

This user survey, and the follow-up discussion, will be used to inform the remaining development of the InetVis research project, and will not be used for any type of statistical analysis.

Participation in this user survey and the follow-up discussion is voluntary, and participants may withdraw at any time. Should you wish to **withdraw** please send your respondent ID, located near the top of this page, via email to *g16j4842@campus.ru.ac.za* and your data will be expunged and not considered for this study.

Your respondent ID, as well as your responses will be kept private and will only be viewed by the researcher. Only aggregate data from user surveys and follow-up discussions will be reported on.

All questions in the user survey are optional.

This research survey has been approved by the Rhodes university CS/IS ethics committee.

2 Procedure

In order to participate in this study please complete the following in order:

1. Ensure you have read the introduction and are willing to participate in the study.
2. Download the *party pack* from the URL provided via email, this contains all of the files necessary to complete the survey.
3. Extract the *party pack* and proceed to either:
 - 3.1. Examine all of the packet capture files using InetVis (see *party pack* for detailed instructions), OR
 - 3.2. Play the video files which show the examination of the packet captures using InetVIs.
4. Complete the user survey in section 3.
5. Respond to the initial email with your completed user survey.
6. The researcher will schedule a time with you for an in-person or audio/video call to perform the follow up discussion.

3 User Survey

Please complete this user survey only once you have either watched the packet capture files being visualised in InetVis, or you have performed the analysis of the raw packet captures yourself using InetVis. Instructions can be found in the *party pack*.

1. What is your current job role?

2. What do you consider your skill level to be, related to your role?

Not skilled Less than Average Average Experienced Expert
☐ ☐ ☐ ☐ ☐

3. Have you performed any kind of network traffic analysis in the past? Please elaborate.

4. How familiar are you with reading *tcpdump*, *tshark*, or similar console output?

Not at all familiar Not too familiar Neutral Somewhat familiar Very familiar
☐ ☐ ☐ ☐ ☐

Comments:

5. Which image, one or two, do you find more useful, and why? (See separate Appendices document, Appendix A: figures 1 and 2)

6. Questions regarding the provided packet captures:

6.1. Did you run InetVis yourself to analyse the packet captures, or did you make use of the provided videos?

Ran the tool	<input type="checkbox"/>
Watched the videos	<input type="checkbox"/>

6.2. How useful did you find InetVis (if you ran the tool)?

Not useful Partially useful Neutral Somewhat useful Very useful
☐ ☐ ☐ ☐ ☐

6.3. How easy did you find it to use InetVis (if you ran the tool)?

Very difficult Somewhat difficult Neutral Somewhat useful Very easy
☐ ☐ ☐ ☐ ☐

6.4. Did you have any trouble obtaining, installing, building, or running InetVis? If so, can you provide details.

6.5. What computer system(s) did you run InetVis on?

6.5.1. Physical or virtual machine:

6.5.2. Operating system:

6.5.3. Processor:

6.5.4. Memory:

6.5.5. Graphics Card:

6.6. Did you find anything particularly hard to understand, or confusing about the InetVis interface?

6.7. Is there anything in particular that you didn't like about the interface, or the tool in general?

6.8. Please answer the following based on which of the packet captures you considered. From a rating of 1 (Very easily) to 5 (not at all) how well did you identify each of the network scan types in InetVis? Whether or not you actually completed the given scan type, enter Y or N. Refer to Appendix C for a description of each of the listed scan types.

Scan	Completed (Y/N)	Ease-of-identification (1-5)
Nmap slow port scan		
Nmap fast port scan		
Port sweep (network scan)		
ICMP network scan		

Comments: _____

6.9. Do you have any comments or criticisms coming from your use of InetVis to perform this analysis?

7. How does InetVis compare to whichever of these tools you've used? What does InetVis do better? What could InetVis improve on? If you have used any other security data visualization tools please specify which in the table below.

	N/A	Much Worse	Worse	The Same	Better	Much Better
AfterGlow	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ELK stack	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
EtherApe	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Flowtag	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Gephi	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GraphViz	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Netgrok	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PicViz	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
_____	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Comments: _____

8. Please rank the following list of potential features and enhancements, from 1 (most useful) to 10 (least useful), or answer 'X' if you do not find the feature useful at all. If you have any additional comments please also provide them. (A brief description of these potential features can be found in Appendix B).

