An Exploration of Geolocation and Traffic Visualisation Using Network Flows to Aid in Cyber Defence

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Sean Niel Pennefather

Grahamstown, South Africa
November 1, 2013

Abstract

A network flow is a record that represents the characteristics associated with a unidirectional stream of packets between two hosts using an IP layer protocol. As a network flow only represents statistics relating to the data transferred in the stream, the effectiveness of utilising network flows for traffic visualisation to aid in cyber defence is not immediately apparent and needs further exploration. The goal of this research is to explore the use of network flows for data visualisation and geolocation.

A prototype system capable of collecting network flows exported using the NetFlow version 9 protocol designed and was implemented as part of this research to aid in this exploration. This system processes the collected flow records and renders the geolocated results on an interactive map in a web browser.

Using conformance testing it is shown that the prototype system is capable of collecting network flows and generating geolocated flow events in 50 milliseconds on the test platform. The system also provides functionality for the generation of heatmaps and tools for replaying flow events from the client browser for further visual analysis. A reporter tool has also been developed to produce monthly reports on the collected network flows.

Acknowledgements

As the writer of this research paper, I would like to acknowledge the support received during this research. I would first like to give thanks to my supervisor Professor Barry Irwin as his guidance and support has been essential to the success of this research.

I am deeply indebted to my family for their continued love and support throughout the course of this year, their care and aid has allowed me to focus on the completion of this thesis.

I would also like to extend thanks to Mr John Richter for his guidance and support during this research.

I would like to thank the NRF and Rhodes University for the financial support that allowed me to complete this research. Finally I would like to acknowledge the financial and technical support of Telkom, Tellabs, Stortech, Genband, Easttel, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

This research makes use of GeoLite data created by MaxMind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Network flow processing has the potential to allow for a large reduction in the volume of data to be processed by monitoring systems when compared to traditional packet processing counterparts. The reason for this reduction in volume is that a network flow is a single record that represents the characteristics associated with an instance of communication between two hosts using an IP layer protocol (Morken, 2010). A flow record does not record the actual data transferred and as a result, the record size is only dependent on the number of characteristics the record must report on rather than the number of packets transferred for the duration of the connection.

This allows network flows to be used to reduce the volume of data that must be processed. This reduction comes at the cost of not recording the actual content of the packets that make up the connection which are required by systems that employ packet analysis techniques as part of processing (Proctor, 2001). Because of this reduction in resolution, the effectiveness of utilising network flows for traffic visualisation to aid in cyber defence is not immediately apparent and needs further exploration.

1.2 Research Goals

As companies expand, supporting networks must evolve to accommodate the resulting traffic requirements. These requirements include support for increased traffic volume and transmission speeds between internal and external networks. This increased volume still

needs to be monitored for both the health of the internal network and its connection to external parties. Though this can continue to be done using purely packet based processing and logging, the amount of memory and processing required increases. The increasing demand for more system resources result in companies needing to purchase more hardware to maintain the current traffic monitoring system.

The aim of this research is not to discount the credibility of continuing to use packet based processing for traffic visualisation and network security but rather to explore the potential of using network flows to achieve a similar goal. The exploration will cover two areas of research:

- An exploration into the use of network flows for traffic visualisation to aid in administration and network security.
- The use of network flows for real-time geolocation and coordinate rendering.

Due to the nature of how network flows are constructed, the default behaviour of the responsible hardware is to report on the flows only after the communication is complete which is discussed in section 2.3.3. This can be overridden so that the reporting device reports on active flows in set intervals rather than waiting for them to complete. As this time can be set, the timing tests will not record the duration from the initialisation of a flow to that flow being exported to the system.

Though geolocation and coordinate rendering falls under data visualisation, it is significant enough to warrant mentioning as host addresses can be exported as a characteristic of a flow. A geolocation database can then be used to convert a host address into geographic coordinates.

In order to explore the feasibility of Network Flows in the above applications, it is necessary to develop a system that is capable of performing both geolocation and other techniques for visually representing stored network flow records. This system must be capable of reading in raw Network Flow packets as they are received from the exporter and handling flow record aggregation as well as representing both the record characteristics and geolocation results visually to the system user.

1.3 Research Scope

Defining the research scope is important to formally describe the problems that are investigated in this research. The investigation will use the IPv4 address space and not the IPv6 address space. The results produced in this research with regards to flow geolocation and data visualisation wilzl be the same for both IPv4 and IPv4. However, to limit complexity and due to the lack of quality IPv6 golocation databases, IPv6 will not be supported.

The system implemented to perform this research is not capable of sending or receiving NetFlow options templates or NetFlow options records. NetFlow options templates and NetFlow options records form part of the NetFlow version 9 protocol and are discussed further in chapter 2. The reason for this is that adding the additional functionality to the system does not effect the research performed. NetFlow options records are used to send information regarding the actual process used to generate the network flows which is not required to achieve the goals described in section 1.2 (Cisco, 2011a).

The research will be host based rather than network based. Network based geolocation and data visualisation require maintenance of multiple source IP addresses and the produced results will need to be generated for each source address. Furthermore, flow exporting system used in this research is Softflowd[1] which is host based and exports flows only for the source IP address on the platform on which it is running.

The results of data visualisation will focus on identifying the type of visualisation that can be performed using network flows. As data visualisation is largely dependent on specific requirements of the user, the results produced for this research are a proof of concept.

For determining the applicability of a realtime system utilising network flows, timing concerns will focus on network flow generation and flow processing by an implemented system. Timings regarding communication between the Server and Client will be considered out of scope due to its reliance on the connecting network path.

1.4 Research Approach

The approach taken to achieve the goals outlined in section 1.2 is to design a prototype system capable of flow geolocation and data visualisation. The designed system is then

[1]Softflowd is a software based flow export system (Miller, 2011)

implemented and tested to insure suitable functionality.

1.5 Document Structure

- An in depth investigation of network flows is covered in chapter 2 which includes a discussion regarding the protocols used to export raw flows from the flow exporting system. Geolocation and data visualisation is discussed along with network treats and common intrusion techniques. An investigation into network flow exporting systems completes the chapter.
- A prototype system is proposed and designed in chapter 3. This chapter identifies goals the proposed system must achieve and the design is partitioned into the components necessary to achieve them.
- Chapter 4 focuses on the system implementation. This chapter notes techniques used to implement the functionality described in chapter 3 as well as identify and promote the selected language for implementation.
- Conformance tests are designed and carried out in chapter 5 to test if the implemented system is capable of successfully performing flow geolocation. Synthesised data is used to time the system components when completing necessary tasks. The data is also in generating statics of the generated system traffic. The tests are followed by an evaluation of other data visualisation aspects of the system.
- Findings of this research are concluded in chapter 6 and is followed by a discussion of future research.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides the necessary background information regarding industry leading protocols for exporting raw flow data. These protocols are used by high level switching or routing devices to export raw flow data to target systems within a closed network. The term 'network flow' is discussed in terms of the definition provided by Cisco which is a unidirectional sequence of packets that all have seven characteristics in common (Cisco, 2012c). These seven characteristics are listed in table 2.1 and are used to define the active flow that a packet belongs to.

Three network flow protocols are discussed in this chapter. As protocols NetFlow version 5 (Cisco, 2006) and NetFlow version 9 (Claise & Systems, 2006) are both provided by Cisco, they are compared first in section 2.2. The comparison is done to highlight the differences between the two versions which includes the advantages and disadvantages of using each. Following the Cisco protocols, literature relating to IP Flow Information eXport (IPFIX) (Claise *et al.*, 2013) is reviewed in section 2.4. The review of each protocol includes a discussion about packet structure and implementation.

This chapter includes a discussion on port scanning in section 2.8 which highlights common techniques used to scan the network ports of an addressable host. This followed by an investigation into worms and denial of service (DoS) attacks. Different systems that are capable of collecting network flows are investigated in section 2.5. The investigation includes a brief overview of each system and how it is used to process collected flows.

Finally, a brief overview of geolocation is discussed in section 2.6 and is followed by a

Table 2.1: Seven Characteristics of a Network Flow

Source IP Address
Destination IP Address
Source Port
Destination Port
Protocol
ToS Byte
Interface

discussion of data visualisation. This chapter concludes in section 2.11 with a summary of the material covered.

2.2 What is a Flow

The concept of a Network Flow was patented by Kerr and Bruins on 28 May 1996. The concept was created as an efficient means to report on network status and traffic patterns as seen by routing devices for a related sequence of packets. Initially, a flow was defined as a set of packets all destined for the same destination IP address and all originating from the same source address. Further identification of a unique flow included the requirement that all packets have the same destination port. (Kerr & Bruins, 1996)

According to Cisco (Cisco, 2012c), a flow is defined as a unidirectional sequence of packets between two end hosts over a network. Each packet in the sequence must display the same 7 characteristics shown in figure 2.1 to be considered part of a single network flow. The direction of the flow is determined by the host that began the communication.

This flow data which is generated by routing or switching devices such as those made by Cisco (Cisco, 2012a) and Juniper (Juniper Networks, 2013). The generated data can then be transferred to other devices for analysis to help identify potential network faults and monitor resource use, typically for billing purposes. This data can further be analysed to only display information pertaining to a particular network mask, a particular date or time, and overall resource use.

The construction of the raw flow data is traditionally done at hardware level in compatible routing or switching devices. Examples of compatible routers include those shown in table 2.2 which are supplied by Cisco and JunOS routers supplied by Juniper (Scheck & CSIRT, 2009). As it is already necessary for such devices to analyse received packets for routing

Table 2.2: Example NetFlow compatable routing devices (Cisco, 2012c)

Platform Name	NetFlow Export Version(s)
Cisco 12000	v5 v8 v9
Cisco 800	v9
Catalyst 65k/7600	v5 v8 v7

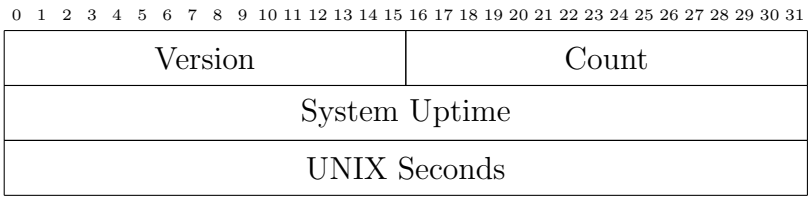


Figure 2.1: Standard NetFlow packet header

purposes, a hardware level approach allows the device to generate the flows using the unpacked packets. Characteristics recorded include the basic components shown in table 2.1 as well as any additional information that the router is configured to collect such as packet length and count.

In order to then transfer the recorded data collected on observed flows to another device for processing, a raw flow exporting protocol is employed. The exporting can occur when the generating system concludes that a flow has expired or in set intervals regardless. Currently, the significant raw flow export protocols are those developed by Cisco called NetFlow version 5 and NetFlow version 9 (Cisco, 2011a). Additionally, a new protocol is currently under development by the Internet Engineering Task Force (IETF) called IPFIX (Claise *et al.*, 2013). Though still in development, implementations of the protocol are currently in use by network components such as the Barracuda NG Firewall[1].

2.3 NetFlow

All versions of the NetFlow protocol currently available use the same packet header format for representing version, count and timings. This helps developers to produce collector software capable of handling different NetFlow protocols. Figure 2.1 shows the format of the first 96 bits of a packet header. The count field indicates the number of flows that are contained within the packet with the exception of NetFlow version 9 where this field indicates the number of flow sets contained (Cisco, 2011a). The concept of a flow set is

[1]Only versions 5.2.3 and above are IPFIX compatible (Barracuda, 2013)

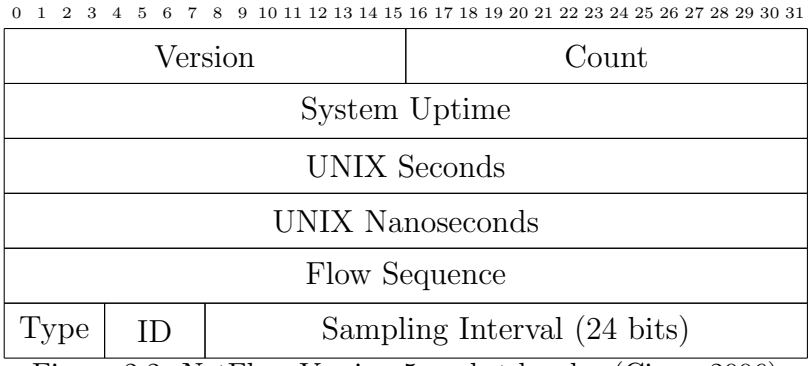


Figure 2.2: NetFlow Version 5 packet header (Cisco, 2006)

explained in more detail in section 2.3.2.

2.3.1 NetFlow Version 5

NetFlow version 5 is currently the most widely used protocol that is developed by Cisco for exporting raw flows from routing devices to the collector (Lee *et al.*, 2010).

Figure 2.2 shows the full NetFlow version 5 packet header which consists of 9 fields and is 24 bytes long. The first 12 bytes of the header that are common to all NetFlow protocols versions and shown above in figure 2.1. This is followed by the residual nanoseconds since the first day UNIX which is defined in RFC 5905 to begin on 1 January 1970 (Mills *et al.*, 2010). The flow sequence number is a running count of the number of flows seen by the export device. The Type and ID fields in figure 2.1 are used to associate the packets exported with a specific export device on the network. The final 3 bytes in the sequence are reserved for the sampling interval where the first 2 bits determine the sampling mode and the remaining 14 bits hold the value.

The Input and Output fields in figure 2.3 are the Simple Network Management Protocol (SNMP) index numbers. The SNMP protocol is responsible for communicating management variables between the managing devices and host as defined in RFC 1157 (Case *et al.*, 1990).Both the First and Last fields in figure 2.3 hold a time value in milliseconds since the recording device was booted. The values are timestamps of the first and last packets routed in the flow.

Next are the fields Source As and Destination As are Autonomous System Numbers (ASN) which are used to identify the Autonomous System that each end host resides in. The Autonomous System (AS) is described in RFC 4271 (Rekhter *et al.*, 2006) as a collection of routers that use an Interior Gateway Protocol along with set metric rules to determine how to route packets. Though the routers may internally use a variety of metrics to

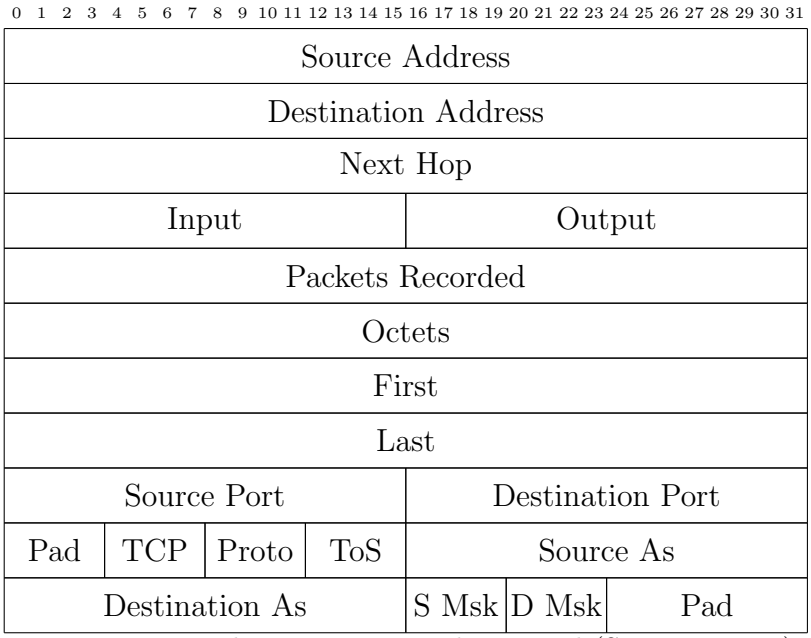


Figure 2.3: NetFlow Version 5 packet record (Systems, 2004)

determine the routing paths between different internal components, the AS is seen to use a single routing algorithm externally.

Two disadvantages of using NetFlow version 5 over its successor variants are that it does not support IPv6 and the structure of exported packets is static (Systems, 2004). IPv6 has become increasingly prevalent as services such those provided by Google are now accessible using IPv6 addressing[2]. Flow records exported under the NetFlow version 5 protocol are limited to only exporting data in the defined fields of the record and cannot change during system runtime. Both of these issues are addressed in NetFlow version 9.

2.3.2 NetFlow Version 9

NetFlow version 9 is a raw flow export protocol that dynamically structures the contents of the exported flow data records according to a previously exported template (Cisco, 2011a). This allows collecting systems to process NetFlow version 9 data packets without knowing the format of the contained records prior to template lookup. The benefit of using this protocol is that it allows the characteristics exported to be changed without restarting either the exporter or collector systems. A result of this is that current NetFlow collectors and parser algorithms will not have to be recompiled to use a new packet structure when when it becomes necessary to export a new characteristic from the export device. Another benefit of using a template based system is that companies can configure export devices

[2]statistics available available at http://www.google.com/intl/en/ipv6/statistics.html

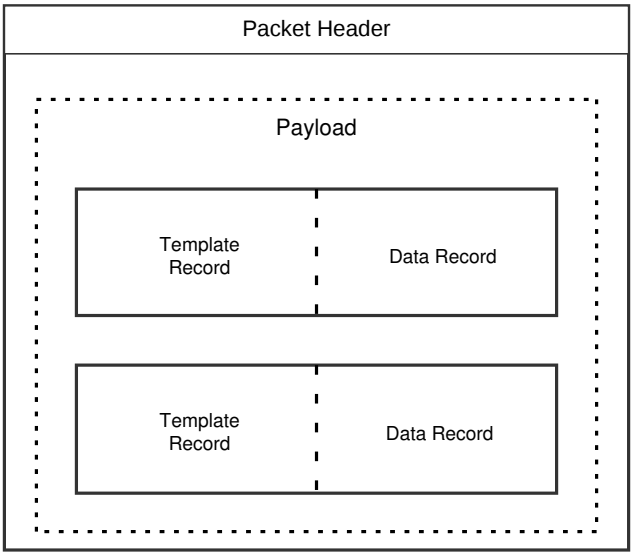


Figure 2.4: NetFlow version 9 Packet Format (Cisco, 2011a)

to export flow records in a format that is optimal to their needs as well as modify them at a later stage without implementing a new protocol.

Figure 2.4 shows a packet exported under the NetFlow version 9 protocol. Th packet begins with a packet header which is followed by a payload. The payload can be made up of any combination of template and data records provided it contains at least one record.

According to Cisco feature guides, NetFlow version 9 is independent of the implemented transport protocol being used to export the packets. Protocols used include UDP, TCP and SCTP as described in NetFlow configuration guide on exporting via SCTP (Cisco, 2012b).

Currently, NetFlow version 9 exports two types of records which are flow records and options records (Cisco, 2011a). Flow records contain information regarding the generated flows and make up for the majority of the records contained in the packet payload. Option records are used by the export device to export additional information regarding the raw flows or the current configuration of the device. This can include the sampling frequency and the algorithm used to generate the flows as well as the number of flows observed.

Both record types consist of template and data records. The format of the data records is determined by a template which is used to pack traffic characteristics into the record. As a result, template records must be received by the system before any dependant data records can be processed. This also implies that the collector needs to maintain a buffer

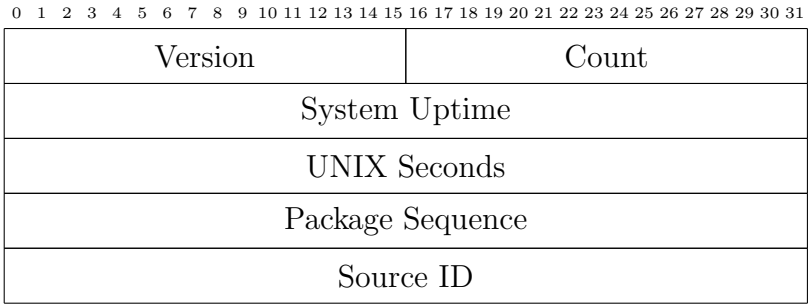


Figure 2.5: NetFlow version 9 Packet Header (Cisco, 2011a)

of received templates which can then be used to parse the corresponding data records. This adds increased complexity when it comes to flow collection as this additional buffer is unnecessary for other protocols.

Packet Header

Figure 2.5 depicts a NetFlow version 9 header which and contains initial data regarding the packet type and the originator. The interpretation of the count field in NetFlow version 9 differers from previous versions of NetFlow. In NetFlow version 9, the count field indicates the total number of records that form the body of the packet rather than the number of flows. This is used to assist the collector in parsing the packet content. (Cisco, 2011a)

The Source ID in figure 2.5 is used to uniquely identify each flow packet that is exported from a specific export device[3]. This allows collector software to combine the source IP address with this ID to uniquely distinguish every NetFlow record currently being exported on the network. Following the header of the packet, the remaining packet components are optional provided that at the packet contains at least one data or template record. (Cisco, 2011a)

Data Template Record

The template record is dynamic in size which allows for the number of characteristics exported using it to differ between templates. An example of this shown in figure 2.6 where the first 64 bits of a template record are compulsory. In order to be dynamic, the Length field contains the total length of that packet in bytes. It is important to note that this length refers to the entire template and so includes the length of the ID and Length

[3]The format of this value is vendor specific

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
FlowSet ID	Length
Template ID	Filed Count
Field 1 Type	Field 1 Length
Field 2 Type	Field 2 Length
::	
Field N Type	Field N Length

Figure 2.6: NetFlow version 9 FlowSet Template Format (Cisco, 2011a)

field. This value can then be used by the collector to determine the end of the current template and the beginning of the next record. The FlowSet ID must contain a value greater than 255. Values between 1 and 255 indicate an options template and the value 0 is reserved. (Cisco, 2011a)

After exporting a template, the exporting device can then export the data FlowSets formatted according to the template exported. In order for the collector to identify the template used to format the received data, the data set must advertise the ID of the template used for formatting. This ID is the value held in the FlowSet ID field.

The body of the template is made for a series of field types followed by the corresponding field length. The field type is a value that corresponds to a table of defined characteristics that NetFlow version 9 is currently capable of exporting. There are 128 possible values that the field type can be assigned to, each of which represents a flow characteristic currently supported by the protocol. A full listing of the possible characteristics is provided in appendix A as only a few characteristics will be highlighted here. (Cisco, 2011a)

Though most values correlate to a predetermined characteristic of the traffic seen, there are a few notable exceptions. The values 43, 51, 65 69, and 87 are associated with a flow characteristic unique to the vendor of the export device and so are not defined in the protocol. Values in the range 105 to 127 are currently reserved by Cisco for future expansion of recorded characteristics. The value has been 100 depreciated by Cisco (Cisco, 2011a). Table 2.3 lists some the more common characteristics exported in a data record. These characteristics make up for six of the seven requirements of a network flow tabulated in table 2.1.

Other fields of particular interest include the value 1, which details the total number of bytes transferred in the flow, and 2 which details the total number of packets transferred in the flow. These two fields can then be used to give the flow analyser an idea of the

Table 2.3: Common characteristics exported

Values	Characteristic
4	IP layer protocol used
5	Incoming Type of Service (ToS)
7	Source port
8	Source IP address (IPv4)
11	Destination port
12	Destination IP address (IPv4)
21	Time when first packet was switched
22	Time when last packet was switched
55	Outgoing Type of Service (ToS)

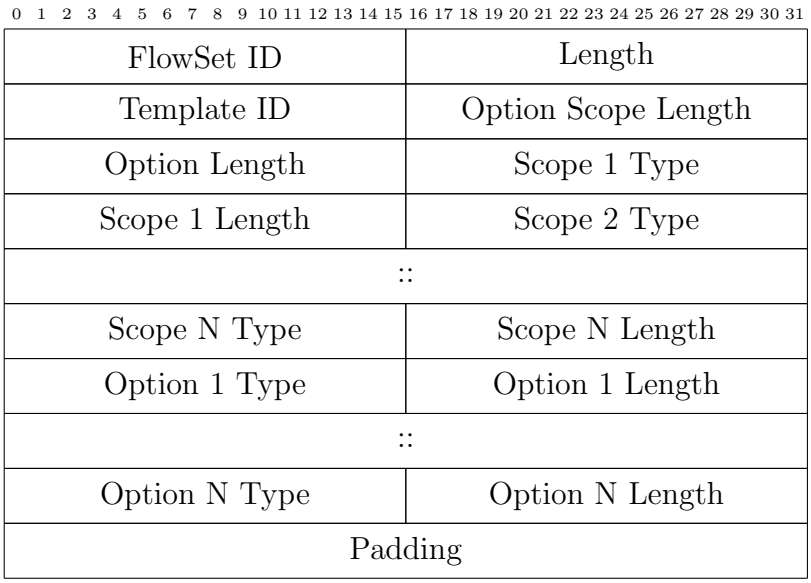


Figure 2.7: NetFlow version 9 Options Template Format (Cisco, 2011a)

average packet size and the amount of data transferred in that particular flow. The values 21 and 22 refer to the system uptime when the first and last packet was processed. These values can be used to determine the duration of a network flow. By using the volume transferred and the duration, the processing system can determine the average bandwidth of the network flow.

