



University of Zurich  
Department of Informatics

# Extending HAPviewer: Time Window, Flow Classification, and Geolocation

*Raphael Thomas Blatter*  
*Grensiols (VS) & Ulrichen (VS), Switzerland*  
*Student ID: 02-430-015*

Supervisor: Prof. Dr. Burkhard Stiller,  
Prof. Eduard Glatz, Dr. Xenofontas Dimitropoulos  
Date of Submission: August 25, 2011



# Zusammenfassung

## Einleitung

HAPviewer [4] ist ein Open Source Tool, welches unter anderem die visuelle Darstellung von Netzwerk-Verkehr auf Host-level<sup>1</sup> in einem 5-teiligen Graph nach dem Berkeley Socket Modell<sup>2</sup> darstellen kann.

## Ziele

In dieser Arbeit entwickle und implementiere ich eine Reihe von Erweiterungen zu HAPviewer, welche einerseits die Benutzung vereinfachen und den Funktionsumfang erhöhen, andererseits erkunde ich anhand von entwickelten Algorithmen und Statistik-Anzeigen die Möglichkeit, gelernte Profile zur Anomalie-Detektion oder zur Benutzer-Identifizierung anhand von Netzwerkverkehr zu verwenden.

## Resultate

Als erste Erweiterung erlaubt die Möglichkeit zur Auswahl eines Zeitfensters dem Benutzer, nur einen bestimmten Teil eines Datensets darzustellen und in Schritten eine Zeitserie von Graphen zu erstellen.

Ein speziell entwickelter Algorithmus erkennt Netzwerk-Flows mit gleichem Zweck, ohne Heuristiken betreffend Ports zu verwenden, und gruppiert solche zusammen. Die integrierte Möglichkeit, zu IP Adressen die dazugehörige Domain zu finden, verbessert das Resultat dieser Gruppierung<sup>3</sup> und erlaubt es dem Benutzer, sowohl im Graph als auch in generierten Flow-Listen eine lesbare Domain anstatt einer IP Adresse zu sehen.

Je zwei Algorithmen benutzen online Lernen und offline Lernen<sup>4</sup> zur Klassifizierung von Flows in zwei Gruppen, die *Basis Flows* - Flows die häufig und verteilt auftreten - und die

---

<sup>1</sup>Nur Verkehr von und zu einem bestimmten Computer wird berücksichtigt

<sup>2</sup>Je eine Spalte für lokale IP, Protokoll, lokalen Port, remote Port und die remote IP

<sup>3</sup>Da häufig eine Domain über viele IP Adressen erreichbar ist

<sup>4</sup>Alle supervised

*dynamischen Flows* - Flows die selten oder nur innerhalb eines bestimmten Zeitfensters auftreten. Während die Anwendung der online Algorithmen es dem Benutzer vereinfacht mithilfe einer Zeitserie von Graphen die Netzwerkaktivität zu analysieren, so bietet die Anwendung der offline Algorithmen zusätzlich weitere Möglichkeiten. Eine Liste mit *Basis Flows* wird erstellt und scheint in ersten explorativen Experimenten indikativ für den Benutzer des Computers zu sein. Die Veränderung des prozentualen Anteils an *Basis Flows* wird visuell angezeigt und kann zwei getestete Anomalien problemlos erkennen. Zusätzlich wird ersichtlich, dass durch das Weglassen von manuell bestätigten *Basis Flows* der Bedarf an Speicherplatz für Netflow Daten um bis zu 99% reduziert werden kann.

Ein weiterer Zusatz erkennt erfolgreich periodisch auftretende Flows, unabhängig von ihren Zyklen, und zeigt diese als Liste an.

Der Geolocation Zusatz erfasst für jeden Kontaktpartner das Ursprungsland und erstellt Statistiken über deren relative Verteilung über die Zeit. Auch wenn die angezeigte Statistik für einen Benutzer informativ ist, so kann aus den durchgeführten Experimenten noch keine abschliessende Beurteilung betreffend potentiellm Nutzen zur Anomalie-Detektion oder zur Benutzer-Identifizierung gemacht werden. Beide Anomalien in den Experimenten werden jedoch erfolgreich erkannt und Vermutungen über deren Art können durch die zusätzlichen Informationen aufgestellt werden.

## Weitere Arbeiten

Als weitere Arbeiten stehen das Testen der Algorithmen an einem breiteren Set von Datensätzen, das systematische Erkunden der Einflüsse der vom Benutzer gewählten Parameter sowie die Implementierung von weiteren Algorithmen auf der hier erstellten Plattform an. Längerfristig ist die Integration der durch die mit den hier erstellten Algorithmen berechneten und gelernten Charakteristiken in ein produktives System zur Anomalie-Detektion oder zur Identifikation von Benutzern anhand von Netzwerkdaten durchaus denkbar.

# Abstract

HAPviewer is a visualization tool of network traffic on a host level using a graph representation. In this thesis, I implemented a set of extensions to increase the usability and the functionality of HAPviewer, and to explore the feasibility of using the characteristics calculated by a set of algorithms developed for anomaly detection and user identification, as follows.

A time window selector was implemented to allow a user to only display flows that occur within a specified time frame and give the option to easily look at the graphlets as a time series.

Flow matching was implemented to group flows with the same purpose together and identifies the service port without using heuristics. I also included reverse DNS lookup to improve flow matching and to increase the usability of HAPviewer by displaying domain names instead of IP addresses.

I developed two supervised online learning algorithms to increase the usability by classifying flows into those occurring often and those occurring rarely or within a small time window; and two offline learning algorithms to make the same classification and provide additional features. Using a list of *base flows* appears promising as a profile for user identification. Analyzing the evolution of the percentage of *base flows* over time and testing it on generated anomalies shows this characteristic to be a promising avenue for use in anomaly detection.

An algorithm for the identification of periodic flows was implemented to be used for better understanding of which applications run in the background of a computer or to detect anomalies in future work.

Lastly, I developed an extension to show geolocation statistics - the distribution of countries of origin of the contact partners over time. While being informative for a user, the results are inconclusive about their applicability in anomaly detection or user identification.



# Acknowledgments

First and foremost I want to thank Prof. Eduard Glatz for giving me the chance to work on his project 'HAPviewer' and especially for his time and a lot of valuable input on my extension. Our meetings did not just help me direct my efforts into promising directions, I also learned a lot of background information about the topic in general.

I also want to thank Prof. Dr. Burkhard Stiller and Dr. Xenofontas Dimitropoulos for supporting my project while still giving me the creative freedom to follow my ideas.

I also want to thank Dr. Lijin Aryananda for the numerous very helpful discussions about my ideas and algorithms and for taking the time to proofread the complete text. I also want to thank her for her understanding and her support while I was working on the project.

Last but not least I want to thank my friends and my family who helped me see passed the horizon while I was immersed in network flows, learning algorithms, and geolocation libraries.





# Contents

<b>Zusammenfassung</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 HAPviewer . . . . .	1
1.2 Motivation . . . . .	3
1.2.1 Extending HAPviewer . . . . .	3
1.2.2 Anomaly Detection . . . . .	3
1.3 Description of Work . . . . .	4
1.4 Thesis Outline . . . . .	5
<b>2 Related Work</b>	<b>7</b>
2.1 HAPviewer . . . . .	7
2.2 Flow Visualization as Graph . . . . .	8
2.3 End Host Profiling . . . . .	9
2.4 Anomaly Detection . . . . .	10
2.5 Traffic Classification . . . . .	11

<b>3</b>	<b>Design and Implementation</b>	<b>13</b>
3.1	Time Window Selector . . . . .	14
3.1.1	Motivation and Assumptions . . . . .	14
3.1.2	Solution Design . . . . .	15
3.1.3	Implementation . . . . .	15
3.1.4	Usage Instructions . . . . .	16
3.2	Flow Matching and Reverse DNS Lookup . . . . .	16
3.2.1	Motivation and Assumptions . . . . .	16
3.2.2	Solution Design . . . . .	17
3.2.3	Implementation . . . . .	18
3.3	Flow Classification . . . . .	20
3.3.1	Online Classification . . . . .	20
3.3.2	Offline Classification . . . . .	23
3.4	Periodic Flow Identification . . . . .	29
3.4.1	Motivation and Assumptions . . . . .	30
3.4.2	Solution Design . . . . .	30
3.4.3	Implementation . . . . .	31
3.4.4	Usage Instructions . . . . .	32
3.5	Geolocation . . . . .	32
3.5.1	Motivation and Assumptions . . . . .	34
3.5.2	Solution Design . . . . .	34
3.5.3	Implementation . . . . .	34
3.5.4	Usage Instructions . . . . .	35

<i>CONTENTS</i>	ix
<b>4 Evaluation</b>	<b>37</b>
4.1 Data . . . . .	37
4.2 Time Window Selector . . . . .	39
4.3 Flow Matching and Reverse DNS Lookup . . . . .	41
4.3.1 Multiple IPs to Same Domain Name . . . . .	41
4.3.2 Domain Name in Graphlet . . . . .	42
4.4 Flow Classification . . . . .	44
4.4.1 Online . . . . .	44
4.4.2 Offline . . . . .	46
4.5 Periodic Flow Identification . . . . .	58
4.6 Geolocation . . . . .	60
<b>5 Summary</b>	<b>65</b>
<b>6 Discussion</b>	<b>69</b>
6.1 Time Window Selector . . . . .	69
6.2 Flow Matching and Reverse DNS Lookup . . . . .	69
6.3 Flow Classification . . . . .	70
6.4 Periodic Flow Identification . . . . .	71
6.5 Geolocation . . . . .	72
<b>7 Future Work</b>	<b>73</b>
<b>Bibliography</b>	<b>75</b>
<b>Abbreviations</b>	<b>77</b>
<b>Glossary</b>	<b>79</b>
<b>List of Figures</b>	<b>79</b>
<b>List of Tables</b>	<b>82</b>

<b>A</b>	<b>Installation Guidelines</b>	<b>85</b>
A.1	Dependencies . . . . .	85
A.2	HAPviewer . . . . .	86
<b>B</b>	<b>Data Collection</b>	<b>89</b>
<b>C</b>	<b>Class List</b>	<b>91</b>
<b>D</b>	<b>Additional Figures</b>	<b>93</b>
D.1	Base Flows of Time Series After Offline Classification . . . . .	93
D.2	Base Flow Percentage Statistics Without Reverse DNS Lookup . . . . .	95
<b>E</b>	<b>Contents of the CD</b>	<b>97</b>
E.1	Source Code . . . . .	97
E.2	Installation Files . . . . .	97
E.3	Report . . . . .	97
E.4	Data . . . . .	98
E.5	Papers . . . . .	98
E.6	Documentation . . . . .	99

# Chapter 1

## Introduction

In this chapter, I briefly describe the basic functionality of HAPviewer without my extensions and then explain the motivation behind this thesis project. Section 1.3 gives an overview of the extensions and algorithms I implemented, followed by an outline of the further contents of this thesis.

### 1.1 HAPviewer

HAPviewer - Host Application Profile Viewer - is a tool created by Prof. Eduard Glatz, which allows to import files containing network flows and lets the user display these flows in a list or as a 5-partite graph according to the Berkeley socket model. Figure 1.1 shows an example graph generated from network flow data. In addition to the visualization functionality, HAPviewer can summarize flows on a per-service level for better overview [4]. This summarization functionality currently includes the summarization of client roles, multi-client roles, server roles, and some peer-to-peer (P2P) roles [5].

HAPviewer works on a per-host level, meaning that one host is selected and only flows coming from or going to the selected host are displayed. This allows the analysis of security incidents by focusing on the traffic involving a suspicious host or it simply gives the possibility to guess what applications are run on a host by looking at its network activity using the HAPviewer [4]. Figure 1.2 shows the host selection menu after opening a file containing network traffic data and after choosing the local network.

At the time of writing, HAPviewer allows the import of network flow files of type pcap, nfdump, ipfix, and cflow. Cflow is a proprietary binary flow format used by HAPviewer. HAPviewer can also import and display graphs saved in the dot language. Exporting flows of selected hosts as cflow for future use is possible as well.

The code is written in C++ using gtkmm <sup>1</sup> for the graphical user interface (GUI). HAPviewer makes use of many libraries, a detailed list of which is provided in appendix A.

---

<sup>1</sup><http://www.gtkmm.org/en/>

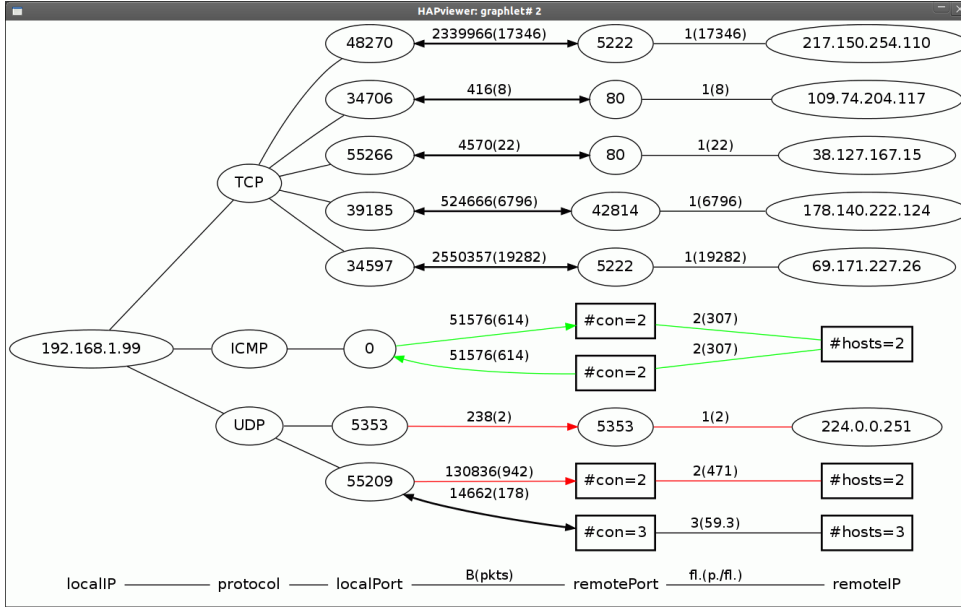


Figure 1.1: Example of a 5-Partite Graphlet Generated by HAPviewer

IP	graphlet	flows	uniflows	protocols	packets	totalBytes
192.168.1.1	0	3010	3010	1	51708	16593172
192.168.1.97	1	2	2	2	6	1104
192.168.1.99	2	11282	3179	3	741793	340875780
192.168.1.136	3	4	4	2	88	16642

Figure 1.2: Host Selection Window of HAPviewer

## 1.2 Motivation

I chose to work on this project because I saw the need for better network analysis tools from personal experience. First, I want to improve HAPviewer by adding additional functionality and classification algorithms. This makes it not only more convenient for an interested user to get to know what is actually going on on his computer, it may also help IT forensics and IT operations. Second, I believe that network based anomaly detection is a field with a lot of potential and implementing a system working on host level is eventually feasible; I want to explore certain network flow characteristics using learning and geolocation in this area.

### 1.2.1 Extending HAPviewer

I strongly believe that the functionality of HAPviewer touches an interesting and promising field - displaying network traffic as a graph according to the Berkeley Socket Model and the flow summarization functionality makes it much easier to quickly see what is going on traffic-wise on a network host of interest. The graphlet view simplifies detecting suspicious flows to be analyzed in depth using a different tool.

Using HAPviewer, I quickly noticed that the graphlet often still contained too many flows for easy analysis. I therefore sought solutions how to further reduce the amount of flows displayed while not losing relevant information. Of course it is important to note, that the term 'relevant' is subjective and depends on the user's goal and approach. Hence I wanted to let the system classify the flows into possibly relevant groups while still giving the user control over the parameters involved in classification and which group and/or which time window to display.

Showing to the user statistics about the flows involving his computer and using classification of flows to make it easier to find the interesting flows from millions may not only give the interested user a better understanding of what is actually going on and running on his computer, it might also help during forensic analysis to more quickly find suspicious flows before one knows exactly what to look for.

### 1.2.2 Anomaly Detection

Anomaly detection is a method often used in network management and security, especially in the field of intrusion detection [16]. Anomalies in network traffic may hint towards a security or network operation problem, but they may also arise from a completely benign behavior, such as a flash crowd or reaction to a marketing campaign. Anomaly detection can of course be applied to other input features than network traffic. In this work, I will however use the term in the context of network flows.

Most commonly used signature-based intrusion detection systems (IDS) are very successful and efficient in detecting previously encountered and known security issues, such as computers running known malware, port scans, or DDoS attacks. They are often also

able to detect known suspicious behavior which tends to be used by malware. However, signature-based IDS are mostly unsuccessful at detecting new, never before encountered threats.

As an addition to the high effectiveness - with low false positives and negatives - and efficiency of signature-based IDS, anomaly detection may add the functionality to possibly detect anomalous behavior or even previously unknown malicious traffic.

An end user's computer has mostly very irregular network behavior. Whether the user is currently using the computer at all, and if so, what the user is currently doing - e.g. writing an email, playing a computer game, or watching movies online - has a big influence on network flow characteristics. To get past this problem, many current anomaly detection systems run on a large amount of aggregated traffic flows from many computers, running on traffic traces at the border between networks or even on bigger backbones. This may show whether there is a big problem occurring inside the system, but it is very difficult to find its origin.

As also stated in e.g. [19], I believe that building a profile of a single host is a promising avenue for anomaly detection. In my opinion, anomaly detection at host level, using nothing else than network traces - which can be collected centrally from e.g. routers - is feasible and would not only enable users and/or network administrators to find out that a problem is happening, but also where exactly it is happening, i.e. exactly which computer. Moreover, as there are less flows having to be considered it is easier to detect which exact flows or services are responsible for the anomaly on a computer.

### 1.3 Description of Work

As pre-thesis HAPviewer <sup>2</sup> can only display all the flows in a file involving a chosen host, the graphs easily get very big with hundreds of thousands of edges for long traffic recordings. To only select a certain selection of flows, be it within a selective time window or following chosen selection criteria, the file containing the flows has to be filtered first using a different tool - e.g. *ethereal* <sup>3</sup>. My first extension enables the user to select a time window using a graphical user interface (GUI). Only flows which fall completely or partly within the selected time window are displayed. Using a click on a button, this time window can easily be moved forwards by a selected step size. This feature allows the user to easily analyze the development of the network traffic of the host of interest over time as a series of graphlets.

An important functionality for my classification algorithms implemented in further steps is to identify matching flows - i.e. flows that probably have the same purpose and use the same service. Simply matching remote IP, protocol, and local and remote port is not enough, as the non-service port is not known from the flow alone. I develop an algorithm to identify the probable service port without using heuristics.

---

<sup>2</sup>The version of HAPviewer without my extensions, as available before my project

<sup>3</sup><http://www.ethereal.com/>



As part of the flow matching feature, the user may choose to use reverse DNS lookup <sup>4</sup>, to reduce the problem that a service may be available over many different IP addresses <sup>5</sup>.

I design and implement a few algorithms, online as well as offline <sup>6</sup>, to classify network flows into two groups: *base flows* and *dynamic flows*. While the former category consists of flows happening often and distributed over time, thus representing a sort of background network traffic of the selected computer and/or user, the latter category consists of flows that happen once or rarely or only in within a relatively small time window. I propose that this distinction not only allows to greatly reduce the flows to display in the graph, characteristic values calculated from this classification may be used for anomaly detection and in future work the set of *base flows* may be used to identify users working on different computers or using different addresses, as a big part of these flows may be characteristic for a user.

A GUI to show the classified flows and different statistics about the classification, as well as export functionality for further calculation is implemented as well.

A feature to identify periodic flows <sup>7</sup> is also implemented. The user is given the possibility to set parameters for the algorithm used and thus may optimize the algorithm for his specific needs. This feature also includes the possibility to only display periodic flows in the graph, to display the flows classified as periodic in a list, and the possibility to export the list to a file for further use.

The option to map the remote IPs to their country of origin and display statistics about the evolution of their relative distribution between countries on a time line is also implemented. In addition to looking at these statistics on graphs, the user can also export them to a file. I propose that the distribution of countries a host communicates with over time may be used for anomaly detection and/or user identification in future work.

In addition to providing the features and algorithms mentioned above, I explore the possibility of using the statistics generated from the classification of flows using learning algorithms and geolocation for anomaly detection and user identification in future work.

## 1.4 Thesis Outline

Chapter 2 gives an overview of related work on HAPviewer, flow visualization using a graph, end host classification, anomaly detection, and network traffic classification.

In chapter 3, I explain the extensions I added to HAPviewer. For easier reading, I organize chapter 3 into sections about individual features implemented. Within each section I briefly describe the motivation for this feature and the assumptions it is based on, the design rationale, the implementation, and how to use it.

---

<sup>4</sup>Determine the domain name corresponding to an IP address using the domain name service

<sup>5</sup>e.g. the Google search engine or amazon cloud

<sup>6</sup>Not in terms of network connectivity, but in terms of learning

<sup>7</sup>Flows that occur with a certain periodicity

A selection of experiments using real life network traffic data is presented in chapter 4 to test the usefulness of the features added and algorithms implemented and to explore the possible application of these features for host-level anomaly detection and user identification in future work.

After summarizing the work done during this project in chapter 5 and discussing the usefulness and possible applications in chapter 6, I give ideas about where to go from here in chapter 7. This includes possible further explorations and improvements of the algorithms developed for this project as well as possible further extensions to HAPviewer.

# Chapter 2

## Related Work

The extensions to HAPviewer implemented for this thesis touch many subfields of the large field of Internet security and network traffic analysis. Even though the field is still relatively young and increasingly active, a large number of papers and books have been written on related topics.

In this chapter, I go into a selection of related work to demonstrate current approaches, point towards avenues of further reading on the topics, and I show that, according to my knowledge, my project explores regions of this vast field that have not yet been looked at.

I start this chapter with a section about pre-thesis HAPviewer. Section 2.2 shows a selection of other work that has been done on flow visualization using graphs, while section 2.3 deals with the topic of end host profiling. Section 2.4 shows example technologies used for anomaly detection and where the approach followed in this thesis is located. Section 2.5 on traffic classification gives some examples from this field and explains an approach by Karagiannis et al. [9] which seems closest to my approach.

### 2.1 HAPviewer

HAPviewer is an approach to visualize network traffic on a host level. Flows can be either shown as a flow list, or, to reduce the cognitive burden, as an annotated 5-partite graph - each column being a key attribute of the Berkeley socket model <sup>1</sup> [5].

In addition, HAPviewer provides a host role summarization feature, currently summarizing the following roles:

- client roles
- multi-client roles
- server roles

---

<sup>1</sup>Local IP, protocol, local port, remote port, and remote IP

- P2P roles

The four processes of insights for effective visualization identified in [20] are seen for HAPviewer in [5] as:

1. providing an overview of all hosts through aggregation of flows on per-host level
2. the ability to adjust the level of abstraction through summarization and through the flow list view
3. the graphlet view facilitates pattern detection as part of natural human perception
4. the graphlet display matches the mental model of the Berkeley socket model

HAPviewer also gives visual cues whether a flow is bidirectional or just a uniflow, and all the edges are annotated with information like number of bytes, number of packets, number of flows, and number of packets per flow.

Further information about pre-thesis HAPviewer can be found in [4] and [5].

## 2.2 Flow Visualization as Graph

As this thesis only uses the visualization feature of HAPviewer using a graphlet [4][5] and only small changes have been made to this feature by the extensions, this section is kept brief.

BLINC as presented in [8] uses the representation of a flow in a 4-partite graph with the columns *source IP*, *destination IP*, *source port*, and *destination port*. The additional column used in HAPviewer - *protocol* - is used as a heuristic in BLINC to refine classification further, but not for the initial algorithm. More details about the BLINC algorithm are explained in section 2.4.

The approach used by Karagiannis et al. in [9] uses a 6-partite graphlet to represent traffic information in a compact form. This is achieved by using the destination IP field twice, resulting in a graphlet with the columns *source IP*, *protocol*, *destination IP*, *source port*, *destination port*, and *destination IP*. Karagiannis et al. use this graphlet representation for the application of an online learning algorithm explained in section 2.4.

Other approaches like [6] use visualization of network traffic as graphs to apply algorithms to detect graph patterns in a network log. The summarization feature of HAPviewer [5] uses the recognition of a small, predefined set of patterns. The approach presented in [6] allows the user to specify patterns to be searched for and highlighted.

## 2.3 End Host Profiling

Many different approaches have been designed to profile end hosts. As HAPviewer tries to create a host application profile and one potential future application of my extensions is to identify users and/or computers in networks by creating a distinct fingerprint of *base flows* and/or geolocation distribution, I mention some approaches from this field in this section.

Tan et al. apply in [18] two algorithms to cluster hosts into groups according to their connection patterns. Their second algorithm also takes into account the change in connection patterns over time. The groups are not predefined and thus their function has to be identified manually. Tan et al. declare in [18] that a 'meaningful' classification was achieved. As opposed to [18], my extensions explore the possibility of getting a unique, in the future identifiable, fingerprint for a host. However, given a distance metric, this fingerprint might be used to improve a clustering algorithm as well.

Chang et al. show a similar approach to [18] in [2]. They apply node behavior profiles on traffic level combined with statistical tests to build models of behavior profiles. These models are used to detect malware by detecting a deviation from this predicted behavior. This approach [2] is similar to this thesis in that after profiling, it uses deviations from the detected profile to detect anomalies.

McHugh, McLeod, and Nagaonkar use profiling from network traffic data from several different kinds of computers in [13]. Profiles are created from the distribution of flows between protocols<sup>2</sup> and the distribution of destination ports<sup>3</sup>. Learning these distributions for a computer as a profile and evolving them over time, big deviations are then detected with the goal of finding anomalies. [13] chooses an interesting profiling approach which is also based on netflow data alone<sup>4</sup>. These elements may be combined with the profiling using *base flows* and geolocation distribution of remote IPs proposed in this thesis, as both have similar goals, using the same data, but different profiles.

Wei, Mirkovic, and Kissel in [19] generate profiles of hosts consisting of the elements presented in table 2.1 from network traces collected over 1 to 2 days.

Using this information, Wei, Mirkovic, and Kissel cluster the hosts into groups and also try to detect anomalous behavior with the help of deviations from known patterns. Their host profile proposed in [19] does not evolve over time; it is created in a first step to build the clusters from it in a second step. In addition, their profile is very elaborate and contains a lot of dimensions. This thesis tries to find profiles that use a lower dimensionality, that evolve over time, and that can uniquely identify a host, the second step in [19] - the clustering - is not implemented in this sense in this thesis, as flows are clustered and not hosts.

---

<sup>2</sup>ICMP, TCP, and UDP

<sup>3</sup>Divided into three classes: 0 - 1024, 1025 - 5000, and > 5000

<sup>4</sup>Thus no payload is available.

Host Feature	Explanation
<i>ip_address</i>	IPv4 address of the host
<i>daily_destination_number</i>	number of distinct remote IP addresses per day
<i>daily_byte_number</i>	total traffic volume per day
<i>average_TTL</i>	reflects the relative location of the host
<i>tcp_services</i>	open TCP ports
<i>udp_services</i>	open UDP ports
<i>communication</i>	detailed description of typical communication
<i>communication_similarity</i>	diversity of all recorded communications

Table 2.1: Host Features Used for Profile Generation In [19]

## 2.4 Anomaly Detection

The field of network-based anomaly detection in IT network security is large, in industry and especially in academics. Several books have been written about the topic in general or specific approaches. For further reading, I name a few examples.