Additional comments: _____

9. Which are the three features you'd most like to see in InetVis, and why?

InetVis User Survey - Appendices

A Images for Comparison

```
listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
15:23:12.870085 ARP, Request who-has 172.16.0.140 tell 172.16.0.1, length 46
15:23:12.870226 IP 172.16.0.1.51871 > 172.16.0.140.80: Flags [SEW], seq 3468327411, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965776164 ecr 0,sackOK,eol], length 0
15:23:12.870266 IP 172.16.0.1.51872 > 172.16.0.140.443: Flags [S], seq 2873692493, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965776164 ecr 0,sackOK,eol], length 0
15:23:14.784908 IP 172.16.0.1.51877 > 172.16.0.140.1: Flags [S], seq 3526972354, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.785011 IP 172.16.0.1.51878 > 172.16.0.140.2: Flags [SEW], seq 2672341312, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.785064 IP 172.16.0.1.51879 > 172.16.0.140.3: Flags [SEW], seq 2342898345, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.785119 IP 172.16.0.1.51880 > 172.16.0.140.4: Flags [S], seq 2169788251, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.785161 IP 172.16.0.1.51881 > 172.16.0.140.5: Flags [SEW], seq 2687430917, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.785302 IP 172.16.0.1.51882 > 172.16.0.140.6: Flags [S], seq 1923644465, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.785355 IP 172.16.0.1.51883 > 172.16.0.140.7: Flags [S], seq 3232251261, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778060 ecr 0,sackOK,eol], length 0
15:23:14.786217 IP 172.16.0.1.51897 > 172.16.0.140.21: Flags [S], seq 386325287, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778061 ecr 0,sackOK,eol], length 0
15:23:14.786286 IP 172.16.0.1.51898 > 172.16.0.140.22: Flags [S], seq 3393827763, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778061 ecr 0,sackOK,eol], length 0
15:23:14.786352 IP 172.16.0.1.51899 > 172.16.0.140.23: Flags [S], seq 443042688, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778061 ecr 0,sackOK,eol], length 0
15:23:14.786433 IP 172.16.0.1.51900 > 172.16.0.140.24: Flags [S], seq 2966874461, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778061 ecr 0,sackOK,eol], length 0
15:23:14.786490 IP 172.16.0.1.51898 > 172.16.0.140.22: Flags [S], seq 409333027, win 4117, options [nop,nop,TS val 965778061 ecr 47], length 0
15:23:14.786515 IP 172.16.0.1.51901 > 172.16.0.140.25: Flags [S], seq 1474731094, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778061 ecr 0,sackOK,eol], length 0
15:23:14.786572 IP 172.16.0.1.51902 > 172.16.0.140.26: Flags [SEW], seq 2794712027, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778061 ecr 0,sackOK,eol], length 0
15:23:14.791369 IP 172.16.0.1.51953 > 172.16.0.140.77: Flags [SEW], seq 3168185109, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778065 ecr 0,sackOK,eol], length 0
15:23:14.791483 IP 172.16.0.1.51954 > 172.16.0.140.78: Flags [SEW], seq 1399406622, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778066 ecr 0,sackOK,eol], length 0
15:23:14.791522 IP 172.16.0.1.51955 > 172.16.0.140.79: Flags [S], seq 2080257375, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778066 ecr 0,sackOK,eol], length 0
15:23:14.791554 IP 172.16.0.1.51956 > 172.16.0.140.80: Flags [SEW], seq 4134652432, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778066 ecr 0,sackOK,eol], length 0
15:23:14.791585 IP 172.16.0.1.51957 > 172.16.0.140.81: Flags [S], seq 489504555, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778066 ecr 0,sackOK,eol], length 0
15:23:14.791628 IP 172.16.0.1.51958 > 172.16.0.140.82: Flags [S], seq 3649676395, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778066 ecr 0,sackOK,eol], length 0
15:23:14.791730 IP 172.16.0.1.51959 > 172.16.0.140.83: Flags [SEW], seq 1015314043, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778066 ecr 0,sackOK,eol], length 0
15:23:14.792901 IP 172.16.0.1.51968 > 172.16.0.140.92: Flags [S], seq 1945428993, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
15:23:14.792962 IP 172.16.0.1.51969 > 172.16.0.140.93: Flags [S], seq 874860761, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
15:23:14.793060 IP 172.16.0.1.51970 > 172.16.0.140.94: Flags [S], seq 2144408368, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
15:23:14.793107 IP 172.16.0.1.51971 > 172.16.0.140.95: Flags [SEW], seq 1689702629, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
15:23:14.793177 IP 172.16.0.1.51972 > 172.16.0.140.96: Flags [SEW], seq 1662401308, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
15:23:14.793216 IP 172.16.0.1.51973 > 172.16.0.140.97: Flags [SEW], seq 3818398314, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
15:23:14.793269 IP 172.16.0.1.51974 > 172.16.0.140.98: Flags [S], seq 2425130898, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 965778067 ecr 0,sackOK,eol], length 0
```