Options Template Record

As in the case of the data template, the options template provides the format used in exporting information regarding the current configuration of the routing device as well as other meta data.

The possible fields that the options data record can be formatted to export are vary similar to a flow data record. The difference between the two records is that the options data is focused on meta data regarding the current configuration of the device and the protocol. In order for the options template to specify what component of the system the fields exported relate to, each field falls into a scope which is also specified in the template. The scope as shown in figure 2.7 precedes the options type and length fields.

2.3.3 NetFlow Export timings

When a packet is routed or switched by a system generating network flows, packet characteristics are logged. The system uses the seven characteristics described in table 2.1 to determine the flow the packet is associated with. If no flow exists and the flow cache is not full, this packet begins a new flow. Once created, the flow exists until it is terminated by the system. To determine whether a flow is to be terminated, NetFlow employs three rules which are used to terminate flows. The first rule applies to TCP connections and requires that a last packet processed for a flow is a FIN packet[4]. The second rule is that a no traffic for a particular flow has been seen for a configurable amount of time, this is the primary method of termination UDP flows. The third rule is implemented to prevent stagnant flows from developing and simply enforces that a flow be terminated after a configurable amount of time, regardless to activity. (Estan *et al.*, 2004)

An alternative is to configure the exporting system so that it periodically reports on network flows regardless of whether they have been terminated or not. This periodic reporting allows the reliant system to define a maximum latency between then a flow begins and when it is reported. This approach does result in a similar latency being defined for knowing when a UDP flow has terminated[5]. This arises from the dependant system needing to wait for the next periodic announce from the export system to detect if the UDP flow is not reported.(Cisco, 2011b)

2.4 IPFIX

IPFIX is an attempt to create a standardised protocol for the export of raw flow records from an export device to a collecting device. IPFIX is based on NetFlow version 9 with

[4]A TCP packet with the FIN flag set to high.
[5]For TCP flows, the flag characteristic can be monitored for a FIN flag.

formal specifications being outlined in Request for Comments (RFC) 3917 (Zseby *et al.*, 2004). IPFIX is well developed in terms of its specification and possible applications in industry.

The IPFIX protocol employs a dynamic packet structure consisting of data and template records. Though similar to NetFlow version 9, IPFIX extends the number of characteristics that can be exported from 128 to 239 (Quittek *et al.*, 2008). As with NetFlow version 9, the collector requires that the template is received before any data formatted according to that template is received.

2.4.1 Transmission Protocols

IPFIX is designed to be independent of the underlying transport protocol used which means it can be transmitted using IP layer protocols such as TCP, UDP, and SCTP. Though all three protocols are a viable choice for transmission, should the transmission path be susceptible to congestion, it is suggested in RFC 5101 (Claise, 2008) that SCTP be used rather than the other two protocols due to the protocol's congestion avoidance capabilities.

2.4.2 Extended Characteristics

The extension to the number of exportable characteristics allows for more detailed information on flows to be exported. Additional characteristics include IP and TCP header lengths and the reasons for flow termination. Other interesting additions include an indicator of whether the packets being routed are part of a multicast address and total counts for TCP flags such as SYN or ACK.

The IPFIX protocol also allows for the export of statistical data about the volume of data processed such as the total number of flow records that have been observed by the export device since it was last booted. Another useful characteristic of the exporter that can be exported using this protocol is the total number of flows that were observed by the router but for some reason, were not exported to the collector. These reasons can include configurations on the exporter to not export flows that exhibit particular characteristics or simply that there was insufficient space to store the record prior to exporting.

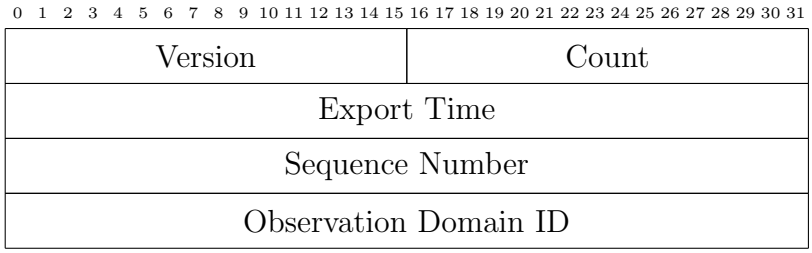


Figure 2.8: IPFIX Packet Header Format (Trammell *et al.*, 2009)

2.4.3 Security Requirements

The IPFIX protocol also has security requirements as mentioned in RFC 3917 (Zseby *et al.*, 2004) to help maintain data confidentiality and integrity which is expanded upon in RFC 5101 (Claise, 2008). There are three broad requirements that the protocol must achieve according to the security specifications. The first requirement is that a mechanism must be provided to maintain the confidentiality of the raw exported flows as they are transported from the exporter to the collector so that sensitive information cannot be be disclosed by intercepting IPFIX packets. Secondly, the integrity of the data must be maintained during the transfer so that information received by the collector can be considered reliable. Lastly, the protocol must include authentication of both exporters and collectors to prevent unauthorised exporters from exporting rogue flows and unauthenticated collectors from receiving legitimate flows as this would break the first security requirement.

2.4.4 Packet Structure

The format of a IPFIX packet is the same as a NetFlow version 9 packet as shown in figure 2.4. The same requirements apply that any combination of headers and template set are allowed in the packet provided the condition that there is at least one header or dater set in the payload is met.

The header of an IPFIX packet differs from the NetFlow version 9 header in figure 2.5 as the System Uptime field has been replaced with the export time of the packet. This export time is the time in seconds since the Coordinated Universal Time (UTC) [6] and is stamped as the packet leaves the exporting device.

The Observation Domain ID field indicates the observation domain [7] that the flow data

[6] A formal discussion of the UTC can be found in RFC 3339 (Klyne & Newman, 2002)
[7] An observation domain is further discussed in RFC 5101 (Claise, 2008)

is exported from. An observation domain consists of one or more observation points which are interfaces that flow recording takes place on. The constructed flows for each observation point can then be aggregated by a metering process for that observation domain. Characteristics of these flows and metering results are then exported from one exporter per domain to the collectors. Observation Domains allow for a single collector to easily identify the segment of a large network the flows being received are being exported from.

2.5 Network Flow Collectors

To collect and process network flows exported by compatible routing or switching devices, it is necessary to have an application that is capable of receiving packets exported under a network flow export protocol. This application should then parse the packets to extract the exported network flows. Three different collector applications are discussed in this section.

2.5.1 Flowd

Flowd is an open source NetFlow traffic collector developed by Miller[8] . The application is currently on release 0.9.1 which was made publicly available on 25th July 2008. flowd-0.9.1 is capable of collecting traffic transmitted under the NetFlow protocol. The versions of the NetFlow protocol supported are; 1, 5, 7, and 9. Flowd is also capable of interpreting both IPv4 and IPv6 addresses. It is also important to point out that Flowd is not an analyser but simply a collector that accepts the above mentioned export formats and writes them to disk in a binary stream format to minimise space usage. In order to read this data for later processing, flowd has support for both Python and Perl calls allowing applications developed in either language to access the stored data directly. (Miller, 2010)

2.5.2 Flow-tools

Unlike Flowd, flow-tools[9] is a complete system for collecting and storing exported NetFlow packets. This software can also be used to process and analyse the stored records. Flow-

[8]http://www.mindrot.org/projects/flowd/

[9]Version 0.68 is available for download from ftp://ftp.eng.oar.net/pub/flow-tools/flow-tools-0.68.tar.gz

tools is capable of generating reports on the current state of the network as well as filtering collected flows (Fullmer, n.d.b).

flow-tools works as a chain of smaller applications where the results of one application become the input to a higher layer application. This means that links in the application chain can be removed or substituted with other applications to generate desired output. The advantage of an application chain is that the entire process can be distributed across multiple systems in a large network and the process can easily be modified by swapping out links.

The flow-tools complete application chain consists of 17 applications developed by Fullmer with the exception of the flow-merge application which is developed by Lidz (Fullmer, n.d.b). A full explanation of each of the applications will not be covered here but an inspection of the capturing application called flow-capture is warranted.

Flow-capture is the first application in the application chain to receive information. This application handles the NetFlow packets as they are exported to the capturing host and can be configured in a number of ways (Fullmer, n.d.a). One configuration of the flow-capture application will limit it to only accept flows from a specific remote IP, otherwise any incoming flow packets will we collected and written to disk.

2.5.3 nProbe

nProbe[10] is an application that can be used in conjunction with nTop for traffic analysis of exported flows (nTop, 2013). As with other mentioned applications, the primary function of nProbe is to act as a collector but is also capable of acting as an exporter to export packets in NetFlow versions 5 and 9 or IPFIX (nTop, 2010). nProbe can run on Linux, OS X and Windows and can export the collected raw flows into MySQL or SQLite databases as well as plain text or binary for analysis. nProbe is also capable of collecting both IPv4 and IPv6 packets and is designed to run with minimal memory use.

2.6 Geolocation

Geolocation is the process of associating a physical region with an IP address or subnet. Though primarily used for geomarketing advertising agencies in offering location specific

[10]The source code can be downloaded from: https://github.com/xrl/nprobe

advertisements to potential buyers (Cliquet, 2006), geolocation can also be used to select the optimal route for region specific clients trying to access a distributed service.

Geolocation can also be used to determine what page a server should return to a client request. This includes country specific web pages which can include data more locally relevant such as language and currency formats. Geolocation can also be used to prioritise results returned from search engines as well as for enforcing location based restrictions on accessible material as mentioned by MacVittie (MacVittie, 2012) when discussing scenarios of geolocation.

The process of geolocation is usually done by looking up an IP address in a geolocation database and finding the smallest subnet that the IP address is contained in. This can be done online by utilising an online geolocation database such as those offered by the Internet Authorities (NCC, 2011), or offline by by using a local geolocation database such as those provided by MaxMind (MaxMind, n.d.).

An example of an implementation of geolocation is HoneyMap. HoneyMap is an interactive map for the visualization of network attacks in real time. The attacks are recorded by honeypots distributed globally that record the IP address of attackers. These IP addresses are then geolocated and the resulting coordinates are used to plot the location of the attack on a world map (Watson *et al.*, 2013).

2.7 Visualisation

An important feature a data analysis system is data visualisation. Network flows will produce considerably less data when compared to other packet sampling based approaches but still handle large volumes of information. This data need to be aggregated so that a user can quickly assess the current situation of the network. To this end, it is important for visual rendering of selective data to be a part of a data analysis application.

In terms of geolocation, visualisation could mean representing geolocated information on a map which is then displayed to the user such as the Honeynet Project (Project, n.d.). This way, a user can quickly see the location of flow endpoints and draw conclusions that would prove difficult for an analyser to perform. Examplles of this include suspicious clustering of addresses in a region known to have been compromised.

Another use of visualisation is to be able to represent flow characteristics over a selected range in terms of each other. Examples can include three dimensional plots such as

InetVis, a visualisation tool for telescope traffic analysis by Van Riel (van Riel & Irwin, 2006), and two dimensional charts and graphs which can be grouped to visually describe a situation.

2.8 Port Scanning

A third technique is Port Scanning which can be used to detect open ports on a host. As discussed by Yang (Yang, 1997), scanning a port is simply the process of sending traffic to a particular port on a potential victim and detecting if there is a reply from the victim which would indicate that its accepting traffic on the target port and thus is open. Both TCP and UDP protocols can be used to perform a port scan with each protocol being used to target specific connections.

One way to perform a port scan is to attempt to create a legitimate TCP connection with the target host on a specific port and wait for a response. If the connection can be established, the target port is open the connection can be closed by performing a legitimately. This method does not need any custom packet crafting but is easy to detect as the port will log successful connection attempts.

An alternative to this is to perform a SYN scan. A SYN scan requires the construction of the initial packet in the TCP handshake which is then sent to a port on the target host. If the host replies with a SYN _ACK packet, then the port is open and unfiltered. In this case, a RST[11] packet should be sent to cancel the connection establishment. The connection must be reset as the target host has allocated the required memory to handle a potential TCP connection. By not sending a RST packet and leaving the connection to time out, the scan becomes more visible due to the larger memory footprint. Should a range of ports on the same host be scanned before these connections time out, the port scan can become a Denial of Service (DoS) attack (Erickson, 2008).

The UDP protocol can also be used to perform a port scan. Due to its simplicity, performing a successful port scan with this protocol is more difficult. One of the main reasons for this is because UDP attempts a best effort approach at transferring data and so there is always a chance that the probe or reply packet never reaches its destination. To account for this, UDP port scans often have to do repeat tests to try reduce the probability of a false response. Another concern with using UDP port scanning is that many modern

[11] A TCP packet with the reset flag set (Braden & IETF, 1989)

firewalls simply block UDP probes from reaching a closed port which will mean that no reply is generated. This lack of reply could indicate a false positive to the scanner.

The process of a UDP port scan is performed by sending a UDP packet or probe to the target host on a specific port. If the port is closed, the scanner will receive an ICMP[12] port unreachable message from the victim (Messer, 2007). If no reply is received then the port can be considered either filtered by a firewall or open.

In order to try reduce ambiguity between whether the port is open or filtered, the scanner can attempt to send crafted UDP packets to the target ports that are designed to trigger a response from the application that the port is currently bound to. If the scanner receives no reply, the port can be considered filtered while a reply can indicate open or closed depending on the response. This process requires that the scanner must know what application is running on the target port or test a possible range of applications.

2.9 Worms

Another threat to end hosts connected to a network is the infection of self propagating malware code. An example of such code is a worm which exploits a vulnerability in the victim to allow the malware to be executed and deliver the attached payload. The infected machine then seeds the worm by scanning other end hosts for a exploitable vulnerability and uploading the malware to those hosts (Zou *et al.*, 2006).

The scanning process utilised by worms varies greatly but five common techniques used are; random scanning, local preferential scanning, target selection, hitlist scanning, and tological scanning.

Random scanning is when the infected host performs port scans on randomly generated IP ranges for potential victims. These random IP ranges are usually created by first generating a 32 bit random number which then acts as the address (Gldi & Hiestand, 2005). This technique allows the worm to spread quickly over a wide range of vulnerable hosts but is very visible as the volume of requests can put noticeable strain on routing devices.

An improvement on pure random scanning is local preferential scanning. In local preferential scanning, IP addresses are still generated randomly except now the most significant octets are fixed so that the infected host first scans random IPs within local

[12]Internet Control Message Protocol (Braden & IETF, 1989)

subnet (mhaske Dhamdhere & Patil, 2012). This method comes with the benefits of more thoroughly scanning local hosts which are more likely to exhibit the same vulnerability. Other advantages include reduced traffic across larger networks which means a reduced likelihood of detection.

Some scanning techniques are not random at all such as hitlist scanning where the infected host will scan a predetermined set of IP addresses. With hitlist scanning, the target IP addresses are hard coded by the attacker prior to the initial infection. This accelerates the initial infection rate of the worm (Staniford *et al.*, 2002). As a result of using a hitlist, the worm will only target hosts explicitly specified by the attacker. A disadvantage of hitlist scanning is that the worm will not be able to scan for hosts in the local network which are more likely to exhibit the same vulnerability. Tological scanning accounts for this by collecting data on computers that exist in the local area network of the infected host (mhaske Dhamdhere & Patil, 2012). This scanning technique then targets hosts in this constructed list first as they are more likely to be vulnerable to the same exploit as the infected host.

NetFlow based approach to intrusion detection has been done by Dhamdhere(mhaske Dhamdhere & Patil, 2012). Dhamdhere describes two systems that make use of NetFlow data and could be used for worm detection. The first system take the approach of parsing data segments for malicious activity, blocking future harmful traffic and then notifying the network administrator. Unfortunately, this system was found to generate many false positives and so a new system called FloWorm was designed. FloWorm quickly identifies and tracks abnormal flow activity. Abnormal traffic is then monitored to detect similarities between it and traffic seen during a worm intrusion. If the traffic does show characteristics seen during a worm intrusion, the flow is tagged and the network Administrator is notified. (mhaske Dhamdhere & Patil, 2012)

2.10 Denial of Service

Denial of service (DoS) or distributed denial of service (DDoS) is another common network threat to any end host the relies on a network connection to provide a service. This threat works by flooding the server providing the service with fake traffic. The increase traffic pushes the server to maximum capacity. When legitimate traffic arrives from a potential client trying to access a provided service, the server is unable to service the request (Mirkovic & Reiher, 2004).

Other versions of this attack include upstream DoS attacks where the target of the attacker are the routers or ISP that route the victims traffic (Mirkovic & Reiher, 2004). These routers become congested with dummy traffic and so legitimate traffic cannot be routed to the victim server successfully.

NVisionIP is a Java based system developed to assist in the identification a DDoS attack on a network. NvisionIP uses data flows collected by NetFlow for statistical analysis of the current traffic being routed or switched by an exporting system. NvisionIP collects network data from the flows received via the NetFlow protocol and after processing, represents the statical results to the user for external analysis. (Lakkaraju *et al.*, 2004)

2.11 Summary

A network flow as defined by Cisco has become a popular format for collecting characteristics about network traffic as it is supported by compliant routing and switching devices. Different protocols exist for exporting theses flows from the export device and these protocols themselves are well defined and quickly gaining popularity. This is especially true of the template based mechanic of NetFlow version 9 what has been incorporated into the IPFIX protocol format.

Two network based attacks that threaten end hosts were discussed and Potential NetFlow based systems that assist in the detection of such attacks are also briefly discussed. From the review, it is clear that there is no simple solution to completely secure a system without significantly compromising the accessibility of the host to legitimate traffic.

Visualisation is an important feature of any application that attempts to represent large volumes of data to a user in a way that is easy to read and quick to interpret. An example of this is representing geolocated data on a map which is displayed to the user so as to quickly identify the physical location of a destination host. Another approach is to use 3D visualisation techniques like those proposed by Van Riel (van Riel & Irwin, 2006).

Chapter 3

Design

3.1 Introduction

To achieve the goals described in the previous chapter, it is necessary to develop a system that is capable to performing geolocation and network visualisation. This system must collect the raw network flows as as they are exported from the responsible system and process them.

This chapter begins by first detailing the goals that this system should be designed to meet in section 3.2. This section also identifies and justifies the network flow export protocol to be used by the system. The system constraints are identified and discussed in section 3.3. These constants do not include the limitations imposed by the research scope in section 1.3. A system overview in given in section 3.7. This overview shows the system composition as well as identifies the components needed for the system to function correctly. Hardware and resource considerations are covered in section 3.5 which includes concerns regarding memory requirements of the web based system component. Security measures are mentioned in section 3.6.

The remainder of the chapter is devoted to detailed designs of each of the required system components. These designs discuss how the components should be implemented to achieve the system goals. This chapter concludes with a summary of the material covered in section 3.11.

3.2 System Goals

The system has two defined goals which it should be able to produce as an end result. These goals are geolocation of the external endpoints of each flow and the generation of visual aids to assist in network visualisation. In order to achieve this, system components for geolocation, report generation, and visually representing recorded characteristics are needed.

3.2.1 Geolocation

To achieve the geolocation goal, two system components are required which are; a geolocation database that can translate IP addresses into latitude and longitude coordinates, and a map that can be viewed and interacted with.

Geolocation database

As discussed in in section 2.6, the geolocation database can simply be a binary file such as those available from MaxMind (MaxMind, n.d.) which will return geolocation details for a given IPv4 address. The resolution of the returned coordinates varies depending on the type of binary file used. For this research, the binary files used are those made available for free by MaxMind. These files do not have the resolution of binary files which are made available after purchasing the appropriate site licence from MaxMind.

Rendering Platform

In order to make and render the interactive map, a component of the system should be designed as a web based application and present the interactive maps to the user using a browser compatible language such as JavaScript. A web based application has been chosen to make the results produced by the system more easily accessible to hosts situated in different physical networks. The possibility of having multiple instances of this component active simultaneously should also be catered for. The rendered map should not be static and must display the current geolocation results without the user being required to request them. To support this, the map rendered in the web application must update itself automatically with new flow endpoints as they are received by the system.

3.2.2 Visualisation

The second goal of the system is to produce visual aids that assist in visualising the recorded information. To achieve this, the system includes three visual components which can be used by the user. Two of these components require a map similar to the one being used for geolocation and so are included as part of the web based application. These two components are; a heatmap for associating a volume or frequency of a flow characteristic with all countries for a specified time range, and a replay component that can be used to replay traffic as it was collected by the system.

The third visual component is a report that can be generated and uses visual aids to represent the network for a given month.

Protocol Selection

In order to achieve either of the above goals, it is also necessary to enable the system to collect the raw flows as they are exported from the exporting system. After reviewing the available flow export protocols investigated in chapter 2, the protocol that this system will accept and be able to interpret is NetFlow version 9.

There are three reasons for selecting this protocol. The first reason is that NetFlow version 9 is currently the most recent protocol developed by Cisco and so is more likely to be compatible with modern export devices.

The second reason for this selection is to take advantage of the dynamic packet structure of this protocol. With this protocol, it is possible to select what flow characteristics the export device should export for each record. This is advantageous as this means that unnecessary characteristics are not exported as they are in previous versions of the NetFlow protocol and the characteristics exported can be specified to meet the system's needs.

Lastly, due the similarities between the NetFlow version 9 protocol and the IPFIX protocol, modifying the defined system to also accept packets being transmitted under the IPFIX protocol can be done with less complications than would be the case if a previous version of the NetFlow protocol was selected.

Table 3.1: Required record characteristics

Source IP Address (IPv4)
Destination IP Address (IPv4)
Source Port
Destination Port
Protocol of flow
Time of first packet switched
Time of last packet switched
Number of packets switched
Number of bytes switched
TCP flags seen

3.3 System Constraints

To produce a feasible system that can achieve the defined goals, some factors of the system have been discussed but not implemented as they are considered out of scope.

In order to simplify the record storage and reporting processes, the system can only store a specific set of characteristics. This requires that a constraint be applied to all templates that the system is allowed to use for collection. The constraint is that any record being processed by the system must at least contain the characteristics depicted in table 3.1. The template is allowed to contain additional characteristics but these will be ignored by the designed system.

Should it become necessary to export additional characteristics, it can be done by extending the Record class and the storage database to include the new characteristics. The functions responsible for constructing the Record object and functions that interface with the record database will have to be modified accordingly.

3.4 System Overview

As the system is defined to include a web based component, the system requires a server component that can serve the web pages and handle communication between the served pages and the rest of the system. Furthermore, for the visualisation components to function, it is necessary for the system to include a record storage component that can store the characteristics of the flow records that are collected.

To achieve this, the system is split into four main components which are; the Collector,

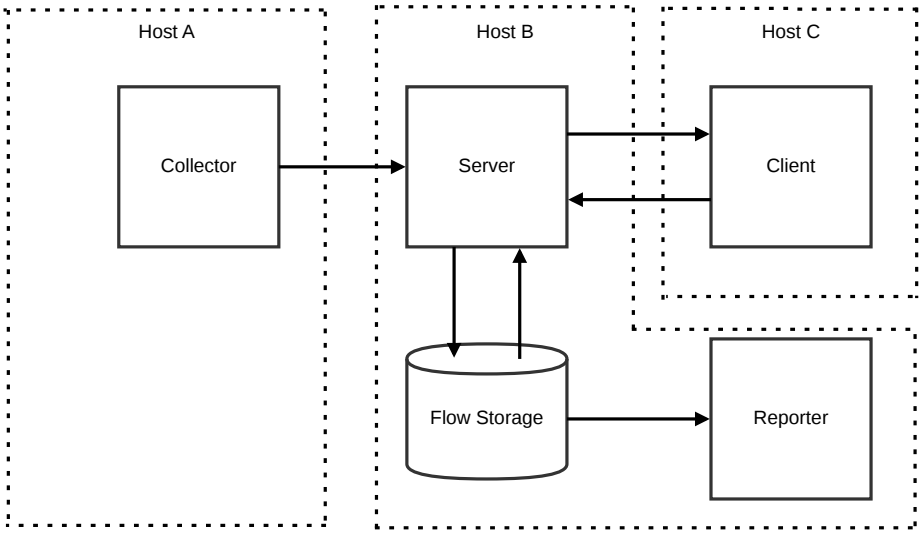


Figure 3.1: System Overview

the Server, the Client, and the Reporter. All of these components must be linked together to form the complete system which is is shown in figure 3.1. This figure uses arrows to represent the communication channels that link different components together to produce the resulting system.