In [7], Hossain uses backpropagation neural networks for anomaly detection on network data. His neural network learns the normal behavior of a user, while recognizing a deviation as an anomaly. He also briefly touches the field of user identification.

A more general overview of the current situation concerning intrusion detection systems is described in [16]. The topic of IDS is analyzed and approaches explained by several specialists in the field.

Fan describes a system to detect anomalies in edge routers of ISPs in [3]. He uses the number of 'unmatched inbound flows' - i.e. the number of inbound flows that have no corresponding outbound traffic within a certain time window - to find anomalies in a system - in this case an ISP.

Many anomaly detection algorithms work on aggregated data from several hosts up to huge numbers of flows in backbones. While there are other challenges present, e.g. the question of efficiency is much more relevant with higher throughput, the task is quite different. The goal with such approaches is often to detect only the presence of an anomaly, host-level anomaly detection however promises to make the identification easier.

Oftentimes when using aggregated flows from many hosts, entropy measures are used alone or in combination with other features. Nychis et al. give a good overview of many approaches using entropy in [15] and evaluate their abilities.

A number of approaches [2], [13], and [19] mentioned in section 2.3 also use their host profiling and classification algorithms to detect anomalies by comparing their profile to the current behavior and measuring the deviation. This thesis follows a similar approach, as it explores the usability of learned or calculated profile features for anomaly detection by analyzing their constancy during normal use and thus deviations during anomalies. The main difference is that this thesis uses different features.

Machine learning is a very vague term [1] and several of the approaches described above use machine learning in one way or another. However, a specialized analysis of these technique being applied to anomaly detection and to intrusion detection in general is given in [12].

## 2.5 Traffic Classification

Classification of network traffic can be understood in two slightly distinct ways. The first one has the goal to identify certain services and thus it is possible to infer applications running on a computer, the second one makes a more coarse-grained classification between less traffic classes.

In general, there are multiple principal types for traffic classification. The first type uses deep packet inspections to analyze the payload of packets and match it to known patterns. The second type takes the transport level characteristics into account, but does not look at the payload. Classification is done according to e.g. port numbers, packet sizes, inter-arrival time of packets, and so on. The third type of classification is based on social interactions of hosts. Interaction patterns with other hosts are analyzed and patterns matched - one way of doing this is matching known subgraph patterns in the graph representation of the traffic flows e.g. using the algorithms developed in [6].

Kim et al. group classification approaches into those based on transport layer ports, host behavior, and flow features [10]. They evaluate these different principal types with several different dataset and come to the conclusion, that for traffic which does not try to avoid being classified, port-based approaches are still very successful. In addition, Kim et al. state that combining different approaches offers synergies and results may be improved.

BLINC - blind classification -, as explained in [8], aims at the classification of flows according to applications. While the paper describes all principal types of traffic classifications mentioned above, BLINC itself uses classification on a social level combined with some elements of transport level characteristics.

Many ways of using techniques from machine learning [1] have been proposed to do network flow classification. One classic example is shown in [21], where only basic attributes like inter-arrival time of packets, mean packet length and its variation, flow duration, etc. are used for classification. Nguyen and Armitage give a comprehensive overview of different approaches in [14]. Nguyen and Armitage compare many different algorithms - BLINC [8] included - which use techniques from the vague field of machine learning [1] and explain many of the methodologies involved.

Karagiannis et al. show in [9] a supervised learning algorithm for the online classification of flows into two coarse-grained groups to form what they call a 'profile graphlet'. This 'profile graphlet' evolves over time to adapt to changes in a user's behavior. The 'profile graphlet' represents a highly compressed version of the total activity graphlet.

Even though the target application for this thesis adds to the one presented in [9]<sup>5</sup>, the principal methods are very similar. The flows that are learned by the algorithm developed by Karagiannis et al. and make up the 'profile graphlet' are analogue to the *base flows* from my algorithms. Just as in [9], I also apply supervised learning algorithms.

However, the online learning algorithms implemented in this thesis are more complex than those in [9] and this thesis includes the supervised offline classification algorithms as well. In addition, while Karagiannis et al. hardcode the parameters into their tool, my extensions to HAPviewer give the user the possibility to change these parameters using a graphical user interface. A more detailed description of the online algorithm used in [9] and the online algorithms implemented for this thesis can be found in part 3.3.1.

---

<sup>5</sup>Karagiannis et al. want to build a compact 'profile graphlet' while this thesis, in addition to exploring the possibilities of using the *base flows* as a digital fingerprint, explores many additional ideas and features.



# Chapter 3

## Design and Implementation

I added features and functionalities to HAPviewer and used HAPviewer's basic functionalities, such as importing many different types of network traffic files, transforming them to the proprietary cflow format, and displaying flows in a 5-partite graph. However, for the implementation of my project I tried to be as minimally invasive as possible, i.e. change the existing code as little as possible. The reasons for this are first that both projects may be continued independently and a programmer may choose to just use the traditional, slimmer version of HAPviewer as a base and still be able to add functionality provided by my extensions later. Second and more importantly, the features and algorithms developed for this project can easily be detached and used with a different base software or even a dedicated, slim file import unit <sup>1</sup>.

An instance of the class `RTBConnector` is the only link between pre-thesis HAPviewer and the current version of my extensions. Hence the parts of pre-thesis HAPviewer only use methods offered by `RTBConnector`, which in turn calls the corresponding functionality offered by other classes.

Even though using my extensions, the user may change the output generated by pre-thesis HAPviewer, i.e. the option to select the time window of flows to display in the graphlet or have the graphlet display domain names instead of IP addresses, those features can always be deactivated in the GUI to get the traditional functionality if so desired.

I also kept the user interface (UI) classes separate from the algorithm classes, which makes it easy to replace the graphical user interface I provide by either a different UI or use the algorithms for automatic processing without UI <sup>2</sup>.

While the algorithms could surely still be made more efficient, I was careful to make the implementation fast on calculation time as well as slim on storage consumption, whenever possible <sup>3</sup>. This is important as a file may very well contain hundreds of thousands

---

<sup>1</sup>Displaying the flows as a graph and the summarization functions provided by pre-thesis HAPviewer are very useful, but are not needed for the functioning of the features I implemented.

<sup>2</sup>In which case the user interface may simply be replaced by a different class providing the same methods, only returning fixed parameter values

<sup>3</sup>As optimization for calculation time is opposed to optimization for storage consumption, in each case the more relevant factor of the two was given more weight.

of flows. I made heavy use of C++ `maps` to allow logarithmic search times instead of linear ones for an element to be located in a data structure <sup>4</sup>. If I reckon it relevant, I describe optimization techniques used in the implementation part of a section, as often it makes sense for a programmer of future improvements as well as for a user to know the approximate complexity and memory consumption of a calculation to be invoked. So far the optimization seems to be sufficient, as the generation of the graphlet is still the bottleneck and not the algorithms implemented for this thesis <sup>5</sup>, at least up to certain flow counts.

Every file, each of the almost 40 classes I created, and every method has been commented in a syntax recognized by doxygen <sup>6</sup>. Therefore it is possible to generate a documentation of the whole tool <sup>7</sup>. A table with all the classes implemented for this thesis with a short description can be found in appendix C.

The following sections describe the features I implemented, why I did so, and how to use them.

## 3.1 Time Window Selector

The time window selector allows the user to only display a part of a file which has been imported into HAPviewer.

### 3.1.1 Motivation and Assumptions

Pre-thesis HAPviewer could only display all the flows in a network traffic file which was imported. To only show a specific time window, the file had to be adjusted accordingly before importing it using a different tool like e.g. `ethereal` <sup>8</sup>. Timing information with varying precision <sup>9</sup> is however available in the `cflow` data structure.

I assume that being able to see network traffic at varying time resolution, drill down to specific time windows, and look at time series of flow graphs is a useful addition to HAPviewer. I experienced it to be easier to import a big file and chose the time window of interest within HAPviewer than having to adapt the file beforehand, as often the exact time window of interest is not known in advance.

---

<sup>4</sup>Of course, if the index of the element looked for is known, an array or `vector` is more efficient and in those cases, these linear data structures are used.

<sup>5</sup>An obvious exception to this is reverse DNS lookup, as it has to look up every unique IP address in the file using the domain name service over the Internet.

<sup>6</sup><http://www.stack.nl/~dimitri/doxygen/>

<sup>7</sup>Simply run `make doc` in the build folder to generate the doxygen documentation, for more details see appendix A.

<sup>8</sup><http://www.ethereal.com/>

<sup>9</sup>Netflow data is known to be less precise when it comes to timing.

### 3.1.2 Solution Design

I decided that the most intuitive and least error-prone way for the user to select a time window is to let him/her provide the two values of the start time and the duration of the time window. For easier time series navigation, I include an additional time entry for the user, which is to be used for the step size, i.e. the amount of time the start time is moved forward from the currently selected position on pressing a button. I did not use the selected duration as the step size as well because a separate input allows overlapping time windows over a time series, which is often nice to have and also smooths changes in the time series statistics.

For matching flows with the selected time window, I chose to consider any flow which is completely or partly within the flow, i.e. it will be displayed even if it started before the time window or ends after it, as long as part of it is within the time window. Using simply start times of flows alone or requiring a flow to be completely within the time window would be possible alternatives. I chose against those possibilities because my solution shows a flow in every time window in which it is actually happening and data is being transferred. No network traffic which is happening is missed when looking at all flows within a selected time window. A user may still apply filters explained below.

### 3.1.3 Implementation

For choosing the start time, only a slider is available, which can be moved between the start time of the first flow and the end time of the last flow. The date and time which corresponds to the currently selected value is always displayed in a label above the slider in an easily readable fashion <sup>10</sup>. As for the other time fields, I chose the resolution to be one second, even though timings are available in milliseconds. The reason for this is that I consider it unlikely that any user needs higher resolution and often the time stamps of the flows in the network traffic files are not exact at millisecond level anyway.

Entering the time window size and the step size can be done by either using a slider or number entry fields for the days, hours, minutes, and seconds respectively. I implemented this input type to always be synchronized between the text fields and the slider <sup>11</sup>. In addition, limit checking is done, making sure that both choices are not less than one second and not bigger than the duration of the imported file. If a value outside the limit is entered, the corresponding limit is automatically set.

A button to refresh the graphlet, considering the currently set time values, and a button to go one step further on the time line are part of the GUI. Figure 3.1 shows the input GUI for selecting the time window to display.

---

<sup>10</sup>While internally calculations are done in seconds or even milliseconds after January 1st, 1970, an actual date and an actual time is displayed for the user.

<sup>11</sup>The slider gets adjusted to the times entered in the text fields when the field either loses focus or enter is pressed.

The image shows a web interface for configuring a time window. It consists of three main sections:
 

- start time:** A text input field containing "06.08.2011 - 21:49:26" with a horizontal slider bar below it.
- duration:** Four input fields for "0 days", "0 hours", "5 minutes", and "0 seconds", with a horizontal slider bar below them.
- step:** Four input fields for "0 days", "0 hours", "5 minutes", and "0 seconds", with a horizontal slider bar below them.

 Below the duration section is a "refresh" button, and below the step section is a "next step" button.

Figure 3.1: Choosing the Time Window to Display

### 3.1.4 Usage Instructions

A time window can be chosen even before a graphlet is displayed. When displaying a graphlet using HAPviewer, the selected time window is automatically taken into account. When a graphlet is already being displayed, clicking the button *refresh* will recalculate the graphlet. Clicking the button *next* will do the same after moving the start time forward by the chosen step size. Be aware that refreshing the graphlet will also recalculate the other algorithms if any parameters have changed and the statistics will be adapted accordingly.

## 3.2 Flow Matching and Reverse DNS Lookup

Flow matching is an important prerequisite for applying the algorithms and calculating the statistics. Flow matching means to identify which flows probably use the same service on the same remote system <sup>12</sup>.

### 3.2.1 Motivation and Assumptions

A single flow provides the information according to the Berkeley socket model, i.e. local IP, protocol (UDP, TCP, or ICMP), local port, remote port, and remote IP, in addition to the timing information, i.e. start of the flow and duration of the flow, and values for the number of packets that are part of the flow and the total number of bytes transferred.

An important part of classifying flows into the groups *base flows* and *dynamic flows* as well as the identification of periodic flows is the ability to know which flows match, meaning which of the flows probably use the same service on the same remote system. I use the

<sup>12</sup>I use the word 'system' because a service is often provided by a server farm and due to load balancing a different physical server may offer the same service at a different moment in time.

word 'probably' because even if there is a high likelihood that the flows that have been matched together actually have the same purpose, there is no guarantee for it <sup>13</sup>.

This matching algorithm assumes that IP to domain name mappings or IPs of a system offering a service <sup>14</sup> do not change too often. I do not use any heuristics about port numbers and thus do not make any assumptions about their importance. The algorithm assumes that each port on each system only offers a single service, which is almost always the case <sup>15</sup>.

### 3.2.2 Solution Design

Even though matching flows sounds like a simple task, it includes mainly two problems. First, a flow uses a remote port and a local port, one of which is usually random, the other - the service port - has to be well defined. Looking at a single flow, it is not possible to know which one of them is the service port. Second, many services are offered by many different servers within the same system and can be accessed using different IP addresses. For example, for reasons of load balancing at different times, a service is available over different IP addresses <sup>16</sup>.

For reducing the first problem, it would be possible to use heuristics, e.g. assuming that every port under 1024 must be the service port <sup>17</sup>. However, many services used today have no assigned port number and use higher numbers. Port numbers below 1024 can also be misused, even if they should not. A possible extension is to use a list of known service ports, which however can result in false positives as even known service ports above 1024 are often randomly used as non-service ports by a host.

I have chosen to avoid using assumptions and develop an algorithm which has then been demonstrated to be very successful, even though it is fairly simple. All flows are first 'put onto piles' according to their remote IP or domain name. Then in every pile, the occurrences of a port number as local port or remote port are simply counted. The highest occurrence is likely to be a service port and all flows having this port in the right position - local or remote - are taken from the 'pile' and the procedure is repeated until no flows remain. If a remote IP has only a single flow or a flow in a pile has no port which matches the others in the pile, the service port can not be determined by the algorithm. However, this is irrelevant, as knowing the service port is only used for flow matching and a single or a unique flow can not be matched anyway.

---

<sup>13</sup>In theory it is possible, however unlikely, that either a host uses two different services with two different ports on the same server and it chooses both times by chance the same random port, or a system might simply at a different time offer a different service on the same port.

<sup>14</sup>The latter is only relevant when reverse DNS lookup is either deactivated or fails.

<sup>15</sup>Rare exceptions include the possibility to use a manipulated DNS server to offer DNS resolution as well as forwarding webpage content if asked for, in order to circumvent access control, both of which is done over port 53.

<sup>16</sup>E.g. a request to the DNS for google.com returns six distinct IP addresses, which also change over time.

<sup>17</sup>All ports below 1024 are assigned by IANA and should not be used as non-service ports.

The second problem is about a domain name possibly being resolved to many different IP addresses, which makes matching by remote IP impossible. I use reverse DNS lookup provided by the domain name system to reduce this problem. Using reverse DNS lookup, the algorithm first tries to match every IP address to its domain name before putting the flows on 'piles' according to domain names. Not all IP addresses can be resolved to a domain name - those which can not are simply sorted according to IP as before.

The proposed system has the problem that IP to DNS mappings may change over time. Although rather rare, it is possible that an IP address which has been resolved some time, e.g. weeks, ago to one domain name is now resolved to a different one. Even though this problem is very slight and causes hardly ever a false matching, it can be reduced by running a reverse DNS lookup for all the flows close to or at the time they have been recorded and save them to a file, which can then be imported again when running the algorithms on the data later. This functionality of exporting and importing learned IP to domain name mappings has also been developed and is included in my extensions to HAPviewer.

### 3.2.3 Implementation

I use special data classes which include the functionality to do the flow mapping. This functionality is included in the classes `RTBDataFlows` and `RTBDataSameFlows`. A reader interested in the low-level detailed is referred to look at the documentation of the methods provided by those two classes.

The current version does not use the protocol for flow mapping, i.e. UDP and TCP flows going to the same service port and go to the same remote system are counted as a match. First, this makes sense, as even if occasionally UDP and TCP connections go over the same service port to the same system, they usually belong together and can safely be classified as matching. Second it is slightly more efficient this way.

If so chosen, the system performs a remote DNS lookup for every remote IP<sup>18</sup> in the flow list. If the reverse DNS lookup is not successful, the IP address as human readable `string`<sup>19</sup> preceded by `*IP*/`<sup>20</sup> is used.

All learned mappings of IP addresses to domain names are saved in a data structure of type `map` with the integer representation of the IP as key. This way it can be determined with  $O(\log(n))$  complexity whether an IP address has already been resolved. If so, the time consuming reverse DNS lookup does not have to be repeated again.

The C++ `map` contains the IP to domain name service, thus every time in the future we need the domain name of a flow, a search of  $O(\log(n))$  would be necessary. As during all the algorithms the domain name is needed for mapping, access is fairly frequent. Introducing a second linear data structure which can be accessed with  $O(1)$ , as the index of the flow in question is known, solves this problem. However, saving a `string` containing

---

<sup>18</sup>The local IP is always the host of interest, thus a resolution, even if available, is not necessary.

<sup>19</sup>Internally, IPs are handled by their numeric value and are of type `uint_32`.

<sup>20</sup>This allows easy and highly efficient identification of an IP vs. a domain name if necessary.

the domain name for every single flow is not only highly redundant - as many flows have the same remote IP - but can also use considerable amounts of memory considering that the number of flows can easily go into the hundreds of thousands or even millions. I avoid this problem by using as the linear structure a second layer of abstraction. Instead of directly saving the `string` for every flow, I simply save a pointer to the corresponding `string`, which is still located in the `map` and is only saved once for every unique IP <sup>21</sup>. At the cost of a single additional redirection per lookup, a lot of memory can be saved while still having access with  $O(1)$ .

One problem that arises is that the complete domain name also includes all subdomains. Currently only the outermost subdomain is removed. Simply trying to identify the domain without any subdomains is also not perfect as it might result in false positive mappings, as different services might be offered on different subdomains <sup>22</sup>. Removing a single level of subdomains has proven successful in most cases, as it removes the location coding <sup>23</sup> used by some systems.

For analyzing flows at a later time and making sure to have the same IP to domain name mapping, or simply to speed up the remote DNS lookup step, it is possible to save mappings from IP to domain name as a csv file. If a collision occurs - i.e. an IP address is imported that is already in the `map` - the new value is simply ignored and the old one kept in the mapping. For exporting, the static member functions of the class `RTBExporter` are used to choose a file in a file selection dialog and check whether the user wants to overwrite the file, if it already exists.

The class `RTBReverseDNSResolver` is responsible for the functionality of doing the reverse DNS lookup and it keeps the mappings as data members. `RTBReverseDNSResolver` has public member functions to build the IP to domain name mapping and access it again. `RTBGuiReverseDNS` provides the GUI.

After having done the reverse DNS lookup, if chosen by the user, the algorithm for matching flows continues by putting all the flows on 'piles' according to their remote IP or domain name. This is again done by using a `map` with the remote IP as the key to have  $O(\log(n))$  complexity on finding the 'pile' a flow belongs to or whether a new one should be created. After this step the source port is identified using the algorithm described above. For technical implementation details, please consult `RTBDataFlows` and `RTBDataSameFlows`.

After the matching procedure, the groups of matching flows are returned in a `vector` of instances of the class `RTBDataSameFlows`. This `vector` may then be used by the algorithms to do classifications and to create statistics.

---

<sup>21</sup>Currently if more than one IP is mapped to a single domain name, the domain name is still multiple times in memory. This could be avoided by using a layer of abstraction in the IP to domain name mapping, having a data structure of type `map<uint_32, *string>` instead of `map<uint_32, string>`. However, this is hardly worth the effort, as the task of searching for already known `strings` to every new IP to domain name needs a lot of processing.

<sup>22</sup>Even if it is highly unlikely, given that the port also has to match.

<sup>23</sup>Using the outermost subdomain to stand for the location of the server

## Usage Instructions

Using a checkbox on the GUI, the user can choose whether reverse DNS lookup should be used or not. As this feature needs a lot of time to create the mapping - if the mapping is not being loaded from a file - it is deactivated by default. When activating it, it is indicated to the user that building the mapping might take a long time and he is asked to confirm his choice.

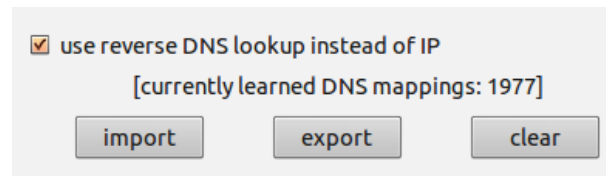


Figure 3.2: Options Concerning Reverse DNS Lookup

The GUI also includes buttons to initialize the import and export of mappings, as well as to clear the mapping currently in memory. A label shows to the user how many IP to domain name mappings are currently in the data structure. Figure 3.2 shows the GUI with options concerning reverse DNS lookup.

## 3.3 Flow Classification

Having a time window selector, it is already possible to easily cut down the number of flows being displayed and the size of the graph, by simply drilling down to the point in time of interest. However, obviously the user already has to already know where to search, even though the feature to go through a time series step by step reduces the number of flows displayed at any one time drastically while still showing all the flows in total.

The choices so far are very limited and the system does not point out flows of interest. This is the goal of the classification system, giving the user the possibility to choose parameters and select which group of flows might be interesting for his specific purpose. The classification done in the implemented learning algorithms is between *base flows* - flows that happen frequently and distributed over time - and *dynamic flows* - flows that happen rarely or even only once. Giving the user the possibility to select parameters for the learning algorithms allows him to adapt the algorithm to his personal needs and make it less selective or more selective for one class of flows or the other.

### 3.3.1 Online Classification

Online classification works on data while it is coming in continuously [1]. The algorithm does not have knowledge about all the flows in a set, only about the flows it has seen in the past <sup>24</sup>.

<sup>24</sup>This does not mean that all the encountered flows are kept in a database. Usually only a very small amount of information about the past is kept.



Online classification has the advantage that it can easily be used on real-time data <sup>25</sup>. Their need for processing power at any time on the time line is relatively low.

### Motivation and Assumptions

My next approach to further reduce the number of flows to be displayed is to use online filtering. Based on an existing approach by Karagiannis et al. [9], the online filters in short help the user to quickly focus on flows of interest while going through a time series. Flows are learned after being shown according to a certain algorithm and are kept in memory. They will not be shown anymore as long as they are still being 'remembered' by the algorithm. After a while however, and according to varying possible rules, learned flows are forgotten again and will be displayed when they happen again.

To the user, this gives the advantage of showing either only the flows he has not seen recently in the time series - the *dynamic flows* - or if he prefers, only the flows that persist or happen often over a certain time and do not just happen very rarely - the *base flows*. Either choice reduces the amount of flows displayed a lot and he may save time mentally sorting out the flows he does not want to see.

These algorithms assumes that flows can easily be matched without keeping a big database all the time and without knowing a large number of flows. For the user it only works, if he looks at flows in a time series, going through them one time step after another successively, while the algorithm can learn from the flows it sees.

### Solution Design

My ideas behind the implemented approach are an extension to a solution implemented and published by Karagiannis et al. in [9]. Karagiannis et al. used a fixed step size and time window and a rather simple forgetting rule. In their work, a flow is learned when it occurs in two successive time windows of 15 minutes, and is forgotten again if it does not happen anymore for 24 hours. Karagiannis et al. are only interested in what I call the *base flow* thus their algorithm filters out flows that occur only sporadically and that way they reduce the number of flows.

The first algorithm I implemented also uses linear forgetting as is do Karagiannis et al. However, most parameters are not fixed and may be chosen by the user. Using the time window selector described above, the user can select the step size and the time window to be displayed and to learn from. That way it is also possible to have overlapping time windows and varying configurations. The linear forgetting algorithm uses a parameter supplied by the user. A flow is forgotten if it does not occur within the specified number of steps and will be classified as dynamic again the first time it occurs.

In the simplest case, the user can make a simple delta algorithm. If the window size and step size are set as equal and the forgetting rate is set to 1, at every step only the flows

---

<sup>25</sup>In fact it is also possible to adapt offline algorithms accordingly, but this usually needs a lot more processing time and memory.

that are new in this window are displayed, thus preventing the user from looking again at and mentally sorting out the flows he has already seen.

The second algorithm I implemented for using online learning is very similar to the first one in many ways. The big difference is the forgetting part of the algorithm. Instead of forgetting a flow if it did not happen within a certain number of time windows, the algorithm counts how many times a flow occurs and keeps a 'persistence' value for every flow. This persistence value is reduced exponentially at every step. If the 'persistence' value falls below a specified threshold, the flow is forgotten.

Using the second approach with exponential forgetting means that it takes longer to forget a flow that happens often than to forget one that only happens rarely. A flow that occurs many times and frequently will not be classified as a *dynamic flow* for a while, even if it stops occurring.

## Implementation

The two online classification algorithms are implemented using a `vector` of instances of the classes `RTBFlowElement` and `RTBFlowElementExp` respectively. Each instance holds all the relevant specific information of the flow, a value used for aging - when it has been last seen for the sliding window <sup>26</sup> algorithm and a 'persistence' value for the algorithm using exponential forgetting.

Both classes provide a method to check whether specifics of a flow passed as parameter match this flow and a method to let it age.

At every time step, every flow is checked against the flows in the data container. If there is no match, it is displayed and the flow is added to the memory. If it is matched, the aging value or the 'persistence' value are adapted. After aging the memory of the algorithm <sup>27</sup>, those who fulfill the conditions for being forgotten are deleted from the data container.

As not the whole set of flows is known in advance, the identification of the service port and the matching of flow can not be done as described in section 3.2. If a flow can not be matched to any existing one, it is added to the data container of learned flows with an unknown service port. As long as the service port is unknown, flows with the same remote IP may have a common source port or destination port to match it. Once either of them is matched to a new flow, it is set as service port and future checks are only made against this port - in addition to the remote IP and protocol.

The online learning algorithms do not make use of reverse DNS lookup. As the total time frame considered by the algorithm is much smaller - usually minutes or a few hours - IP to domain name mapping do not change as often. However, the use of reverse DNS lookup may easily be included in future versions.

---

<sup>26</sup>I call it 'sliding window' because one can imagine a sliding window following the user on the time line, and none of the flows that have already occurred in this sliding window are shown.

<sup>27</sup>The data container containing the flows

## Usage Instructions

Using the GUI, the user has to activate learning by setting the tick in the checkbox. Using radio buttons, *online* learning can be chosen and within online learning the *sliding window* algorithm. There, one can set the number of windows in the past to consider.