Figure 1: Tcpdump Output

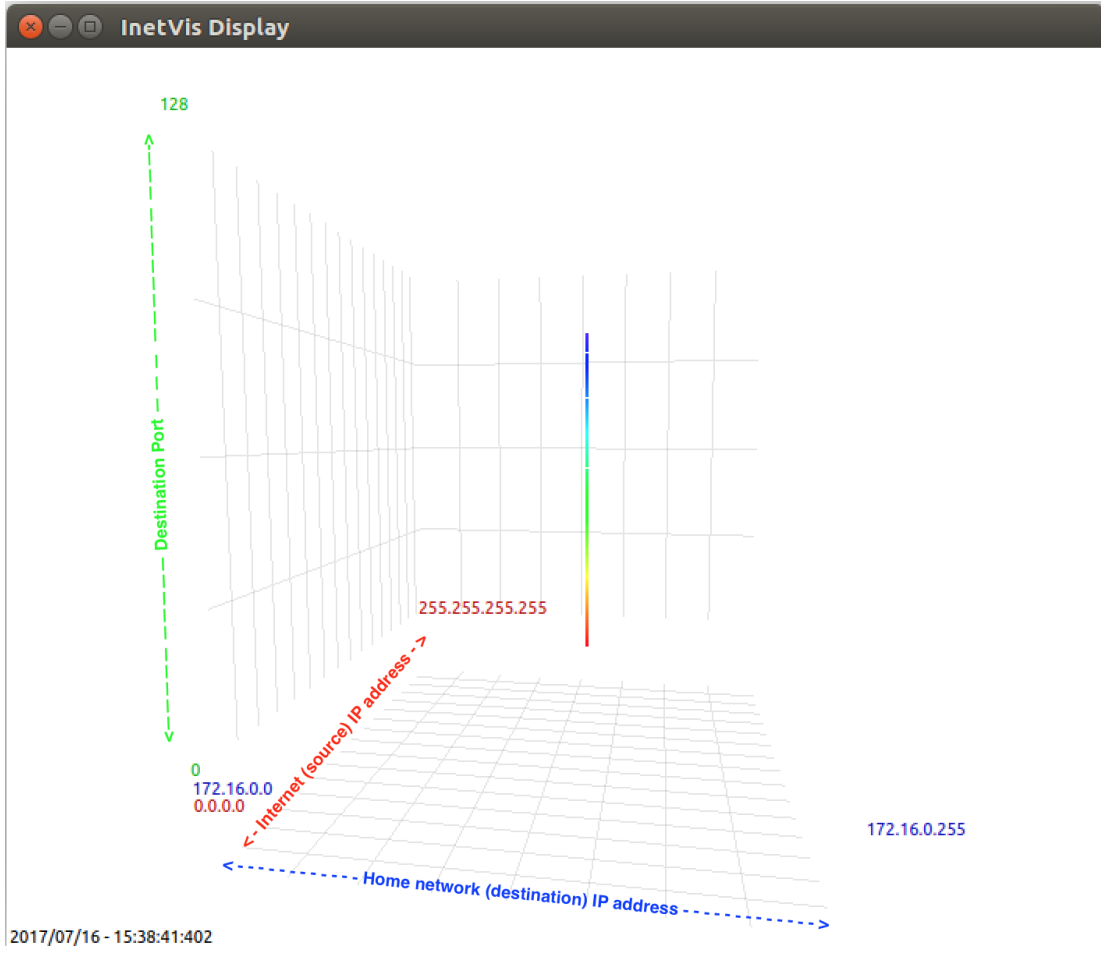


Figure 2: InetVis

B Description of Proposed Features and Enhancements

The following are extensions that are being considered to the existing Inetvis tool.

B.1 A Module Framework

Introducing a module framework to InetVis will pave the way for easier and more flexible future expansion by developers and users. A module framework will be introduced that hooks into various aspects of the system, such as when certain events occur, and allows the user to write a module that can override and/or augment the standard logic of said event.

For example, this could allow the user to write a simple module that would change the way in which events are drawn on the screen, such as change the co-ordinate system, colour scheme, offset, etc... Allowing users to do this in self contained modules, and not in the core business logic of the application greatly improves the simplicity of the action.

The module framework could also be used for functions such as displaying various custom heads-up display (HUD) widgets, and so much more.

B.2 Update to the Latest Version of OpenGL

InetVis was last updated in 2007, and thus could only make use of the OpenGL functionality available at that time. OpenGL has improved drastically in the last ten years, and there is much potential for the code to be updated to make use of the new features of the framework. Improvements that could be implemented are improved OpenGL video output, and various other improvements to the InetVis cube visualisation.

B.3 Alternate Input support

In order to more efficiently make use of InetVis, by improving cube navigation, toggling options, playback speed, and so on, having support for alternate forms of input would be useful.

This could take the form of support for joysticks and gamepads, allowing for much easier navigation of the cube, as well as for the hardware buttons on these devices being mapped to the most useful functionality provided by InetVis.

B.4 Option to Colour / Highlight events by Filter Match

Given the vast amount of information that InetVis is able to display at one time, having some way of colouring or highlighting network events based on a filter would be very useful. This would most usefully be implemented using BPF filters, which are already supported by InetVis for other filtering.

With this feature it would be possible to colour events based on their netblock, AN, or any other option that can be expressed by using a BPF filter.

B.5 A Heads Up Display (HUD) with Widgets