The four components are distributed into three different groups; host A, host B, and host C. The grouping indicates a direct dependency between the contained components while the communication between groups is done over TCP connections. By developing the system to communicate between the different groups over a network, the system can easily be fragmented with each group running on a different host.

By allowing the different components to communicate in this manor, it becomes possible to situate the different components in different networks so that the Server can become more accessible to a network segment that is separate from the segment in which the raw flows are being exported. This allows for improved security as the Clients do not need to access to the same network as the exporter to request pages from the server.

As mentioned in section 3.2, the Client should be designed to run in the host's browser and so is not required to be run on the same host as any other component of the system. Another import point to note about the Client is that the system should be capable of handling multiple instances of the Client simultaneously. To achieve this, the Client should be implemented as a web page which is returned by the Server to any requesting host.

3.5 Hardware and Resource Considerations

This system is designed so that it can run at a high level and does not require any dedicated hardware. Though it is segmented as shown in figure 3.1, it is possible to run the entire system on a single host and, if the network packets can be internally simulated, does not require a network connection.

3.5.1 Record Storage

Record storage is an aspect of the system that must be considered for this design. Due to the dependence on previously collected records, the storage component must be robust and easily accessible by the relevant functions. As reliability must also be considered, a database should be used rather than writing the data out into files.

By using a database such as MySQL, concerns regarding algorithms for better lookup efficiency, data corruption, and scalable storage can become the responsibility of the database. Using a database also provides advantages in security as it includes access control. This is discussed in section 3.6.

3.5.2 Bandwidth Considerations

Exported packets from softflowd to the Collector are transmitted using the UDP protocol. This is because all necessary information can be transmitted in a single packet and the overhead of a TCP connection is necessary. The reason this is possible is due to a limitation in softflowd where the characteristics that are exported are predetermined and so cannot be changed. This limitation may not exist in other implementations of a NetFlow exporter and thus a different transmission protocol could be used.

Data being passed between the Collector and Server occurs in set volumes. Each packet contains a single serialised record. As the serialised data is small enough to be encapsulated in a single UDP packet, the overhead of a TCP connection is not warranted. Concerns regarding the loss of packets is warranted however as neither the Collector or Server is capable of knowing that a packet was both successfully sent and received. Should this concern become significant, communication between the Collector and Server could be modified to use the TCP protocol rather than the UDP protocol.

The bandwidth requirements between the Server and the different instances of the Client are a significant concern. In order to keep an open channel between the Client and Server, a TCP connection between the two components needs to be maintained.

The advantage of using a TCP connection is the additional security the connection adds for preventing packet injection and protection from data loss due to packets being dropped.

The disadvantage of a TCP connection is the volume to traffic that will be transmitted between the Server and the Client. This is particularity a concern for transferring large volumes of data in response to a traffic replay request.

The volume of data can be reduced by changing the format used to transmit the replay information. Examples can include transmitting the data according to a template to reduce data wasted on formatting. Sending the information incrementally could also be considered but will disrupt replay if congestion occurs on the network between the Client and Server.

As already mentioned, this system is designed as a prototype and proof of concept so bandwidth concerns are limited to a discussion only and implementations to resolve them are not considered within scope.

3.5.3 Memory Considerations of the Client

As the Client will continuously be receiving information from the Server and rendering that information, memory management of the received information is a concern.

The real-time interactive map renders the flow events as they are received from the Server. The number of rendered events should be limited while minimising the effect on the visualisation of the system.

A option to reduce the sprite count is to not generate a sprite for every IP address received. If an IP address has not been rendered on the map yet, then generate a new event for that IP address and store a handle to it. If an event for the IP address already exists on the map, then use the stored handle to that event to reanimate it. Another approach is to limit the number of events allowed on the map at any given time or to apply a lifetime to each sprite so that only recent activity is rendered.

The approach taken in the design of this system is to enforce a sprite limit on the map. When the number of rendered sprites exceeds a specified limit, the least recently added element is removed from the map.

3.6 Security Measures

As this system has been defined as a prototype to assist in the exploration of Network Flows, it is not production ready in terms of security. That being said, some security measures have been implemented and this topic must become a significant component and be reinvestigated should this system be used in future research in the field of Network Flows.

3.6.1 Exporter-Collector Security

A security concern is the corruption of the network traffic stream to the collector. In the current implementation, softflowd exports raw flows under the NetFlow version 9 protocol but using the UDP protocol. UDP packet streams are more vulnerable to malicious packet injection as it is unable to maintain connection state or implement security. This results in the traffic between the exporting device and the Collector being vulnerable to malicious traffic injection.

Currently, the collector is designed to accept packets being exported to a predetermined port on the host and only confirms that the packets are being transmitted under the correct protocol. The Collector performs no additional checks before parsing the packets into a format readable by the rest of the system.

This results in the system being vulnerable to packet injection provided that the crafted packets are not malformed and are being transmitted under the correct protocol and using the correct template. An attacker could use this to inject fictitious flows into the system or terminate flows prematurely.

In an attempt to recognise and mitigate this vulnerability, the Collector component can be restricted to only accept flows from a specific IP address. This restriction is employed by setting a variable at the top of the Collector script file.

3.6.2 Collector-Server Security

A similar problem to the one stated above exists between the Collector component and the Server component and is solved in the same manor by setting the respective IP address and port of the Collector and Server in the relevant script files before start up.

3.6.3 Client-Server Security

For communication between the Client and the Server, checks need to be employed to insure the system only reacts to valid traffic. A white list approach to identifying the traffic the Server will accept is followed.

Each Client request has to follow a specified format, otherwise the request will be ignored by the Server. This format is discussed in chapter 4. The contents of each request is then sanitised in an attempt to mitigate SQL Injection vulnerabilities.

The server itself is limited to only communicate via a websocket with the exception of the first page request. Upon page request, the server returns the index HTML file along with the associated JavaScript to the requesting host. After rendering the page, all additional communication must happen over a websocket.

3.6.4 Database Security

Another area where security is considered is the data storage. Many components of the system are dependent on having access to the stored data in order to function correctly. This security is necessary to prevent data from being deleted, added, or modified unless it is explicitly required by the system. As a result, the database must be accessible to the relevant system functions but should be restricted elsewhere.

A MySQL database has been selected for this implementation of the system and the security of the database itself will not fall into the scope of this discussion. The focus of the system security in this field will be on securing communication with the database.

To achieve this, the credentials are not hard coded into the system scripts but are passed to the system on start up. Each connection to the server persists for the duration of the query after which, the connection is immediately closed. Components with access to the database are currently only Server components but as a consequence of this, the Reporter component can only access the database through the server and so must run on the same host.

3.7 Collector

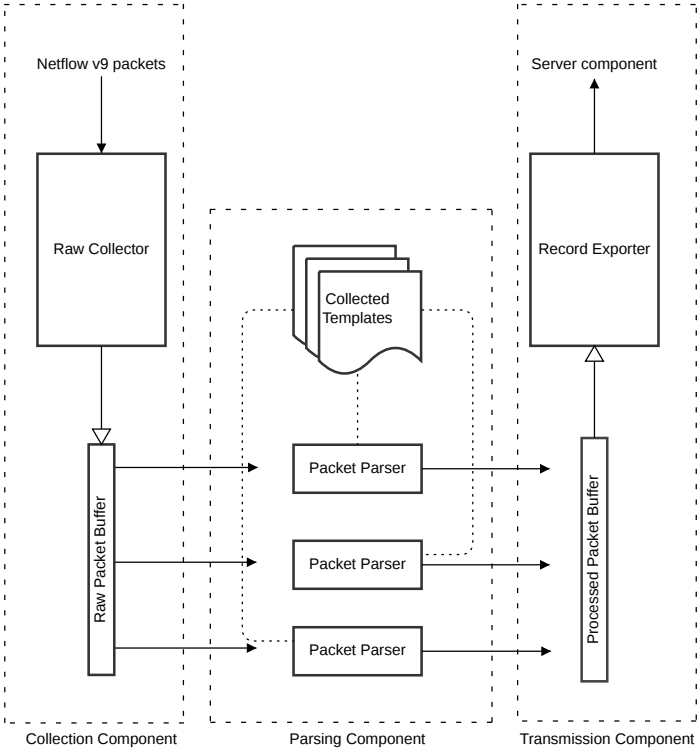


Figure 3.2: Collector Design

Before processing and storage of network flows and be performed, the exported packets need to be collected and parsed. The process followed by the system is depicted in figure 3.2.

The Collector can be broken up into three subcomponents; a collection component, a parsing component, and a transmission component. The collection component is responsible for reading raw packets from a specified port and storing the packet contents in the Raw Packet Buffer. The parsing component then reads the packets from the Raw Packet Buffer, processes them and stores the result in the Processed Packet Buffer. The transmission component reads the Processed Packet Buffer and transmits the buffer contents to the Server.

3.7.1 Collection Component

The collection component can be represented as a flow chart as shown in figure 3.3. When a packet is received from the specified port, the collection component first parses the packet header in order to confirm that it is being transmitted under the NetFlow version

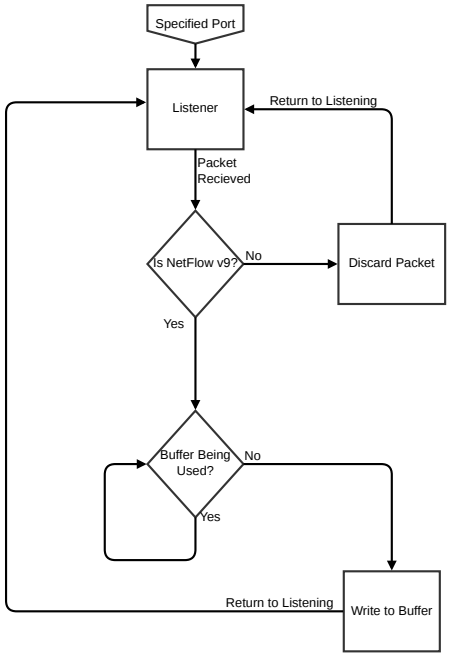


Figure 3.3: Collection Component

9 protocol. If the packet is not using the NetFlow version 9 protocol for transmission, the packet is dropped and the collection component returns to listening on the port.

If the transmission protocol is correct, the collection component checks if the Raw Packet Buffer is currently in use. If the buffer is free, the collection component stores the packet in the buffer and returns to listening on the port.

If the buffer is being used, the collection component waits until the lock being held for the buffer is released. Once the Raw Packet Buffer is free, the collector can then insert the collected packet and lock the buffer for the duration of the action. Once the packet is inserted, the collection component must immediately return to listening on the port for any additional packets as they arrive.

3.7.2 Parser Component

The goal of the parser component is to remove elements from the Raw Packet Buffer and create threads responsible for processing them. This is achieved by monitoring the Raw Packet Buffer for changes in volume. While the Raw Packet Buffer contains elements, the parser component will attempt to remove a single element to be processed. Before the element can be removed, the parser component must first insure that the number of currently active parse threads is below the thread limit. If the thread limit has not been

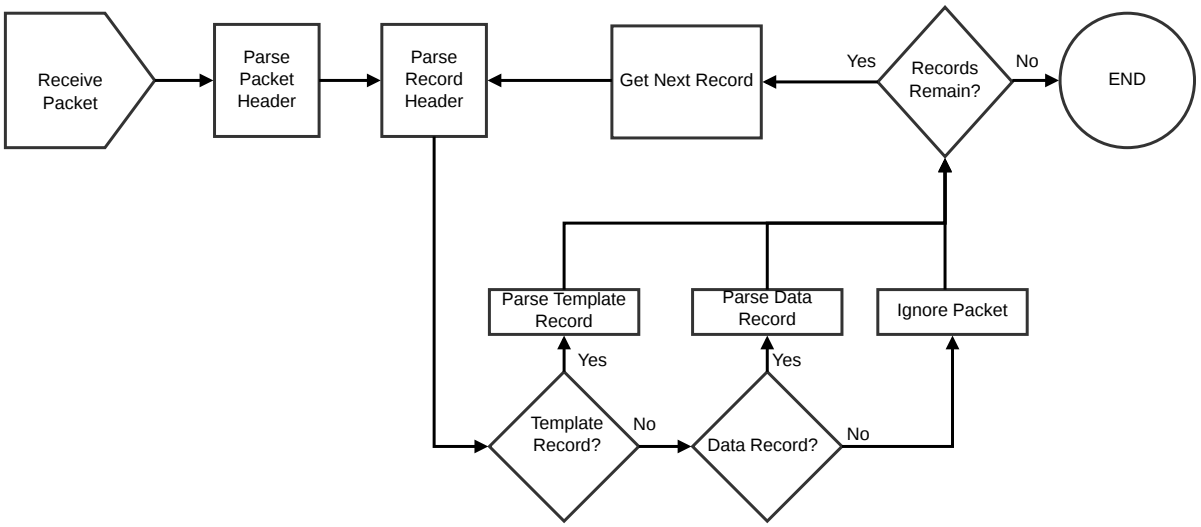


Figure 3.4: Parse Component

reached, an element is removed from the Raw Packet Buffer and a new thread is created to process it. If the thread limit has been reached, the parser component must wait until an active parse thread has completed before continuing.

Each thread that the parser component generates begins by first processing the packet header as shown in figure 3.4. From the packet header, the parser component can extract the record count for that packet as well as the system uptime. The record count indicates how many records are contained in the packet and is used by the parser component to determine when all the records contained in the packet have been processed.

For each record in the packet, the parser begins by processing the header to determine the record ID. This ID is used to determine the type of record that is associated with the processed header. Should the value of the ID be 0, the associated record is considered to be a data template record and should be processed accordingly. If the ID value is 1, the record is an options template and is ignored by the current implementation of the system. Otherwise, the template is considered to be a data record and is treated as such.

If the thread determines that the record to be parsed is a template, the thread parses the record and creates a Template object from its contents. The process followed is depicted in figure 3.5. The process begins by processing the record header to determine the template ID and the number of fields contained. As the field length is set, the the process then cycles through the record until all fields are extracted. A Template object is then created from the extracted fields. This object is then serialised and the serialised data is stored in a file named after the template ID. Should a file with that ID already

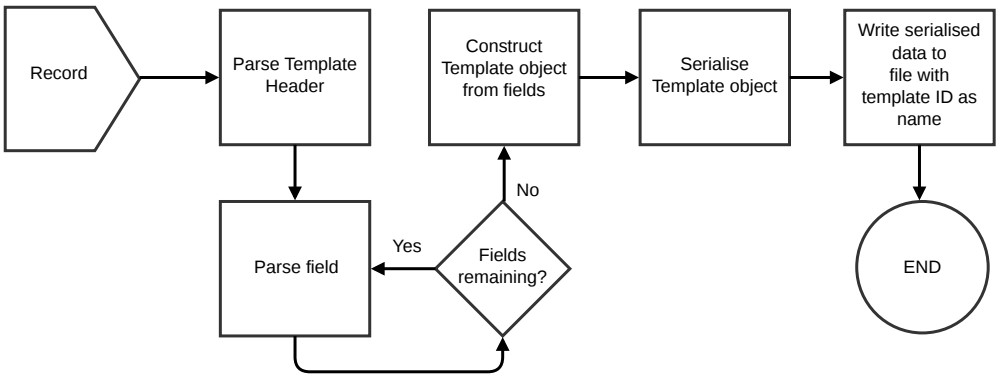


Figure 3.5: Processing a Template Record

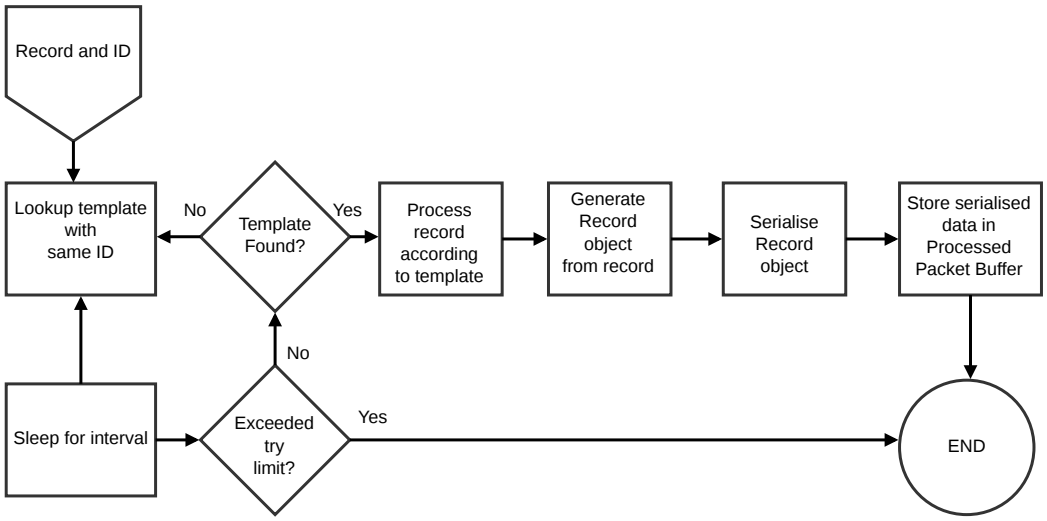


Figure 3.6: Processing a Data Record

exist, it is overwritten as future records with that ID will be formatted according to the new template.

If the thread determines that the record to be parsed is a data record, the thread processes the record according to the specified template and creates a Record object form its contents. This process is described in figure 3.6. The process begins by attempting to load a file named after the template ID associated with the record. Should this be successful, the loaded template is de-serialised and used used to process the data record. A Record object is created using the processed fields of the data record which must the elements described in section 3.3. This object is serialised and stored in the Processed Packet Buffer.

If a file with the associated template ID cannot be found, it could mean that the thread responsible for processing the template has not completed yet. To account for this, the requesting thread sleeps for a short interval after unsuccessfully loading the file. This step

is repeated until the thread successfully loads the file or exceeds the maximum number of attempts. If unsuccessful, the thread announces such and drops the record.

3.7.3 Transmission Component

The transmission component is responsible for passing the parsed records from the Processed Packet Buffer to the Server.

The process begins by checking if any parsed records have been stored in the Processed Packet Buffer. If this is the case, the process then attempts to acquire a lock for the buffer. If the lock is being held by another process, the transmission component spins until the lock is released. The transmission component then removes a single element from the buffer and sends it to the specified IP address and port. This is done using a UDP connection and the advantages of this connection are discussed in section 3.5.

After the record has been successfully transmitted, the transmission component returns to checking if a record has been stored in the Processed Packet Buffer.

3.8 Server

Once the records are processed by the Collector, they are passed to to the Server (shown in figure 3.7) to be stored and later broadcast to connected clients via the webserver component. The functionality of the Server can be grouped into to parts; the processing component that is responsible for receiving and storing records, and the webserver component which handles communication between the Clients and the Server.

3.8.1 Processor Component

The processor component is responsible for collecting the records as they are transmitted from the Collector. This is done by a thread that is separate from the main thread so that it can run concurrently with the webserver component. This way the threads can run asynchronously so that the records can be processed without interfering with the webserver component. The process that this component follows is shown in figure 3.8 which details the flow of the raw record from the Record Collector to the platform database and Broadcast Call queue.

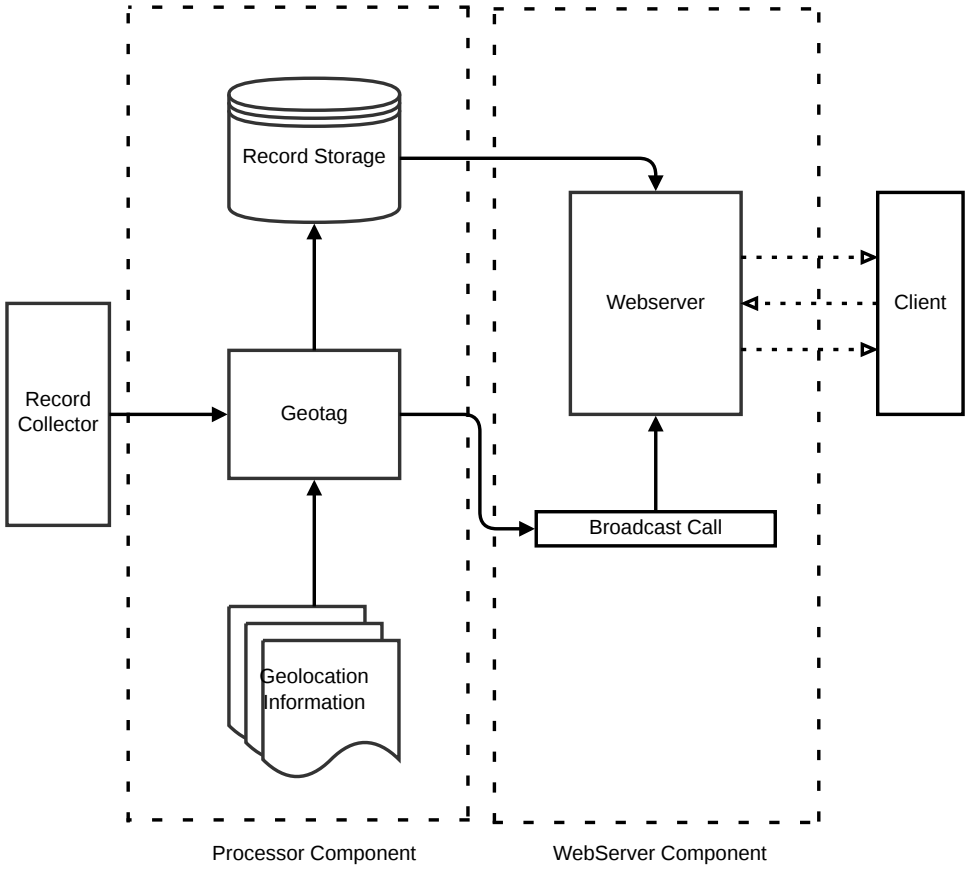


Figure 3.7: Server Component

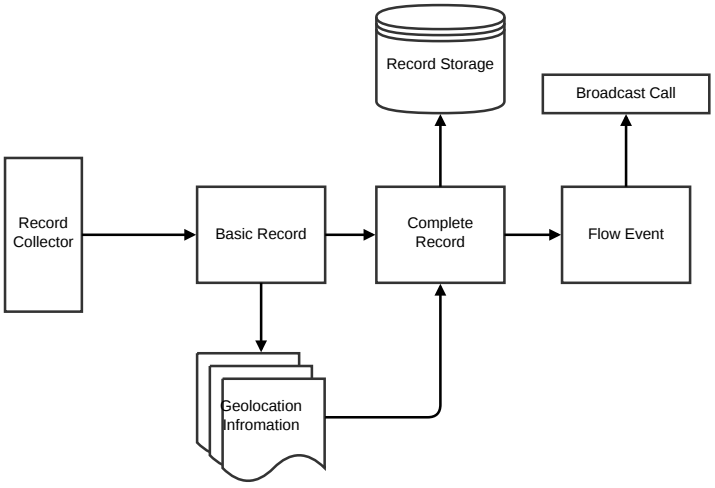


Figure 3.8: Server Processor

Record Geolocation

Once the records are received from the Record Collector, the first thing the processor component does is connect to the geoIP database and request the associated geolocation details for the IP address stored in the record. From these details, the Country, Country tag, and coordinates are extracted and associated with the record as described in figure 3.8.

Record Storage

After the geolocation is complete, the record is stored in the database labelled Record Storage as shown in figure 3.7 and 3.8. Following this, a flow event is built from the IP, Country, Country tag and geolocation elements of the flow. This flow event is then sent to the webserver component by using a broadcast call so that the event can be broadcast to all connected clients.

3.8.2 Webserver Component

Once a the flow event is sent to the webserver component, it becomes the responsibility of that component to broadcast the flow event to all clients that are currently connected to the Server. In order to perform a broadcast, the webserver component needs to be able to collect and store handles to all Clients when they connect. This must then be removed when the Client later disconnects. This can be done using a list and is further discussed in the implementation of this component in Chapter 4. Once the webserver component has a list of connected clients, it can simply iterate through the list and send the flow event to each Client.