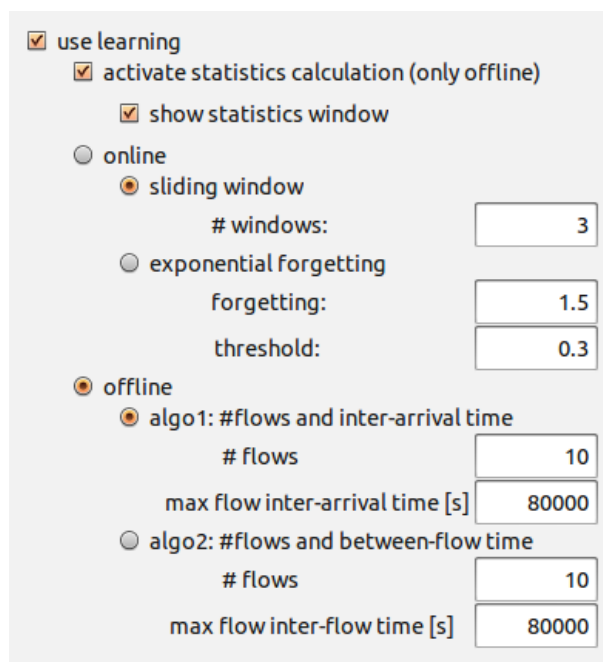


Figure 3.3: Parameter Selection for Classification

The selection of the time window size and the step size <sup>28</sup> is done using the time selector described above. Having made a selection, the user can then use the button *next* to go through the flow list step by step while only the chosen class of flows are displayed.

For the second algorithm, the user selects *exponential forgetting* within *online* learning. There, he can set the forgetting rate, which has to be greater than or equal to 1 <sup>29</sup>, and the threshold. If the 'persistence' value of a flow falls below this threshold value, it will be forgotten. The use of the time window selector and the button *next* is analogous to the sliding window algorithm. Figure 3.3 shows the GUI with the options available to the user concerning flow classification.

### 3.3.2 Offline Classification

Unlike an online classification algorithm, an offline algorithm has the advantage of having knowledge about all the flows in a set at the time of classifying. This means e.g. that already the very early flow may be classified based on the complete set [1].

<sup>28</sup>And thus, together with the number of windows, the size of the sliding window to consider.

<sup>29</sup>Setting it to 1 means that none of the flows seen are ever forgotten.

Calculation time is usually more extensive, but it only has to happen once and all the flows in the set can be classified.

## Motivation and Assumptions

The online classification algorithms are fairly simple. Nevertheless they seem to be fairly useful for visual traffic data analysis. The online algorithms implemented in this thesis have one major drawback, namely that the set of *base flows* changes over time - at every step new flows are added and old ones deleted - and that every flow will be part of the *base flows* at least at one time <sup>30</sup>, even if it happens only once.

This drawback is not important for simple visual inspection of network traffic step by step. It however limits the application possibilities in anomaly detection and user identification. No clear fingerprint can be created and the low level of consistency in the *base flows* makes it hard to find deviations from the normal using the *base flows*, as the 'normal' changes all the time.

To explore this direction further, I implemented two offline classification algorithms. After running any of them, a list of flows classified as *base flows* considering the whole data set is compiled and every flow in the flow list is classified either as *dynamic flow* or *base flow*. This classification does not change over time.

I want to mention here that adapting the set of *base flows* over time is possible and may make sense at some point. In relatively small time windows of hours or a few days, I consider adapting the *base flows* not to be necessary. This however would be an avenue to follow up in the future.

For my offline classification algorithms, I assume a single or small number of users using the computer used as local host in the classification. A computer in an Internet café or a public workstation is probably not suited for this kind a learning, as not only the behavior of the host itself, but mainly the behavior of the person using it is identified and condensed into a fingerprint consisting of the *base flows*. A server on the other hand would be very well suited for my algorithms, as its behavior is fairly constant over time.

In the current version, I also assume no long holidays or absences from the computer. However, this assumption may be taken out in future versions by adding a feature to take out time windows of complete inactivity before applying the algorithm.

## Solution Design

The algorithms implemented for this explorative project are fairly simple. The effectiveness and usability in principle shall be explored without using too many parameters and calculations, which however may be added in the future.

---

<sup>30</sup>When it occurs until it is forgotten again

I implemented two similar algorithms, which I call *algo01* and *algo02* <sup>31</sup>. Even though the number of parameters shall be kept low, the possibility to choose those few parameters shall be given to the user.

Both algorithms are based on the **vector** of matched flows calculated as described in section 3.2. For every collection of matching flows in this **vector**, a method is called to check whether the criteria defined by the algorithm and the parameters are fulfilled for this type of flow to be classified as a *base flow* or not.

*Algo01* uses the number of flows of the same type within the whole imported file in combination with a maximally allowed inter-arrival time <sup>32</sup> between matching flows. The first parameter is fairly obvious, as a type of flow only occurring a few times should not be classified as *base flow* <sup>33</sup>. The second check is needed, as according to my subjective view, a type of flow which occurs many times, but only during a very limited time window should not be classified as *base flow*, it may be only a unique appearance. Using a parameter to limit the maximum inter-arrival time allowed for classification as *base flow* ensures a certain distribution of this flow over time.

*Algo02* is very similar in that it also uses the number of flows of the same type. As a measure for the distribution over time, between-flow time <sup>34</sup> is used instead of inter-arrival time. This change makes it more likely that flows going on for a long time are classified as *base flow*. If this is desired, *Algo02* should be chosen.

After classifying all flow types as either *base flow* or *dynamic flow*, the whole flow list is matched against this classification. Every single flow is checked if it matches the pattern of remote IP and service port. A linear array with matching index to the flow list is then filled with the result <sup>35</sup>.

My extensions to HAPviewer do not only include the algorithm for classification and the possibility to only show either *base flows* or *dynamic flows* in the graphlet, they also include the display of the learned *base flows* in csv format in a list with the opportunity to export it and statistics display features for this classification.

As shown in figure 3.4, values of the relative occurrence of flows classified as *base flows* to those classified as *dynamic flows* are shown on three separate charts. The first chart shows the percentage of the number of flows classified as *base flows*, the second shows the percentage of the number of packets belonging to flows classified as *base flows*, and the third shows the same statistics for the number of bytes depending on classification. In addition to the total value over the whole file, a time line is displayed showing the respective percentages for every time step. Again the step size and the size of the time window can be chosen by the user in the time window selector described in section 3.1.

---

<sup>31</sup>I initially chose these names temporarily but they stuck while working on the project.

<sup>32</sup>The time between the starts of two successive flows of the same type

<sup>33</sup>What should and should not be classified as a *base flow* is a very subjective matter and is a basic problem in this thesis. I am aware that my classification algorithm is subjective and I try to make it less so by giving the user the opportunity to adapt the parameters.

<sup>34</sup>The time between the end of a flow and the start of the next flow of the same type

<sup>35</sup>Whether a flow is a *base flow* or a *dynamic flow*

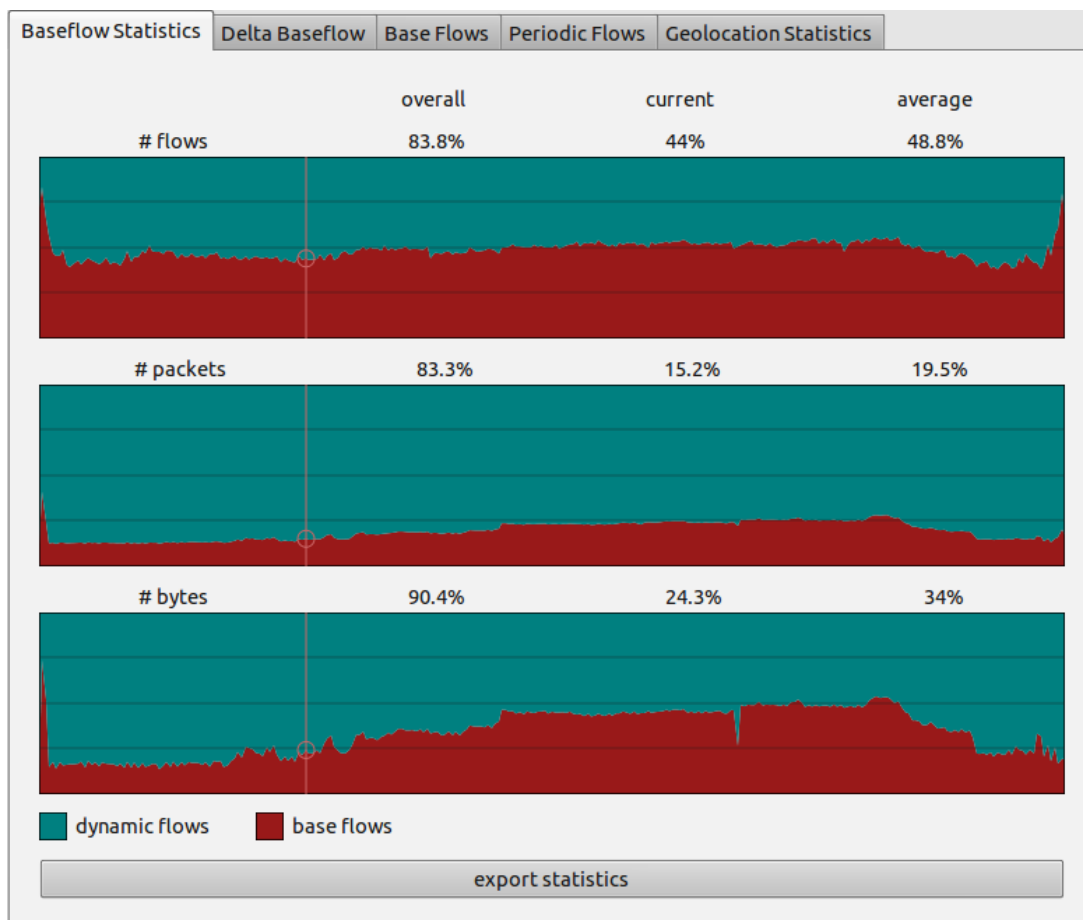


Figure 3.4: Statistics Screen for *Base Flow* Percentages

In addition to the total percentage and the percentage over the time line, the average percentage over all time steps can also be calculated <sup>36</sup>.

In addition to simply showing the statistics, the GUI gives the option to export the complete statistics set to a csv file for further processing.

I use this statistics display to give the opportunity to simply explore the effectiveness of the *base flow* percentage for anomaly detection. This makes it possible to check the assumption that the percentage of *base flows* stays approximately constant over time under normal conditions. Different time window sizes and classification algorithms may be used to check this hypothesis using the display possibilities implemented.

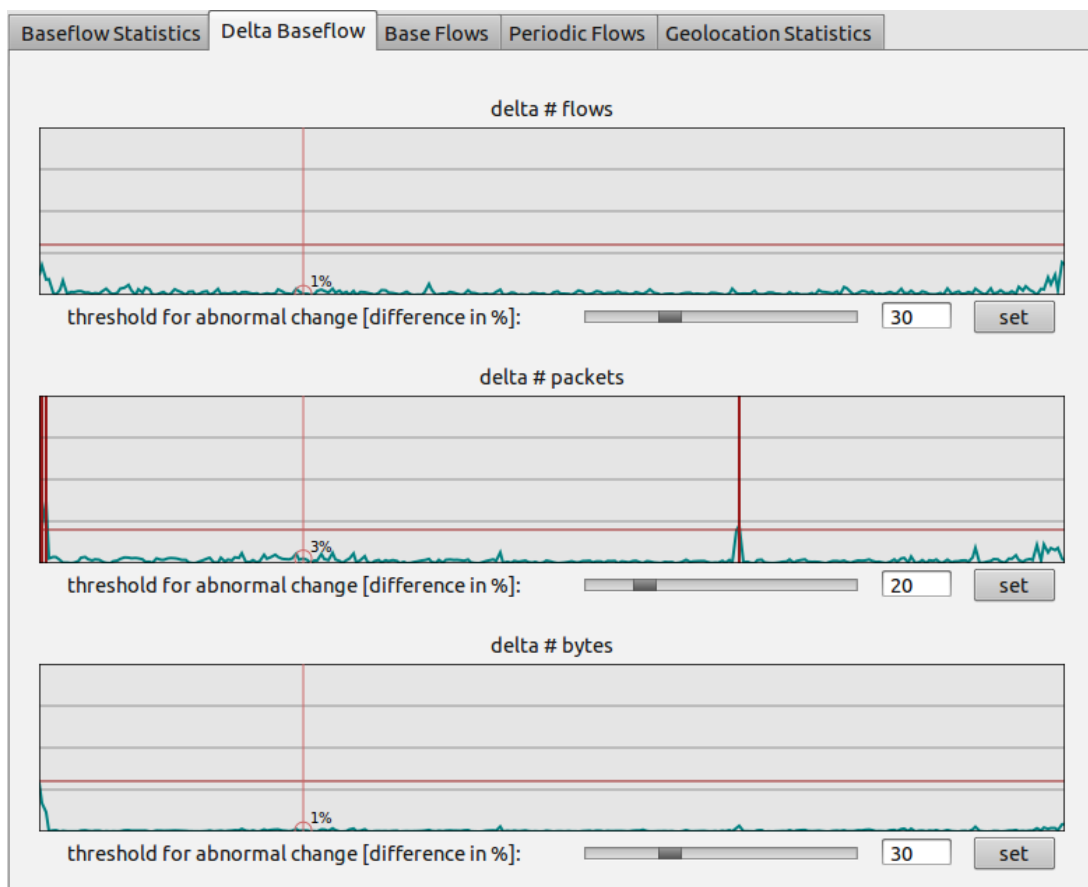


Figure 3.5: Delta *Base Flow* Statistics

To simplify the exploration, a second derived statistics screen as seen in figure 3.5 is available, which shows the delta <sup>37</sup> of percentages between time steps for the number of flows classified as *base flows*, their number of packets, and their number of bytes respectively. The delta is the absolute number of percentage points between two consecutive time steps. This second screen allows the user to set a threshold, automatically marking on the statistics screen all changes that are above this threshold.

<sup>36</sup>Which is not equal to the total percentage, as for the total values, time windows with a lot of flows, packets, or bytes have a lot of weight, in some datasets even dwarfing the other time steps

<sup>37</sup>The difference between the values of two successive time steps

With offline classification active, the time window selector can be used just like before. The user can now also choose whether to display only the flows that are classified as *dynamic flows* or those classified as *base flows* within the set time window in the graphlet.

## Implementation

The functionality of the offline classification algorithms is provided by the classes `RTBCalcAlgo01` and `RTBCalcAlgo02` respectively. Both classes have a pointer to the parameter input user interface - `RTBLearningSelector` - and make use of an instance of a helper class, `RTBCalcHelper`, which provides functionality to match flows in the flow list with a list of classified flows, in this case *base flows*.

To make it easy to change or add classification algorithms, all the common functionality like e.g. flow matching is kept in separate classes. In future work, with this setup of the code, more complex classification algorithms can quickly be included in the code and their effectiveness can be tested using the statistics calculation functionality already provided.

I wrote the statistics display window code from scratch specifically for this purpose. However, it is general enough that it can display any similarly formatted data. As I use object-oriented programming and fairly modular design, just parts of a statistics window can be used for other purposes as well.

The current implementation and functionality can also be seen as a framework to easily allow the testing of new algorithms and it makes it easy to display statistics about the results and export them to a csv file. While programming, a focus has been set on making the tool extensible while already providing much of the basic functionality for future features and algorithms.

## Usage Instructions

To use the offline learning functionality, the checkbox *use learning* has to be checked. The radio button for *offline* learning has to be selected and within that, either *algo01* or *algo02*. For both algorithms, the parameters - number of flows and the maximally allowed inter-arrival time of flows for *algo01* and the number of flows and the maximally allowed between-flow time for *algo02* - can be set using text entry boxes. I use my own text entry boxes to automatically check content entered: whether it is a numerical value as required, if the value is within limits set in the constructor, and the content is rounded to the closest value set as step size in the constructor. The parameter input GUI is shown in figure 3.3.

To calculate and display the statistics, the corresponding checkboxes on the left side have to be activated<sup>38</sup>. If the statistics are not calculated, the classification is still done and affects the graphlet display, but no statistics will be available.

The right side of the window contains the two statistics screens *Baseflow Statistics* and *Delta Baseflow*. If selected, these will automatically be filled. *Baseflow Statistics* contains

---

<sup>38</sup>Both are checked by default.



a button to export the complete set of percentages to a csv file. Each of the three graphs in *Delta Baseflow* contains an entry field with a synchronized slider to set the desired threshold. The chosen threshold is then shown as a vertical red line on the chart. Every time step with a delta exceeding the threshold will be marked with a vertical red line. Note that this input only affects the display of the respective chart and has no effect on the underlying data whatsoever.

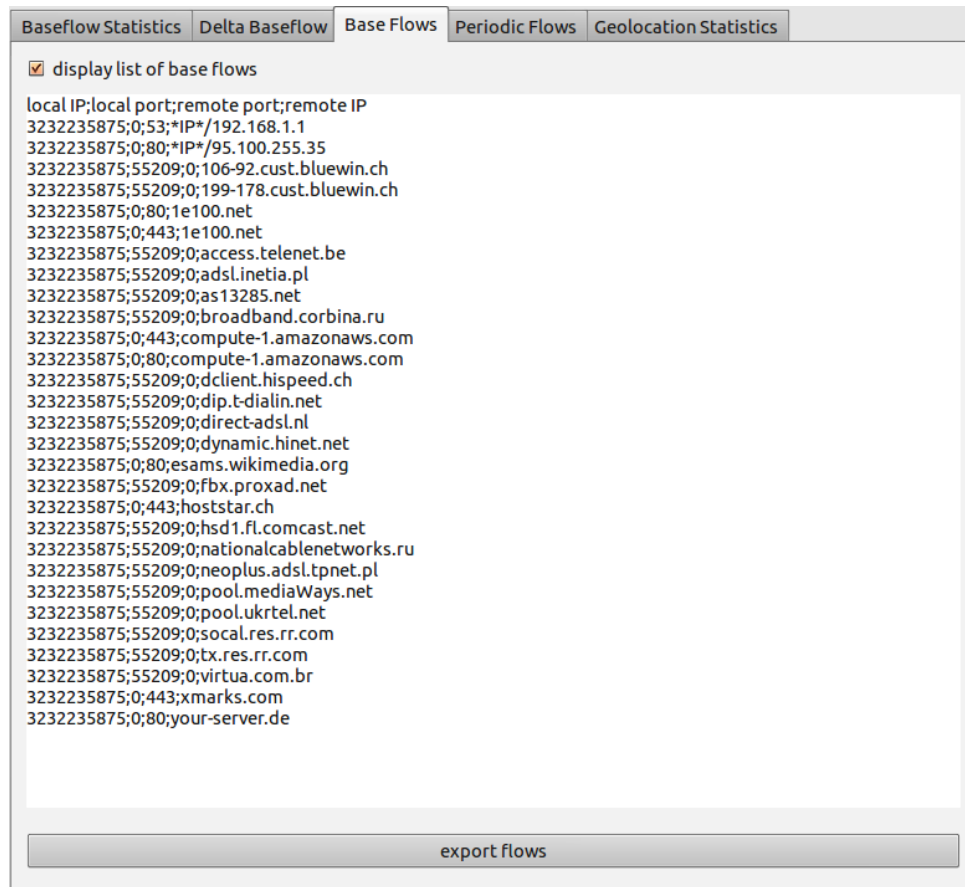


Figure 3.6: List of Flows Classified as *Base Flow*

On the right side of the window as well is a page called *Base Flows*, which shows, when activated using the checkbox, all the types of flows currently classified as *base flows* in csv format. This view can be seen in figure 3.6. The flows in this format can be either copied and pasted or directly exported to a file using the button located at the bottom of the page.

### 3.4 Periodic Flow Identification

Periodic flows are flows that happen with a certain regularity, at a more or less constant interval. I added an algorithm to identify periodic flows within the imported file, provide the option to display only the periodic flows in the graphlet, and implemented the feature to export the periodic flows to a csv file.

### 3.4.1 Motivation and Assumptions

Several network flows in real-life datasets occur at a certain interval. This is not only true for servers which are being backed up regularly and often have actions scheduled, but also for home computers - programs or the operating system check for updates, the mail client checks for new mail on the server, or the cloud storage solution looks for changed files, just to name a few common examples.

In my opinion, knowing which services are called by a computer at regular intervals is very interesting and highly informative. It can show what programs run on the computer, thus extending the possibilities of one of the original purposes of HAPviewer: host application profiling.

While informative on end user's computers, identifying periodic network flows on servers can be very useful as I have experienced myself in a company. Manually finding out which backup processes and data transactions are set up to run at specific times is very cumbersome and time consuming, Having an algorithm show you all the periodic flows from network data alone - in companies often available for forensics purposes - can help save time and money.

My current algorithm assumes that periodic flows mostly <sup>39</sup> have a very similar interval, e.g. periodic flows with alternating intervals will not be identified as periodic. For a periodic flow to be identified, it has to have run a sufficient amount of times in the network traffic file imported, e.g. it is impossible to classify a flow as periodic if it only occurs twice or even once in a file, even if it in fact is periodic on a larger time scale.

### 3.4.2 Solution Design

The algorithm has been improved over several versions as I detected the classification of false positives and false negatives by previous versions. In this subsection I describe the individual steps and why this still failed, to show exactly why I have chosen the parameters I did.

For a type of flow to be classified as periodic, it stands to reason that the variability - in my case expressed by the standard deviation - of the inter-arrival times of the individual flows of the same type has to be low.

This is by far not enough. As mentioned above, a certain minimum amount of flows of the same type have to be observed to even attempt classification as a periodic flow <sup>40</sup>.

This is not yet enough as e.g. a large number of matching flows at the same time <sup>41</sup> still has an inter-arrival time of zero and thus also a standard deviation of zero. Adding a parameter to require a minimum inter-arrival time reduces this problem.

---

<sup>39</sup>Only mostly because the algorithm allows for outliers to be removed

<sup>40</sup>The standard deviation of inter-arrival times of a flow type with two instances is always zero, as the only sample corresponds to the mean.

<sup>41</sup>Some services seem to work like this.

In this version, I detected an obviously periodic flow type <sup>42</sup> which was not classified as periodic. Even though it occurred every hour, two flows were initiated at the same time. Mathematically speaking alternating inter-arrival times of 0 and almost exactly 3600 seconds resulted in a very high standard deviation. Therefore, I included a parameter for the minimally required inter-arrival time of two consecutive flows to be used as a single flow, thus two matching flows happening at the same time or very close to each other are now rated as a single flow.

Turning off the computer overnight, a temporary network failure or e.g. shutting down the email client for some time, did result in all or some periodic flows to be misclassified as non-periodic, as only a single outlier mostly increased the resulting standard deviations above the threshold. Therefore, I added the functionality to remove a chosen amount of outliers before calculating the standard deviation.

While already fairly successful, some flows with fairly small inter-arrival times were sometimes still misclassified, as flow types with hundreds of individual flows tend to have more outliers in absolute numbers, as my explorations have shown. Thus I added the option to select a relative amount of outliers <sup>43</sup> instead of an absolute number.

All flows, which I myself consider to be periodic, are now correctly identified and only certain special conditions lead to a false positive classification.

The user can choose to show only the flow identified as periodic in the graphlet. He can then see a list of the periodic flow types in csv format, and a feature is included to export the list to a csv file. This list also includes the average period in seconds of the periodic flow type.

### 3.4.3 Implementation

The identification of periodic flows uses the `vector` containing the already matched flows as described in section 3.2. The functionality itself can be found in the class `RTBCalcPeriodic`, which uses a temporary data class called `RTBIndexedDouble` containing inter-arrival times and deviations as well as an index. Elements of type `RTBIndexedDouble` can be compared according to the absolute value of their deviations and thus can easily be sorted accordingly.

The calculation of the algorithm is straight-forward, but I would like to mention the removal of the outliers. After all inter-arrival times corresponding to a type of flows have been put into a `vector` of instances of type `RTBIndexedDouble`, the mean is calculated and then the deviation for every inter-arrival time. Until the chosen number of outliers has been removed, iteratively the comparator of `RTBIndexedDouble` is used to sort the elements according to the absolute deviation. Then the inter-arrival time with the highest deviation is removed, the mean of the deviations of the remaining instances of `RTBIndexedDouble` is calculated again and subtracted from every remaining deviation, resulting in the new deviation without the removed outlier. These steps are iteratively repeated until the required number of outliers have been removed.

---

<sup>42</sup>A connection to xmarks (<http://www.xmarks.com/>) to synchronize bookmarks

<sup>43</sup>A percentage of the total number of flows of the same flow type

### 3.4.4 Usage Instructions

<input checked="" type="checkbox"/> calculate periodic flows	
min. # flows necessary	7
min. average inter-arrival t [s]	10
min. inter-arrival t as unique	20
max. accepted standard deviation [s]	200
allowed outliers	
<input type="radio"/> absolute	
# of flows	3
<input checked="" type="radio"/> relative	
% of flows [in %]	25

Figure 3.7: Parameter Selection for Periodic Flow Identification

On the left side of the window, on the page *Periodic Flows*, the user can check the box to calculate periodic flows. The calculation is very efficient, thus this box is checked by default. On this page, the user may also set the parameters - the minimum number of flows required for a flow type to be considered as a potential periodic flow, the minimum average inter-arrival times, the minimum inter-arrival time for flows to be considered as a unique flow, the maximally allowed standard deviation, and the type and amount of allowed outliers. Note that after outlier removal, the flow type still needs the minimum number of flows set as a parameter to be considered for being periodic. The input GUI for periodic flow identification parameter is shown in figure 3.7

On the right side of the window, on the page *Periodic Flows*, the user can check the box to have a list of periodic flows including the average inter-arrival time displayed, as shown in figure 3.8. Using the button *export flows* at the bottom, the user can export the list to a csv file.

## 3.5 Geolocation

Geolocation, in the context of this thesis, means the mapping of an IP address to its country of origin. The IANA assigns IP address ranges to the corresponding authorities of countries<sup>44</sup>. This information is available in databases and can be used to find out where a service with a given IP address is located<sup>45</sup>.

As a further extension to HAPviewer, I use this mapping to identify the distribution of communication partners of the host of interest.

<sup>44</sup><http://www.iana.org/>

<sup>45</sup>I will not go into possibilities of 'faking' this mapping e.g. using proxies, just note that this mapping is not always 100% correct.

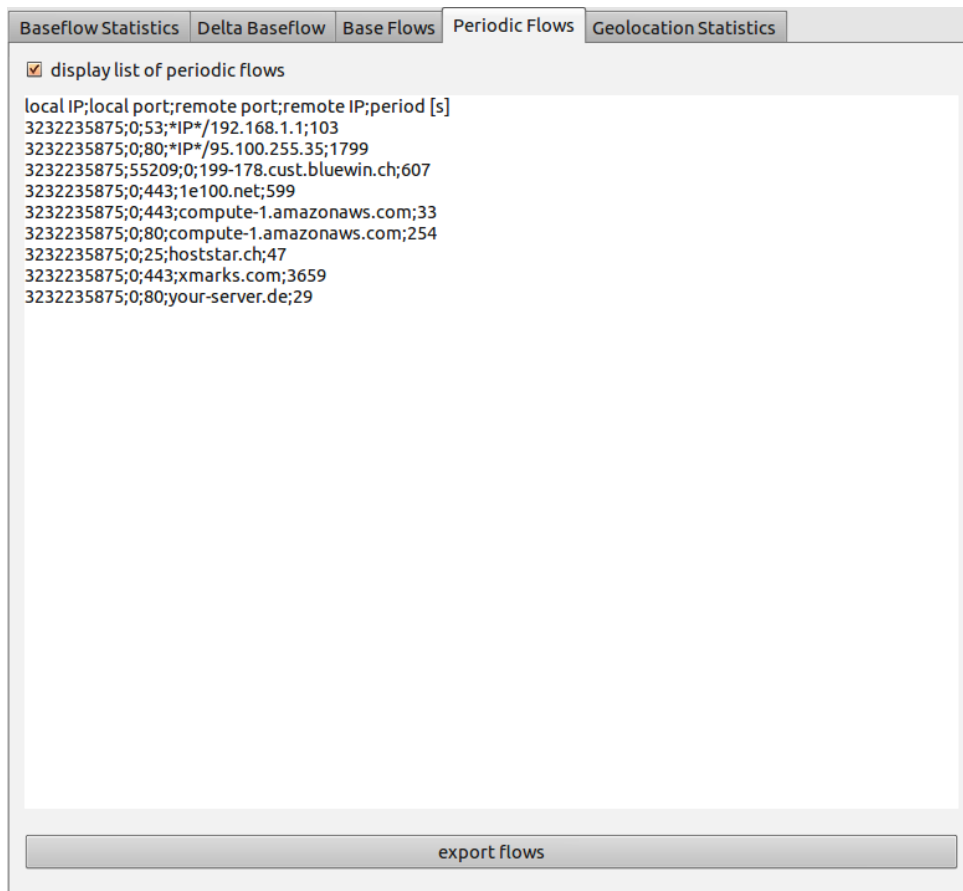


Figure 3.8: List of Flows Classified as Periodic

### 3.5.1 Motivation and Assumptions

I propose that depending on a person's country of origin, his language abilities, his interests, and general Internet use, the communication partners of his computer may have a characteristic distribution in terms of country of origin.