The InetVis display cube displays all the basics of the information obtained from packet captures, however there is still some very useful information that can be obtained from the source data, that is not visualised. If this feature were to be implemented it would allow for various new types of information to be displayed on screen together with the cube.

Examples of this information include: a graph of the number of packets processed per second, the rate of incoming packets, spark lines displaying useful information. etc...

Another candidate for a heads up display widget is to have a popup window appear when mousing-over of a network event. This would allow details of the event to be displayed, without having to resort to manually exploring the packet capture file.

B.6 Overhaul of the UI and Colour Scheme

This feature would involve doing a complete review and potential overhaul of the application UI, basing the new design on solid UI principles, and taking advantage of the new UI and UX features present in Qt 5. These features include improved options for screen layout, such as tabbed interfaces, which could be used to supplement the current UI.

This will be done with the aim of improving the usability and user experience of the application.

B.7 Support for Video Output

At present InetVis only supports taking a snapshot of the display cube, as well as exporting a frame capture for each frame, as a packet capture is replayed. It would be very useful for InetVis to have the capability of exporting the replay of a packet capture file to video. This would make further review and demonstration of the analysed data to be more useful.

This feature ties in somewhat to the feature for updating to take advantage of the latest version of OpenGL, although this feature is not dependent on the update.

B.8 A Server / Client Architecture

The idea behind this extension is to split the InetVis application up into a client and server components. This will allow for much more flexibility, as well as much improved performance improvements that will be gained by decoupling the UI from the backend packet processing.

The server component could run on a powerful server, potentially in the cloud, and would take advantage of the increased processing power to process large packet captures, or to monitor live network interface/s that are receiving large quantities of traffic. An efficiently compressed event stream would be generated by the server and made available to clients.

The client application would then connect to the server, receive the event stream, and then visualise the events.