Client Connection

As the webserver component handles all communication with the Clients, it is necessary that a communication channel is established and maintained to allow the two components to exchange information. This must be done so that the Server can push information to the Client without having to wait for prompting first as this would greatly hamper the

real time rendering of records. To facilitate this, a websocket connection must be set up between the Client and the Server. In order to connect to the Server and create the websocket, the Client must initiate a websocket connection upon start up. This connection is what the webserver component stores for later communication. If the client makes a request to the Server, it is though this websocket connection and must follow a specific format. Upon receiving a valid request from the Client, the webserver component must then connect to the database to make the appropriate queries and process the result. This result is then packed into a JSON string which is tagged with the appropriate tag for the client and sent back to it through the same websocket connection that the request was received from.

The Server component will respond to three types of requests from the client; a country details request, a heatmap request, and a replay request.

Country Details Request

The request for country details occurs when the user of the Client application wishes to view details associated with a particular country. The user indicates such by clicking on the country in question on the realtime interactive map. This causes the Client to generate the appropriate request for that country. The request is then sent to the Server via the websocket associated with that Client where it is handled by the method responsible for processing country detail requests. The responsible method than constructs a query to request details about the country in question from the Record Storage and processes the results returned. The processed results show the total number of flows and the total number of bytes associated with each IP address recorded for that country. This data is packet into a JSON string and returned to the requesting Client.

Heatmap Request

A heatmap request occurs when a user of the Client component wishes to generate a heatmap of a specific NetFlow characteristic within a specified time range. This information is received by the webserver component of the Server and used to construct a SQL statement. The statement is executed to extract information from the Record Storage regarding the characteristic that the heatmap must be generated for. The returned data contains all flow information relating to the request type for each country. The returned data is weighted to lie in the range from 0 to the maximum heat and each weighted result

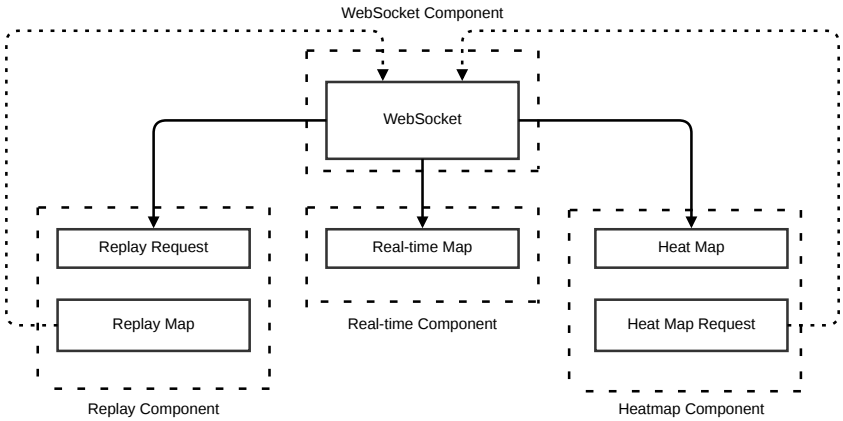


Figure 3.9: Client Component

is then paired with its associated country. This processed list is then formatted into a JSON string and returned to the requesting Client.

Replay Request

A replay request occurs when a user of a client wishes to replay an instance of recorded traffic at a specified speed. The time range that flow events must be replayed from is sent as a request to the Server via the opened websocket. This information is received by the webserver component and the specified range of flow events is returned from the Record Storage. Each result returned contains; a Country tag, a set of coordinates indicating where exactly in the country the flow occurred, and the start time of the flow. This data is used to construct a JSON formatted list which is returned to the Client requesting the data. The efficiency concerns regarding sending large volumes of information over a tcp connection are discussed in section 3.5. As the efficiency and bandwidth consumption of this technique has little impact on the results this component of the system attempts to produce, it is limited to a discussion only.

3.9 Client

The goal of the Client component of this system is to provide a web based user interface as specified in the system definition. To achieve the defined goals, the Client is split into four components which are; the websocket component, the real-time component, the replay

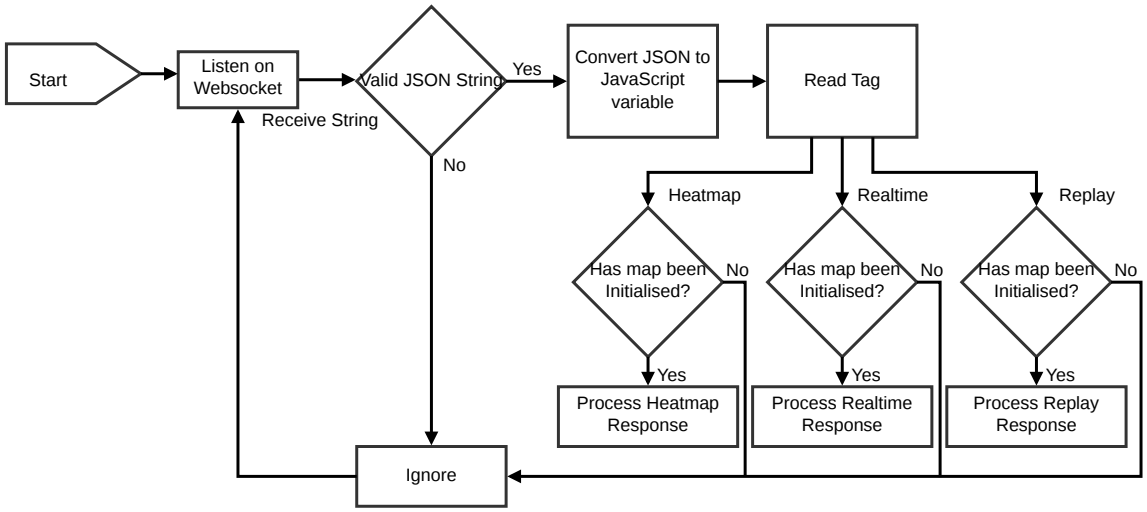


Figure 3.10: Websocket Component

component, and the heatmap component. The websocket component is responsible for handling communication between the Client and Server.

The remaining three components of the Client are the human interface components of the system. Each of these components is designed to allow the user to view the recorded network data in terms of its location, time of generation, and characteristic.

The replay component contains an interactive map for replaying traffic. The real-time component handles displaying flow tags on an interactive map. The heatmap component is an interactive map that can be coloured to produce heatmaps of flow characteristics. The way that each of these components connect is shown in figure 3.9 along with the contents of each component.

3.9.1 Websocket Component

For each of the front-end subcomponents to work, they must communicate with the websocket component which, in turn, communicates with the Server. As stated in the Server section, the Client communicates via a websocket which is created as part of the startup sequence of the Client. After initialisation, the websocket component listens for incoming traffic from the Server and responds according to what components of the front-end have been activated by the user. If traffic is received for a component that the user has yet to initialise, the traffic is ignored. A component can be initialised by

selecting the appropriate tab on the web page. This allows the user to reduce the resource consumption of the Client if they only wish to use it to generate a single heatmap or replay a piece of traffic without the real-time map running in the background. Further resource considerations regarding the Client component of the system are discussed in section 3.5.

3.9.2 Real-time Component

The realtime component of the Client generates a map that, as flows are received by the system, is populated with events representing the origin of that flow. As the map becomes increasingly populated, the countries change colour according to the number of flows seen from that location. This rendering begins upon initialisation of this component and continues until the Client is terminated.

3.9.3 Heatmap Component

The heatmap component of the Client is designed to represent ratios of a characteristic seen between countries. This is done by using a colour ratio where the brighter the colour, the more dominant a particular country is with regards to the selected characteristic. In order to generate a heatmap, the user must generate a heatmap request which specifies the characteristic that the user is interested in as well as the time range the heat map must represent. Once this request has been generated and correctly formatted, it is sent to the Server via the websocket component. When the webwsocket component receives data from the Server with the relevant tag indicating that the JSON formatted data is destined for the heatmap, the data is processed into an array and the heatmap is coloured according to the values contained. Should the user then wish to view a heatmap for a different time range or characteristic, the process is simply repeated and the heatmap is cleared and updated with the new values.

3.9.4 Replay Component

The replay component of the client is designed to allow the user to replay traffic that has already been recorded by the system. To use the replay map, the user creates a replay request which defines a time range to be replayed. An additional feature of this component is to allow the user to define the speed of the replay. This will allow the user to set the speed of the replay. The replay request is then sent to the websocket component

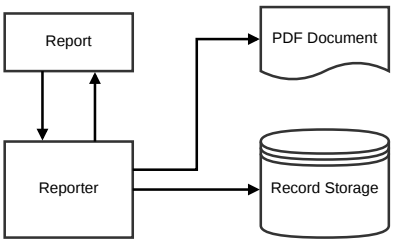


Figure 3.11: Reporter Overview

which in turn forwards the request to the Server. The returned JSON string includes all the flow events that happened in the specified duration and the replay map can recreate the event just like the realtime map but now with a scaled time factor.

3.10 Report Generator

An additional method or representing the Network Flows that are collected by the system is to generate a printable report. This report is generated by the Report Generator component of the system and must be capable of producing summaries and comparisons between stored data both textually and visually. In order to generate a report that can include both images and text as a single item, the Report Generator component includes a pdf generator that can build a canvas from a series of paragraphs, tables, and drawings and render the resulting canvas as a pdf.

3.10.1 Report Structure

The format of the report is static in order to simplify the generation process. The report consists of four components. The first component is a summary of significant characteristics seen by the system in the specified month. The second component is a bar chart of flows seen to the top ten IP addresses in the specified month. The third component consists of two pie charts comparing flows and data for the specified month for the top 10 countries. The fourth component is two line graphs showing flow and byte count per country over the specified month.

3.10.2 Characteristic summary

The characteristic summary component of the report provides a short list of characteristics that could be of use to the viewer. The characteristics that are currently reported on are:

- Bytes Transferred
- Flows Recorded
- Number of IPs Seen
- Number of new IPs seen
- Number of Countries seen
- Number of new countries seen
- Number of ports assigned to ingress flows
- Number of ports assigned to egress flows

To goal of this component is to give a brief textual overview of the network for the given IP address in the given month. This will allow the user to quickly see if the selected IP address is sending or receiving traffic as well as get an idea of the traffic volume that is being transferred.

3.10.3 IP graph

The bar graph is generated from the number of flows recorded between external IP address and a specified IP address for which the report is being generated for. The data is sorted so that the IP address to which the largest number of flows has been seen are first. This data is then used to render a horizontal bar graph which is converted into a drawing and inserted onto the pdf canvas.

3.10.4 Pie Charts

Pie charts are generated to display the volume to traffic seen to each country relative to other countries. These charts are designed to show at most, ten segments with the tenth segment being a concatenation of all all additional data sets. The information presented is the total number of flows and bytes seen to each country. This information is displayed

as a coloured segment for each country and is ordered by volume. All segments whose volume is too small to be generated and all segments beyond the top ten are merged together into an 'Others' segment.

In order to represent the volume of both bytes and flows to each country, two pie charts are used. The segments of the first pie chart are generated from the total number of bytes transferred and is ordered buy volume in a clockwise direction. The segments of the second pie chart are generated from the total number of flows seen to each country. The ordering of the countries in this chart are ordered to follow the same order as the first pie chart rather than being ordered by volume.

The colours used for each country in these pie charts are kept constant throughout the rest of the report.

3.10.5 Line Graphs

Line graphs are used to represent the data displayed in the pie charts as a daily progression rather than a ratio of the totals. A separate line graph is generated for bytes and flows. The countries represented here are chosen to be the same as the countries displayed in the respective pie chart.

3.11 Summary

This chapter covers the design of a system to be used to investigate the use of network flows for geolocation and traffic visualisation. During the definition of the stsyem goals, it was decided that the system component responsible for generating the geolocation results and some of the visual aids should be web based. In the discussion regarding system constraints, it was decided that the characteristics collected and processed by the system should be limited to simplify later implementation.

To account for record storage in the designed system, a MySQL database has been chosen so that issues relating to data storage and security do not need to be considered during design. Approaches to security concerns discussed in section 3.6 are to restrict the IP addresses each component will accept a connection from.

The majority of the chapter focuses on detailing the design of the components that make up the system. These design discussions focus on highlighting the goal of each component

and its functionality in terms of the rest of the system. Approaches taken to achieve this functionality are then detailed. The next chapter details the implementation of a prototype system that follows the design logic detailed in sections 3.7, 3.8, 3.9, and 3.10.

Chapter 4

Implementation

4.1 Introduction

In chapter 3, the system and its constituents are defined and further described. This includes a discussion regarding the design of each component and the necessary subcomponents to enable it to achieve the defined goals. The implementation of each of these components is described here.

This chapter begins by discussing and supporting the programming languages used to implement the system. This is followed by a brief description of significant libraries that are used to implement the system components.

The remainder of the chapter is focused on the implementation of the system. The structure of the implementation discussion follows that used previously in chapter 3. Each component is considered separately and segmented further into the subcomponents identified during system design. The implementation of each subcomponent is then discussed. This chapter concludes with a summary of the material covered.

4.2 Programming Language Section

To implement the system, it is first necessary to select a suitable set of programming languages. For this implementation, cross platform compatibility and rapid development are important factors that a language should feature to be considered suitable. One of the system goals described in section 3.2 is that the system should be segmented and dif-

ferent components should be able to run on different hosts. Furthermore, section 3.2 also identifies that the Client component of the system should run as a web based application. The result is that the system will be implemented using multiple programming languages.

Languages to consider can be grouped into two categories; Compiled languages and Interpreted Languages. As this system relies heavily on network communication and the management of data, object orientated programming and data serialisation should also be considered in selecting the appropriate languages.

4.2.1 Compiled Languages

Compiled languages provide the advantage of compiling down to byte code that is native to the architecture that they are compiled on. This gives optimal performance as compilation occurs before runtime and can be optimised by the compiler. Should the performance gains resulting in using a compiled language such as C be significantly more important than the advantages provided by interpreted languages which are discussed next, than a complied language should be selected for implementation.

4.2.2 Interpreted Languages

Alternatively, interpreted languages do not have the performance edge over compiled languages discussed above. The performance loss occurs as a result of the each instruction having to be converted into byte code native to the underlying architecture during run time. This intermediate step increases execution time which reduces performance and does not allow for compiler optimisations.

By implementing this intermediate step though, interpreted languages gain advantages over compiled languages. These advantages can include dynamic typing, dynamic scoping, and platform independence. Dynamic typing and scoping implies that the the type or scope of a variable may not be known until runtime.

Dynamic typing allows for more generalised code to be written that can function for different data types without implementing generics. Platform Independence results form being able to abstract the interpreted source code away from the architecture. This is done by making compatibility the responsibility of the interpreter running on the architecture.

4.2.3 Supporting Python as a Language

For this implementation of the system, an interpreted language has been decided on over an compiled language. The decision results from the benefits of producing a system that is platform independent and using a language that is simple and promotes rapid development. As a result, both Java and Python were considered as potential languages for implementation.

The specific language that has been decided on is Python as it is both Interpreted and Object Orientated with a large set of libraries that can easily be included for object serialisation and server implementation. Such libraries include Tornado[1], which is a Python library that provides asynchronous networking and a web framework, and pyGeoIP[2], an API for interfacing with geolocation databases.

4.2.4 Supporting Javascript as a language

To implement a system component to run in a user's browser such as the Client application, a language capable of running in this environment is necessary. The language chosen for this implementation is JavaScript because of its compatibility with the majority browsers currently available. An additional advantage of using JavaScript is a large collection of libraries and widgets that can be used to simplify the implementation of the application.

4.2.5 Pickle and Object serialisation

Object serialisation is the process of converting an object in memory into a format that can be stored in a file or transmitted across the network. This is significant as the data transmitted from the Collector to the Server is implemented as a serialised Record object.

As the chosen language is Python which is discussed in the section 4.2.3, object serialisation will be discussed and motivated based on how it is implemented in Python.

Serialisation in Python can be done using a native library called Pickle. This library contains functions to serialise all native data types such as booleans, strings, bytes, and integers. Pickle is also capable of serialising more complex data structures such as Lists

[1]http://www.tornadoweb.org/en/stable/
[2]https://pypi.python.org/pypi/pygeoip

and Dictionaries that implement these data types as well as classes and instances of these classes.

The process of object serialisation in pickle is initiated by calling the [.dump()] method which takes in a serialisable object and converts it into a binary stream. This binary stream is generated from the object according to a Python specific Pickle protocol.

There are two main factors promoting the use of object serialisation over simply writing the raw data into a formatted text file. These advantage they are space optimisation and complexity abstraction.

By serialising the objects and storing or transmitting the binary stream, additional formatting to help the different components of the systems to distinguish elements is removed. This is because the application can rely on the protocol to have formatted the data in a known format.

Secondly, by serialising the objects, the complexity revolving around extracting the necessary information form the object and writing it to a file is removed. This also removes the difficulties involved in maintaining the format across different applications. The same applies when the stored data must be read in and the original object be reconstructed. Though this is similar to what the object serialisation and de-serialisation does anyway, by using the included libraries, maintaining the process does not become a concern of the programmer.

4.2.6 Tornado Web Server

Tornado is an asynchronous networking library and web framework that is an open source version of the FriendFeed web server (FriendFeed, 2013). The library uses an event queue to allow it to become non-blocking and scale to handle multiple simultaneous connections (Facebook, 2013).

This library is used to create and maintain WebSockets as they are created and destroyed by connecting Client applications. By using this library, concerns regarding methods blocking communication between the Clients and Server can become the responsibility of the created HTTPServer.

Communication with the individual Client applications is also facilitated by this library. Messages can be sent to the applications by using the Client's WebSocket handle and adding the message to the HTTPServer callback queue. The main loop of the server

will cycle through this queue and service the request when it is safe to do so. A similar approach can be used to broadcast messages to all connected Client applications.

4.2.7 ReportLab

ReportLab is an open source library for creating documents and drawings. The library allows the programmer to create documents in Adobe's Portable Document Format (PDF) which can include both text and drawings (ReportLab, 2013).

For the purposes of this implementation, reportlab is used to construct a PDF document from a series of drawings, tables, and paragraphs. This is done by building a list of drawable objects which are generated from the characteristics to be reported on. This list is then built into a document by using the build function included in a SimpleDocTemplate class.

4.2.8 pyGeoIP

Pure Python GeoIP API (pyGeoIP) is a Python library used for determining geolocation information associated with an IP address (Ennis, 2013). The library was ported from the Pure PHP GeoIP API which was developed by Jim Winstead and Hans Lellelid (Lellelid, 2013).

The library allows for the creation of a GeoIP class that accepts a file name as a parameter. This file name is the path to a binary GeoIP database. The class includes a IP lookup method that accepts an IP address as a parameter and returns the geolocation details associated with the smallest subnet which includes that address.

Binary GeoIP databases are supplied by MaxMind which the pyGeoIP library is developed to support (MaxMind, n.d.). MaxMind supplies databases based on the users requirements which can be downloaded after purchasing the appropriate site licence. MaxMind also supplies a free version of the GeoIP City database which is referred to as the GeoLite City database (Maxmind, 2013).

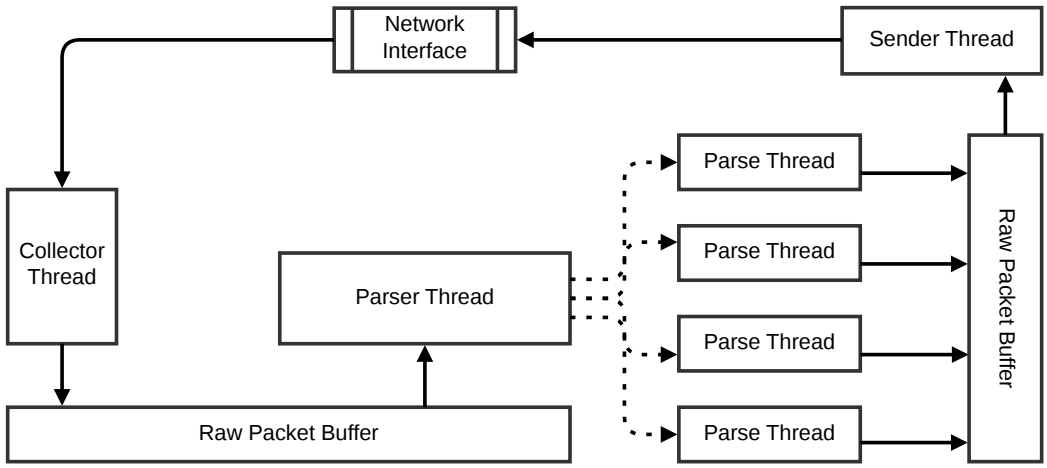


Figure 4.1: Collector implementation

4.2.9 Threading

To create a system that can communicate over an asynchronous medium such as a network, it is important that the implementation be capable of handling multiple events. As these events can happen simultaneously, the system needs to be developed to service them in parallel rather than sequentially. To facilitate this, the system employs threading to run the different subcomponents [3].

4.3 Collector Implementation

The Collector is implemented as a multi-threaded application. To achieve this, the Client is split into three threads which are spawned from the main thread when the Collector is run. These threads are the collector, parser, and sender threads. In figure 4.1, the interaction between threads is done using two buffers.

After the three threads are spawned, the main thread goes into a cycle so that the application does not terminate. In future implementations, the main thread could be used to monitor and report on the status of the application as it could do so without interfering with the functionality of the other threads.

[3]This is done using the Python Threading library.

Upon initialisation, both the collector and sender threads open a socket and bind them to the ports specified in the Collector and Processor variables. For this implementation of the system, the buffer for receiving packets is set to 512 bytes as this is the maximum packet size used by SoftflowD.

4.3.1 Network Protocols

Both the collection and transmission components of the Collector as can be seen in figure 3.2 use the Python Socket library to open a port for UDP communication.

UDP has been chosen over TCP for the transmission as the data being transmitted from the Collector to the Server is small enough to be encapsulated in a single datagram. Because of this, the transmission does not warrant the overhead resulting from setting up and tearing down the TCP connection. The reason for using UDP communication between the collector and the exporter is because SoftflowD exports the generated flows using the UDP protocol which is further discussed in section 3.3.

4.3.2 Collector Thread

The collector thread is implemented as a subclass of the threading class which is included from the python threading library. As described in section 4.2.9, this subclass requires a run method which is called to spawn the thread. During initialisation of the Collector thread, the UDP socket is opened using the credentials specified near the top of file.

When the run method is called, the thread goes into a cycle where is listens on the opened socket for an incoming packet from the exporter. When the collector thread receives a packet from the socket, it attempts to store the packet in the Raw Packet buffer. To do this, the thread first attempts to acquire a lock for the Raw Packet Buffer. If the lock is successfully acquired, the packet is deposited in the buffer. The Raw Packet buffer is implemented as a list with global scope so all threads spawned by the Collector application can access it. Associated with this buffer is a lock which must first be acquired before attempting to use the buffer.

4.3.3 Sender Thread

Implementation of the sender thread uses the same techniques as the collector thread. A subclass of the threading class is defined which includes a run method. During initialisation of a sender thread object, a socket with the Server component is opened.

When called, the run method causes the thread to go into a loop where it continuously checks to see if the Parsed Packet buffer is not empty. The Parsed Packet buffer is implemented as a list with global scope akin to the Raw Packet buffer. Associated with this buffer is another lock which is used by the application to indicate if the buffer is in use or not.

If the buffer is not empty, the sender thread attempts to acquire the associated lock. On a successful attempt, the thread removes a single element from the buffer and passes it to the Server component via the opened socket. Details regarding how the element is packed into a datagram and transmitted is abstracted away by using the python socket library.

4.4 NetFlow Packet Parsing

The actual parsing of the packets is done sequentially by the parser script. The process begins by parsing the header of the packet to determine if it is a data or template record.

If the is a template record, the parser then proceeds to parse the remainder of the record and generates a Template object from the resulting data. This Template object is then serialised and written to a file in the local templates directory using methods supplied by the Pickle library. These files are then available to be read in by other instances of the parser script.

Should the record be a data record, the parser then attempts to read in the associated template record from the local templates directory. If successful, the template file is serialised back into a template object and the data record is parsed according to the template to create a record object. This record object is then returned to the caller of the script. Should the script fail to find the associated file, it will wait for 200 ms before trying again and repeat this three times. If after the third iteration the file is not available, the parsing of that record fails and the record is dropped.

4.4.1 Parser Thread

The parser thread is responsible for removing elements from the Raw Packet buffer and spawning additional parse threads to to parse the elements. It achieves this by continuously monitoring the Raw Packet buffer for elements. When the buffer is not empty, the parser thread checks if the number of currently active parse threads exceeds the thread limit specified near the top of the script.