It is possible that this distribution may in some ways be used in the future for host-level anomaly detection and/or user identification.

In addition, I consider that an interested computer user finds looking at an overview of the countries his computer communicates with to be informative. In my opinion, looking at such a graph gives a lot of information on little space.

To explore these possibilities, I added a feature to HAPviewer to display such information visually using charts.

My assumption is that a user's abilities and his interests are reflected in the distribution of countries his computer is connected to.

### 3.5.2 Solution Design

I use a library provided by MaxMind <sup>46</sup> to do the mapping of IP addresses to country of origin. For every flow the country of origin is resolved. From this mapping, aggregated statistics to be displayed on a chart are created and the user has the option to export the distribution to a csv file.

As with other features, three distinct charts are shown, one for the distribution of number of flows according to country, one for the number of packets sent to and received from a country, and the third for the number of bytes. Each of these charts consists of several parts. On one, the distribution of the total amount of flows, bytes, and packets respectively is displayed. The next part displays the average of percentages over a complete time series. The biggest part of the chart shows the time series itself, i.e. the relative distribution of communication partner countries in respect to number of flows, number of packets, and number of bytes respectively for every single time step in the imported file.

### 3.5.3 Implementation

For the mapping, I use the open source GeoIP C API by MaxMind <sup>47</sup>, currently with the free GeoLite country database. For a fee, it is possible to get a more accurate proprietary database, but so far I did not have any problems with the free version which seems to be good enough for my purposes. The database is saved locally. This has the advantages of having the feature work even without Internet connectivity and the IP to country

---

<sup>46</sup><http://www.maxmind.com/>

<sup>47</sup><http://www.maxmind.com/app/c>

resolution is extremely fast. As a disadvantage, some mappings may not be completely accurate all the time and the database should be updated from time to time.

To avoid compilation problems, I added the library as a requirement to the cmake project. For detailed installation instructions, please refer to appendix A. I would like to mention that if future versions may have compatibility problems, the constant `GEO_IP_ACTIVE` in the file `RTBconsts.h` can be defined as 0, which makes the preprocessor remove all connections to the library. While HAPviewer then works without requiring the library, the geolocation statistics would remain empty.

The functionality, which includes but is not limited to interfacing to the GeoIP library, can be found in the class `RTBGeoAnalyzer`. Methods of this class are also responsible for calculating the statistics and inserting them into a data structure of type `map`. The statistics are saved in a instance of the class `RTBCountryData`<sup>48</sup>. This class includes helper methods, a data structure of type `map` to represent the total distribution and an element of type `vector<map<int, double> >`. The `vector` includes for every time step a `map` mapping the integer country code to the corresponding amount of data. This combination, given a known time step and country id, provides highly efficient read access to the values it contains.

### 3.5.4 Usage Instructions

To activate the geolocation feature of HAPviewer, the corresponding tick has to be set on the page *Geostatistics* on the left side of the window. To display the statistics, the tick just below has to be set. As this feature is very efficient, calculation and display are activated by default.

The statistics as shown in figure 3.9 can be viewed on the right side of the window on the page *Geolocation Statistics*. On the left of the charts is an ordered list<sup>49</sup> of labels naming the country corresponding to the color. On the bottom of the page is a button which allows the export of the statistics currently displayed in csv format for further calculations using different software.

---

<sup>48</sup>One instance each for number of flows, packets, and bytes

<sup>49</sup>According to the total relative occurrence of number of flows, packets, or bytes respectively

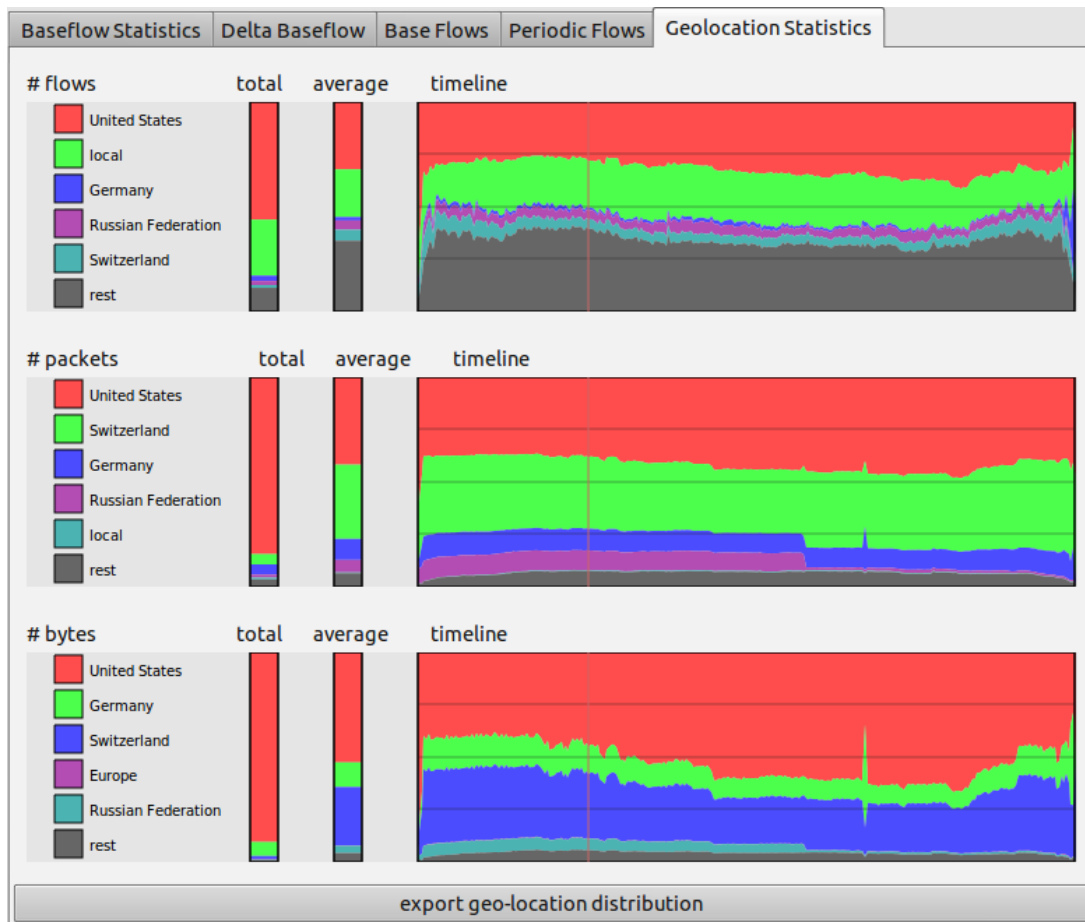


Figure 3.9: Geolocation Statistics



# Chapter 4

## Evaluation

This thesis includes the implementation of new features, the building of a platform for the implementation and testing of new algorithms, and exploring the usability of simple algorithms in terms of how useful they may be for a user to more easily analyze network traffic and to preliminarily assess potential suitability for anomaly detection.

Much of the evaluation has been done while implementing, and the results have been taken into considerations for further improvements and more and better features. As an example, see subsection 3.4.2 about how preliminary tests have shown which direction to take and how to further improve the algorithm.

This chapter shall give some example applications and some quantitative analysis on the usefulness of the features and algorithms implemented for this thesis. I am aware, that even using the simple algorithms already available - i.e. without even taking the ability to use the tool as a framework to implement and test new algorithms - countless experiments can be run on many kinds of data. This chapter shall give a selection I find instructive for the reader.

The usefulness of a feature and an algorithm is inherently a subjective matter. It does not only depend on the task at hand, but also on the person using it. Being aware of this restriction, I demonstrate applications I reckon useful and I show quantitative analysis when possible.

While section 4.1 explains the data used in this chapter, the following sections are ordered according to the features implemented, as already seen in chapter 3. Many evaluations however show several features at once - e.g. most parts use the time window selector and flow matching. The evaluations are placed in the section I consider most suited and other features presented are mentioned there.

### 4.1 Data

During the implementation process, several datasets have been used for evaluation and for identifying next steps. In this chapter, I chose three datasets which seem to be repre-

sentative and may be used to illustrate the evaluation of the implemented features. They have been chosen because they have a certain length in time and they show computers with different purposes and at different dates.

HAPviewer - the pre-thesis version just as well as the versions with my extensions - works on a host level, i.e. it is used to analyze the network traffic going to and coming from a single host. Even though the algorithms may also be employed on servers, my primary interest lies in analyzing end host behavior. Therefore most of my data chosen for this evaluation is coming from a normal desktop computer in use. To add a different view, I also included one dataset from a server, running several services for end hosts.

To make sure the results are constant over time, I use two sets of data collected in April 2011 and one set collected in August 2011.

All the network traces are in netflow format <sup>1</sup> and are available on the attached CD as described in appendix B. The data has not been anonymized and manipulated in any way. As the datasets have been recorded by me on computers and servers used by me in a local network, there are no privacy concerns and the data may be used for any other purposes as well. Netflow data contains aggregated metadata of flows and no payloads, hence no messages can be read or content analyzed.

Table 4.1 and 4.2 show the datasets used in this chapter with their specific information.

Dataset Name	Computer Type	Recording Time	File Name	Size
<i>PD1</i>	desktop computer	April 21 to May 3, 2011	<code>nfcapd.pd1</code>	11.0 Mb
<i>PD2</i>	desktop computer	August 6 to August 11, 2011	<code>nfcapd.pd2</code>	18.8 Mb
<i>GS1</i>	server	April 21 to April 26, 2011	<code>nfcapd.gs1</code>	5.4 Mb

Table 4.1: Datasets Used for Evaluation [1]

Dataset Name	IP Address of Recorder	Number of Flows	Special
<i>PD1</i>	192.168.1.99	50745	
<i>PD2</i>	192.168.1.99	46088	port scan towards the end
<i>GS1</i>	192.168.1.97	43251	bittorrent client briefly running

Table 4.2: Datasets Used for Evaluation [2]

In addition to the datasets themselves, I also provide IP to domain name mappings for all of the datasets, which have been recorded not long after the network traces have been recorded themselves. Importing the IP to domain name mappings does not only solve the problems of IPs changing their mapping to different domain names over time, it also makes calculating much faster. For details please see appendix B.

I am aware that, because of the limited number of datasets used, the results presented in this chapter are not statistically relevant. Given the subjective nature of the evaluation,

<sup>1</sup><http://www.cisco.com/web/go/netflow>

focus on the implementation, and the scope of this thesis, I have chosen to focus on exploring features and algorithms and showing their applications in this chapter.

## 4.2 Time Window Selector

The functionality of the time window selector is straight forward and its usefulness is likely to be evident. Having network traffic files like the ones used in this evaluation as mentioned in section 4.1 with dozens of thousands of flows, displaying all of them makes a graphlet impossible to work with.

In addition, displaying the flows in a graphlet does not show the timing information of flows, this information is implicitly shown when going through a dataset step by step or simply when selecting a time window to display.

Displaying a graphlet of the dataset *PD1* without selecting a time window results in a graph with 132948 edges, even after applying all the role summarization functions already implemented in pre-thesis HAPviewer.

I would have liked to show here at least a part of the complete graphlet, but after calculating the graphlet display for over an hour, the resulting graphlet was so big even though every node in the resulting graph was exactly 3 pixels high <sup>2</sup>, the total height of the resulting graph was still 32767 pixels. Such a graph does not provide any overview and not only the need for a time selection feature is evident, it also shows that reducing the number of flows to display by doing some classification - as shown in different features of this thesis - makes sense.

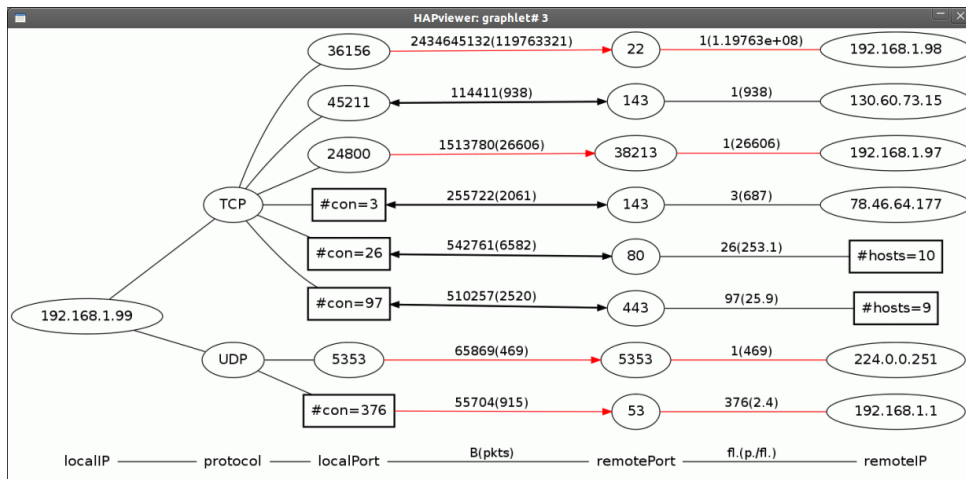


Figure 4.1: Step 1 of Time Series of Flows

Figures 4.1 to 4.4 show as an example four consecutive time windows of one hour each, with a step size of one hour. Just like the complete graphlet mentioned above, these time steps have been taken from the dataset *PD1*.

<sup>2</sup>And thus unreadable

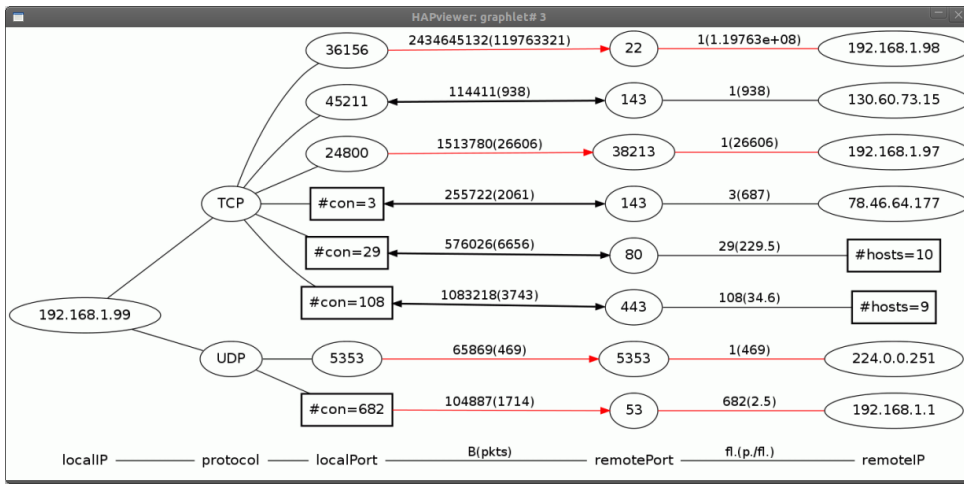


Figure 4.2: Step 2 of Time Series of Flows

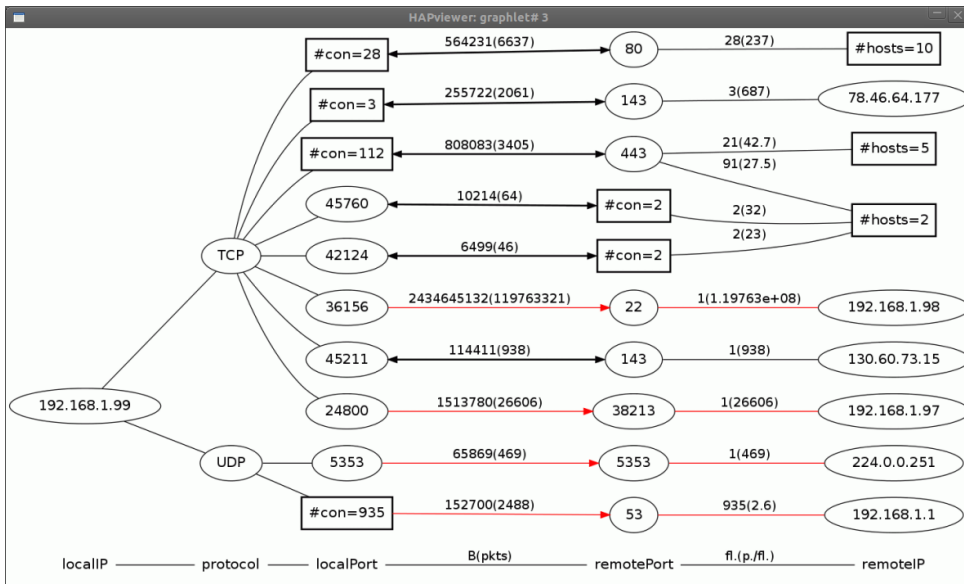


Figure 4.3: Step 3 of Time Series of Flows

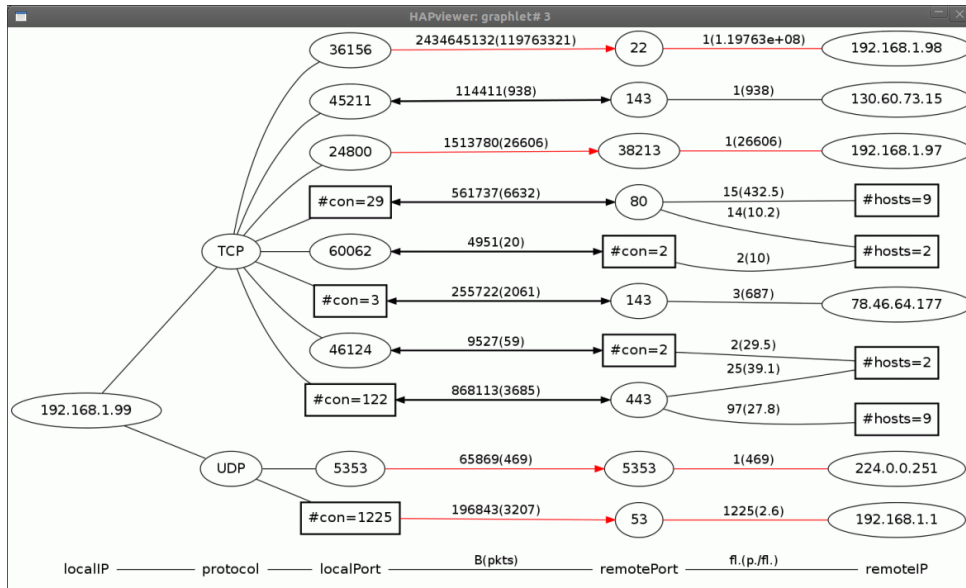


Figure 4.4: Step 4 of Time Series of Flows

Note, that in these figures only a time window has been selected, no classification takes place, no reverse DNS lookup is used, and the only filtering applied is based on the times of the flows. The additional features will be shown in the following chapters.

## 4.3 Flow Matching and Reverse DNS Lookup

Reverse DNS lookup demonstrated to be much more important than originally expected. It turns out that many of the services I would classify as *base flows* use so many different IP addresses, that many of them could not be classified as *base flows* without reverse DNS lookup in some datasets, especially smaller ones.

In this section I show examples of multiple IPs being mapped to the same domain name and how reverse DNS lookup improves the usefulness of the graphlet by replacing the remote IP address by its domain name.

The effect of reverse DNS lookup on classification algorithms is shown in subsection 4.4.2.

### 4.3.1 Multiple IPs to Same Domain Name

As a first evaluation of the usefulness of reverse DNS lookup I take all the unique mappings from the two datasets *PD1* and *PD2*. I choose these two datasets, because they come from the same end host, only from different times, and thus should be using similar services.

It turns out that Google's domain *1e100.com* has been mapped to over 100 distinct IP addresses. I assume the host connects to this domain when using Google calendar <sup>3</sup>.

<sup>3</sup><http://www.google.com/calendar/>

Other examples - just a few are shown here - of domains that correspond to services often used, which are mapped to many IP addresses, include `canonical.com` for Ubuntu One <sup>4</sup> cloud services, `compute-1.amazonaws.com` for ZumoDrive <sup>5</sup> cloud services, `facebook.com` for the social networking platform Facebook <sup>6</sup> and chatting, and `xmarks.com` for synchronization of bookmarks <sup>7</sup> for the browser Firefox <sup>8</sup>.

Domain Name	Number of IP Addresses	Purpose of Service
<code>1e100.com</code>	113	Google calendar
<code>canonical.com</code>	17	Ubuntu One cloud services
<code>compute-1.amazonaws.com</code>	94	ZumoDrive cloud services
<code>facebook.com</code>	43	social networking and chatting
<code>xmarks.com</code>	7	bookmark synchronization

Table 4.3: Examples of Numbers of Distinct IP Addresses Being Mapped to Same Domain Name

Table 4.3 shows examples of multiple distinct IP addresses being mapped to the same domain name with their respective number of distinct IP addresses.

This shows that reverse DNS lookup is an important feature for correct flow matching.

### 4.3.2 Domain Name in Graphlet

Using the reverse DNS lookup part implemented for flow matching, I extend the functionality of HAPviewer by giving the user the option to display domain names instead of remote IPs in the graphlet.

Figure 4.5 shows a randomly selected graphlet from the dataset *PD1* with remote IP addresses as displayed with pre-thesis HAPviewer. Figure 4.6 shows the same selection with domain names instead of remote IP addresses, at least those that can be resolved.

In my opinion, adding this feature helps a user a lot by removing the need to manually decipher IP addresses or trying to guess the domain from port numbers and previous knowledge of the computer in question.

The user does however not have to show the domain names. If he wishes not to, he can simply not activate the corresponding checkbox in the GUI. The user has also the option to either display the complete domain names or the shortened ones. Figure 4.7 shows the same random flow selection as above with complete domain names.

Please note that the remote IP addresses are simply replaced by the domain name in the graphlet, no flow matching takes part for the graphlet calculation!

<sup>4</sup><https://one.ubuntu.com/>

<sup>5</sup><http://www.zumodrive.com/>

<sup>6</sup><http://www.facebook.com/>

<sup>7</sup><http://www.xmarks.com/>

<sup>8</sup><http://www.mozilla.com/en-US/firefox/>

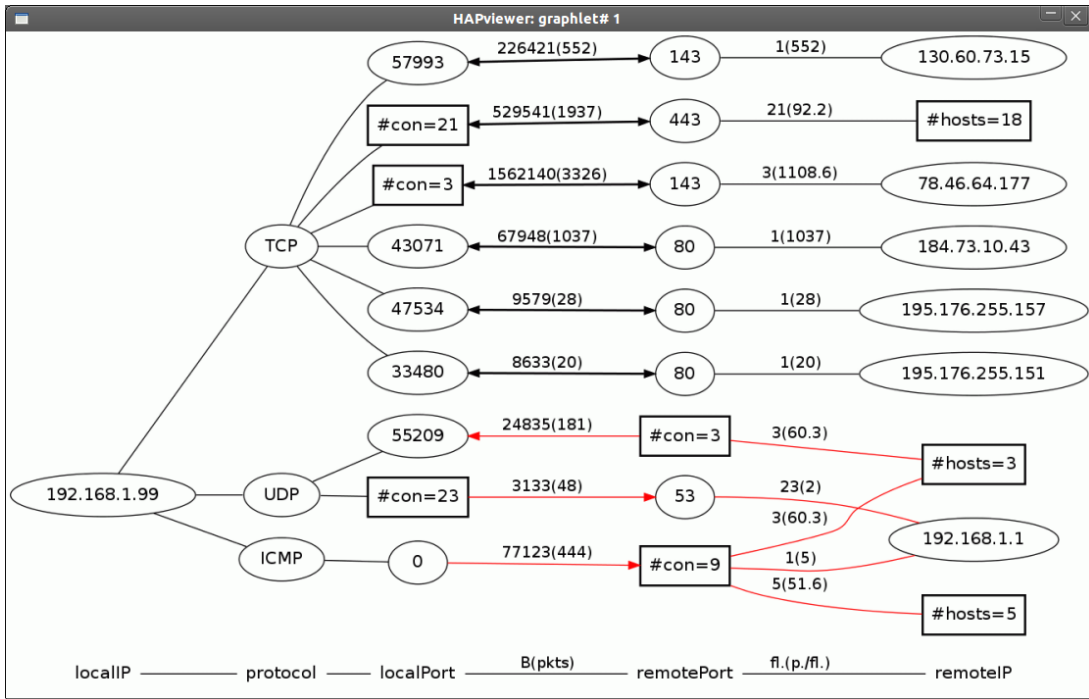


Figure 4.5: Graphlet With IP Addresses

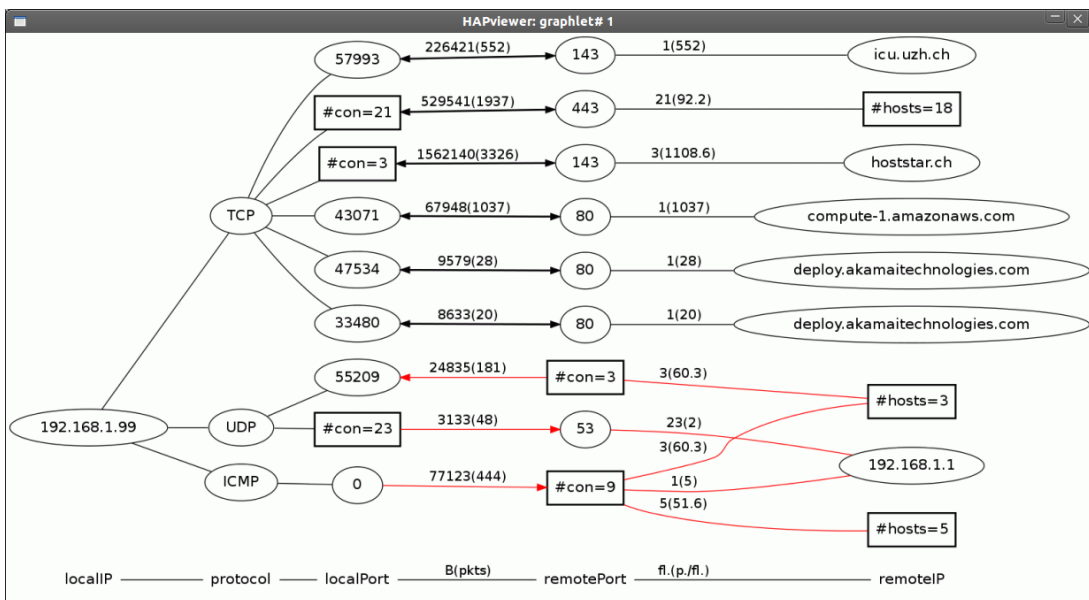


Figure 4.6: Graphlet With Shortened Domain Names

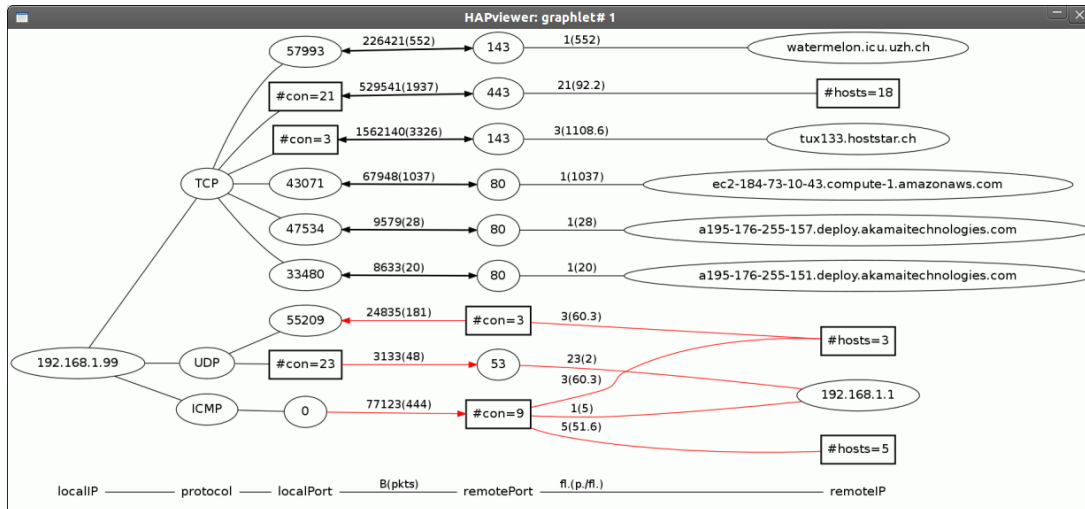


Figure 4.7: Graphlet With Complete Domain Names

## 4.4 Flow Classification

During flow classification, all flows are assigned to one of two groups, either *base flows* or *dynamic flows*. While this assignment may and often does change over time for online classification, it stays constant for offline classification.