Another useful option here would be to have multiple clients supported by a single server instance. This could allow an incident response, or blue team members to all monitor the same interface, potentially filtering on and focusing on different aspects.

B.9 Ability to Dump All Events on Screen to a Packet Capture File

It is currently possible to dump all of the network packets received, or processed from a packet capture, into a new output file. This function works in an on/off mode, and thus does not allow easy output of all of the events that are currently on screen.

Having this functionality would allow the user to easily explore any events of interest that are currently on screen.

B.10 Support for Time-Series Compressed Files

This feature would allow very large packet captures to be loaded into InetVis seamlessly with improved performance. This can be accomplished by adding support for a time-series compressed packet capture format that was devised by Nottingham in his PhD research¹. This is described in detail in the GPU accelerated protocol analysis for large and long-term traffic traces section, page 122 shows the Index and Time index format used.

By implementing this feature it will enable InetVis to visualise much larger packet captures than it is currently able to, and while memory is not as much of an issue nowadays, this feature would further remove physical memory from the performance equation.

C Scan Types

C.1 Nmap Slow Port Scan

A port scan, also known as a ‘vertical scan’, is a probing technique whereby a single source address targets a single destination address on several different destination ports. In essence an attacker focuses their energies on a single host, and probes all, or a subset, of the destination ports on said host.

An Nmap slow scan is achieved when Nmap is run using the `-T 0-5` command line option. This command line option tells Nmap to use a certain timing template². `-T0` represents the slowest possible timing, and `-T5` represents the fastest possible scan³.

-T paranoid—sneaky—polite—normal—aggressive—insane

`-T0` waits five minutes between sending each probe, `-T1` 15 seconds, and up to `-T5` being done in parallel and much, much faster.

In this case the `-T1` option was used to adequately show the results of a *slow* scan.

¹<https://github.com/anottingham/PhdResearch>

²<https://linux.die.net/man/1/nmap>

³<https://nmap.org/book/man-performance.html>

C.2 Nmap Fast Port Scan

A port scan, also known as a ‘vertical scan’, is a probing technique whereby a single source address targets a single destination address on several different destination ports. In essence an attacker focuses their energies on a single host, and probes all, or a subset, of the destination ports on said host.

An Nmap fast scan is achieved when Nmap is run using the `-T 0-5` command line option. This command line option tells Nmap to use a certain timing template ⁴. `-T0` represents the slowest possible timing, and `-T5` represents the fastest possible scan.

-T paranoid—sneaky—polite—normal—aggressive—insane

In this case the `-T5` option was used to adequately show the results of a *fast* scan.

C.3 Port Sweep (Network Scan)

A port sweep, also known as a network scan, is a probing technique whereby a single source address targets multiple destination addresses using a particular destination port. In essence an attacker scans an entire network, but only checks a certain destination port on each host within the network.

C.4 ICMP Network Scan

An ICMP network scan can be defined as a single source address probing multiple destination addresses by making use of the ICMP ping protocol.

⁴<https://linux.die.net/man/1/nmap>

APPENDIX D

InetVis Videos

In the user survey, videos were created giving survey participants a brief introduction, and then a more detailed UI overview of InetVis. Videos were also created to show the loading and visualisation of the sample packet capture files, showing various types of network scans in order to allow participants who did not wish to run the software themselves, or who had difficulty doing so, the opportunity to see how it functions, and how it visualises the given packet captures. These videos are public on a YouTube channel specifically for the Internet Visualisation project, and the relevant videos are referenced, together with URLs to their locations on YouTube, in Table D.1.

Table D.1: Overview of source code structure.

Video Name	URL
InetVis Introduction	https://youtu.be/VB5sB7tXgsU
InetVis UI Overview	https://youtu.be/m9ZiLZoohpI
Nmap Polite Scan Visualisation v2	https://youtu.be/qq3vM1au3F0
Nmap Insane Scan Visualisation v2	https://youtu.be/jX_XHUj5BDI
Nmap Port Sweep Visualisation v2	https://youtu.be/RcrcuVmARgY
Custom ICMP Scan Visualisation v2	https://youtu.be/IUIENQ4wcj8