When the limit is not exceeded, the thread attempts to acquire the lock for the Raw Packet buffer to remove an element. Once an element is removed, a new object of the parse thread class is created. This thread is passed the element and the associated run method is called.

The thread count is incremented and the parser thread goes back to monitoring the Raw Packet buffer.

4.4.2 Parse Thread

The parse thread then calls the parse method from an imported script. This method parses the packet and if it is a data record, returns a record object. The parse thread then takes this record object and serialises it using method supplied by the Python Pickle library. After the appropriate lock is acquired, the resulting data string is inserted into the Processed Packet buffer.

Once complete, the parse thread then decrements the thread count before terminating.

4.5 Server Implementation

The implementation of the server is done in Python relies on the pygeoip and tornado libraries amongst other to function correctly. The implementation of this server is done so that the separate components discussed in chapter 3 can run asynchronously.

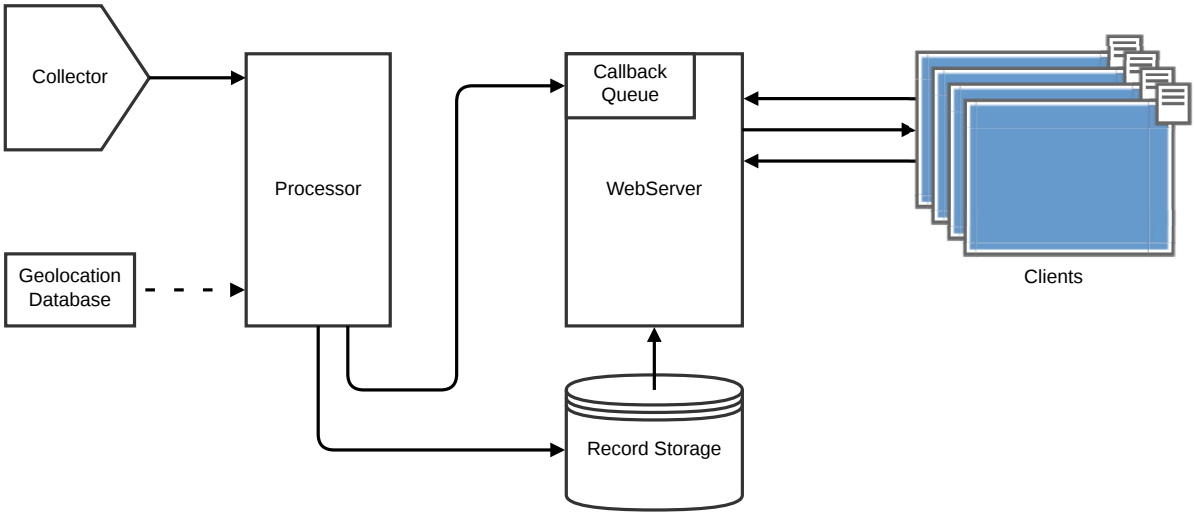


Figure 4.2: Server implementation

4.5.1 Processor Component

As with the transmission and collection components of the Collector, the processor component of the Server uses the Python Socket library. This is used to open a connection for receiving UDP packets as they are sent from the Collector to the Server. Each packet contains exactly one record that has been serialised using the Python library Pickle which is discussed in section 4.2.5.

In figure 4.2, processor component runs as a separate thread so that it does not block other components of the script from executing. This is important as the processor component will block while listening on an opened socket.

Packet Collection

The process of collecting and storing the records is the responsibility of a thread spawned by the main thread of the Server during the start-up phase of the script. As with the threads in the Collector, this thread is implemented as a subclass of the threading class provided by the threading library. This thread opens a socket using the collector credentials and listens for UDP packets. The collector credentials are set in a variable near the top of the script so that a user can easily edit it prior to execution of the script.

When a packet is received from this socket, the thread uses the Python Pickle library to de-serialise the packet payload back into the record object sent from the Collector.

Flow Geolocation

Once the record is de-serialised, the thread then compares the record destination IP address with a list of Internal IP addresses. If the IP address matches any element in this list, it is considered to be an ingress flow and the geolocaion should happen on the Source IP address rather than the destination IP address. Due to the limitation of using softflowd as the network flow exporter, development of the system as been done as a host based system that handles flows for a specific host. As a result, development of the system to handle multiple source ip addresses is limited.

Once the appropriate IP address is determined, the thread uses the pygeoIP Python library to look the address up in a geolocation database. For this implementation, the geolocation database used is the 'GeoLite City' database which is provided for free by MaxMind (MaxMind, n.d.). The lookup returns a geolocation record from the database that includes the country name, country tag, and the geolocation coordinates of the IP address that was used to perform the lookup.

Record Storage

The country tag and geolocation coordinates are extracted and associated with the flow record. In order to store the collected information in the local flow record database, it is first necessary to see if there is a record associated with the flow which has not been terminated yet.The reason for this additional requirement on storing the records is because the exporter is instructed to report on all flows that are active regardless of whether or not they are complete in set intervals. The reasons for this are discussed in section 3.5.

To achieve this, the components of the record that form the primary key are used along with the 'Completed' field which indicates whether the TCP FIN flag has been seen yet to form a SQL query. If the database response to the query is empty, then this is the first time that the flow has been seen and it can be inserted directly by means of an INSERT SQL statement. If the database response is not empty, then the record returned must be merged with the new record to update the duration, bytes transferred, packets transferred, and whether the TCP FIN flag has now been seen. This updated record is then inserted into the database by means of a REPLACE query to update the stored record.

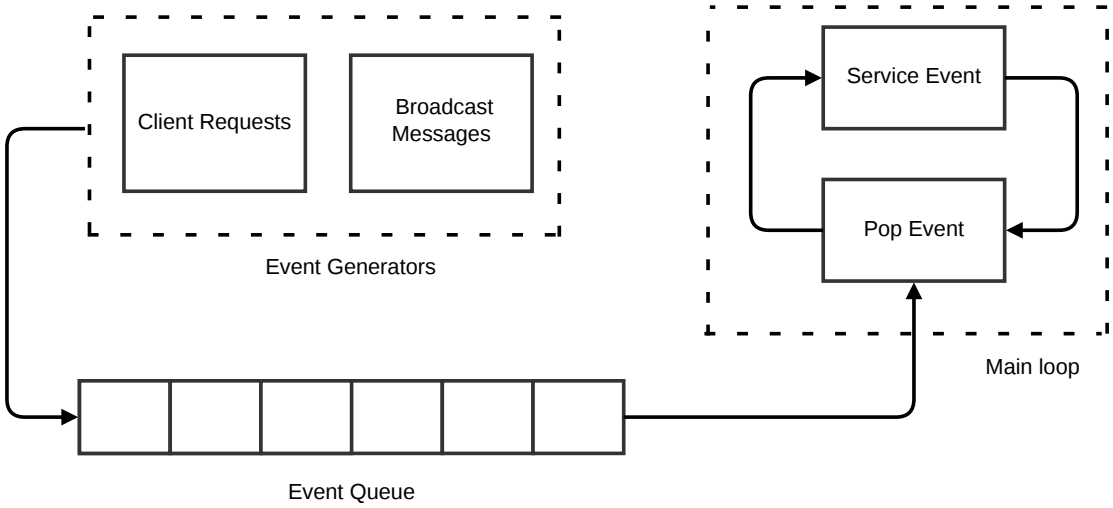


Figure 4.3: Overview of the event handling by the WebServer

Adding to the Callback Queue

Once the record is stored, the thread instructs the tornado main thread to add a signal broadcast call to its callback queue. This method call takes the record that was stored as a parameter which includes the geolocation information and extracts the IP address, the coordinates, and the Country tag. This information is then used to create a JSON string which is then sent to all currently connected clients by using the write message method associated with a tornado WebSocket object.

4.5.2 WebServer Component

The second component of the Server is the webserver component which interfaces with the processor component by servicing callbacks added to an event queue shown in figure 4.3. The implementation of this component is done using tornado which is discussed in section 4.2.6. The webserver component also acts as the communication point between connected Clients and the rest of the system. The communication is handled by maintaining a list of WebSocket objects, one for each connected Client. Broadcasting messages such as a flow tag for real-time rendering is done by iterating through this list and writing the broadcast message to each WebSocket. The webserver component can be seen adjacent to the processor component in figure 3.7.

Table 4.1: List of implemented request tags

Tag No.	Request Type
0	Country Details Request
1	Flow heatmap request
2	Btye heatmap request
3	Packet heatmap request
4	IP heatmap request
5	Flow even replay request

Setting up the WSHandler

The WebSocket is set up by adding a WSHandler object to the tornado HTTPServer class which allows the script to add additional functionality to the basic methods. To add the functionality, the WSHandler bust include a method with the same signature as basic method to be enhanced. This allows the HTTPserver to know to call the method in the WSHandler once it has completed execution of the basic method with the same signature. By using this functionality, the basic 'open' method of the HTTPServer can be overridden to include functionality for storing references to opened web sockets. This is done by maintaining a list of references which the new websocked is appended to.

The same technique is used to allow the HTTPServer to remove a WebSocket from the list when the 'on close' method is called. In this case, the maintained list is iterated through and the reference WebSocket being closed is removed.

Handling Server Responses

For the HTTPServer to respond appropriately to client requests, a list of the possible response functions is kept with the index to each method in the list being associated with a tag element. This element is used in every message between the Client and Server to distinguish between different requests and the associated responses. Should the HTTPServer receive a request with an unknown tag, the request is simply ignored. The possible requests that the Server will respond to are listed int table 4.1.

Country Detail Request

A country detail request requires a country tag which is checked against an internal list to insure the requested tag is valid. The details associated with that country tag are then

collected from the flow record database and returned to the method handling the request. The method then uses the returned information to construct a list of the different IP addresses seen from that country. Associated with each of these addresses is the number of bytes and the number of flows that are recorded for that IP address. This data is packed into a JSON string and sent to the requesting WebSocket.

Heatmap Request

The heatmap requests are differentiated according to the characteristic that is being requested. For each request, a start and end time is required and indicates the duration over which the the country heats must be constructed. As with the country detail request, a query is built and used to request the details from the system database. How the method handling the request then processes the results depends on the type query that was executed which in turn depends on the characteristic that was requested. For the flows, bytes, packets, and IP characteristics, the process is the same; the characteristic is tallied for each returned country and both the minimum and maximum value is recorded. These values are used to create a range. This ratio is then applied to the value associated with each country in the list. The scaled results are used to construct a JSON string to be returned to the requesting WebSocket.

Replay Request

Replay requests are used to generate a list of flow events for the requesting client to render. After the event information is returned from the system database, the method responsible for handling the request constructs a FlowEvent object for each record returned. A FlowEvent requires the IP address, the coordinates, the country tag, and the start time of the event. This object acts as a container that can later produce a string of the elements in the correct order. A JSON string is generated using this functionality of the objects and sent to the requesting WebSocket.

4.6 Web Client Implementation

As discussed in chapter 3, the Client component of the system is designed to run as a web application within the user's browser. To achieve this, the Client component

is implemented as web page and written using HTML and JavaScript, specifically the JQuery library.

4.6.1 Interface Structure

As the Client must contain three distinct components discussed during its design, the interface must be segmented accordingly. For this implementation, segmentation is achieved by using tabs. The interface consists of four tabs, one for each component and an additional tab for the introduction.

For the replay and heatmap components, additional dialogues are required to allow the user to specify details for selecting the information to be rendered. Dialogues are produced using bpopup by Klinggaard[4] to produce the dialogues on user request.

4.6.2 Initialisation

After the Server has returned the Client to a requesting web browser, the Client application begins by rendering the HTML file and calling the JavaScript 'ready' function associated with the rendered document. This function in turn begins the initialisation of the Client by creating a WebSocket connection with the server.

Once the WebSocket has been created, the Client sends the value '-1' to the server. This is done to open the connection between the two components otherwise the WebSocket exists but remains closed until data needs to be passed from the Client to the Server.

Once connection with the Server has been established and the web page is rendered, the start-up phase is considered complete. Initially, none of the Client components are enabled and any information received from then Server regarding them is ignored. When the users wishes to use a Client component, the initialisation of that component will begin when the relevant tab is selected.

4.6.3 WebSocket Communication

Once start up is complete, the Client listens on the opened WebSocket. When data is received via this socket, it is run through the parseJSON function of the jQuery library.

[4]http://dinbror.dk/bpopup/

After the string has been parsed, the application enters a switch statement using the 'tag' element in the parsed string as the switching expression. The cases of this statement determine which component of the Client the parsed string is for. If the associated component has not been initialised yet, or the tag is not serviceable by the Client, it is discarded.

Sending data to the Server via the WebSocket is the responsibility of the relevant components. The format of each message is a string of comma separated values. The first value is a tag which indicates the type of request and the remaining values are discussed further in the different component sections.

4.6.4 Map Initialisation

The initialisation of each interactive map is achieved by following the same format.

When a tab is selected for the first time, the interactive map for that component is initialised. To do this, a Raphael object is created using the current tab window size. Following the object creation, a path is added to the Raphael object for each path list in the paths JavaScript file. Associated with each path list is the country name, flow count, and map colour.

Two dictionaries are created for storing the generated paths, a dictionary for storing country names, and a dictionary for storing the actual paths. In the country names dictionary, a country name of each path is stored by using the path id as a key. The second dictionary also uses the country path id as a key but pairs it with the entire path which is then stored.

Functionality for responding to a mouse hover event is added to each path which renders a small popup box which includes the country name. Additional functionality provided is discussed in the separate components. After all paths have been generated, the dictionary of paths is then scaled to fit the Raphael object. An example of the resulting interactive maps is depicted in figure 4.4 and is similar to the map generated by the Honeynet project [5]. This similarity is expected as the JavaScript path file used in this implementation was generated using the paths associated with the Honeynet Project map.

[5]The Honeynet Project map HoneyMap is discussed in section 2.6

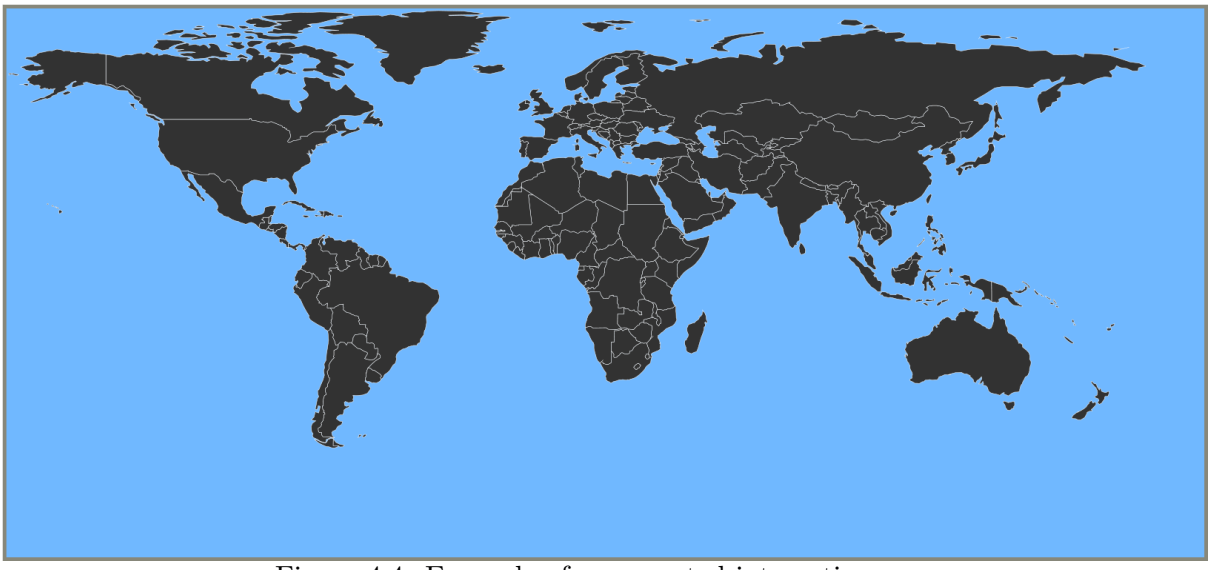


Figure 4.4: Example of a generated interactive map

4.6.5 Live Communication

The initialisation of this component begins when the user selects the tab labelled 'Realtime'. The creation of the interactive map follows the format described in section 4.6.4 above with the exception of the functionality added to the paths. In addition to displaying the country name when the 'onHover' event occurs, the realtime map also reports on the number of flows seen to that country.

Additional functionality is also added for on click events. When the user selects a country, a pop is displayed which includes a table. The able is populated with the IP addresses recorded from that country along with the total number of flows seen from that IP address and the total volume of data transferred.

The rendered information is received as a response to a realtime request from the Server application. The request is generated when the user selects a country. This request includes the appropriate tag and the time range the response should cover. The time range is set to be the duration the realtime component has been active for.

Once the map has been generated, any flow event the Client receives from the Server will be rendered. In order to render a new event, the geolocation coordinates are first transformed into position coordinates on the map. This is done by scaling the geolocation coordinates by the realtime map dimensions. Using these scaled coordinates, a Raphael circle object is created to be rendered at that location on the map. This object is coloured yellow and is animated to 'pulse' during rendering. The new circle object is then stored in a JavaScript array.

Table 4.2: Possible heatmap characteristics

Tag	Characteristic
1	Flows
2	Bytes
3	Packets
I4	P addresses

Once the number of stored events exceeds a limit of 200 elements, the JavaScript shift function is called which shifts the array contents to the left and the first array element is removed.

4.6.6 Heatmap Generation

The heatmap component is initialised when the user selects the Heatmap tab and beings by generating the interactive map. The generation of the heatmap display follows the format described in section 4.6.4 and does not include any additional functionality.

To use this component, the user must select a generate heatmap button which causes the Client to render a popup window containing components to aid in generating a heatmap request. The components displayed are; a drop down list for specifying the characteristic being rendered, a colour picker to select the heat colour of the generated map, and a date range slider for specifying the time window that the Server should generate the map for.

The items in the drop down list are characteristics that the Server can produce heat mapping of. The current implementation supports four characteristics tabulated in table 4.2.

The colour picker is a component that is not used to generate the Server request but the result is stored by the Client. Until changed, all heatmap response messages received from the Server are rendered using the selected colour. This colour is changed to the default colour on each heatmap request unless specified otherwise.

The date range slider is implemented using the jQDateRangeSlider library supplied by Gautreau[6]. The slider allows the user to select a time range with day resolution by moving and resizing the slider.

When the user clicks submit, the values from the date range slider and drop down list are used to generate a string of three comma separated values. The first value is an integer

[6]http://ghusse.github.io/jQRangeSlider

representing the selected characteristic. The remaining two values are the start time and end time respectively. The time values are measured as seconds since the epoch. The generated string is then sent to the Server via the maintained WebSocket.

The structure of the Server response message is discussed in section 4.5.2 which is converted from a JSON string into a container containing a tag and an array. To render the response data as a heatmap, the Client iterates through the response array and colours the heatmap countries accordingly. The colouring is done as an animation. An example rendered heatmap for each characteristic shown in table 4.2 can be found in appendix C. The four heatmpas were generated using the same source data.

4.6.7 Historical Replay

The replay component of the Client is initialised once the user selects the Replay tab. Once selected, the Client generates an interactive map as described in section 4.6.4. After the map has been rendered, the user can initiate a replay action by selecting the replay button which generates popup window.

The popup contains two components to aid in generating the replay request string. The first component is a simple slider for selecting the speed at which the traffic should be replayed. The second component is a range slider for selecting the time range for replaying traffic.

The traffic replay speed is stored by the client while the date range is used to generate a comma separated string along with the appropriate to act as a request to the Server. When the WebSocket receives a replay message from the Server and the replay map has been initialised, replay can begin.

The data returned form the Server includes an array of flow events. This array must then be iterated through at the user specified speed.

The replay function uses the JavaScript SetInterval function which calls a function passed as a parameter in set intervals. The called function uses a global variable called position that indicates the replay position in the event array. Once the position reaches the end of the array, the replay function is stopped as all events have been replayed.

On each iteration, the called function loops through the events, checking the associated time of each until a time is found that exceeds the current replay time. For each checked event, a Raphael circle object is created and rendered in the same manor as described in

subsection 4.6.5 and the global position variable is incremented. Once the current replay time is exceeded, the time is increased according to the user defined speed or replay and the function ends.

4.7 Reporter Implementation

To implement the Reporter discussed in chapter 3, a Python library called reportlab has been used. This libray is further discussed in section 4.2.7. The Reporter is dependant on the host providing the system database and is not available from the web client. As a result, the implementation of this system is done with the requirement that it be run with a database connection.

The application begins by ensuring the correct arguments have been passed upon start-up. Regular expressions are used to match the date and IP arguments against patterns. The date pattern insures that the month component lies between 1 and 12 and the year component is a four digit number. The pattern for matching the IP address requires that a IPv4 address is passed in dotted quad notation. This pattern checks to insure that the address lies in the range from '0.0.0.0' to '255.255.255.255'.

The moth argument is the passed to a function which returns the start date and end date of the month in seconds since the epoch. This is done using the 'mktime' and 'strptime' functions of the python time library. These two times are used in all data requests as the time range for the request.

4.7.1 Chart Colours

To keep colours uniform throughout the report and insure it that once a country has been assigned a colour, no other country may use that colour, a dictionary of colours is created. Associated with this dictionary is a function 'getColour' that returns the colour associated with the country passed to it as a parameter.

This function works by attempting to extract a colour from the dictionary by using the passed parameter as a key. If the parameter does not exist in the dictionary, a colour is popped off a list of predetermined colours and associated with it. The pair is added to the dictionary and the colour is returned.

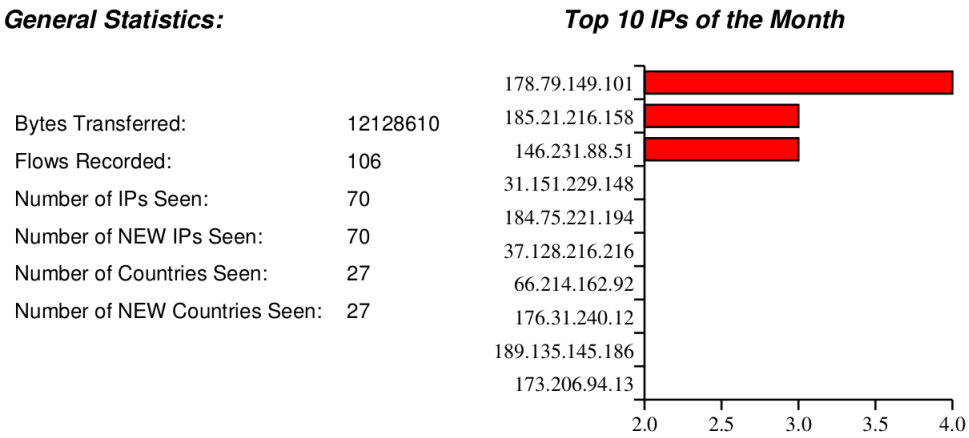


Figure 4.5: Example of a generated list of statistics and IP chart

4.7.2 Report Structure

As stated in chapter 3, the structure of the generated report is static which allows the Reporter application to generate the report procedurally. An example of the generated report can be found in appendix B. To generate a procedural report, a list containing all the drawings is used and built into the PDF document. The order in which the drawable elements appear in the list indicates the order in which they are rendered.

The structure of the report begins with a title which includes the Month that the report is generated for. This is the first object to be appended to the report list.

To have multiple objects render with the same vertical coordinates, a Table object is needed. The table can then have multiple columns, each with a Drawing object contained. A table is used regularly throughout the remainder of the report.

Characteristics List and Bar Chart

The first two components of the report are a list of general statistics and a bar chart. An example of these two components is depicted in figure 4.5 which is placed directly beneath the report title.

For each characteristic shown in figure 4.5 and discussed in chapter 3, a SQL query is performed to get the associated value from the system database. The query is limited to time ranges representing the beginning and end of the specified month.