In this section, I show some examples of using this feature of HAPviewer and make some - very limited - quantitative analysis. The first subsection shows one example of using online learning to show its advantages, while the second subsection shows a series of evaluations of the offline classification possibilities.

As the user is able to choose parameter values to his liking and depending on the task at hand, whole books could be written comparing different parameter settings and their effect on the classification - the usefulness of which is very subjective. Due to time and size constraints of this thesis, I have to skip a detailed analysis of parameter settings and leave it for future work. I choose parameters which have demonstrated to be successful in my opinion while working on the thesis to run the exploration and the feature presentations shown in this thesis.

### 4.4.1 Online

To demonstrate this feature, I choose the sliding window online learning algorithm. In many ways, the exponential forgetting online learning algorithm has very similar results, the difference being that flows that occur many times are forgotten a lot slower than flows that occur only once or a few times.

As parameter, I selected a sliding window size of 3, this means that every flow that has occurred at least once within the last three time steps will not be displayed. As a time window, I use one hour and the step size is also exactly one hour. Depending on the



amount of details a user wants or how active the network was at the point in time in question, it makes sense to choose a smaller time window size, or a bigger one.

To demonstrate the use, I display the same time series of flows already shown in figures 4.1 to 4.4 in section 4.2, from the start of the dataset *PDI*. However, this time I activate online learning. The resulting time series can be seen in figures 4.8 to 4.11

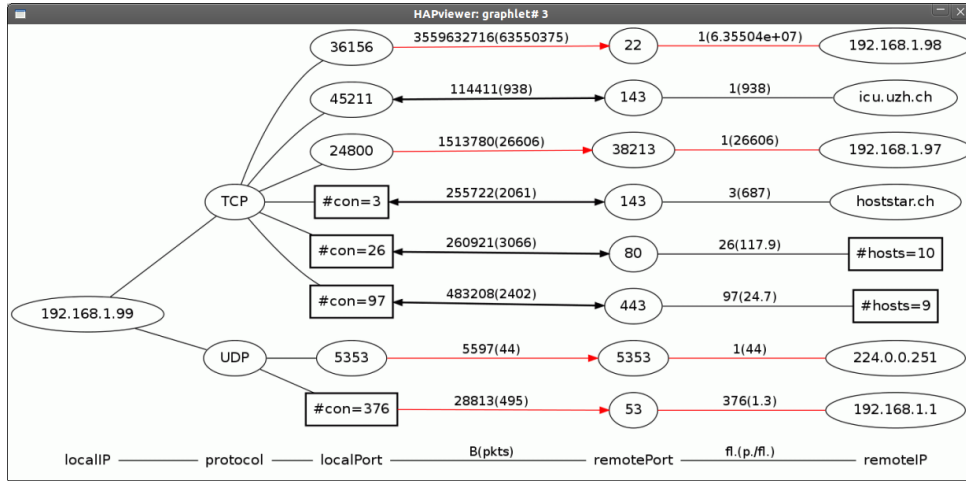


Figure 4.8: Step 1 of Time Series With Online Learning - *Dynamic Flows*

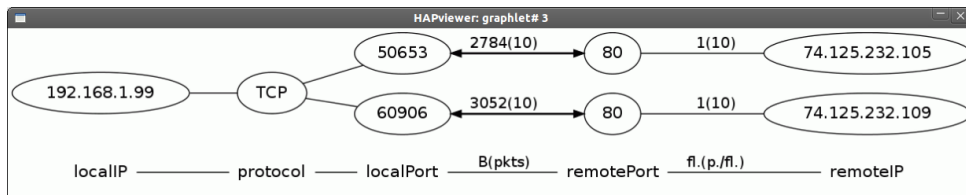


Figure 4.9: Step 2 of Time Series With Online Learning - *Dynamic Flows*

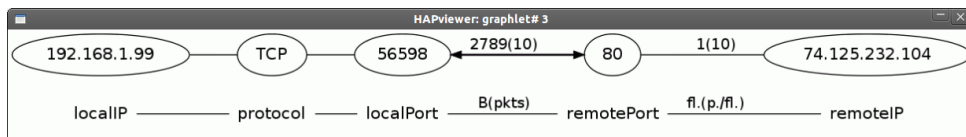


Figure 4.10: Step 3 of Time Series With Online Learning - *Dynamic Flows*

In addition to learning, I also activate the display of shortened domain names instead of remote IP addresses as described in subsection 4.3.2.

Note that the first graphlet of the time series is exactly the same. The reason for this is, that no flow has been learned yet and the algorithm has to start from scratch. However, in all the figures 4.2 to 4.4 a big difference can be seen. As - like in most cases - many flows appear in several consecutive time windows.

In many cases, this simple algorithm can already reduce the number of flows having to be displayed in the graph to a fraction of the original amount, thus showing the user only

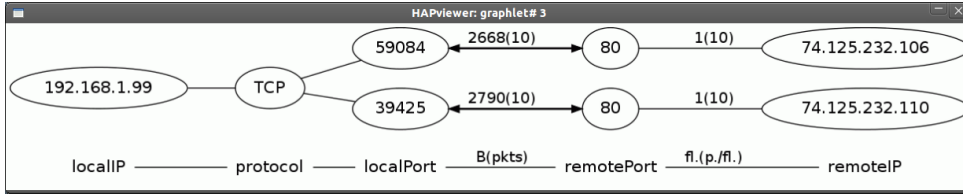


Figure 4.11: Step 4 of Time Series With Online Learning - *Dynamic Flows*

the flows that have changed and relieving him from the laborious task of identifying the flows that are still the same.

In this case, displaying the learned flows - the *base flows* - does not necessarily give much information. It may however be possible to use this feature, e.g. to only display flows that are persistent - i.e. occur multiple times within a specified time frame, and not just once.

The *base flows* from step 2 to step 4 are shown in figures 4.12 to 4.14. Note that the first step is not shown, as it would be an empty graph as no flows have been learned yet as online learning is used.

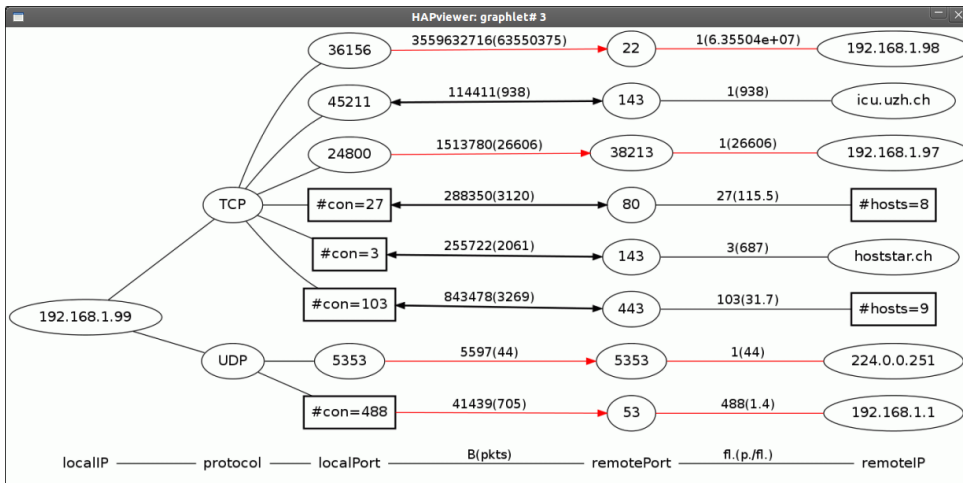


Figure 4.12: Step 2 of Time Series With Online Learning - *Base Flows*

### 4.4.2 Offline

The offline classification algorithms implemented in this thesis work very differently from the online classification algorithms. In addition, from the classification of flows into *base flows* and *dynamic flows* by the offline learning algorithm, statistics are calculated, which can be analyzed and potentially used e.g. for anomaly detection.

In addition to the ratio of the number of *base flows* to the total number of flows overall and as a time series, the difference - the delta - of this ratio is calculated for every step to easily see big changes over time.

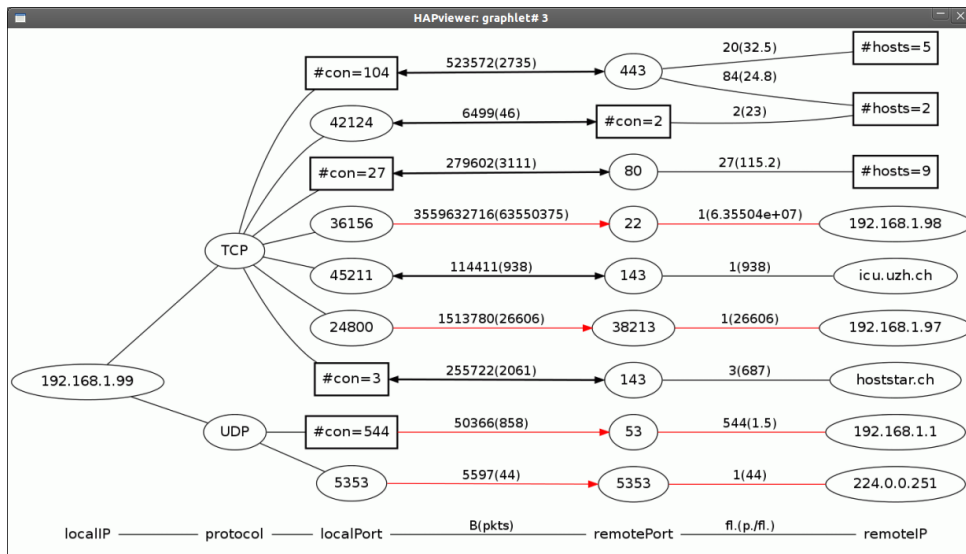


Figure 4.13: Step 3 of Time Series With Online Learning - *Base Flows*

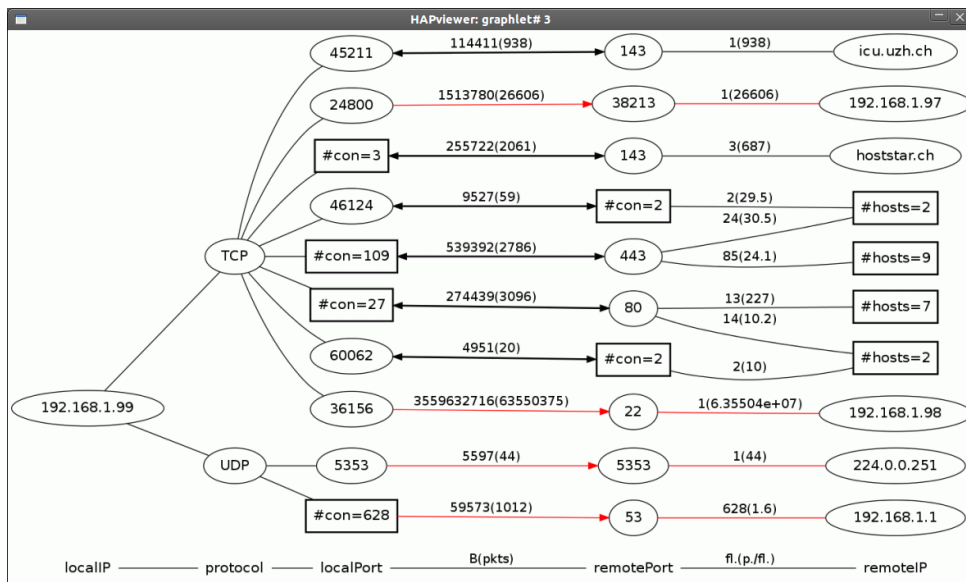


Figure 4.14: Step 4 of Time Series With Online Learning - *Base Flows*

In this subsection, I show first a list of flows classified as *base flows* running the algorithm on the dataset *PD1*. Then I show how the time series used above looks when applying offline classification, before showing and analyzing the statistics of *base flow* percentages and their delta. Statistics are shown for all three datasets to be compared. The next part shows the effect of peer-to-peer downloading and of a port scan on the *base flow* percentages. The last part looks at the effect of using reverse DNS lookup on the percentage of *base flows*.

For this subsection, I use the algorithm *algo01*, which is based on the number of flows and the inter-arrival time of flows.

### Analysis of Base Flows

In this part, I use as parameter *# flows* the value 50 and for *max flow inter-arrival time* 200000. This means that a flow must occur at least 50 times in total and the maximum inter-arrival time must be lower than 200000 seconds, which is a bit over two days. Considering that the dataset spans over almost two weeks and the computer has not been on all the time, these settings seem reasonable.

As mentioned above, it is possible to write books about how to choose the 'best' parameter in every situation alone, thus I can not go into much details. These values proved to give good results, this is why they are taken for this presentation. Table 4.4 shows the parameters used in this part.

Parameter	Value
<i># flows</i>	50
<i>max flow inter-arrival time</i>	200000

Table 4.4: *Algo01* Parameters Chosen for Dataset *PD1*

Running the algorithm *algo01* on the complete dataset *PD1* returns a list of the types of flows classified as *base flows*, as shown in table 4.5. Note that a port of 0 may take any possible value. In the last column of table 4.5, I write what the flow does, if known.

Without looking into every single flow, most of the classifications seem reasonable. The *base flows* contain many entries for Skype contacts, as Skype is almost always open on my computer. It is reasonable to classify the flows to friends that are online often as *base flows*, as they occur very often.

Other *base flows* include DNS requests, connections to the amazon cloud used by ZumoDrive cloud services, Ubuntu One cloud services, synergy for mouse and keyboard sharing between computers, connections to several email accounts, update checking, and bookmark and password synchronization. Also a few webpages I use often - e.g. nzz.ch and one for e-banking - have been classified as *base flows*.

In addition to these *base flows* that would occur on any of my personal computers for probably years, there are some types of flows that correspond to current interests, in the

Local P.	Remote P.	Remote IP / Domain	Comment
0	53	*IP*/192.168.1.1	DNS request
24800	0	*IP*/192.168.1.97	Synergy (input sharing)
0	55209	*IP*/192.168.1.99	Skype (P2P chatting)
0	80	*IP*/212.103.68.58	e-banking
0	80	*IP*/213.173.171.80	
0	80	*IP*/217.33.112.245	SHL (online assessment center)
0	80	*IP*/95.100.255.35	Akamai technologies
0	80	*IP*/95.100.255.58	Akamai technologies
55209	0	106-92.cust.bluewin.ch	Skype
55209	0	199-178.cust.bluewin.ch	Skype
0	80	1e100.net	Google calendar
0	443	1e100.net	Google calendar
55209	0	5-85.cust.bluewin.ch	Skype
55209	0	62-188.cust.bluewin.ch	Skype
0	80	LastPass.com	online browser password vault
0	80	acelb.sj.mozilla.com	browser update checker
0	80	atrilanet	
0	80	boozco.atlasworks.com	Booz & Company
55209	0	broadband.corbina.ru	Skype
0	80	canonical.com	Ubuntu One cloud services
0	80	co106.spacenet.de	
0	443	compute-1.amazonaws.com	ZumoDrive cloud services
0	80	compute-1.amazonaws.com	ZumoDrive cloud services
0	443	dclient.hispeed.ch	
0	55209	dclient.hispeed.ch	Skype
55209	0	dip.t-dialin.net	Skype
55209	0	dynamic.hinet.net	Skype
0	80	eclipse.org	eclipse IDE project
0	80	facebook.com	social networking
0	80	gnome.org	gtkmm documentation
0	143	hoststar.ch	private email account
0	80	hoststar.ch	private email account
0	80	iad2.webtrends-live.com	
0	143	icu.uzh.ch	university email account
0	80	insign.ch	
0	80	lastpass.com	
0	80	microspot.ch	online shop
0	443	microspot.ch	online shop
0	80	nyip.net	
0	80	nzz.everyware.ch	NZZ online portal
55209	0	pool.ukrtel.net	Skype
0	80	telebermuda.com	
0	80	toppreise.ch	Swiss price comparison
0	80	tpimg.ch	
0	80	xmarks.com	bookmark synchronization
0	443	xmarks.com	bookmark synchronization

Table 4.5: List of *Base Flows* After *Algo01* Classification

case presented above namely programming with eclipse using gtkmm - obviously for this thesis - and job search. These flows would only be found during a certain time of flow recording. If a bigger network traffic file ranging over a longer time window would be chosen, I am convinced those flows would not be classified as *base flows* anymore.

In summary, in my opinion, almost all the flows above can be explained and it makes sense to classify them as one class, be it to display only the *base flow* as fingerprint or to not display the *base flow* in every time step they occur, as this allows the user to focus on the more dynamic part. I also suggest that this exploration shows that following the *base flow* approach is a promising avenue for fingerprinting a user to detect him on a network.

### Time Series Using Offline Classification

Using the same parameter settings as above and as shown in table 4.4, I run the *algo01* algorithm on the dataset *PD1*.

To compare graphlets shown after applying this algorithm with online classification (as shown in subsection 4.4.1) and the graphlet without applying any classification (as shown in section 4.2), figures 4.15 to 4.18 show the first four time windows of one hour length from the start of the dataset *PD1*.

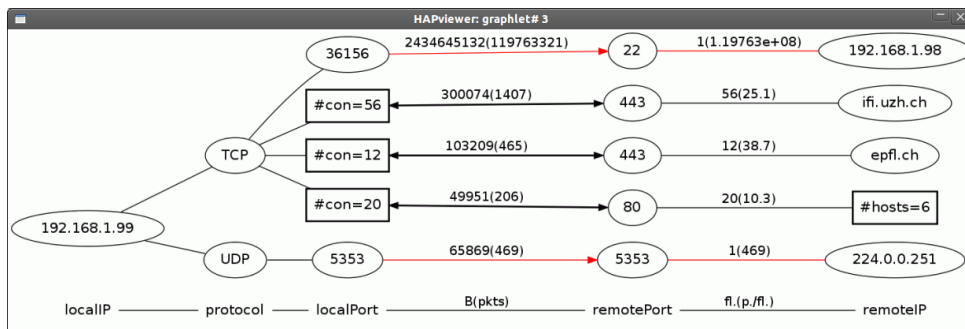


Figure 4.15: Step 1 of Time Series With Offline Learning - *Dynamic Flows*

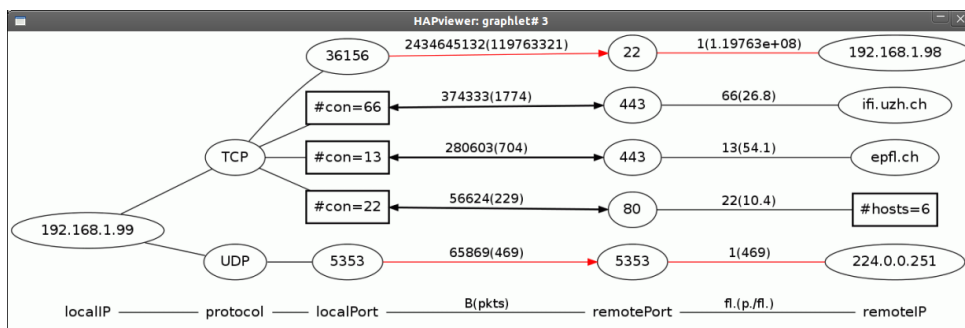


Figure 4.16: Step 2 of Time Series With Offline Learning - *Dynamic Flows*

Figures 4.15 to 4.18 show only those flows that have been classified as *dynamic flows*. Many of the flows are not being displayed anymore, as they have been classified as *base*

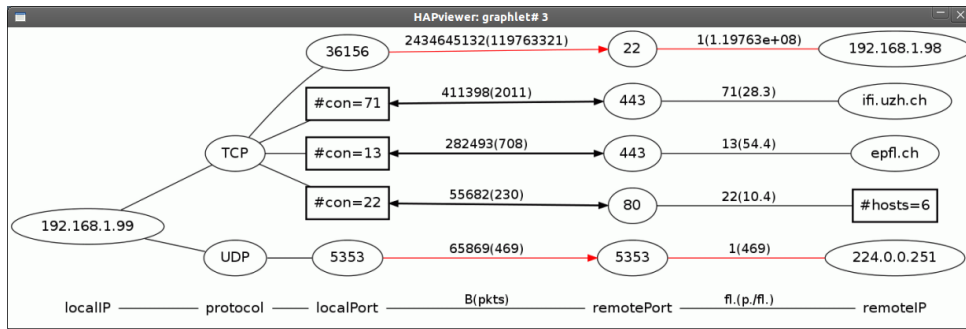


Figure 4.17: Step 3 of Time Series With Offline Learning - *Dynamic Flows*

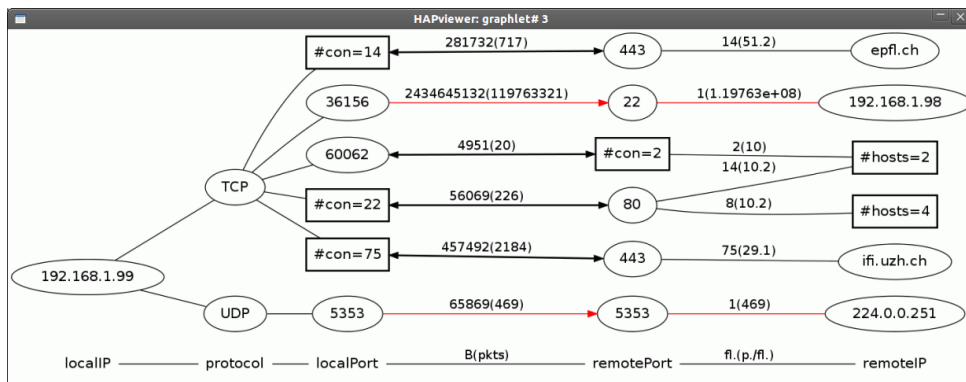


Figure 4.18: Step 4 of Time Series With Offline Learning - *Dynamic Flows*

*flows*. This reduces the number of flows a user has to look at. After he analyzes everything classified as *base flows* using the list as shown above or using the graphlet, the user can focus his attention on those that are not happening all the time, which reduces the time necessary to spend on analyzing a time step.

Also note that, unlike using when online classification, many flows have already been filtered in the first time window. This is one of the advantages of using offline classification, as already during the first step of a series all the information for classification is available to the algorithm.

When looking at small time steps, in my opinion it makes only little sense to display the *base flows*, as they will be similar over many time steps and their total number is not very large. However, to show which flows have been subtracted from the graphlet and are not shown, please look at the figures in section D.1 in the appendix.

### Base Flow Percentage

In this part, I use the statistics feature implemented as an extension to HAPviewer to analyze what percentage of flows, of packages, and of bytes is being classified as *base flow* in total and as a time series. I do this analysis for all three datasets, *PD1*, *PD2*, and *GS1*.

The first goal is to see if the number of flows being classified as *base flow* is large enough for such a classification to be useful in reducing the number of flows to consider for a user and potentially for reducing the storage space needed, as possible approved *base flows* do not need to be stored for forensic purposes anymore.

The second goal is to explore the potential of using any of the percentage measures in host-level anomaly detection. If the percentage is constant enough over time, any big deviation can be seen as an anomaly.

For looking at the deviations, another new feature implemented for HAPviewer is used, namely the possibility to show statistics about deltas in percentages for every time step as explained in subsection 3.3.2.

For the dataset *PD1*, I use the same parameters as above and as shown in table 4.4. However, for the datasets *PD2* and *GS1* I modify the parameters slightly, i.e. the parameter *# flows* is set to 20 instead of 50 while the parameter *max flow inter-arrival time* stays the same. The reason for this is, that the datasets *PD2* and *GS1* are only about five days long, which makes it reasonable to require a smaller absolute number of flows of the same type for it to be classified as *base flow*. The parameters used for the datasets *PD2* and *GS1* are shown in table 4.6

Parameter	Value
<i># flows</i>	20
<i>max flow inter-arrival time</i>	200000

Table 4.6: *Algo01* Parameters Chosen for Datasets *PD2* and *GS1*



Again, a time window of one hour has been chosen and the step size is one hour. Smaller time windows and step sizes will probably lead to less constant behavior, however, the detection of the optimal values has to be left for future work.

Figure 4.19 shows the statistics generated for the percentages of flows, packets, and bytes classified as *base flows* for the dataset *PD1*. Figure 4.20 shows the same statistics generated for the dataset *PD2* and figure 4.21 for *GS1*.