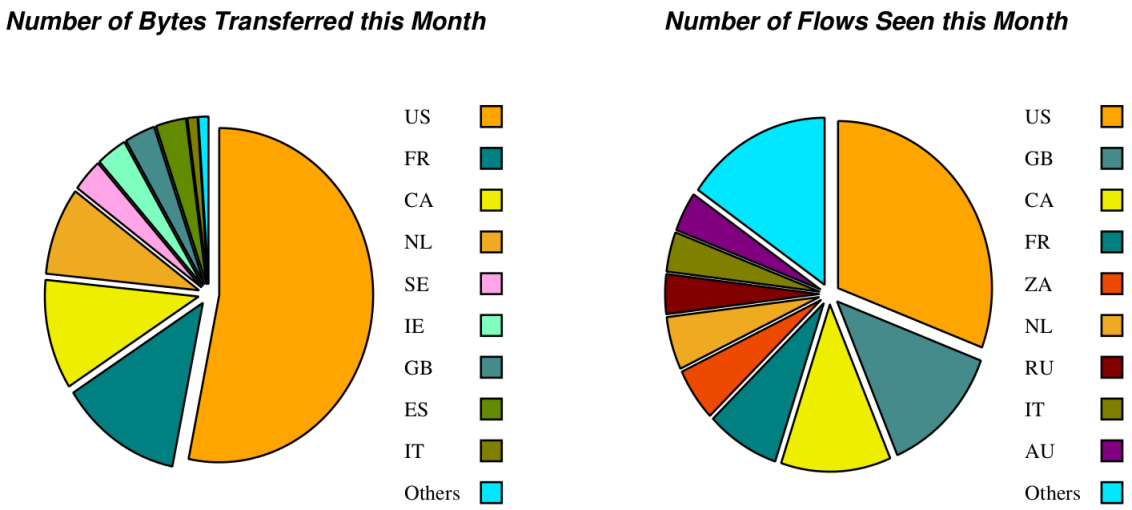


Figure 4.6: Example of generated pie charts

The horizontal bar chart represents the top 10 IP addresses seen in the specified month. The IP addresses are retieved from the system database usng a SQL query and the image is rendered from a HorizontalBarChart object. In figure 4.5, the x-axis represents the flow count and the value range is designed by report lab to span from the smallest element to the largest element rendered [7].

Pie Charts

The pie charts are are used to represent the amount of data transferred and the number of flows seen to up to ten countries. Figure 4.6 shows and example of the pie charts that the current implementation can generate. For each chart, the associated function call is made to get a weighted list of countries from the system database. The weight of each element in the returned list represents the volume of traffic seen and is determine wedge size.

In both cases, the generation of the chart is the same. The list of countries and values that is returned from the function call is split into two separate lists, one for the country names and the other for the values. A colour list is then generated using the names list and the 'getColour' function.

Using the generated lists, a reprotlab Pie object is created and populated. A reportlab Legend object is also created for presenting the countries and the associated colours. A

[7] Axis labels are not possible in the current version of reportab and so no axis in this report are labelled.

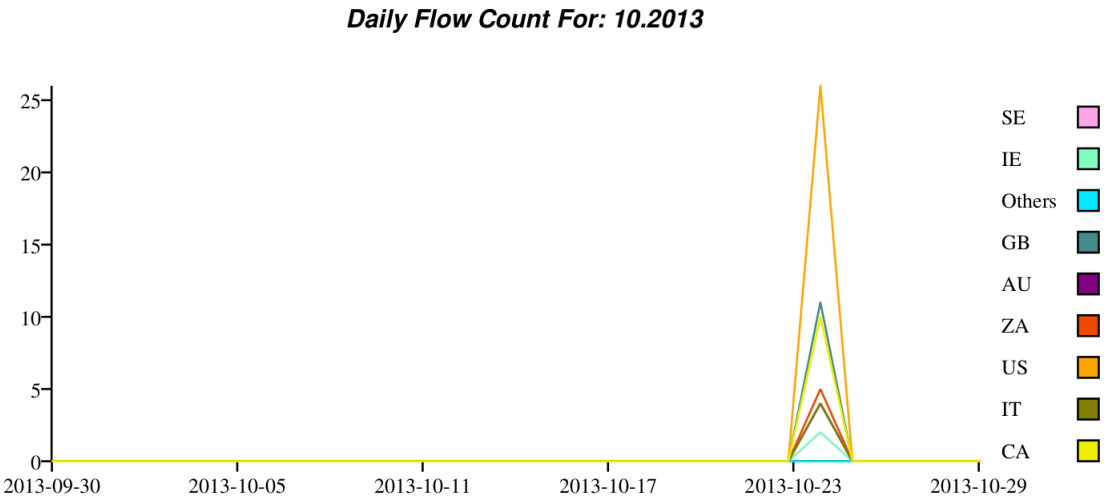


Figure 4.7: Example of a generated line graph

separated object is necessary for the legend as the Pie object does not have a native legend component.

Line Graphs

The report line graphs towards the end of a report represent the volume of bytes and number of flows seen by the system. This data is represented as a daily progression for the 10 most significant countries seen previously in the report [8].

The example line graph in figure 4.7 was generated for a single instance of traffic and depicts each country rendered using a different colour. Similar to other generated images, a database query is used to extract the relevant data from the system database. This data is a list of the 10 countries to be rendered along with traffic values for each day in the specified month. The values are dependant on the type of line chart generated.

As with the pie charts, the returned data is split into separate lists and the country names are used to create a list of colours. A reportlab LinePlot object is created using these lists and a legend object is associated with it. The legend, positioned to the right of the image, relates each colour to its assigned country code.

[8] Country significance is currently determined by the average amount of data seen each day.

Report Generation

After all the necessary drawings have been added to the report list, the list is built on a document template by calling the build function associated with that template. This function handles the generation of the document and stores the result as a pdf.

4.8 Summary

Python and JavaScript were the programming languages selected to implement the system. This selection was motivated in section 4.2 which is followed by a brief overview of significant libraries used as part of the implementation. The remainder of this chapter is devoted to the implementation of system designed in chapter 3. The approach taken in each component is documented and discussed in further detail to highlight how the design logic was implemented.

The next chapter discusses the testing and evaluation of the system implemented here. The system is also run in a live environment to further test and demonstrate the functionality of the implementation.

Chapter 5

System Evaluation

5.1 Introduction

This chapter focuses on testing and verification of the implementation of the system described in chapters 3 and 4. The testing checks conformance of the geolocation components as well as system performance with regards to timing and generated traffic. These tests are done to insure the implementation is capable of achieving the system goals defined in section 3.2.

Following testing, the system is used in a live environment to demonstrate some of the implemented functionality. The demonstration is performed on a large file transfer using a peer to peer file sharing. The system was then used to visualise the network traffic for the duration of the transfer.

This chapter concludes with a chapter summary which briefly discusses the results found during the testing of the implemented system.

5.2 Geolocation conformance testing

The system implemented in chapter 4 is capable of performing geolocation on network flows and rendering the result on a world map. To confirm that the implementation is correct and the system performs the geolocation adequately, conformance testing is necessary.

The conformance testing will use an external application to perform geolocation on a known IP address. The coordinates returned will be rendered on a google map to visibly display the address location. Network traffic will be simulated which uses the known IP address and replayed to softflowd. This simulated traffic will then be passed to the system and the rendered flow event on the world map will be recorded. The two rendered maps will be compared and a conclusion will be drawn regarding the accuracy of the system in terms of flow geolocation.

5.2.1 Tools Used

The system tests were performed using generated test data for which the results were recorded. In order to generate the test data and record the results, a series of applications were employed during the testing phase of the system.

tcpdump As part of the generation of test data, a valid sample traffic set needed to be recorded for modification. The collection of these packets was done with tcpdump[1]. tcpdump was set to listen on the loopback interface of the testing platform and record all traffic seen. The application then stored the recorded traffic in a pcap file.

tcpreplay Once traffic has been collected and stored by tcpdump, the next stage was to modify the file to produce the appropriate test data. These modified files were then replayed back onto the interface for the system to detect. The replaying was performed using tcpreplay[2].

scapy After traffic has been saved into a pcap file, it then needs to be modified to so that it can represent traffic to and from specified host addresses. Packet modification was performed using a Python script build using the Scapy library. This library was written by Biondi and provides functionality for packet manipulation as well as saving and loading network traffic stored in pcap files[3].

softflowd The system has been implemented to function as a network flow collector and processor but cannot generate the network flows itself. To generate the flows, an application called softflowd[4] was employed.

[1]http://www.tcpdump.org/release
[2]http://prdownloads.sourceforge.net/tcpreplay
[3]http://www.secdev.org/projects/scapy/files/scapy-latest.tar.gz
[4]http://code.google.com/p/softflowd/downloads/list

Maxmind Online Geolocation To test the accuracy of the geolocation component of the system, an application capable of performing geolocation was required. The results from this application would then be compared with the result of the implemented system. The application chosen was the online GeoIP2 Precision Demo provided by MaxMind (MaxMind, 2013). This application uses the same geolocation database as the system and so will make inaccuracies incurred by the current implementation apparent.

Google Maps To actually render the geolocation results from the online geolocation demo, an system was needed that could accept coordinates and plot them on a map. The system used during testing to achieve this was Google Maps which is an online map rendering service designed by Google[5].

Wireshark The traffic produced by the system was recorded using a network protocol analyser called Wireshark[6]. The traffic was then filtered and statistics of the traffic were generated.

5.2.2 Method

To generate the test traffic to be used, two options were considered. The necessary packets could be constructed entirely in the test bed or legitimate traffic could be collected and modified to suit the needs of the test bed. After considering the difficulties involved in either approach, it was decided that modifying legitimate traffic would be the optimal choice.

The legitimate traffic was generated using a simple Client/Server model which consists of a Client script and a Server script. The Server script creates and listens on a socket for TCP connections. Then a connection is received, the server reads the contents of the traffic being passed through the connection and prints it before closing the connection. The Client script begins by creating a TCP connection with the Server script. Once the connection is established, the Client passes a string message to the Server script before terminating.

The traffic generated from running this model was written to a pcap file using tcpdump. The resulting pcap file was then read into a Python script which used the Scapy library to modify the packet headers. The modified data was then written into a new pcap file.

[5]https://maps.google.com
[6]http://www.wireshark.org/download.html

Table 5.1: Table of IP addresses used in conformance testing

IP Address	Country
42.69.78.7	Taiwan
189.60.150.30	Brazil
119.81.44.63	Singapore
142.231.123.92	Canada
82.154.2.224	Portugal
146.231.123.92	South Africa

To replay the modified traffic back onto the wire, tcpreplay was used which read the pcap file and reproduced the necessary packets.

The replayed traffic was processed by by softflowd which in turn generated the raw flows and exported them to the implemented system. The system results were then rendered on the realtime map of a connected Client via the web browser.

Addresses Used

A list of six IP addresses were chosen to to be used in the geolocation testing and are tabulated in table 5.1. These six were chosen as the coordinates they resolve to lie near the edges of the rendered map. Furthermore all these addressees lie near a coastline. These conditions are advantageous for testing as it makes any inaccuracies in the geolocation rendering component more apparent.

External Geolocation

After selecting a set of IP addresses, the next step was to find the associated coordinates of each. To do this, the online GeoIP2 Precision Demo was used. The resulting table is displayed in figure 5.1.

Google Map Rendering

GeoIP2 Precision City/ISP/Organization Results

IP Address	Country Code	Location	Postal Code	Coordinates	ISP	Organization	Domain	Metro Code
42.69.78.7	TW	Taipei, Taipei, Taiwan, Asia		25.0392, 121.525	Mobile Business Group, Chunghwa Telecom Co., Ltd.	Mobile Business Group, Chunghwa Telecom Co., Ltd.	hinet.net	
189.60.150.30	BR	Rio de Janeiro, Rio de Janeiro, Brazil, South America		-22.9, -43.2333	Virtua	Virtua	virtua.com.br	
119.81.44.63	SG	Singapore, Asia		1.3667, 103.8	SoftLayer Dutch Holdings B.V.	SoftLayer	softlayer.com	
142.231.123.92	CA	Vancouver, British Columbia, Canada, North America	V6B	49.2836, -123.1041	BCnet	BCnet		
82.154.2.224	PT	Lisbon, Lisbon, Portugal, Europe		38.7167, -9.1333	PT Comunicacoes	PT Comunicacoes	telepac.pt	
146.231.123.92	ZA	Rhodes, Province of Eastern Cape, South Africa, Africa	9787	-30.7954, 27.9647	Rhodes University	Rhodes-Univ	ru.ac.za	

Figure 5.1: Geolocation information returned by the GeoIP2 Precision Demo

System Geolocation Rendering

Using the coordinate results tabulated in figure 5.1 and Google Maps, the geographic location of each coordinate was rendered. Figure 5.2 shows a screen capture for each result.

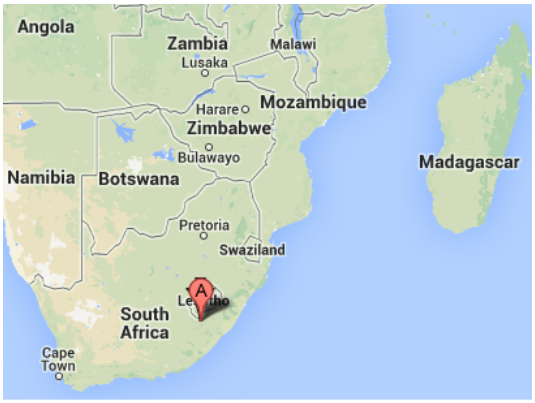
The figures in 5.3 were generated from screen captures after replaying the simulated traffic for each IP address onto the loopback interface of the testing platform. This traffic was recorded by softflowd which generated the appropriate network flows. The flow events were then rendered on the interactive map of a connected Client showing the system geolocation results for each IP addresses.

5.2.3 Results

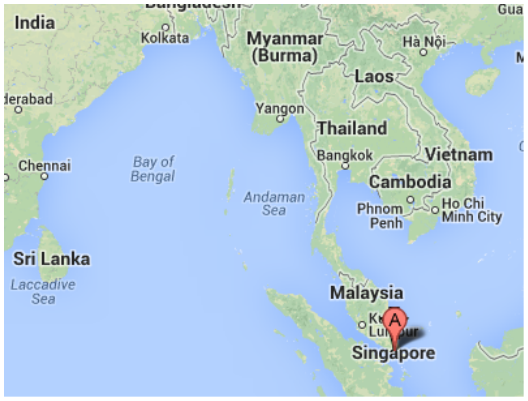
Comparing the images rendered in google maps to the images rendered from the implemented system, it is clear that there is a close correlation. To confirm that the coordinates generated by the system implementation match the coordinates displayed in figure 5.1, the associated records are extracted from the system database. The returned records are shown in table 5.2.

Inaccuracies do however exist in the geolocation databases that the implemented system relies on. The IP address geolocated in sub figures 5.2(b) and 5.3(b) belong to Rhodes university which is not situated at the geolocated coordinates. The physical location of this institute is recorded using Google Maps as marker B in figure 5.4. The distance between the actual location of the host and the geolocated position is 310.69 kilometres. Relative to a global scale, this inaccuracy is not large to consider geolocation unsuccessful but should warrant concern regarding geolocation accuracy.

(a) Google map lookup for IP: 189.60.150.30



(b) Google map lookup for IP: 146.231.123.92



(c) Google map lookup for IP: 119.81.44.63

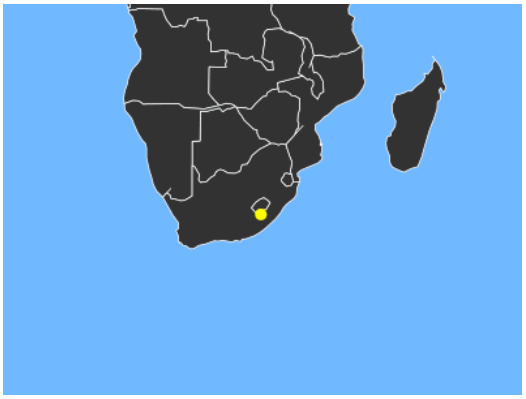
(d) Google map lookup for IP: 142.231.123.92

(e) Google map lookup for IP: 42.69.78.7

(f) Google map lookup for IP: 82.154.2.224

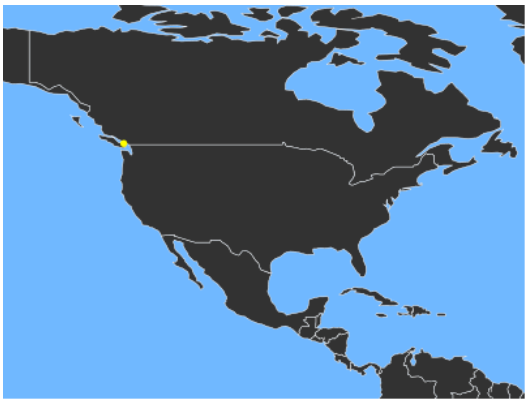
Figure 5.2: Geolocation using Google Maps for coordinates from 5.1

(a) Rendered flow event for IP: 189.60.150.30



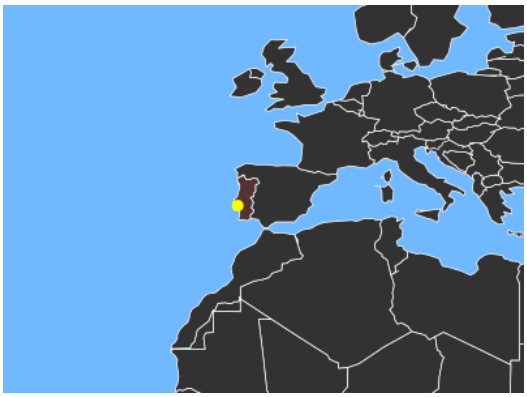
(b) Rendered flow event for IP: 146.231.123.92

(c) Rendered flow event for IP: 119.81.44.63



(d) Rendered flow event for IP: 142.231.123.92

(e) Rendered flow event for IP: 42.69.78.7



(f) Rendered flow event for IP: 82.154.2.224

Figure 5.3: Images rendered by system for geolocation

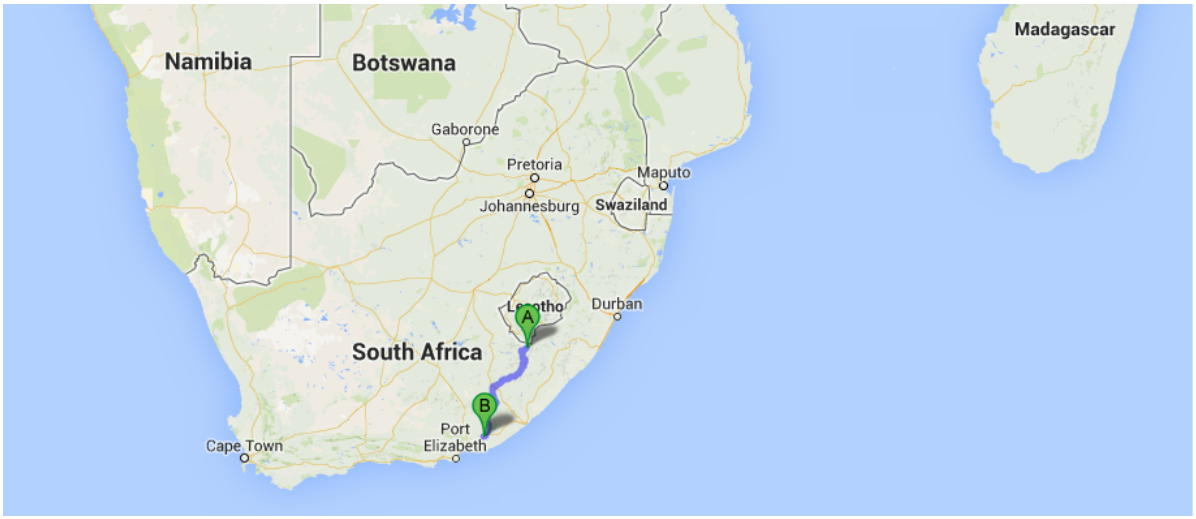


Figure 5.4: Physical[B] and Geolocated[A] location of IP 146.231.123.92.

Table 5.2: Database Records Extracted

Destination IP	Packets	Bytes	Latitude	longitude	Country Code
42.69.78.7	5	280	25.0392	121.5250	TW
189.60.150.30	5	280	-22.9000	-43.2333	BR
119.81.44.63	5	280	1.3667	103.8000	SG
142.231.123.92	5	280	49.2836	-123.1041	CA
82.154.2.224	5	280	38.7167	-9.1333	PT
146.231.123.92	5	280	-30.7954	27.9647	SA

5.3 System Performance Testing

To test the implemented system, performance with regards to timing and data transferred must be monitored while the system is in use. Tests are performed for different stages of the system to determine the time required to generate a geolocated flow event from an exported raw network flow.

Traffic use was recorded using Wireshark for interactions between the Server and the Client. These tests were performed multiple times to determine if excessive traffic was being generated by the implemented system.

5.3.1 Timing and Accuracy

Timing tests were performed for significant stages in the implemented system. These stages include the time taken to interface with the database for record storage as well as the time taken to perform a geoIP lookup in the geolocation binary file. Other stages of interest are the time taken to parse a raw packet received from the export device and the time required to serialise a single record for transmission.

System Process Time

The time taken for the implemented system to process a raw flow and produce a flow event is recorded to be approximately 0.05 seconds or 50 milliseconds. This is performed by recording the time when the system receives a raw network flow to the time the associated flow event is broadcast to the connected Clients. The recorded times are shown in table 5.3.

These results are measured from the time a packet is received by the Collector to when it is broadcast from the Server. As a result, it excludes the transmission time from the Server to the Client. It was decided not to perform timing tests between the implemented Server and connected Clients as the duration is heavily dependant on the quality of the network path between the two components and considered out of scope for the prototype system as mentioned in section 1.3.

Table 5.3: Sample recorded results of time taken to process a network flow

Test No.	Time Packet Received	Time First Record Sent	Duration [s]
1	1381000626.1056400	1381000626.1600400	0.0543940
2	1381001156.9537300	1381001157.0111500	0.0574150
3	1381001211.8136500	1381001211.8485900	0.0349381
4	1381001301.8416500	1381001301.8949600	0.0533080
5	1381001356.3536600	1381001356.4073800	0.0537219
6	1381001536.2776300	1381001536.3262600	0.0486290
7	1381001686.5536400	1381001686.6038200	0.0501890
8	1381001771.8176900	1381001771.8710300	0.0533390
9	1381001866.4496300	1381001866.4996200	0.0499859
10	1381001921.6336400	1381001921.6834500	0.0498040

Table 5.4: Recorded results of time taken to parse received packets

Test No.	Parse Time	Record Count	Average Time per record
1	0.001602	5	0.000320
2	0.000115	2	0.000058
3	0.000284	9	0.000032
4	0.000174	6	0.000029
5	0.000178	4	0.000045
6	0.000313	8	0.000039
7	0.000249	9	0.000028
8	0.000335	9	0.000037
9	0.000300	8	0.000038
10	0.000518	9	0.000058

Packet Parse Time

To test the time taken to parse a received packet, the time the packet was removed from the Raw Packet buffer and the time the parsing of it completed is recorded. The number or records returned is also recorded. A selection of the recorded results is shown in table 5.4.

As can be seen in table 5.4, the first entry is an order of magnitude out in the average time taken to compute a single record. The reason for this is because the first packet received from the collector included template records as well as data records. The recording process only counted the number of data records returned ad so parse time is not accurately presented in the first row. Omitting the first row and computing the average time taken to parse a single record for these results yields a result of 40.2 microseconds. The first row has been omitted when determining the average as the effect will become insignificant

Table 5.5: Recorded results of time taken to serialise records

Test No.	Serialise Time	Record Count	Average Time per Record
1	0.000117	4	0.0000293
2	0.000052	1	0.0000520
3	0.000073	3	0.0000243
4	0.000053	2	0.0000265
5	0.000084	3	0.0000280
6	0.000085	3	0.0000283
7	0.000081	2	0.0000405
8	0.000237	4	0.0000593
9	0.000060	2	0.0000300
10	0.000044	1	0.0000440

when using larger sample sets.

Record Serialisation Time

The time taken to serialise a set of records is also recorded. The time before and after the serialisation of a record set is recorded as is the number of records serialised. From these results, the average time taken to serialise a single packet is computed. The recorded result are shown in table 5.5. The average time taken to serialise a single record for the recorded data shown in table 5.5 is 0.0000362 seconds which is equivalent to 36.2 microseconds.

Geolocation and Record Storage

Once the flow records propagate to the Server component of the system, the time taken to perform geolocation lookups and record storage is recorded. The testing is done for 20 records and the recorded results are displayed in table 5.6.