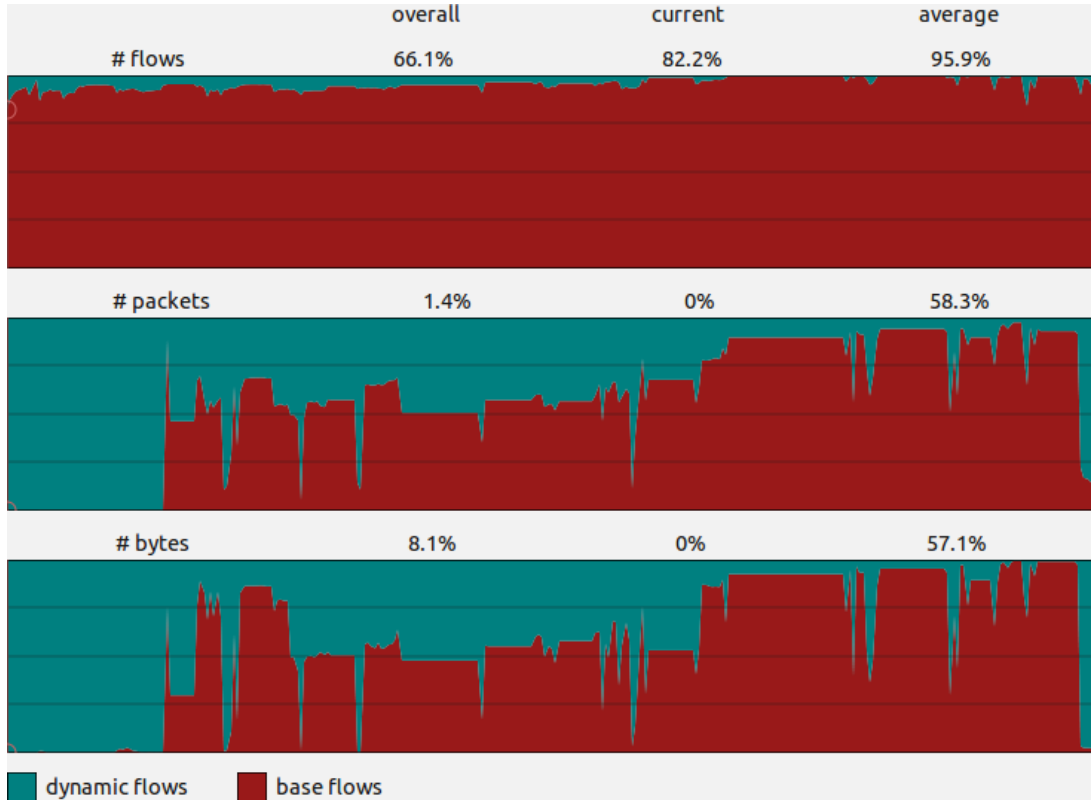
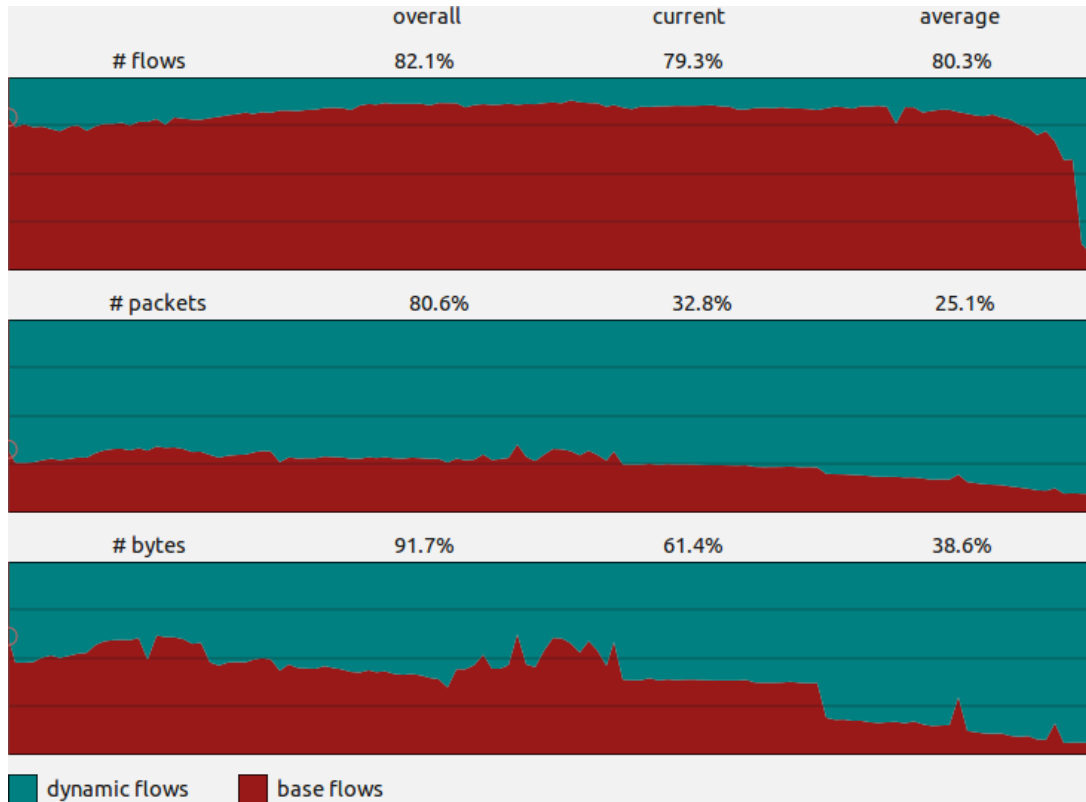
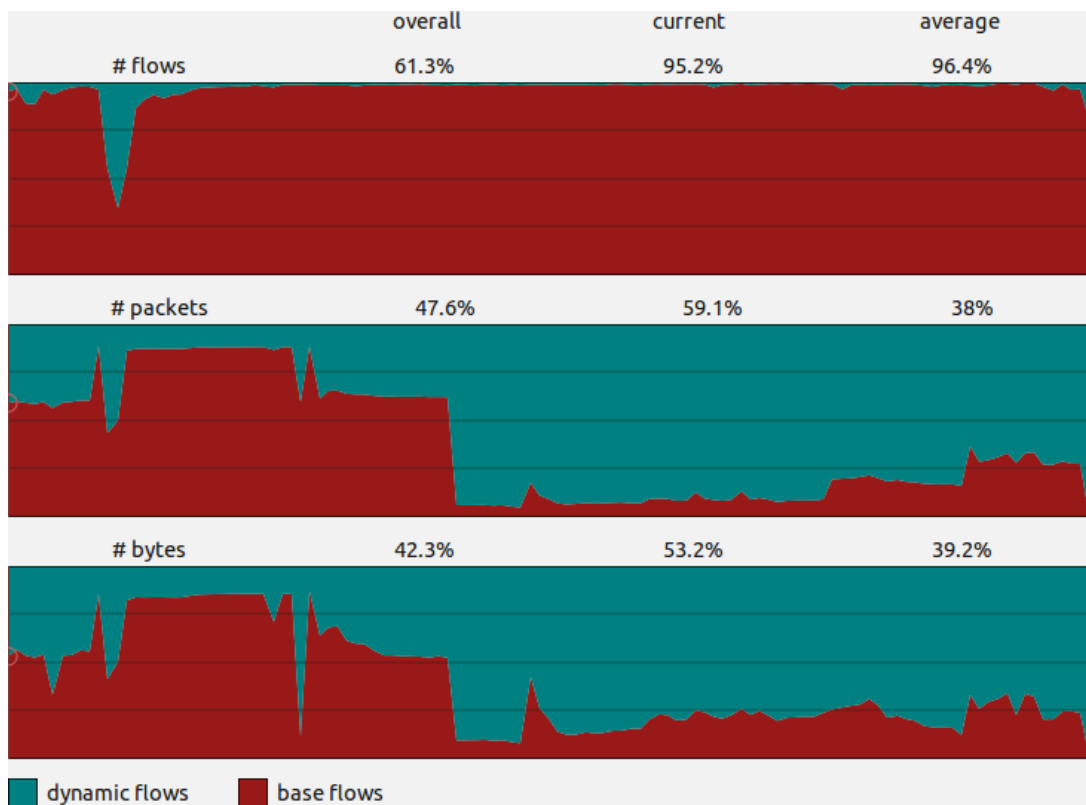


Figure 4.19: *Base Flow* Percentage Statistics of Dataset *PD1*

Figure 4.19 shows that using the parameters shown in table 4.4, on average almost 96% of flows are classified as *base flows* in every time step. This is a considerable amount given that the *base flows* consist only of the few types shown and analyzed above. Given the parameters in table 4.6, the average percentage of *base flows* are 80.3% for *PD2* and 96.4% for the dataset *GS1*, which represents a server. Given that both of those two datasets contain a special case as described below, these amounts are also considerable. Table 4.7 shows the numbers collected when applying *algo01*.

Dataset	Average Percentage	Overall Percentage
<i>PD1</i>	95.9%	66.1%
<i>PD2</i>	80.3%	82.1%
<i>GS1</i>	96.4%	61.3%

Table 4.7: Percentages of *Base Flows* After Application of *Algo01*

Figure 4.20: *Base Flow* Percentage Statistics of Dataset *PD2*Figure 4.21: *Base Flow* Percentage Statistics of Dataset *GS1*

It is expected that a server has more *base flows*, as its behavior is generally limited to a select number of services being run, this assumption is shown true by the numbers calculated in this explorative evaluation.

Another thing to note is, that the overall percentage <sup>9</sup> can be very different from the average percentage <sup>10</sup>. The reason for this is, that the overall percentage gets heavily influenced by outliers, i.e. time steps with very high traffic volume. These outliers can completely skew the overall percentage, while they have little influence on the average percentage, given enough time steps.

Figures 4.19, 4.20, and 4.21 show that the percentage of *base flows* relative to the total number of flows stays fairly constant over time - with a few notable exceptions as described below.

Concerning the statistics about number of packets and number of bytes transferred, the numbers are fairly erratic. These numbers are by no means constant and thus this exploration shows that while the percentage of classified flows in terms of their number is a promising avenue, the usability of the percentages of packets and the percentages of bytes is doubtful.

To further analyze the percentages over the time series, I apply the feature to calculate the difference - the delta - between any two steps. The resulting charts are shown in figure 4.22 for the dataset *PD1*, figure 4.23 for the dataset *PD2*, and figure 4.24 for the dataset *GS1*.

For the display, I chose a threshold of 30%. This means that if the difference of *base flow* percentages in percentage points between any two consecutive time steps is bigger than 30, this time step is marked. The choice of the threshold is arbitrary, given the data and the time windows it seems a good compromise. For possible further application in anomaly detection, an optimal threshold to minimize false positive and false negatives has to be chosen.

As already clear from figures 4.19, 4.20, and 4.21, the deltas in terms of packets and bytes are erratic for two out of three datasets, which suggests their limited usability.

The delta for flow percentages confirm the constancy of percentages of flows classified as *base flows* over time. With the exception of a short period in the dataset *PD2* and a short period in the dataset *GS1*, the deltas stay way below the threshold of 30%.

In-depth analysis of the time windows with the anomalies <sup>11</sup> shows, that during the anomaly detected in the dataset *GS1*, a bittorrent client <sup>12</sup> was running on the server. This activity reduced the percentage of flows classified as *base flows* a lot, because many new, *dynamic flows* are active during this time. This anomaly is detected and the percentage returns to a normal value, as soon as the bittorrent client is turned off again.

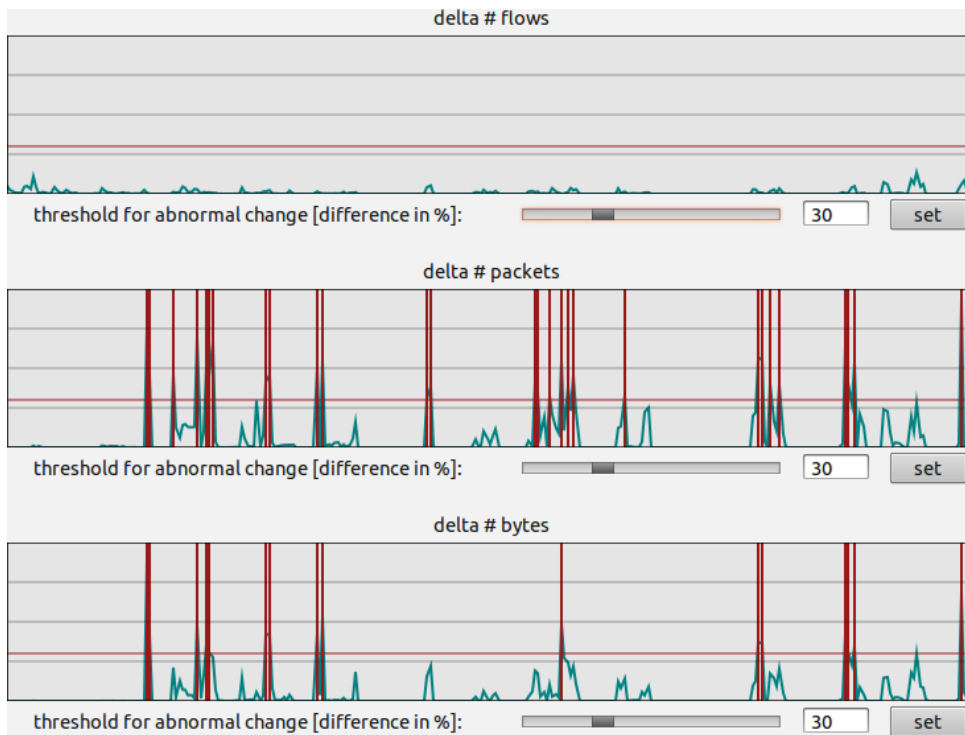
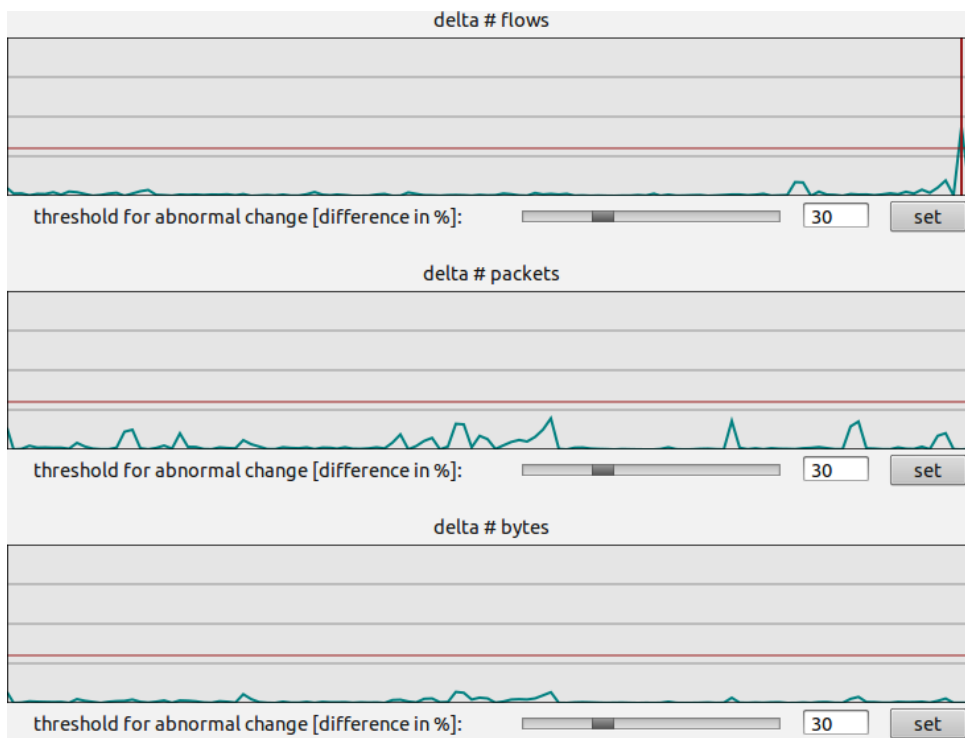
---

<sup>9</sup>Total number of *base flow* divided by the total number of flows in the dataset

<sup>10</sup>Average over percentages of every single time step

<sup>11</sup>Using the time window selector and the graphlet display

<sup>12</sup><http://www.bittorrent.com/>

Figure 4.22: *Base Flow Percentage Delta of Dataset PD1*Figure 4.23: *Base Flow Percentage Delta of Dataset PD2*

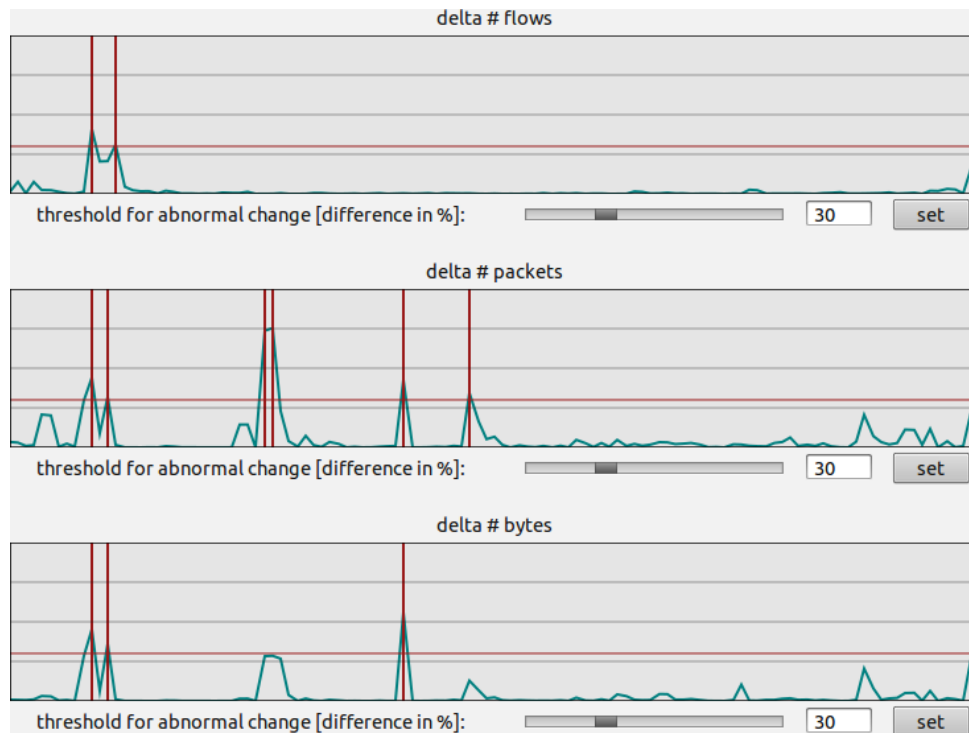


Figure 4.24: *Base Flow Percentage Delta of Dataset GS1*

The anomaly in the dataset *PD2* is due to a port scan which has been run against this computer towards the end of the data recording session. The port scan can easily be detected, as the percentage of *base flows* drops to around 10% from around 80%, as the flows originating from the port scan are correctly identified as *dynamic flows*.

The results shown here are consistent over all three datasets shown in this evaluation and over other dataset used during programming as well. Hence, using *base flow* percentages for anomaly detection in my opinion is feasible and a promising avenue for future work.

### Offline Classification Without Reverse DNS Lookup

Reverse DNS lookup has an influence on the performance of the offline classification algorithms. However, the influence depends a lot on the parameters chosen. Using domain names instead of IP addresses for flow matching makes matching flows using the same service over different IP addresses possible, as described in section 3.3.

This part analyzes the influence of deactivating reverse DNS lookup on the ratio of *base flows* to total flows. Calculating the statistics for all three datasets *PD1*, *PD2*, and *GS1* gives the values shown in table 4.8.

In all three cases, the average percentages are only slightly - between 1% and 5% - below those calculated with reverse DNS lookup active as shown in table 4.7. However, the overall percentage differs by up to 20%. Even as many domain names point to the same IP, given the settings and datasets used in this chapter, many flow types still happen often

Dataset	Average Percentage	Overall Percentage
<i>PD1</i>	94.8%	56.8%
<i>PD2</i>	75.4%	77.5%
<i>GS1</i>	95.1%	42.6%

Table 4.8: Percentages of *Base Flows* After Application of *Algo01* Without Reverse DNS Lookup

enough to be classified as *base flow*. The number of *base flow* types is higher, as now the same type of flow corresponds to several *base flows*, each using a different IP pointing to the same domain name.

Using different parameters and especially a smaller dataset, reverse DNS resolution has demonstrated to be even more useful. However, as shown in this part, given the right parameters and a sufficiently large dataset, the influence of reverse DNS resolution on the statistics calculated for the percentages of *base flows* is on average not very large <sup>13</sup>.

For more details, the graphs showing the statistics calculated without reverse DNS resolution can be found in section D.2 in the appendix.

## 4.5 Periodic Flow Identification

Another extension I implemented for HAPviewer is the detection of periodic flows. Many parameters can be set, which define how periodic a flow has to be, how many outliers are allowed, and so on. Analyzing the effect of each of the parameters with real network traces is not possible within the scope of this thesis, hence all values are chosen according to experience.

Table 4.9 shows the values chosen for this evaluation. For information on the meaning of a parameter, please look at section 3.4.

Parameter	Value
<i>min. # flows necessary</i>	30
<i>min. average inter-arrival t</i>	20
<i>min. inter-arrival t as unique</i>	20
<i>max. accepted standard deviation</i>	400
<i>allowed outliers</i>	relative
<i>% of flows</i>	25

Table 4.9: Parameters Chosen for Periodic Flow Identification

<sup>13</sup>In some cases as seen with the dataset *GS1*, the difference in overall flow classification is still quite large.

As output, it is currently possible to display the periodic flows within the selected time window in the graphlet, or to simply show a list of all periodic flows including their average inter-arrival time.

Periodic flow identification makes use of the flow matching feature described in section 3.2 and thus may use reverse DNS lookup. For the evaluation shown here, reverse DNS lookup is active.

Table 4.10 shows a list of the periodic flows found by the identification algorithm in the dataset *PD2*. The column *Period* shows the average inter-arrival time of the periodic flow in seconds. If a port is set to 0, any port may be substituted for a flow to match the type.

Local P.	Remote P.	Remote IP / Domain	Period	Comment
0	443	*IP*/107.20.50.147	32	ZumoDrive cloud services
0	49887	*IP*/178.254.248.162	1799	
0	53	*IP*/192.168.1.1	70	DNS
80	0	*IP*/192.168.1.130	25	
0	80	*IP*/80.150.21.202	300	Conrad online shop
0	80	*IP*/95.100.255.35	1799	Akamai technologies
55209	0	199-178.cust.bluewin.ch	576	Skype
0	80	1e100.net	1710	Google calendar
0	443	1e100.net	553	Google calendar
0	443	compute-1.amazonaws.com	33	ZumoDrive cloud services
0	80	compute-1.amazonaws.com	243	ZumoDrive cloud services
0	55209	dynamic.hispeed.ch	1799	Skype
0	24470	kellogg.northwestern.edu	1799	
0	80	static.quiettouch.com	14399	
0	443	xmarks.com	3655	bookmarks synchronization
0	80	your-server.de	29	

Table 4.10: List of Periodic Flows of *PD2*

Several flows connected with the ZumoDrive cloud services turn out to be periodic, as are the flows connected with Google calendar, and xmarks bookmarks. These flows are obviously periodic, as the service has to check at periodic intervals whether something has been updated online to synchronize.

In addition, several flows connected to Skype are periodic. As Skype works in a P2P fashion, it seems like the computer checks at regular intervals whether a contact is still online or not. Those contacts that are online often enough appear in the list of periodic flows.

The identification of DNS requests as periodic flows is an artifact of the applied algorithm. As a large number of all the flows are DNS requests and using the part of the algorithm to make matching flows that are close together in time count as one, this type of flow is identified as periodic, even if it is not in reality.

Several other flows seem to be highly periodic, sometimes with large periods. I can not identify them, but a user may be interested in following up on the source of these flows and

find out why they are occurring on his system. After using this extension to HAPviewer for identification of the flows of interest, a user may now drill down and analyze packet payload using a tool like ethereal.

## 4.6 Geolocation

The geolocation extension to HAPviewer gives the option to map all the remote IPs to their country of origin and calculate statistics about their distribution. These statistics include three sets of charts for the distribution concerning the number of flows, the number of packets, and the number of bytes. Each of the sets shows the top five in terms of total occurrence, relative distribution over a time series, and the average relative distribution over the time series.

In this section, I briefly explore the possibilities given by this extension and demonstrate the usage on the three datasets *PD1*, *PD2*, and *GS1*.

The only parameters to set for the usage of this feature are the time window size and the time step size. As consistently throughout this chapter, I choose a value of one hour for both of them.

Figure 4.25 shows the calculated geolocation distribution for the dataset *PD1*, while figure 4.26 and figure 4.27 show the distributions for the datasets *PD2* and *GS1* respectively.

The statistics for the number of packets and the number of bytes seem to be more erratic than the values for the number of flows. Again it can be seen in all figures that the overall distribution gets skewed a lot more by a few outliers than the average distribution over the time series.

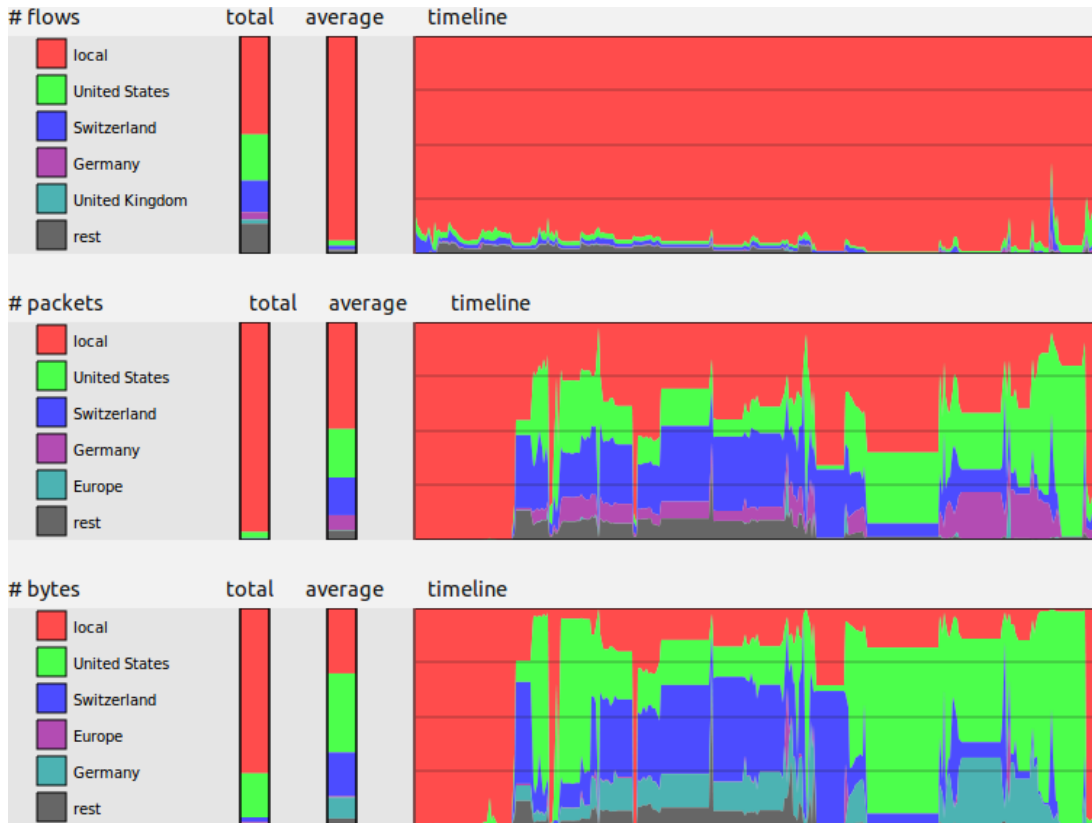
In figure 4.26, the port scan can be detected easily towards the end of the time series in the set of statistics showing the number of flows. Towards the end, during a short period of time, the majority of flows come from a local source - the source of the port scan. The change in the distribution is very clear and points to an anomaly.