The average time taken to perform a single geolocation lookup and a single record storage are computed to be:

```
Geolocation lookup average = 0.0001500588 seconds
Record storage average     = 0.0251362941 seconds
```

From these results, the average time required for the individual components is 0.0252 seconds. The remaining 0.0248 seconds are required for passing the serialised records over

Table 5.6: Times recorded for geolocation lookup and record storage

Test No.	Geolocation Time	Storage Time
1	0.000036	0.041609
2	0.000069	0.000483
3	0.000036	0.024312
4	0.000084	0.000325
5	0.000276	0.037505
6	0.000106	0.000351
7	0.000071	0.000387
8	0.000063	0.060549
9	0.000090	0.000327
10	0.000269	0.032560
11	0.000212	0.030819
12	0.000235	0.033044
13	0.000203	0.032948
14	0.000230	0.033151
15	0.000186	0.032813
16	0.000177	0.033023
17	0.000208	0.033111

the network from the Collector to the Server. As this time will vary depending on the network path between the Collector and Server, timing analysis is not performed.

5.3.2 Bandwidth

Traffic statistics are recorded for communication between the Server and a connected client. The stitics produces show the volume of traffic seen between the two applications during a request and response to different user requests.

Realtime Traffic

As traffic for realtime is continuous and independent of the user, the traffic statics are generated for a period of time. The number of flow events exported over this duration is also recorded. From these two recordings, the average volume of traffic per flow event is determined. The results of these tests are recorded in table 5.7. From the recorded statics, the average size of a flow event is calculated to be 204.33 bytes. A host averaging 10 flows a minute would generate a total of 1.4 MB of traffic in 24 hours which is an insignificant amount of traffic to be passed through an internal network.

Table 5.7: Recorded statistics of realtime traffic

Test No.	Event Count	Packet Count	Duration [s]	Total Bytes	Average Event Size [b]
1	394	789	648	80576	204.51
2	1019	2038	595	207769	203.89
3	2857	5714	9760	583940	204.39
4	19	38	70	3886	204.53

Table 5.8: Recorded statistics of heatmap traffic

Test No.	Characteristic	Packet Count	Total Bytes
1	Flow	4	5251
2	Packets	3	5185
3	Bytes	3	5186
4	IP	3	5185
5	Flow	3	5185
6	Packets	3	5185
7	Bytes	3	5186
8	IP	3	5185

Heatmap Traffic

Traffic statics for the heatmap component are constant between requests and the same volume of data is returned per request and the request id formatted to be a set size. Traffic generated for different requests is monitored using Wireshark and the resulting statics are reported in table 5.8.

From the recorded data, the average number of bytes required to respond to a heatmap request is calculated to be 5193.5 bytes or 5.07 kilobytes. As an individual Client can only request one heatmap at a time the volume of traffic transferred for a heatmap request will not scale for an individual Client. The volume of traffic will scale for depending on the number of Clients should simultaneous requests occur.

Replay Traffic

The replay component of the Client is capable of producing the greatest volume of traffic per request. Traffic testing for this component is done for different requests. For each request, traffic statistics of the response from the server are tabulated 5.9 along with the number of events returned.

Form the recorded traffic, the average number of bytes required to transmit a single record

Table 5.9: Recorded statistics of replay traffic

No.	Record Count	Packet Count	Total [b]	Average Record Size [b]
1	724	11	69794	96.40
2	724	9	69662	96.22
3	724	7	69530	96.04
4	724	7	69530	96.04
5	724	7	69530	96.04
6	724	7	69530	96.04
7	724	11	69794	96.40
8	724	9	69662	96.22
9	724	7	69530	96.04
10	724	7	69530	96.04

from the Server to the Client is calculated to be 96.15 bytes. This implies that transferring up to 10905 flow events would generate 1 MB of traffic. The number of flow events that the system will return to a connected Client is dependant on the time range the Client has selected and the number of flows seen during that time range. The generated traffic will also scale according to the number of simultaneous requests the server receives.

5.4 System Demonstration

To demonstrate the functionality of the implemented system, a large file was downloaded using peer to peer file sharing system. The file downloaded was a distribution of the Linux Mint 15 operating system which is free to download[7] which is 959.4 MB in size.

The system was initialised and softflowd was used to monitor the download and generate the raw network flows. The file downloaded at an average of 18 MiB/s and a system report was generated. This report can be found in appendix B. After the download had completed, the system was used to generate a series of heatmaps for the different flow characteristics.

From the generated maps, the comparison of two characteristics is of particular interest.

[7] http://www.linuxmint.com/edition.php?id=132.

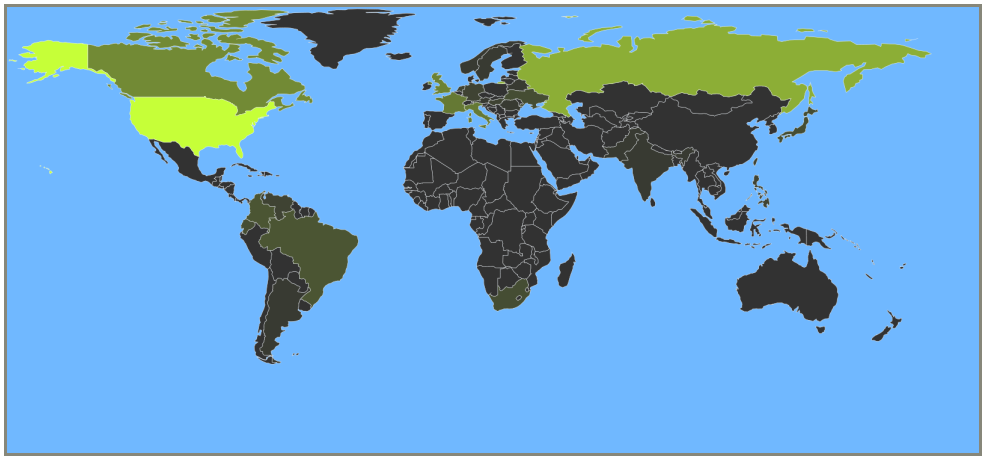


Figure 5.5: Generated heatmap for flows seen

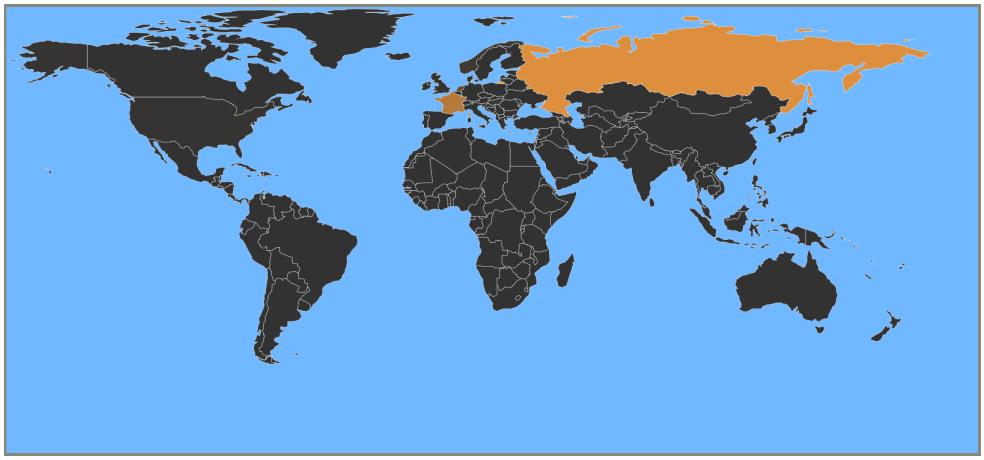


Figure 5.6: Generated heatmap for data transferred

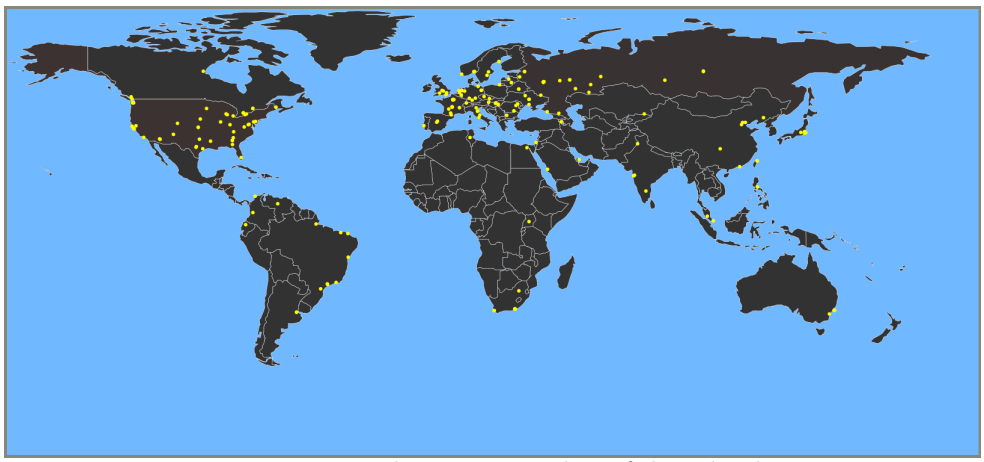


Figure 5.7: Flow event replay of download

Table 5.10: System processing time

Component	Time performed [ms]	Time performed [%]
Parsing	0.04	00.0804
Serialisation	0.072	00.1448
Transmission	24.6	49.2021
Geolocation	1.5	00.3001
Storage	25.136	50.2726

The first characteristic is figure 5.5 which is a heatmap where the colouring represents the number of flows seen to and from a specific country. The second characteristic depicts byte transfer where the heat indicates the volume of data seen from a specific country. The byte heatmap is depicted in figure 5.6.

Figure 5.5 indicates the countries to which were connected to during the download. As expected, the heatmap shows results similar to the replay map shown in Figure 5.7 which shows the geolocated origins of these flows. Figure 5.6 describes a different result and shows countries Russia and France outweigh the rest of the world with regards to where the data was transferred from. This difference is also highlighted by comparing the line graphs of the generated report in Appendix B. This comparison shows how the system could be used to show the difference between flow count and flow volume. Similar comparisons can be done for packet count and the number of different IP addresses connected to in each country.

5.5 Summary

This chapter describes the testing performed on the implemented system. This testing begins by confirming the accuracy of the geolocation rendering by by comparing map renderings generated by Google Maps with the realtime map generated by the system. Traffic between the Server and a Client application is also recorded to determine the volume of data and number of packets used by each subcomponent of the Client.

Form the results of the test performance, it is shown that the system geolocation behaves correctly and accurately plots the test data. Inaccuracies do exist but are due to geolocation errors in the geolocation database and are not a fault of the prototype system. From timing results, it is seen that the system is capable of generating flows in a suitable time with the largest delay arising from database communication.

Further analysis of the timing results shows the distribution of the time required to process a NetFlow packet into a geolocated flow event. Theses timings are presented as percentage values of the total time taken and tabulated in table 5.10. These timings show that 99.47% of the time taken to parse a single record is required to interface with the MySQL database and transmit the record from the Collector to the server using the TCP protocol. The remaining 0.53% is used to parse, serialise, de-serialise, and geolocate the record. This indicates that, should if become necessary to reduce the time required to parse a network flow record, efforts should focus on minimising the timing requirements relating to communication between the two components and interfacing with the record storage.

The bandwidth tests show that the volume of traffic generated by the implemented system is not excessive and could be handled but most modern networks. Bandwidth requirements will scale according to the number of simultaneously connected Clients as well as the duration of a replay request. The system is then used to demonstrate some of the implemented functionality and the visualised results are discussed. A report is also generated as part of the demonstration which can be found in Appendix B. The next chapter concludes the results recorded in this chapter and relates these findings to the research goals stated in chapter 1.

Chapter 6

Conclusion

6.1 Introduction

This chapter begins with a summary of the chapters included in this thesis. This summary is followed by a conclusion of the results found from performing this research in section 6.3. Section 6.4 restates the research goals and the approach taken to achieve them. Section 6.5 concludes the research covered and relates the results recorded to the research goals. This chapter ends with a discussion of future work that could be done to extend the current system or the develop a new software based network flow export system.

6.2 Summary of Chapters

A description of the problem statement is introduced in section 1.1 which is followed by a discussion of the research goals. The scope of this research is presented in section 1.3 which specifies and justifies limitations imposed on the research. The chapter concludes with a document structure that describes the format followed by the remainder of this document.

Literature relating to an investigation into network flows is presented in chapter 2. The term network flow is explained in section 2.2 and this includes a discussion of devices that support the generation of network flows. Protocols NetFlow version 5 and NetFlow version 9 are reviewed in section 2.3 where the benefits and difficulties of each are compared. Following this, IPFIX is described and compared to NetFlow version 9. A brief investigation into network flow collectors is done to gain a better understanding of some

of the common approaches taken in collection and processing. This chapter also discusses security threats such as DDoS, port scanning, and worms. Both the DDoS and worm discussions include an example system designed to detect the threat using network flows.

The design of the prototype system is covered in chapter 3. The goals of the system are defined in section 3.2 as well as system constraints in section 3.3. These constraints are employed to simplify system development by eliminating features that are not required to achieve the system goals. Hardware and other resource considerations are discussed in section 3.5 and are focused on bandwidth and memory requirements. The remainder of the chapter is devoted to the design of the necessary system components and concludes with a chapter summary.

Chapter 4 continues on from chapter 3 by documenting the implementation of the designed system. This chapter also justifies the use of Python and JavaScript as the languages used for implementation in section 4.2. This section also covers the libraries used as part of the system implementation. Once the implementation of each component has been documented, the chapter concludes with a summary of the material covered.

Testing of the implemented system is the focus of chapter 5 and is split into three parts. The first part is conformance testing of the geolocation component the system. The test was carried out by preforming geolocation on known IP addresses and comparing the recorded results with images rendered by the system for the same test data. System performance tests were then done to determine the time required for the system to process an exported flow into a geolocated flow event. The final testing component was performed to demonstrate the functionality of the implemented system by running it in a live environment. The system recorded flow events associated with a torrent download. From the recorded results, two heatmaps are generated and compared. A report of the event can be found in appendix B.

6.3 Concluding results

This research was performed to explore the use network flows in traffic visualisation and geolocation to aid in cyber defence. Two goals are defined in chapter 1 which this research attempts to achieve. The first goal is to attempt the development of a prototype system capable of geolocation and traffic visualisation. The second goal is an investigation into using network flows and geolocation to produce a real time map rendering of flow endpoints.

6.3.1 Prototype System

It is shown through conformance testing in section 5.2 that the geolocation provided by the implemented system is suitable for geolocating network flows on a global scale. Comparisons between the coordinates produced using the online GeoIP2 Precision Demo and the coordinates stored in the system database show that the geolocation and storage components function correctly. The flow events rendered using these coordinates match tags produced using Google Maps.

Inaccuracies do however exist in the geolocation databases which are discussed in section 5.2.3. This error is due to inaccuracies in the geolocation database rather thatn a fault of the implemented system. This issue could be rectified by using a more accurate geolocation database but for the current implementation, the database used is still considered acceptable.

Bandwidth results generated for server responses showed that traffic generated by the system is mostly dependant on the number of connected Clients. For traffic replay events, the range specified to be replayed and the number of flows recorded during that period also effects the volume of traffic generated. As a result, the traffic generated by the system will not be excessive if the number of connected clients is limited.

By using the implemented system in a live environment it is shown that the results rendered by using the visualisation components are useful in analysing an event. Comparing the heatmap in figure 5.5 with the heatmap depicted in figure 5.6 is is shown that there is no immediate relationship between the number of flows seen and the volume of traffic transferred from a specific country for a torrent. This result is expected and the goal of the demonstration is to show how the heatmaps can be used to visualise different characteristics by using network flows.

6.3.2 Real-time Geolocation

System timings reveal that the time taken by the system to render a received flow event is approximately 50 milliseconds. This testing was done on specific hardware so results my vary depending on the platform. The total timing arising from network communication will increase if the system is hosted across multiple platforms and if the Client is situated a considerable distance from the Server. Considering this, timings of communication between the Client and Server have not been tested.

In section 2.3.3, timings relating to when a network flow export system will announce the expiration of a network flow are discussed. This will only occur when the network flow expires as the default configuration of most devices is to only exporter expired flows. An alternative is to configure the system to periodically broadcast the state of all monitored flows which allows the geolocation system to specify a maximum latency between when a flow begins and when it is exported to the collector. The consequence of using a periodic broadcast is the volume of traffic generated and the additional processing overhead the collector will require as discussed in section 2.3.3. As these configuration options do not form part of the NetFlow protocol specifications so timings regarding network flow export in non Cisco systems do not need to conform to these configurations.

Should the exportation of network flows be performed with suitable latency, constructing a real-time geolocation system similar to the prototype system developed as part of this research is feasible. A consequence is that the designed system will be dependent of the configurable options of the system responsible for generating and exporting the network flows.

6.4 Research Review

This research had two goals that it attempted to achieve which were:

- An exploration into the use of network flows for traffic visualisation to aid in administration and network security.
- The use of network flows for realtime geolocation and coordinate rendering.

The approach taken in this research was to design and implement a prototype system capable actually performing traffic visualisation and geolocation. The system could then be used to explore the timing concerns that could hinder the development of a realtime geolocation system using network flows. The prototype system could also be used to visualise the recorded network flows for different characteristics and to produce a report displaying statistics of the recorded traffic.

6.5 Closing Statement

Considering the results collected from testing the prototype system and functionality shown during the system demonstration, it can be concluded that network flows are a feasible source of data for traffic visualisation and geolocation. Treating the exported characteristics associated with each flow as packet summaries, analysis using network flows allows the user to quickly acquire a good overview of the condition of the associated network. A drawback to this approach is the loss of resolution and so the user will be unable to evaluate packet contents if using network flows.

One use of network flows found in this research is for the geolocation of end hosts that communicate through the monitored network. By using flows, geolocation systems can have the source or destination host of a flow exported as a characteristic. The advantage of this is the reduced amount of data needed to be monitored for new IP addresses to geolocate. In traditional systems, this is done by evaluating the header of every sampled packet to determine id a new host address has been seen. Using network flows, the monitoring needs only be done once for each network flow when it is first announced by the exporting system.

Two concerns were raised during this research regarding the feasibility of using network flows for geolocation. The first concern relates to the timing and latency of using network flows and is discussed in section 2.3.3. It is concluded that, with proper configuration of the exporter, a maximum latency can be established that is short enough to be suitable without causing the exporter to generate excessive traffic volumes. The second concern relates to the accuracy of the geolocated results. It was concluded that this is not the a consequence of using network flows but an inaccuracy in the implemented geolocation database. As a result, this concern is valid for geolocation systems regardless of whether network flows or packets are used as the traffic source data.

6.6 Future Work

- As specified in the system constraints in section 3.3, the current implementation does not attempt to support IPv6. The reason for this is IPv4 addresses are suitable to explore the use of network flows for geolocation and data visualisation. Should the system later be required to monitor and geolocate IPv6 addresses, then the system should be extended to do so.

- The protocol the implemented system accepts is NetFlow version 9. The reasons for this decision is discussed in section 3.2, one of which was the similarities between NetFlow version and IPFIX. As a result, an extension of the current system could be to allow the system to accept and process raw network flows being exported under the IPFIX protocol.
- The reporting tools in the current implementation have been designed as a proof of concept and to show the type of information that the system can report on. This section can be improved to include more thorough reporting tools, especially if the system is extended to accept a larger set of characteristics. The reporting functionality should also be tailored to suit the field that the system will be implemented in.

6.7 Development of a Suitable Flow Export System

Throughout the development and testing of the implemented system, the source of network flows has been Softflowd. As discussed in chapter 3, this is a software based application capable of generating and exporting raw network flows. Softflowd uses the UDP protocol for packet transmission which incurs risks as dependent systems require that exported packets are guaranteed to arrive, especially packets containing template records. Another issue with using Softflowd is that the exported characteristics are defined in the source code and cannot be specified by the connected system or the user at run time[1]. Future work in developing a suitable software based flow export system should be considered. Such a system could be based of softflowd or extend it to make it more suitable for systems reliant on a range of characteristics and flow durations. Features the implemented to the extended system should include:

- Develop the system to be platform independent.
- Allow records to be exported using the IPFIX protocol.
- Allow the user to specify the underlying protocols that the application should use to export the flows.
- Functionality to allow the user to specify the characteristics exported when using the NetFlow version 9 or IPFIX protocol.

[1] http://code.google.com/p/softflowd/source/checkout

- Proper handeling of timing configurations to mimic a NetFlow compatible router or switch developed by Cisco as discussed in section 2.3.3.
- Complete documentation of system functionality.

References

Barracuda. 2013. How to Configure Audit & Reporting With IPFIX. TechLibrary. Available from: http://techlib.barracuda.com/pages/viewpage.action?pageId=6979841. Accessed on 27 October 2013.

Braden, & IETF. 1989 (October). *Requirements for Internet Hosts – Communication Layers*. RFC 1122. Internet Engineering Task Force (IETF).

Case, Fedor, Schoffstall, & Davin. 1990. *A Simple Network Management Protocol (SNMP)*. RFC 1098. Network Working Group.

Cisco. 2006. NetFlow Export Datagram Format. Online. Available from: http://www.cisco.com/en/US/docs/net_mgmt/NetFlow_collection_engine/3.6/user/guide/format.html. Accessed on 12 May 2013.

Cisco. 2011a. Cisco IOS NetFlow Version 9 Flow-Record Format. Online. Available from: http://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.pdf. Accessed on 16 May 2013.

Cisco. 2011b. Configuring NetFlow and NetFlow Data Export. Online. Available from: http://www.cisco.com/en/US/docs/ios-xml/ios/NetFlow/configuration/12-4/cfg-nflow-data-expt.html. Accessed on 14 May 2013.

Cisco. 2012a (Jan). *Catalyst 6500/6000 Switches NetFlow Configuration and Troubleshooting*. Cisco. Document ID: 70974.

Cisco. 2012b. NetFlow Reliable Export With SCTP. Online. Available from: http://www.cisco.com/en/US/docs/ios-xml/ios/NetFlow/configuration/12-4t/nf-12-4t-book.pdf. Accessed on 14 May 2013.

Cisco. 2012c. NetFlow Services Solutions Guide . Online. Available from: http://www.cisco.com/en/US/docs/ios/solutions_docs/NetFlow/nfwhite.html. White Paper Accessed on 23 October 2013.

Claise, Systems, Cisco, Trammell, & Zurich. 2013 (Feb). *Specification of the IP Flow Information eXport (IPFIX) Protocol for the Exchange of Flow Information.* RFC.

Claise, B. 2008 (Jan). *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information.* RFC 5101. Internet Engineering Task Force(IETF).

Claise, B., & Systems, Cisco. 2006 (October). *Cisco Systems NetFlow Services Export Version 9.* RFC 3954. Network Working Group.

Cliquet, Gerard. 2006. *Geomarketing. Methods and Strategies in Spatial Thinking.* Vol. 1. ISTE Ltd.

Ennis, Jennifer. 2013. pygeoip 0.2.7. Online. Available from: `https://pypi.python.org/pypi/pygeoip/`. Accessed on 11 October 2013.

Erickson, Jon. 2008. *Hacking: The art of exploitation.* Second edition edn. No Starch Pr.

Estan, Keys, Moore, & Varghese. 2004. Building a Better NetFlow. *SIGCOMM*, **1**(September), 245 – 257.

Facebook. 2013. Tornado. Online. Available from: `http://www.tornadoweb.org/en/stable/`. Accessed on 5 October 2013.

FriendFeed. 2013. FriendFeed API. Online. Available from: `http://friendfeed.com/api/`. Accessed on 4 August 2013.

Fullmer, Mark. flow-capture. Man page. Available from: `http://www.splintered.net/sw/flow-tools/docs/flow-capture.html`. Accessed on 14 May 2013.

Fullmer, Mark. flow-tools. Online. Available from: `http://www.splintered.net/sw/flow-tools/docs/flow-tools.html`. Accessed on 15 Jun 2013.

Gldi, Christoph, & Hiestand, Roman. 2005 (April). *Scan Detection Based Identification of Worm-Infected Hosts.* Ph.D. thesis, Institut fr Technische Informatik und Kommunikationsnetze.

Juniper Networks, Inc. 2013. Junos OS Routing Protocols Overview. Online. Available from: `http://www.juniper.net/techpubs/en_US/junos13.1/information-products/pathway-pages/config-guide-routing/config-guide-routing-overview.pdf`. Accessed on 21 May 2013.

Kerr, & Bruins. 1996. Network flow switching and flow data export. Online. Available from: `http://www.google.com/patents/US6243667`. Patent.

Klinggaard, Bjrn. jQuery.bPopup.js. Online. Available from: `http://dinbror.dk/bpopup/`. Accessed on 12 August 2013.

Klyne, Graham, & Newman, Chris. 2002. *Date and time on the internet: Timestamps.* Technical report. IETF. RFC. 3339.

Lakkaraju, Kiran, Yurcik, William, & Lee, Adam J. 2004. PNVisionIP: NetFlow visualizations of system state for security situational awareness. *Pages 65–72 of: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security.* ACM.

Lee, Changyong, Kim, Hwankuk, Jeong, Hyuncheol, & Won, Yoojae. 2010 (June). *Analysis of SIP Traffic Behavior with NetFlow - based Statistical Information.*

Lellelid, Jim Winstead; Hans. 2013. Net GeoIP. Online. Available from: `http://pear.php.net/manual/en/package.networking.net-geoip.php`. Accessed on 14 October 2013.

MacVittie, Lori. 2012. Geolocation and Application Delivery. Online. Available from: `http://www.f5.com/pdf/white-papers/geolocation-wp.pdf`. Accessed on 14 May 2013.

MaxMind. MaxMind. Online. Available from: `http://www.maxmind.com/en/home`. Accessed on 12 May 2013.

MaxMind. 2013. GeoIP2 Precision Demo. Online. Available from: `http://www.maxmind.com/en/geoip_demo`. Accessed on 22 October 2013.

Maxmind. 2013. PGeoLite Free Downloadable Databases. Online. Available from: `http://dev.maxmind.com/geoip/legacy/geolite/`. Accessed on 14 June 2013.

Messer, James. 2007. *Secrets of Network Cartography. A Comprehensive Guide to Nmap.* Second edition, revision 2 edn. Professor Messer Publication.

mhaske Dhamdhere, Vidya, & Patil, Prof. G.A. 2012. NetFlow method used for internet worm detetion. *International Journal of Scientific & Engineering Research*, **3**(March), 1–7.

Miller, Damien. 2010. Readme file of flowd. Online. Available from: `http://code.google.com/p/flowd/wiki/README`. Accessed on 14 May 2013.

Miller, Damien J. 2011. Softflowd. Online. Available from: `http://www.mindrot.org/projects/softflowd/`. Accessed on 14 June 2013.

Mills, D., Delaware, U., J. Martin, Ed., ISC, & Burbank, J. 2010 (June). *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. Internet Engineering Task Force (IETF).

Mirkovic, Jelena, & Reiher, Peter. 2004. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, **34**(2), 39–53.

Morken, Jan Tore. 2010 (June). *Distributed NetFlow Processing Using the Map-Reduce Model*. Computer and Information Science, Norwegian University of Science and Technology.

NCC, RIPE. 2011. The Internet Registry System. Online. Available from: `http://www.ripe.net/internet-coordination/internet-governance/internet-technical-community/the-rir-system`. Accessed on 14 June 2013.

nTop. 2010. nBox, nProbe, n2disk Users Guide. Online. Available from: `http://www.nmon.net/UsersGuide.pdf`. Accessed on 23 May 2013.

nTop. 2013. nProbe: An Extensible NetFlow v5 v9 IPFIX. Online. Available from: `http://www.ntop.org/products/nprobe/`. Accessed on 16 October 2013.

Proctor, Paul E. 2001. *The Practical Intrusion Detection Handbook*. Vol. 1. Prentice Hall PTR.

Project, The Honeynet. Honeymap. Online. Available from: `http://map.honeynet.org/`. Accessed on 15 May 2013.

Quittek, Jürgen, Aitken, Paul, Meyer, Jeff, Claise, Benoit, & Bryant, Stewart. 2008. *Information model for ip flow information export*. Technical report. IETF.

Rekhter, Y., Li, T., & Hares, S. 2006 (Jan). *A Border Gateway Protocol 4 (BGP-4)*. RFC 4217. Internet Engineering Task Force (IETF).

ReportLab. 2013 (May). *ReportLab PDF Library User Guide*. Users Guide 2.7. ReportLab, ReportLab PDF Library User Guide ReportLab Version 2.7 Document generated on 2013/05/07 20:18:53 Thornton House Thornton Road Wimbledon London SW19 4NG, UK.

Scheck, Michael, & CSIRT, Cisco. 2009. NetFlow for Incident Detection. Online. Available from: `www.first.org/global/practices/NetFlow.pdf`. Accessed on 27 August 2013.

Staniford, Stuart, Paxson, Vern, & Weaver, Nicholas (eds). 2002 (Aug). *How to 0wn the Internet in Your Spare Time.* USENIX Security Symposium, vol. 11. USENIX Association, San Francisco, California, USA.

Systems, Cisco. 2004. *Catalyst 4500 Series Switch Cisco IOS Software Configuration Guide.* 12.2 edn. Cisco, Cisco Systems, Inc. 170 West Tasman Drive San Jose, CA 95134-1706 USA.

Trammell, Boschi, Mark, Zseby, & Wagner. 2009 (Oct). *Specification of the IP Flow Information Export (IPFIX) File Format.* RFC 5655. Internet Engineering Task Force (IETF).

van Riel, Jean-Pierre, & Irwin, Barry. 2006. InetVis, a visual tool for network telescope traffic analysis. *Pages 85–89 of: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa.* ACM.

Watson, David, Clune, Arthur, Riden, Jamie, & Mumford, Steve. 2013. The Honeynet Project. Online. Available from: `http://www.honeynet.org/`. Accessed on 15 October 2013.

Yang, Guang. 1997 (Nov). *Introduction to TCP/IP Network Attacks.* M.Phil. thesis, Iowa State University, Department of Computer Science Iowa State University Ames, IA 50011.

Zou, Cliff, Towsley, Don, & Gong, Weibo. 2006. On the performance of Internet worm scanning strategies. *Performance Evaluation*, **63**(7), 700–723.

Zseby, Tanja, Quittek, Juergen, Claise, Benoit, & Zander, Sebastian. 2004. *Requirements for IP Flow Information Export (IPFIX).* Technical report. Internet Engineering Task Force(IETF).

Appendix A

Exportable Characteristics

The following table lists all characteristics currently exportable using the NetFlow version 9 protocol (Cisco, 2011a). Selected characteristics have been discussed in chapter 2.

Value	Field Type	Length (bytes)
1	IN_BYTES	N (default is 4)
2	IN_PKTS	N (default is 4)
3	FLOWS	N
4	PROTOCOL	1
5	SRC_TOS	1
6	TCP_FLAGS	1
7	L4_SRC_PORT	2
8	IPV4_SRC_ADDR	4
9	SRC_MASK	1
10	INPUT_SNMP	N
11	L4_DST_PORT	2
12	IPV4_DST_ADDR	4
13	DST_MASK	1
14	OUTPUT_SNMP	N
15	IPV4_NEXT_HOP	4
16	SRC_AS	N (default is 2)
17	DST_AS	N (default is 2)
18	BGP_IPV4_NEXT_HOP	4

19	MUL_DST_PKTS	N (default is 4)
20	MUL_DST_BYTES	N (default is 4)
21	LAST_SWITCHED	4
22	FIRST_SWITCHED	4
23	OUT_BYTES	N (default is 4)
24	OUT_PKTS	N (default is 4)
25	MIN_PKT_LNGTH	2
26	MAX_PKT_LNGTH	2
27	IPV6_SRC_ADDR	16
28	IPV6_DST_ADDR	16
29	IPV6_SRC_MASK	1
30	IPV6_DST_MASK	1
31	IPV6_FLOW_LABEL	3
32	ICMP_TYPE	2
33	MUL_IGMP_TYPE	1
34	SAMPLING_INTERVAL	4
35	SAMPLING_ALGORITHM	1
36	FLOW_ACTIVE_TIMEOUT	2
37	FLOW_INACTIVE_TIMEOUT	2
38	ENGINE_TYPE	1
39	ENGINE_ID	1
40	TOTAL_BYTES_EXP	N (default is 4)
41	TOTAL_PKTS_EXP	N (default is 4)
42	TOTAL_FLOWS_EXP	N (default is 4)
43	*Vendor Proprietary*	
44	IPV4_SRC_PREFIX	4
45	IPV4_DST_PREFIX	4
46	MPLS_TOP_LABEL_TYPE	1
47	MPLS_TOP_LABEL_IP_ADDR	4
48	FLOW_SAMPLER_ID	1
49	FLOW_SAMPLER_MODE	1
50	FLOW_SAMPLER_RANDOM_INTERVAL	4
51	*Vendor Proprietary*	
52	MIN_TTL	1
53	MAX_TTL	1
54	IPV4_IDENT	2
55	DST_TOS	1
56	IN_SRC_MAC	6

57	OUT_DST_MAC	6
58	SRC_VLAN	2
59	DST_VLAN	2
60	IP_PROTOCOL_VERSION	1
61	DIRECTION	1
62	IPV6_NEXT_HOP	16
63	BPG_IPV6_NEXT_HOP	16
64	IPV6_OPTION_HEADERS	4
65	*Vendor Proprietary*	
66	*Vendor Proprietary*	
67	*Vendor Proprietary*	
68	*Vendor Proprietary*	
69	*Vendor Proprietary*	
70	MPLS_LABEL_1	3
71	MPLS_LABEL_2	3
72	MPLS_LABEL_3	3
73	MPLS_LABEL_4	3
74	MPLS_LABEL_5	3
75	MPLS_LABEL_6	3
76	MPLS_LABEL_7	3
77	MPLS_LABEL_8	3
78	MPLS_LABEL_9	3
79	MPLS_LABEL_10	3
80	IN_DST_MAC	6
81	OUT_SRC_MAC	6
82	IF_NAME	N
83	IF_DESC	N
84	SAMPLER_NAME	N
85	IN_ PERMANENT _BYTES	N (default is 4)
86	IN_ PERMANENT _PKTS	N (default is 4)
87	* Vendor Proprietary*	
88	FRAGMENT_OFFSET	2
89	FORWARDING STATUS	1
90	MPLS PAL RD	8 (array)
91	MPLS PREFIX LEN	1
92	SRC TRAFFIC INDEX	4
93	DST TRAFFIC INDEX	4
94	APPLICATION DESCRIPTION	N
95	APPLICATION TAG	1+n
96	APPLICATION NAME	N
98	postipDiffServCodePoint	1
99	replication factor	4
100	DEPRECATED	N
102	layer2packetSectionOffset	
103	layer2packetSectionSize	
104	layer2packetSectionData	
105 to 127	UNUSED	

Appendix B

Sample system report

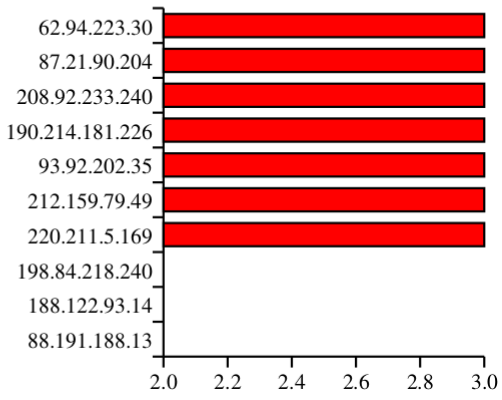
Below is a report generated by the implemented system. The report was generated from the flows processed during the file download which is discussed in section 5.4

Flow Report For: 10.2013

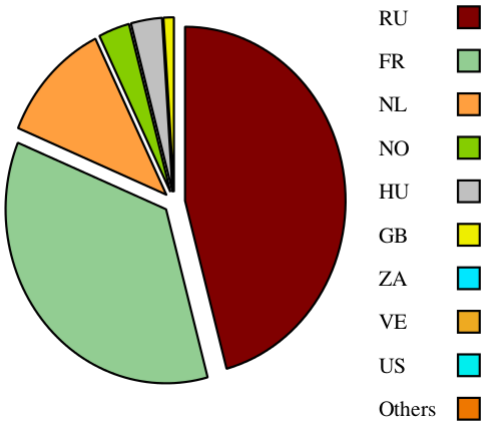
General Statistics:

Bytes Transferred:	14315318
Flows Recorded:	112
Number of IPs Seen:	75
Number of NEW IPs Seen:	75
Number of Countries Seen:	26
Number of NEW Countries Seen:	26

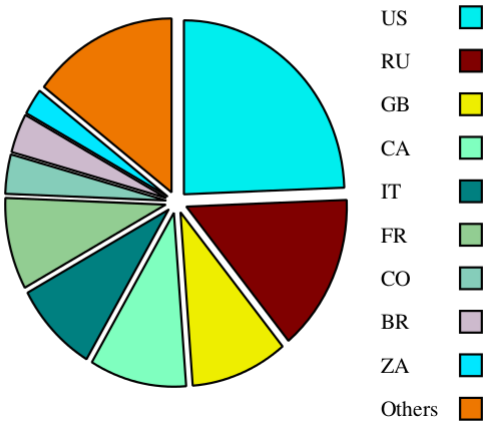
Top 10 IPs of the Month



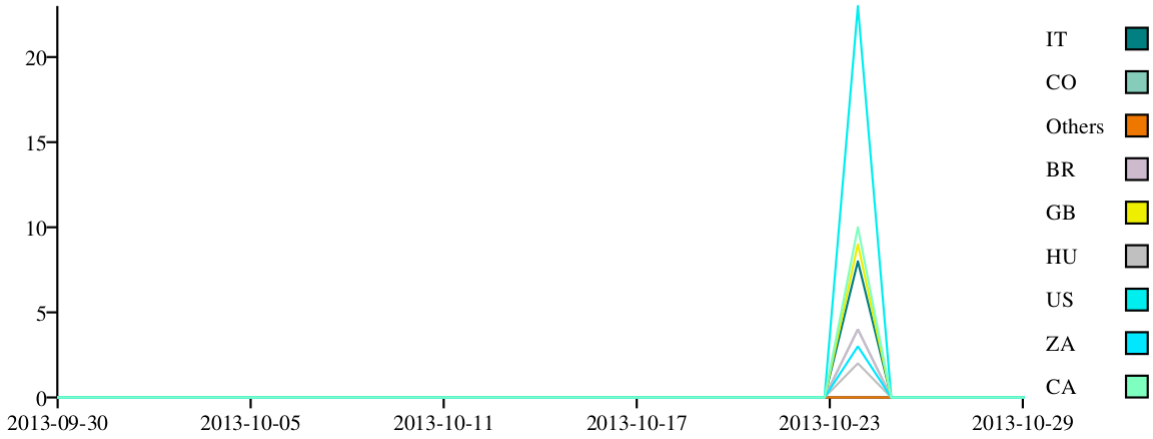
Number of Bytes Transferred this Month



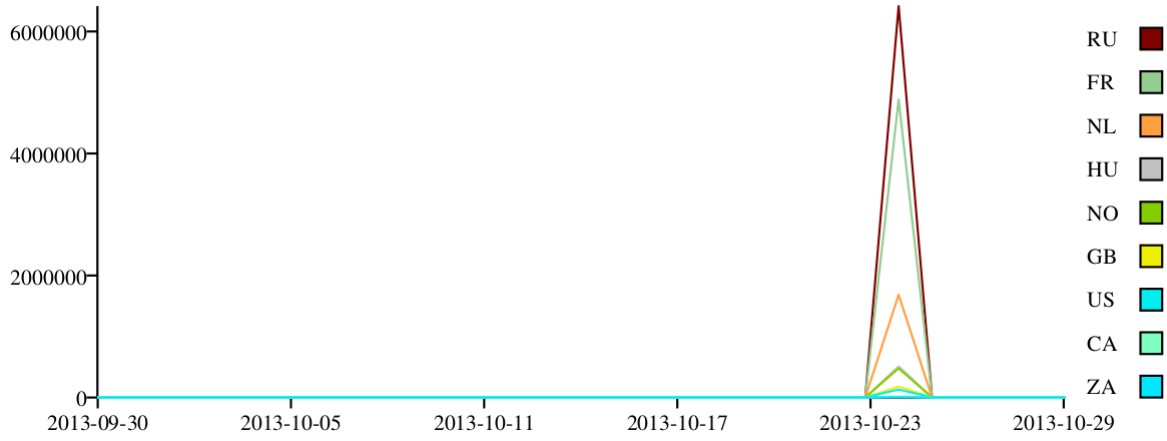
Number of Flows Seen this Month



Daily Flow Count For: 10.2013



Daily Byte Total For: 10.2013



Appendix C

Sample Heatmaps

Below are heatmaps generated by the prototype system for a singe sample data set. The sample data set was generated by performing a large download of the Linux distribution CrunchBang 11 "Waldorf"[1]. This file was downloaded using peer to peer file sharing.

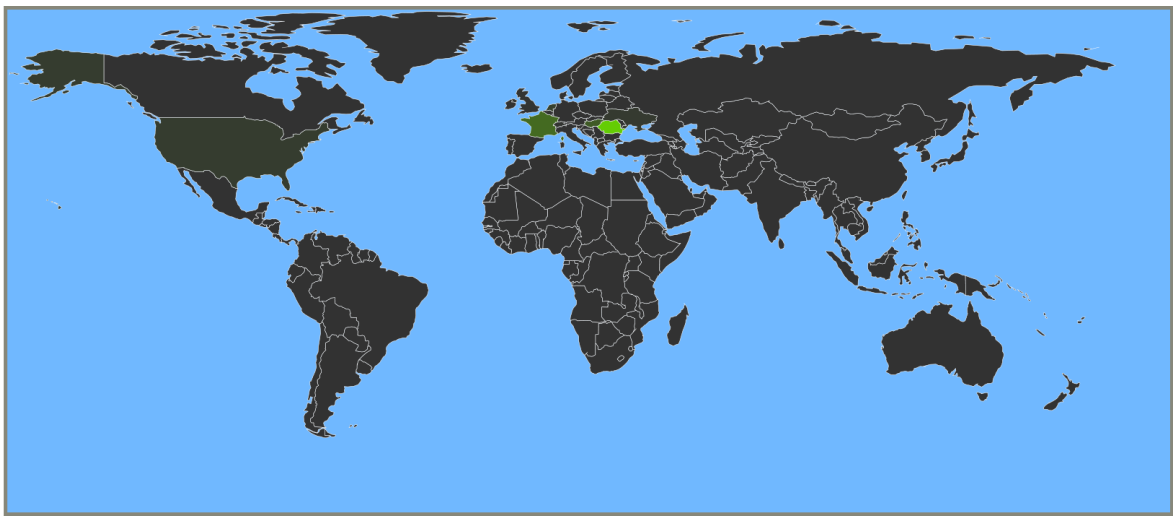


Figure C.1: Heatmap of total bytes transferred

[1] http://crunchbang.org/download/

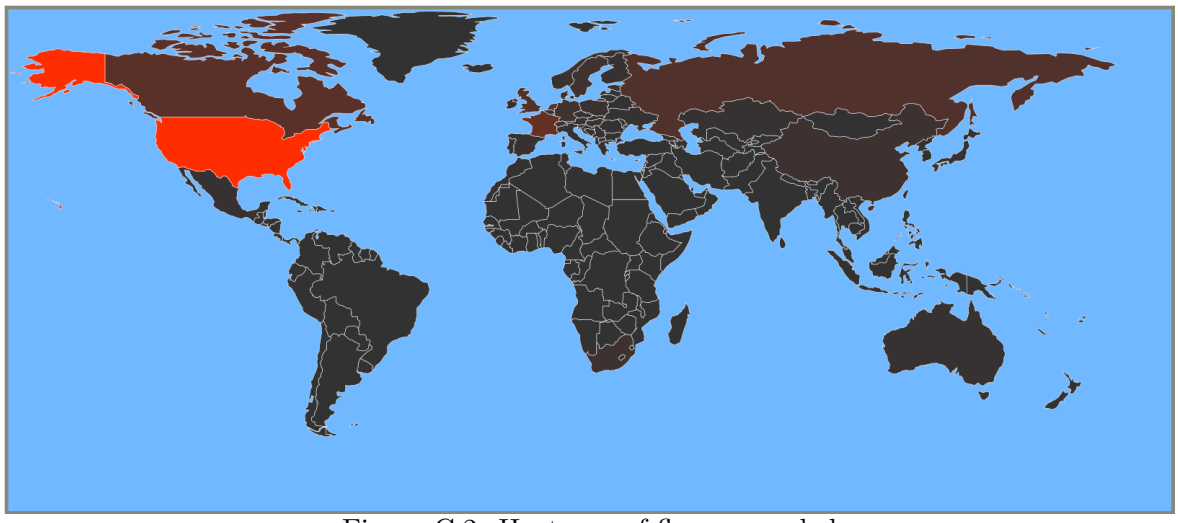


Figure C.2: Heatmap of flows recorded

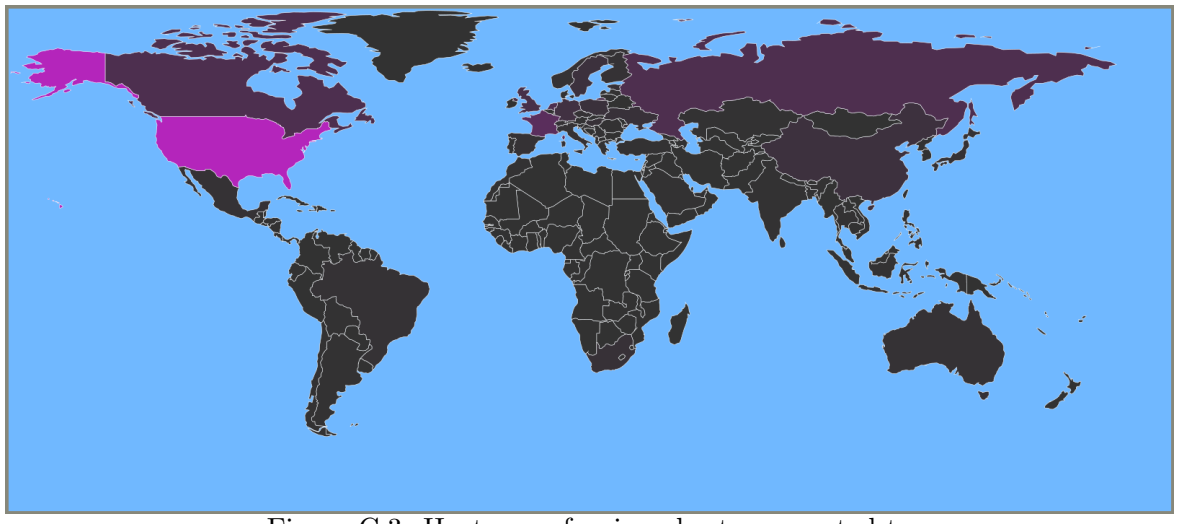


Figure C.3: Heatmap of unique hosts connected to

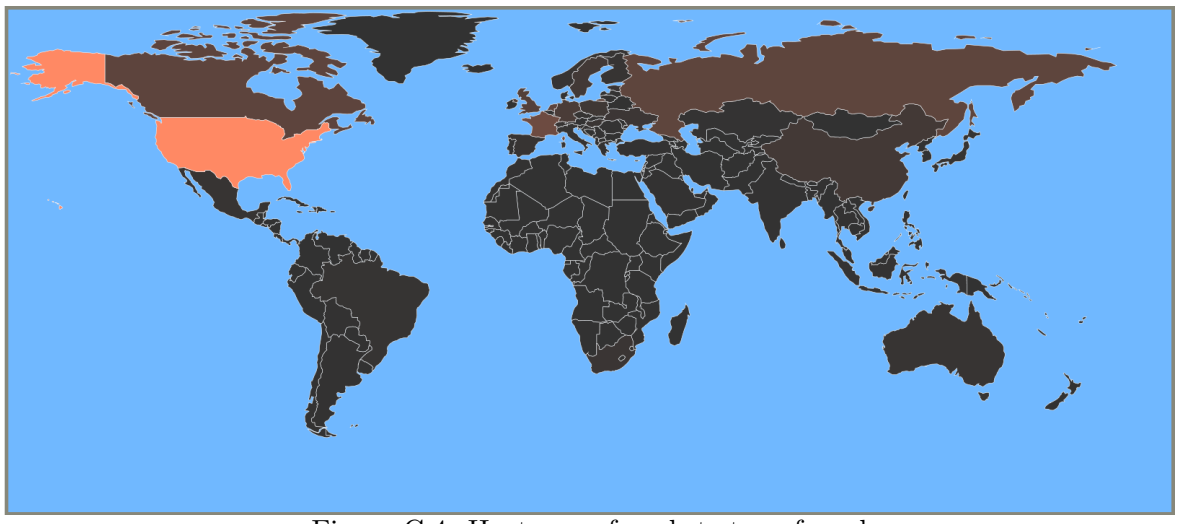


Figure C.4: Heatmap of packets transferred