Figure 4.26 also shows that while in the beginning hardly any flows come from Germany, at some point this percentage increases a lot and stays on a high level. Something similar happens with Serbia, as at some point many packets are exchanged with an IP located in Serbia, which then stays like this for the rest of the dataset. I do not know the reason for either of those phenomena, but this extension to HAPviewer can point a user to such events which may require further investigation.

In the dataset *PD1*, almost all flows go to the local network as can be seen in figure 4.25. One reason for this significant difference to the dataset *PD2* is, that I used the tool synergy to share a keyboard and a mouse over a local network with other computers, which results in many flows to the local network. I did not use this tool anymore in August.

Figure 4.27 clearly shows the time when a bittorrent client has been running on the server, early during traffic trace recording. The distribution changes completely from the usual



Figure 4.25: Geolocation Distribution for Dataset *PD1*

situation when almost all the flows go either to the United States or are local. Knowing the functionality of the server, this two-part distribution makes sense, as the server offers several services to local computers, and in addition is connected to ZumoDrive cloud services located in the United States for backups.

The initial high traffic load to the local network is due to the transfer of large files over the local network using SSH. The large number of flows afterwards while having a small transfer volume probably originates from the use of synergy, as the server is one of the computers sharing the keyboard and the mouse with the desktop computer from datasets *PD1* and *PD2*.

The results from the explorative evaluation of this feature are not conclusive. In my opinion, information shown like the distribution of countries with which a computer communicates can be highly interesting and show to a user where to investigate further. However, more datasets and more experiments are needed to assess the usefulness of geolocation information for anomaly detection and user identification. Given that activating a bittorrent client as well as a port scan can easily be detected in the distribution as an anomaly and that the distribution gives a hint what kind of anomaly it is - high percentage of flows from a single origin for port scans and a wider distribution of countries of origin for running a bittorrent client - it may be possible to use information provided by geolocation as part of a host-level anomaly detection system.

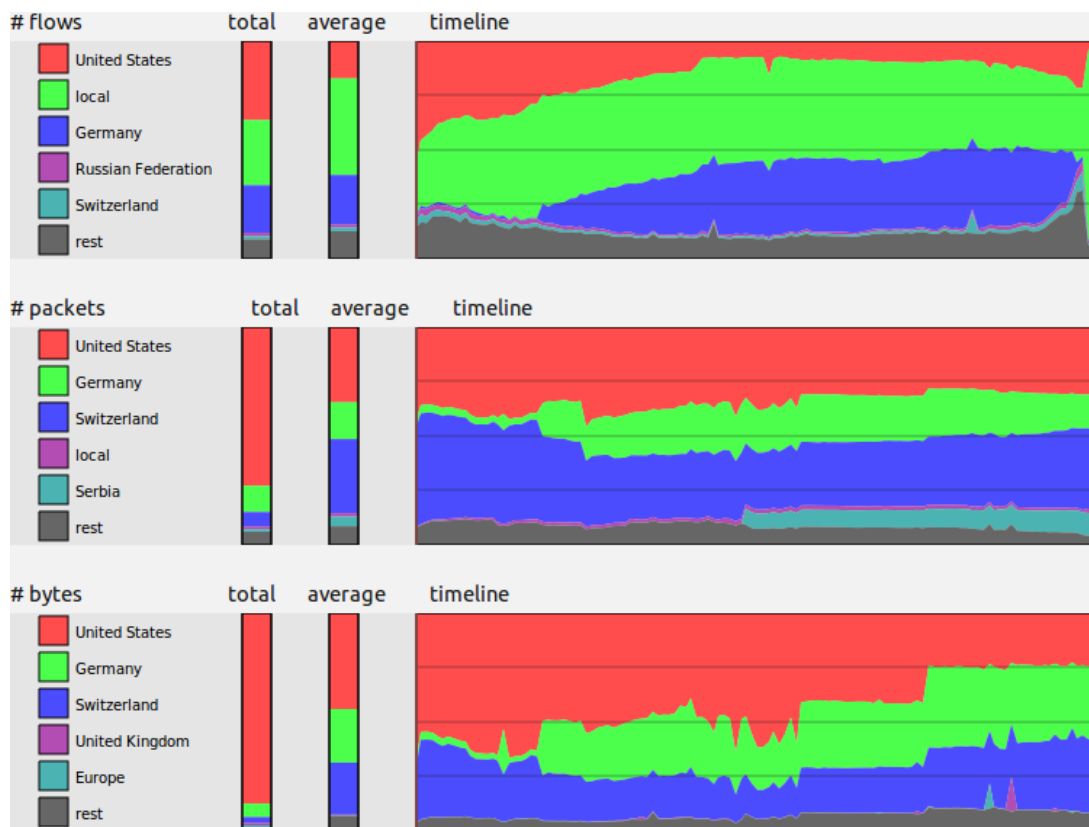
Figure 4.26: Geolocation Distribution for Dataset *PD2*

Figure 4.27: Geolocation Distribution for Dataset *GS1*



# Chapter 5

## Summary

HAPviewer is a tool which can visualize network traffic imported from files as a 5-partite graph following the Berkeley socket model <sup>1</sup>. It works on a per-host basis <sup>2</sup> and contains flow summarization features, i.e. flows can be aggregated according to identified role as client, multi-client, server, or participant in a P2P network.

To extend the usability of HAPviewer and to explore possible applications for anomaly detection and user identification <sup>3</sup>, I design and implement a set of extensions, algorithms, a GUI for user interaction, and a variety of statistics output and export possibilities. The type of programming is chosen to enable the extensions as a platform to easily implement and test further algorithms in future work as well.

A time window selector allows the user to specify precisely to only display flows that either completely or partly fall within a defined time window. It also makes it possible to simply advance this time window by a set amount of time to get a time series display of active flows.

An offline algorithm for flow matching calculates which flows are of the same type. Identification of the service port is done without the use of heuristics. Flow matching also includes the possibility to use reverse DNS lookup on remote IPs to prevent flows going to distinct remote IPs being mapped to the same domain name from being identified as of a different type. To reduce the problem of changing IP to domain name mappings over time and to increase speed, mappings can be exported to a file and imported to be used again.

I develop and implement two supervised online learning algorithms and two supervised offline learning algorithms to classify a set of flows into those occurring often - the *base flows* - and those occurring rarely or only in a limited time window - the *dynamic flows*. The user can select to only show the *base flows* or the *dynamic flows* in the graphlet. When using the offline learning algorithm for classification, a list of learned *base flows*

---

<sup>1</sup>Using a graph with the nodes placed in five columns: local IP, protocol, local port, remote port, and remote IP

<sup>2</sup>Meaning that all flows going to or coming from a selected host are shown

<sup>3</sup>Also labeled fingerprinting

is displayed and may be exported. In addition, a set of statistics concerning the ratio of *base flows* to the total flows can be displayed and exported for further analysis, the difference of the calculated values between any two consecutive time steps can be shown in a separate set of statistics charts.

To detect flows that happen periodically, I implement a special algorithm, which can - just as the classification algorithms - be configured by the user by adapting parameters using the GUI. This algorithm identifies periodic flows, lists them for export, and gives the possibility to show them in the graphlet.

One extension determines the country of origin for every remote IP. It then shows a series of statistics, including the evolution over time, calculated from the distribution of the contact partners in terms of country. This feature also includes the option of exporting the statistics to a file.

The time window selector makes handling large files using HAPviewer possible and introduces the dimension of time with the time series functionality. Dividing the complete duration of a network traffic file into steps is a requirement for the implemented online learning algorithms.

Using the flow matching algorithm, flows are matched successfully to enable the classification and the periodic flow identification algorithms. Even though reverse DNS lookup has a big effect on matching flows correctly, using this feature on large datasets has only a limited effect on the classification statistics.

The online learning algorithms implemented reduce the cognitive overload of a user when going through a time series of graphlets by either only displaying flows he has not already seen recently - the *dynamic flows* - or only those that persist over time - the *base flows*.

Applying the offline learning algorithms produces a set of *base flows* that seem to be highly indicative of the person using the computer and the programs running regularly. Removing all flows belonging to a small set of types classified as *base flows* from a dataset, reduces the total number of flows by between 60% and 95%, depending on the dataset in use. The percentage of *base flows* stays rather constant between time steps, as can be seen in the statistics displaying the differences. Only the statistics concerning the percentage in numbers of *base flows* stays similar, the same statistics for the number of packets or the number of bytes belonging to *base flows* is erratic. A port scan as well as running a bittorrent client results in a big change between time steps - in both cases over 30 percentage points - and thus can be easily identified.

Periodic flows can be identified with a fairly good accuracy, even though one false positive - DNS requests - is still included. Looking at a list of periodic flow may help a user detect what services are running or being used on his computer in the background.

Statistics concerning the geolocation distributions <sup>4</sup> over time show an inconclusive picture. Changes over time are not erratic, but still higher than for the *base flow* percentages. Further exploration is needed to clearly identify or reject the use of geolocation distribution for user identification or anomaly detection. However, the port scan as well as running

---

<sup>4</sup>The distribution of countries of origin of the remote IP addresses

a bittorrent client can easily be detected using geolocation distribution and information can be deduced about the kind of anomaly<sup>5</sup>.

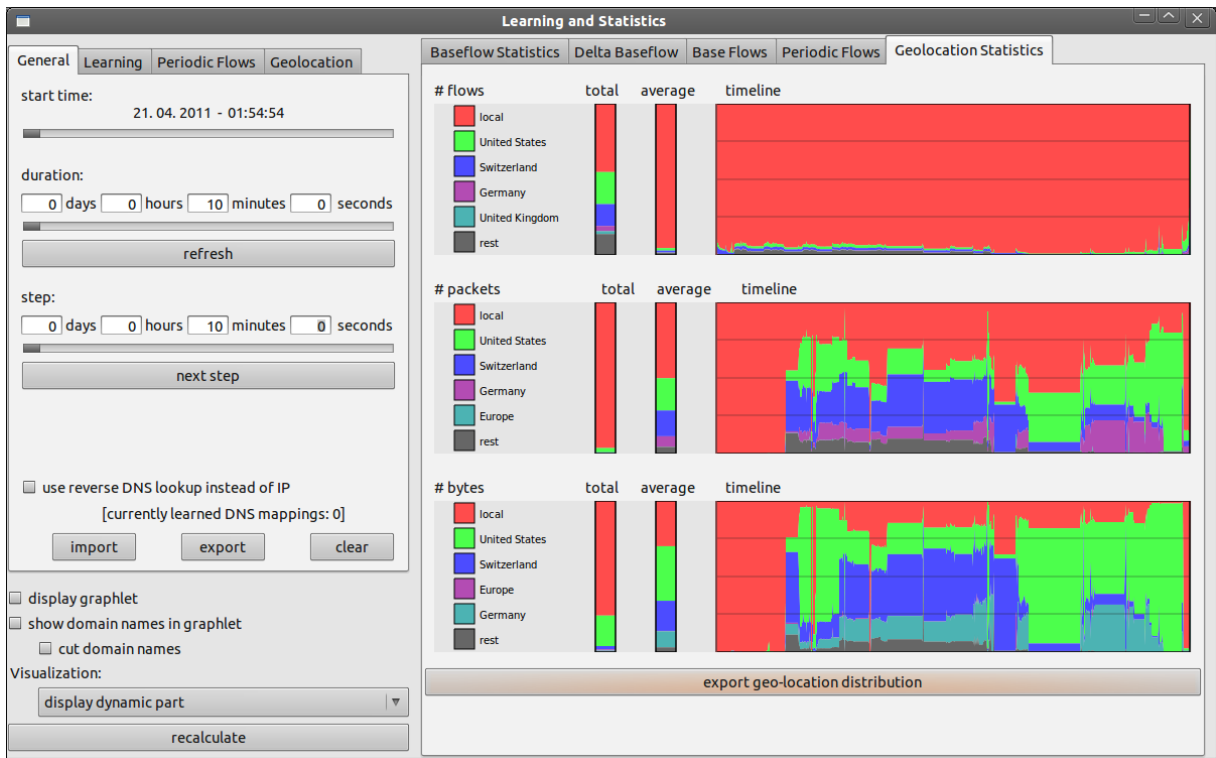


Figure 5.1: Overview of HAPviewer Extensions

Figure 5.1 shows an example of the GUI implemented for the extensions written for this thesis. The extensions to HAPviewer seem to be useful for network traffic visualization and the results of the explorative line of action seem promising for further analysis. The following chapter shows proposed ways of using the extensions, possible applications, and interpretations.

<sup>5</sup>While during the port scan, the distribution is much more concentrated than usual, when running a bittorrent client, the distribution is a lot wider than usual.





# Chapter 6

## Discussion

### 6.1 Time Window Selector

Even though the time window selector allows drilling down in time within a file and thus the import of very large datasets, this practice is currently not necessarily suggested. It is no problem to work on datasets of several hours, up to a day, but working on larger dataset like I did during the evaluation may result in a significant delay. This is due to the fact that HAPviewer was originally not developed to include the possibility to go through time series and as explained in chapter 3, I wanted to minimize invasive programming, thus the data structures and algorithms currently implemented are not optimized for time series display.

A related problem is that currently for every time step, the graphlet is calculated from scratch, which takes time and results in the possible rearrangement of the same flow between two consecutive graphlets. This is hardly a problem when using the online classification algorithms, but might be distracting when having learning deactivated or when using offline classification. One possible solution would be to include the pattern preserving layout algorithm of PatternGravisto presented in [11].

Although there are limitations with the current implementation of the time window selector, in my opinion this extension makes HAPviewer much more usable, as now datasets do not have to be manually preprocessed and adapted to include exactly the interesting parts before importing them, now HAPviewer can be used for finding interesting parts by itself.

### 6.2 Flow Matching and Reverse DNS Lookup

One potential problem with the flow matching algorithm comes from certain P2P applications, namely those that choose both ports seemingly at random. However, to better work behind firewalls and to avoid the need of a central server to set up connections, most

current P2P applications have a constant local port. This results in near flawless flow matching.

In the current version, the flow matching algorithm simply enables the offline classification and periodic flow identification algorithms. This algorithm can however very well be used for different purposes, e.g. for improving the summarization functionality of pre-thesis HAPviewer, to use its output as further input features for other algorithms or to generate a different kind of statistics, i.e. the number of unique flows being classified as *base flows* in every time window instead of the total number of flows.

Reverse DNS lookup has shown to have a big effect on the results of the classification algorithm in smaller datasets, whereas in bigger datasets the effects are between 1% and 20%. Still the usefulness for a normal user is given, as e.g. displaying a flow list with a domain name instead of a remote IP or displaying domain names in the graphlet can be much more easily understood. Reverse DNS lookup reduces the cognitive load and the time necessary for a user to identify what service a specific flow belongs to.

Currently the time necessary for resolving a large number of IP addresses is considerable. Using the option to export an IP to domain name mapping requires this process to run only once per dataset. Further improvement could be made by parallelizing the reverse DNS lookup requests in the code using many threads, instead of doing them in sequence.

## 6.3 Flow Classification

The online learning algorithms, though simple, appear to be very effective in reducing the cognitive load necessary to analyze a time series of graphlets. These algorithms may also be used in a way as has been done by Karagiannis et al. in [9]<sup>1</sup>, but I suggest using offline classification algorithms for that purpose. Implementing more sophisticated online classification algorithms is always possible, but looking at the usability of already the simple algorithms implemented, it might not be necessary at all.

Classification of flows into *base flows* and *dynamic flows* using the offline learning algorithms has shown to be promising in many ways.

First of all, a dataset can be compressed drastically by removing the *base flows* or not storing them in the first place. Experiments with different datasets have shown that this reduction in space can be up to 95%, being between 60% and 85% for the datasets presented in this thesis, even with anomalies. Removing those flows would mean, that after accepting a detected flow as *base flow*, these types of flows, e.g. DNS request to the DNS server, synchronizing cloud services to a known domain name, or synchronizing the mailbox content between mail client and known mail server, would not be stored anymore. This in my opinion makes sense, as those flows, once manually accepted, are hardly malicious. This compression possibility has not been intended initially, it occurs as a side effect of the classification.

---

<sup>1</sup>The creation of a profile

Another promising field is generating a user fingerprint with the set of *base flows* found in his network traffic. Although not statistically relevant, the exploratory results show, that the list of *base flows* contain a small computer specific part, a large permanent user specific part, and a small temporary user specific part <sup>2</sup>. Further studies can be made in creating distance metrics between *base flow* lists and then analyze how well a user may be identified using this profile.

The port scan as well as temporarily running a bittorrent client have been successfully detected as anomalies using a threshold on the delta-metrics calculated on the percentage of *base flows* over time. I am aware that both of these 'anomalies' are fairly easy to detect, but most published academic approaches on anomaly detection are also tested on exactly these kinds of events. Given that the percentage of *base flows* over time stays fairly constant during normal computer use and changes drastically during extraordinary events, using these characteristics for anomaly detection either alone or in combination with other features is in my opinion very promising.

Classification using the offline learning algorithms has also shown to increase the usability of HAPviewer. I suggest displaying *dynamic flows* when looking at a small time window or at time series of graphlets, to avoid having the flows occurring most of the time being a part of the graph. Displaying the *base flows* makes especially sense when looking at larger time windows. This shows only the flows that persist over a certain time and avoids displaying the countless flows that only happen rarely.

## 6.4 Periodic Flow Identification

The current periodic flow identification algorithm, although already gone through many generations, still has an artifact of displaying a type of flow that happens extremely often as being periodic, even if strictly speaking it is not. Knowing the algorithm, this behavior can be explained, as the standard deviation of inter-arrival times is very small, as all the inter-arrival times are very small. In all the datasets tested, this artifact only happens for DNS requests, thus there is one type of flow classified as false positive.

Seeing that periodic flows are classified with high accuracy, a possible application would be to build a database of these periodic flows and set up expectations - i.e. at what time a flow of a certain type is supposed to occur in the future. If such an expected flow does not occur a chosen number of times or the deviation - be it in start time, flow size, or flow duration - is bigger than a threshold, an alarm may be raised or an anomaly communicated. Possible application scenarios include a backup failing to run, a virus scanner not checking for updates anymore, or a regular data transfer not happening. Of course the database of periodic flows could evolve over time.

---

<sup>2</sup>Which is only there if the dataset is small enough. Examples from the datasets used in the evaluation are searching for a job and programming on eclipse using gtkmm

## 6.5 Geolocation

Showing the geolocation distribution of remote IP addresses over time results in interesting graphs to look at, which give additional information about the network traffic happening on a computer and possibly point a user to a potential problem <sup>3</sup>. A user can use these pointers to further drill down and investigate.

However, the evaluation does not conclusively show the usefulness of geolocation distribution data for anomaly detection and user identification. Even though the port scan and running the bittorrent client have both been successfully detected, further studies with a wider array of datasets are needed. In my opinion it is worth to further investigate in this direction, as even if geolocation distributions are not sufficient by themselves, it is possible that synergies arrive when coupling these characteristics with others.

---

<sup>3</sup>One example from the datasets shown in this thesis includes a constant large amount of packets going to Serbia, starting at a clearly defined point in time

# Chapter 7

## Future Work

Some refactoring has already been done and the GUI went through several different generations. However, in the near future, it makes sense to refactor the code and create a better, more intuitive GUI. The current code and the GUI have grown and evolved, it has not been clear from the beginning what features will be included and what exactly they will look like. Refactoring and adapting the GUI would make the extensions easier to use and would simplify further programming using the code.

For performance reasons and to use the new features in the pre-thesis part of HAPviewer it could make sense to further integrate the new extensions with HAPviewer. However, such a step has to be well considered as one would lose the advantages mentioned in chapter 3.

Further studies using the algorithms implemented in this thesis should include more datasets to get statistically relevant results. The usability of the algorithms for the proposed applications can only be conclusively shown by running more experiments, continuing on the basis of the exploratory research done in this thesis.

For the experiments in this thesis, I chose parameters from experience and by comparing results produced with different settings by common sense. It might be useful to run systematic experiments covering a wide range of possible parameter settings. This may show the exact effects of choosing a parameter in a certain way. Using e.g. a manually classified set of *base flows* it is also possible to apply techniques like artificial evolution or simulated annealing for finding a good combination of parameters, as the dimensionality can be quite large for systematic experiments.

Using the platform provided and the statistics output and export possibilities, it makes sense to develop and implement more sophisticated algorithms - for offline classification, online classification, or periodic flow identification. Once developed, implementing an algorithm idea is an easy task using the available platform. It also makes sense to develop algorithms which use more characteristics than the ones used so far.

The applicability to anomaly detection should be tested using more datasets and less obvious anomalies. It also makes sense to test the performance of the algorithm against poisoning attacks as proposed in [17].

Implementing the additional functionalities or applications suggested in chapter 6 like building a database of periodic flows with expectancies or testing the uniqueness of generated user profiles against other users and their constancy over a larger time window can be done in the long run.

I encourage anyone to continue the work on HAPviewer and extend its functionality and usability. I personally would like to use the experience collected while programming for this thesis and the results from the exploration performed in the evaluation section to further improve HAPviewer, that it may widely be used by interested users and network administrators for analysis and visualization of network traffic.

# Bibliography

- [1] Alpaydin, E.: *Introduction to Machine Learning (Adaptive Computation and Machine Learning Series)*, The MIT Press, 2009.
- [2] Chang, S. and Daniels, T. E.: *Correlation based Node Behavior Profiling for Enterprise Network Security*, In Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies, 2009, pp. 298-305.
- [3] Fan, J.: *Network Centric Traffic Analysis: Network Centric Anomaly Detection and Network Centric Traffic Classification*, VDM Verlag Dr. Müller, 2008.
- [4] Glatz. E.: *HAPviewer - Host Application Profile Viewer*, <http://http://hapviewer.sourceforge.net/>, seen April 2011.
- [5] Glatz. E.: *Visualizing Host Traffic through Graphs*, In Proceedings of the 7th International Symposium on Visualization of Cyber Security (VizSec'10), September 2010, pp. 58-63.
- [6] Godiyal, A., Garland, M., and Hart, J. C.: *Enhancing Network Traffic Visualization by Graph Pattern Analysis*, <https://wiki.engr.illinois.edu/download/attachments/169574460/NetflowPatternGraph.pdf?version=1&modificationDate=1238354953000>, seen August 2011.
- [7] Hossain, M.: *Intrusion Detection with Artificial Neural Networks: Anomaly based Intrusion Detection using Backpropagation Neural Networks*, VDM Verlag, 2009.
- [8] Karagiannis, T., Papagiannaki, K., and Faloutsos, M.: *BLINC: Multilevel Traffic Classification in the Dark*, In Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'05), 2005, pp. 229-240.
- [9] Karagiannis, T., Papagiannaki, Taft, N., and Faloutsos, M.: *Profiling the End Host*, In Proceedings of the 8th International Conference on Passive and Active Network Measurement (PAM'07), 2007, pp. 186-196.
- [10] Kim, H., Claffy, K. C., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K. Y.: *Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices*, In Proceedings of the 2008 ACM CoNEXT Conference (CoNEXT'08), 2008, pp. 1-12.

- [11] Klukas, C., Koschützki, D., and Schreiber, F.: *Graph Pattern Analysis with Pattern-Gravisto*, Journal of Graph Algorithms and Applications, Volume 9, Number 1, 2005, pp. 19-29.
- [12] Maloof, M. A.: *Machine Learning and Data Mining for Computer Security: Methods and Applications (Advanced Information and Knowledge Processing)*, Springer, 2011.
- [13] McHugh, J., McLeod, R., and Nagaonkar, V.: *Passive Network Forensics: Behavioural Classification of Network Hosts Based on Connection Patterns*, In ACM SIGOPS Operating Systems Review, Volume 42, Issue 3, April 2006, pp. 99-111.
- [14] Nguyen, T. T. T. and Armitage, G.: *A Survey of Techniques for Internet Traffic Classification using Machine Learning*, Communications Surveys & Tutorials, IEEE, Volume 10, Issue 4, 2008, pp. 56-76.
- [15] Nychis, G., Sekar, V., Andersen, D. G., Kim, H., and Zhang, H.: *An Empirical Evaluation of Entropy-based Traffic Anomaly Detection*, In Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC'08), 2008, pp. 151-156.
- [16] Pietro, R. and Mancini, L. V., eds.: *Intrusion Detection Systems (Advances in Information Security)*, Springer, 2008.
- [17] Rubinstein, B. I. P., Nelson, B., Huang, L., Joseph, A. D., Lau, S., Rao, S., Taft, N., and Tygar, J. D.: *ANTIDOTE: Understanding and Defending against Poisoning of Anomaly Detectors*, In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement (IMC'09), 2009, pp. 1-14.
- [18] Tan, G., Poletto, M., Guttag, J., and Kaashoek, F.: *Role Classification of Hosts within Enterprise Networks Based on Connection Pattern*, In Proceedings of the annual conference on USENIX (ATEC'03), 2003, pp. 15-28.
- [19] Wei, S., Mirkovic, J., and Kissel, E.: *Profiling and Clustering Internet Hosts*, In Proceedings of DMIN'2006, 2006, pp. 269-275.
- [20] Yi, J. S., Kang, Y., Stasko, J. T., and Jacko, J. A.: *Understanding and Characterizing Insights: How Do People Gain Insights Using Information Visualization?*, In Proceedings of the 2008 Conference BEyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV'08), 2008, pp. 1-6.
- [21] Zander, S., Nguyen, T. T. T., Armitage, G.: *Automated Traffic Classification and Application Identification using Machine Learning*, In Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05), 2005, pp. 250-257.



# Abbreviations

API	Application Programming Interface
csv	Comma-Separated Values
DDoS	Distributed Denial of Service
DNS	Domain Name Service
GUI	Graphical User Interface
HAP	Host Application Profile
HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IDS	Intrusion Detection System
ISP	Internet Service Provider
IT	Information Technology
P2P	Peer-to-Peer
PDF	Portable Document Format
PNG	Portable Network Graphics
PS	PostScript
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface



# Glossary

**base flow** Network flow that occurs often and occurrence is distributed over time.

**Berkeley socket model** Local IP, protocol, local port, remote port, and remote IP.

**cflow** Cflow is a proprietary network flow format developed as part of HAPviewer.

**dynamic flow** Network flow that either occurs rarely or all occurrences happen within a small time window.

**flow list** A list of all flows in the imported network traffic file in cflow format.

**geolocation** Mapping of an IP address to its country of origin.

**graphlet** A 5-partite graph representation of flows according to the Berkeley socket model generated by HAPviewer.

**HAPviewer** A tool created by Prof. Eduard Glatz able to display flows in a 5-partite graph and provide flow summarization functionality (see section 1.1 for more information).

**host** A single computer, usually part of a network.

**matching flows** Flows that probably have the same purpose and use the same service.

**non-service port** Each flow has a local port and a remote port. Most of the times, one of them is chosen to identify a service, while the other one is random. The non-service port is the port that is not used for service identification.

**pre-thesis HAPviewer** The version of HAPviewer without my extensions, as available before my project.

**reverse DNS lookup** Determine the domain name corresponding to an IP address using the domain name service.



# List of Figures

1.1	Example of a 5-Partite Graphlet Generated by HAPviewer . . . . .	2
1.2	Host Selection Window of HAPviewer . . . . .	2
3.1	Choosing the Time Window to Display . . . . .	16
3.2	Options Concerning Reverse DNS Lookup . . . . .	20
3.3	Parameter Selection for Classification . . . . .	23
3.4	Statistics Screen for <i>Base Flow</i> Percentages . . . . .	26
3.5	Delta <i>Base Flow</i> Statistics . . . . .	27
3.6	List of Flows Classified as <i>Base Flow</i> . . . . .	29
3.7	Parameter Selection for Periodic Flow Identification . . . . .	32
3.8	List of Flows Classified as Periodic . . . . .	33
3.9	Geolocation Statistics . . . . .	36
4.1	Step 1 of Time Series of Flows . . . . .	39
4.2	Step 2 of Time Series of Flows . . . . .	40
4.3	Step 3 of Time Series of Flows . . . . .	40
4.4	Step 4 of Time Series of Flows . . . . .	41
4.5	Graphlet With IP Addresses . . . . .	43
4.6	Graphlet With Shortened Domain Names . . . . .	43
4.7	Graphlet With Complete Domain Names . . . . .	44
4.8	Step 1 of Time Series With Online Learning - <i>Dynamic Flows</i> . . . . .	45
4.9	Step 2 of Time Series With Online Learning - <i>Dynamic Flows</i> . . . . .	45

4.10	Step 3 of Time Series With Online Learning - <i>Dynamic Flows</i>	45
4.11	Step 4 of Time Series With Online Learning - <i>Dynamic Flows</i>	46
4.12	Step 2 of Time Series With Online Learning - <i>Base Flows</i>	46
4.13	Step 3 of Time Series With Online Learning - <i>Base Flows</i>	47
4.14	Step 4 of Time Series With Online Learning - <i>Base Flows</i>	47
4.15	Step 1 of Time Series With Offline Learning - <i>Dynamic Flows</i>	50
4.16	Step 2 of Time Series With Offline Learning - <i>Dynamic Flows</i>	50
4.17	Step 3 of Time Series With Offline Learning - <i>Dynamic Flows</i>	51
4.18	Step 4 of Time Series With Offline Learning - <i>Dynamic Flows</i>	51
4.19	<i>Base Flow</i> Percentage Statistics of Dataset <i>PD1</i>	53
4.20	<i>Base Flow</i> Percentage Statistics of Dataset <i>PD2</i>	54
4.21	<i>Base Flow</i> Percentage Statistics of Dataset <i>GS1</i>	54
4.22	<i>Base Flow</i> Percentage Delta of Dataset <i>PD1</i>	56
4.23	<i>Base Flow</i> Percentage Delta of Dataset <i>PD2</i>	56
4.24	<i>Base Flow</i> Percentage Delta of Dataset <i>GS1</i>	57
4.25	Geolocation Distribution for Dataset <i>PD1</i>	61
4.26	Geolocation Distribution for Dataset <i>PD2</i>	62
4.27	Geolocation Distribution for Dataset <i>GS1</i>	63
5.1	Overview of HAPviewer Extensions	67
D.1	Step 1 of Time Series With Offline Learning - <i>Base Flows</i>	93
D.2	Step 2 of Time Series With Offline Learning - <i>Base Flows</i>	94
D.3	Step 3 of Time Series With Offline Learning - <i>Base Flows</i>	94
D.4	Step 4 of Time Series With Offline Learning - <i>Base Flows</i>	94
D.5	<i>Base Flow</i> Percentage of Dataset <i>PD1</i> Without Reverse DNS Lookup	95
D.6	<i>Base Flow</i> Percentage of Dataset <i>PD2</i> Without Reverse DNS Lookup	96
D.7	<i>Base Flow</i> Percentage of Dataset <i>GS1</i> Without Reverse DNS Lookup	96

# List of Tables

2.1	Host Features Used for Profile Generation In [19]	10
4.1	Datasets Used for Evaluation [1]	38
4.2	Datasets Used for Evaluation [2]	38
4.3	Examples of Numbers of Distinct IP Addresses Being Mapped to Same Domain Name	42
4.4	<i>Algo01</i> Parameters Chosen for Dataset <i>PD1</i>	48
4.5	List of <i>Base Flows</i> After <i>Algo01</i> Classification	49
4.6	<i>Algo01</i> Parameters Chosen for Datasets <i>PD2</i> and <i>GS1</i>	52
4.7	Percentages of <i>Base Flows</i> After Application of <i>Algo01</i>	53
4.8	Percentages of <i>Base Flows</i> After Application of <i>Algo01</i> Without Reverse DNS Lookup	58
4.9	Parameters Chosen for Periodic Flow Identification	58
4.10	List of Periodic Flows of <i>PD2</i>	59
C.1	Classes of the HAPviewer Extensions	92
E.1	Files in Data Folder	98





# Appendix A

## Installation Guidelines

With all the necessary dependencies needed in the right versions, it is fairly complex to install HAPviewer and being able to run it. This chapter shows all the necessary steps for HAPviewer to run on Ubuntu 10.04, 10.10, and 11.04. I have neither run nor tested HAPviewer on any other system.

Section A.1 shows what libraries and other dependencies are required for HAPviewer to run and how to install them, while section A.2 explains how to install and run HAPviewer itself.

For both parts I have provided a small script - `install/install_dependencies.sh` and `prepare_project.sh` - which does all the necessary steps automatically. Even though I use these scripts on my systems, I suggest using them as a reference and doing the steps one by one. In any case, use the scripts provided at your own risk!

### A.1 Dependencies

HAPviewer has a lot of dependencies, which have to be installed for the program to run properly. The dependencies to install are shown in the following list:

- `make`
- `cmake`
- `cmake-curses-gui`
- `g++`
- `doxygen`
- `graphviz`
- `libgtkmm-2.4-dev`
- `libglib2.0-dev`
- `libglibmm-2.4-dev`
- `libgraphviz-dev`
- `libboost-dev`

- libboost-iostreams-dev
- libboost-thread-dev
- libboost-filesystem-dev
- libboost-regex-dev
- libpcap0.8-dev
- libpcap++-0.0.2
- libfixbuf-0.9.0
- GeoIP-1.4.8

Using some workarounds I have managed to install, run, and extend HAPviewer on Ubuntu 10.04, 10.10, and 11.04. The installation on Ubuntu 11.04 works but has not been thoroughly tested.

To facilitate the installation of all dependencies on new systems, I wrote a small script, located on the attached CD under `install/install_dependencies.sh`, which has been successfully used on Ubuntu of the versions mentioned above <sup>1</sup>. It is important to run the script from the directory `install`. Even though the script is very small and simple, use at your own risk! It is safer to use the script to see what has to be installed how and to then do it manually step by step. Also be aware that the script installs several packages only needed e.g. for the generation of the doxygen documentation, but are not needed to simply run HAPviewer.

Be aware that the libraries *libpcap++-0.0.2* <sup>2</sup> and *libfixbuf-0.9.0* <sup>3</sup> provided on the attached CD are not exactly the same as can be downloaded from their respective websites. I had to apply some workarounds to the code for them to successfully run on newer operating systems.

In addition, a softlink has to be set as well - see the dependencies installation script - as HAPviewer does not look for a dependency in a folder where it will be installed in newer systems.

The GeoIP library <sup>4</sup> - the only library added to the requirements by the extensions implemented during this thesis -, is the newest one which can be downloaded from the Internet at the time of writing, without any changes. It is easily possible to replace this one with a more recent one over <http://www.maxmind.com/app/geolitecountry> if so desired. I added the current version at time of writing to the CD, because it has been tested and works free of charge <sup>5</sup>.

## A.2 HAPviewer

HAPviewer uses cmake for easy installation. Dependencies are checked and the user informed if something is not available. Again, for easy installation, I made a small script

---

<sup>1</sup>Ubuntu 10.04, 10.10, and 11.04

<sup>2</sup><http://libpcappp.sourceforge.net/>

<sup>3</sup><http://tools.netsa.cert.org/fixbuf/>

<sup>4</sup><http://www.maxmind.com/app/c>

<sup>5</sup>I do not know if and when MaxMind might decide not to provide a free version anymore

located at `prepare_project.sh`. Please use this script at your own risk, I suggest using it as a reference and going through the commands one by one.

After entering the command `ccmake .`, you will be prompted with an input screen. There you have to press `c` to configure and then `g` to apply the settings and exit.

The last line - namely `cmake -G"Eclipse CDT4 - Unix Makefiles" ../hapviewer/` - creates an eclipse <sup>6</sup> project and thus is only needed if one intends to use the eclipse IDE.

Having run the `prepare_project.sh` or having followed the steps in it, the tool is ready to be compiled. Run `make` from the folder `build_hapviewer` to create the tool. Compilation can take several minutes. When ready, the executable file to run HAPviewer with the extensions can be found at `build_hapviewer/bin/HAPviewer`.

To automatically create the doxygen documentation from the comments in the source code, run `make doc` from the folder `build_hapviewer`. The generated documentation in HTML format may then be found at `build_hapviewer/doxygen_docu/`. Open `index.html` to start browsing through the documentation.

Using the scripts provided and these instructions, it should be easy to get the program running under Ubuntu 10.04, 10.10, and 11.04 <sup>7</sup>. In case you encounter any problems with the installation on these operating systems, don not hesitate to contact me under `raphael@blatter.sg`.

---

<sup>6</sup><http://www.eclipse.org/>

<sup>7</sup>When running HAPviewer under Ubuntu 11.04, be sure to look for the menubar at the top of the screen as with all programs, not on the program window itself



# Appendix B

## Data Collection

For chapter 4 and for testing features while programming, I needed to collect data in different formats and of different length. I do not want to go into details comparing the different formats and look at advantages and disadvantages, for that the reader is referred to other literature or the specific manuals. In this brief chapter I want to show one easy way on how to record traffic traces on a computer, which then can be opened by HAPviewer and can be analyzed using the features provided.

For the purpose of looking at the HAPviewer features and to analyze what is going on on a computer, I suggest using netflow<sup>1</sup> data. Netflow data does not contain the content being transmitted, it only contains information about every flow that occurred, e.g. the start time of the flow, the duration, the number of bytes transferred, the number of packets, the destination IP, the local port, the destination port, and so on. As HAPviewer does not use the payload data anyway and only storing metadata about flows saves a lot on memory consumption, I suggest using netflow, especially if one is interested in larger time scales.

Linux provides two easy to use tools to make data collection simple<sup>2</sup>. I suggest using a combination of the tool `fprobe` and `nfdump` which includes `nfcapd`.

`Fprobe` is a local probe that collects traffic data and combines it to form netflow data. It then sends this netflow data to a selected address and port, where a netflow collector should be listening for data storage. `Nfcapd` is such a collector daemon. It captures the netflow information sent to it over the selected port and writes it to a file, which then can be further manipulated and/or imported to HAPviewer.

Without going into details, `fprobe` may be started with the following command:

```
sudo fprobe -i <INTERFACE> 127.0.0.1:9555
```

<INTERFACE> has to be substituted by the network interface of interest - e.g. `eth0`.

---

<sup>1</sup><http://www.cisco.com/web/go/netflow>

<sup>2</sup>I am aware of the `flow-tools` package, which is also available to do the same task, however, I did not use it

To start the netflow collector daemon, run the following command on the same machine <sup>3</sup>:

```
nfcapd -t 86400 -p 9555 -B 1000000 -I <NAME> -l <FOLDER>nfddata/
```

<NAME> can be replaced by an arbitrary name to be attached to the flows, e.g. marking them with their computer of origin. <FOLDER> designates where the produced files should be stored on the harddisk.

The tool collection of `nfdump` provides several tools to further manipulate the files stored. Flows can be e.g. filtered according to some criteria or can be combined together. Please note that during this thesis the netflow files have only been concatenated to allow the analysis of a larger time span from netflow files containing one day of flows each. No other manipulation has been performed in order to have authentic data.

This is only one way of collecting network data to be used in HAPviewer. The user is welcome to use e.g. `flow-tools`, `tcpdump`, or `ethereal` to record data. The options I provided can also be adapted according to the specific needs, these instructions however are supposed to help an inexperienced user to quickly get his own network traffic traces to work with.

---

<sup>3</sup>By substituting the IP address, it is also possible to collect the netflow data on a remote machine

# Appendix C

## Class List

The source code for my extensions to HAPviewer consists of almost 40 classes. A detailed description of all the classes, especially of all the member functions and their usage, may be found in the doxygen documentation, which is available on the attached CD or can be built at any time from the source code using the instructions given in appendix A.

Table C.1 shows an overview of all the classes added with a short description.

Class Name	Description
RTBCalcAlgo01	Algorithm class for the Algo01-algorithm
RTBCalcAlgo02	Algorithm class for the Algo02-algorithm
RTBCalcHelper	A helper class for flow matching
RTBCalcPeriodic	Algorithm class to calculate periodic flows
RTBCountryData	Data class for geolocation
RTBCountryDisplayElement	GUI element for statistics
RTBCountryGraph	Graph displaying geolocation data
RTBCountryStats	GUI displaying the statistics of geolocation
RTBDataFlows	Data class for flows
RTBDataSameFlows	Data class containing matching flows
RTBexporter	Helper class for exporting
RTBFlowElement	Class used for Online filtering
RTBFlowFilter	Class for applying filters
RTBGeoAnalyzer	Class providing reverse DNS lookup
RTBGuiElementSliderEntryH	Synchronized slider and number entry
RTBGuiGeoOptions	GUI for geolocation options
RTBGuiInPeriodic	Input GUI for periodic flow calculation
RTBGuiOfflineLearner	GUI element to choose parameters for offline learning
RTBGuiOutList	Displaying and exporting lists of flows
RTBGuiOutStatsDelta	GUI displaying the delta statistics
RTBGuiOutStatsDeltaElement	GUI element for statistics
RTBGuiOutStatsDeltaGraph	Graph displaying delta data
RTBGuiReverseDNS	Interface to the reverse DNS functionality
RTBGuiStats	Input GUI for statistics display selection
RTBIndexedDouble	Data element to sort times
RTBKnownFlows	Data structure for flows
RTBLearningSelector	GUI for choosing learning option
RTBNumberEntry	Entry box with label and checking
RTBOptionsWindow	Window to learning selection and statistics
RTBPortsTimes	Small data class for flows
RTBReverseDNSResolver	Functional class for reverse DNS resolution
RTBSmallSquare	Square for statistics labels
RTBStatsDisplay	GUI displaying the baseflow statistics
RTBStatsElementData	Data class used for statistics
RTBStatsElementDisplay	GUI element for baseflow statistics
RTBStatsGraph	Graph displaying baseflow data
RTBTPointField	Entry object for times
RTBVisualSelection	Selector for flows to display

Table C.1: Classes of the HAPviewer Extensions



# Appendix D

## Additional Figures

This appendix includes some figures and explanations which may help better understand the points made in the text, but can very well be ignored if need be.

### D.1 Base Flows of Time Series After Offline Classification

Figures D.1 to D.4 show all the *base flows* that occur in the first four time steps of one hour length of the dataset *PD1* after classification with *algo01*. These have been subtracted from all the flows to only display the *dynamic flows* as shown in part 4.4.2.

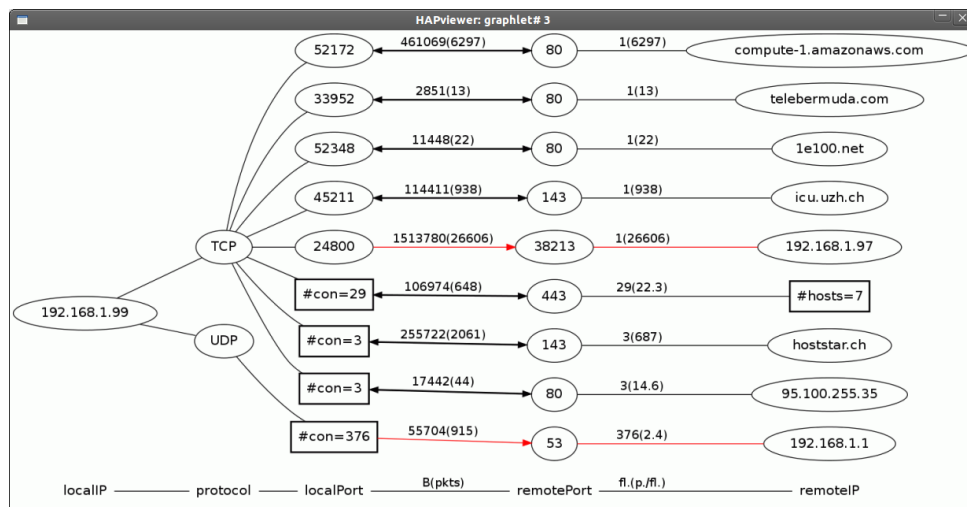


Figure D.1: Step 1 of Time Series With Offline Learning - *Base Flows*

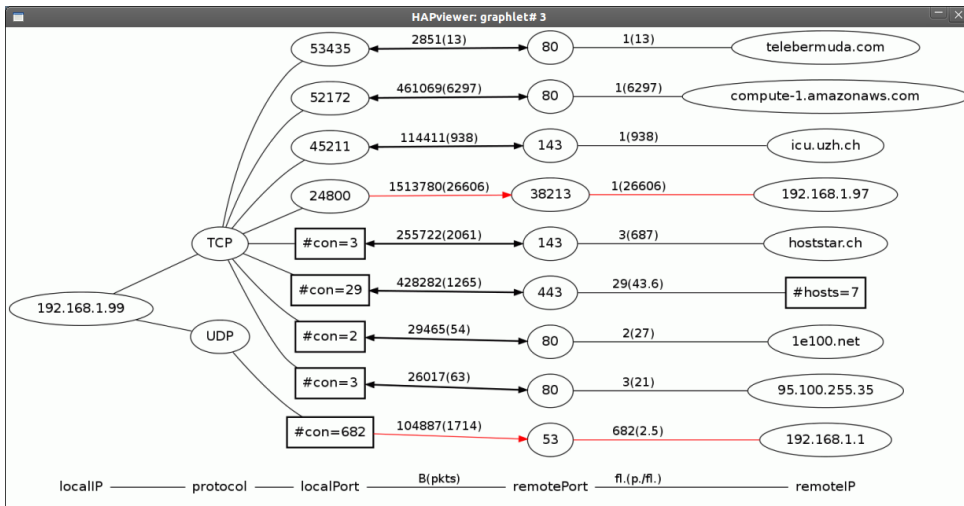


Figure D.2: Step 2 of Time Series With Offline Learning - Base Flows

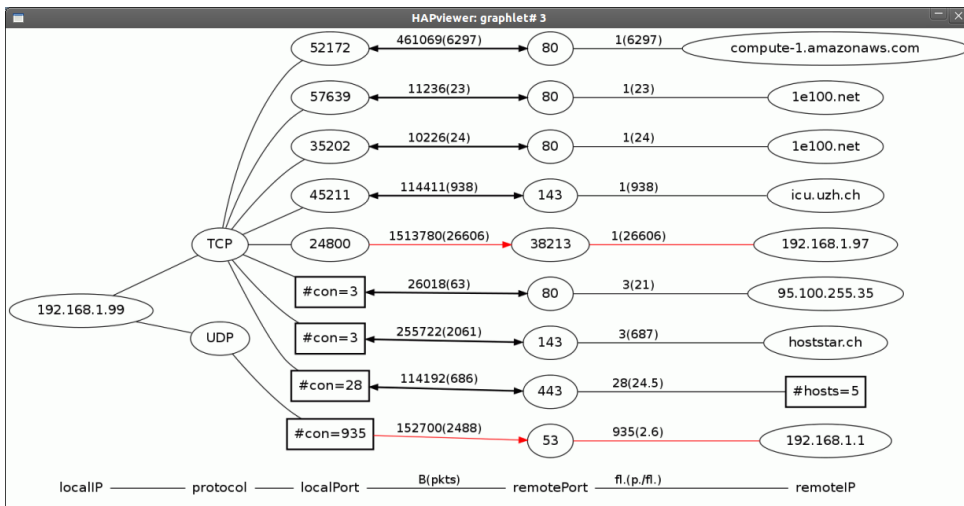


Figure D.3: Step 3 of Time Series With Offline Learning - Base Flows

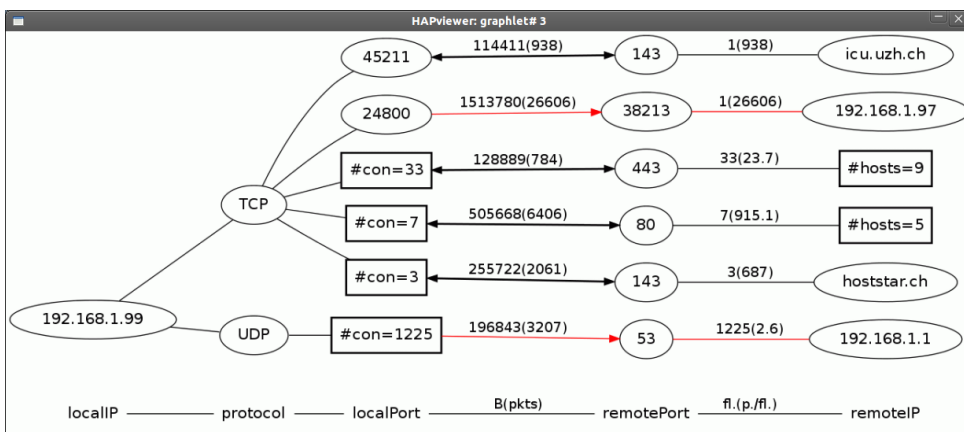


Figure D.4: Step 4 of Time Series With Offline Learning - Base Flows

## D.2 Base Flow Percentage Statistics Without Reverse DNS Lookup

This section contains the charts of the statistics generated applying the algorithm *algo01* on all three datasets *PD1*, *PD2*, and *GS1* without activating reverse DNS lookup. Figure D.5 shows the statistics for the dataset *PD1*, while figures D.6 and D.7 show the statistics for the datasets *PD2* and *GS1* respectively.

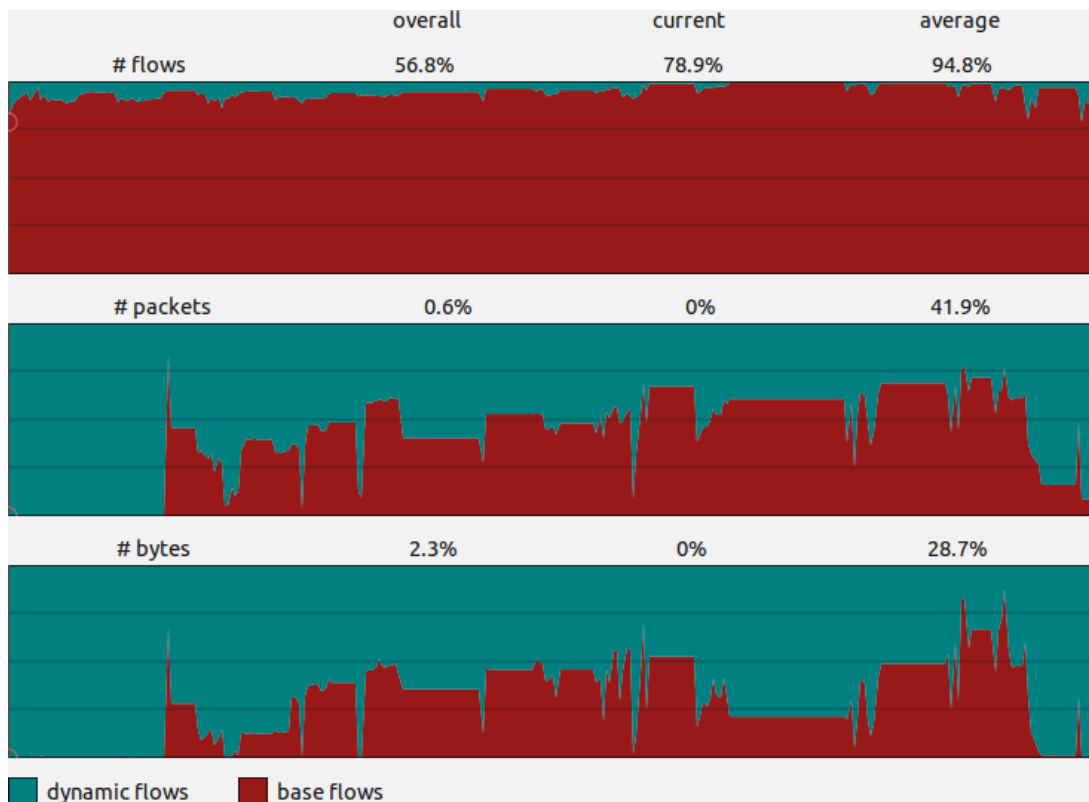
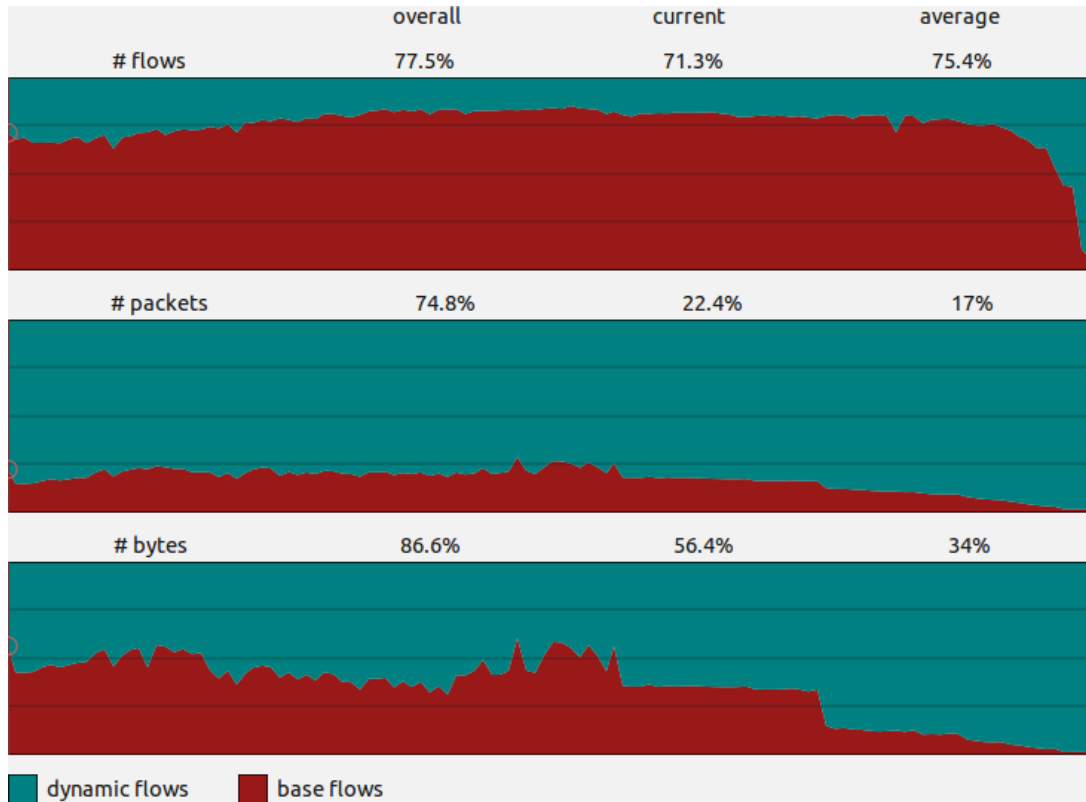
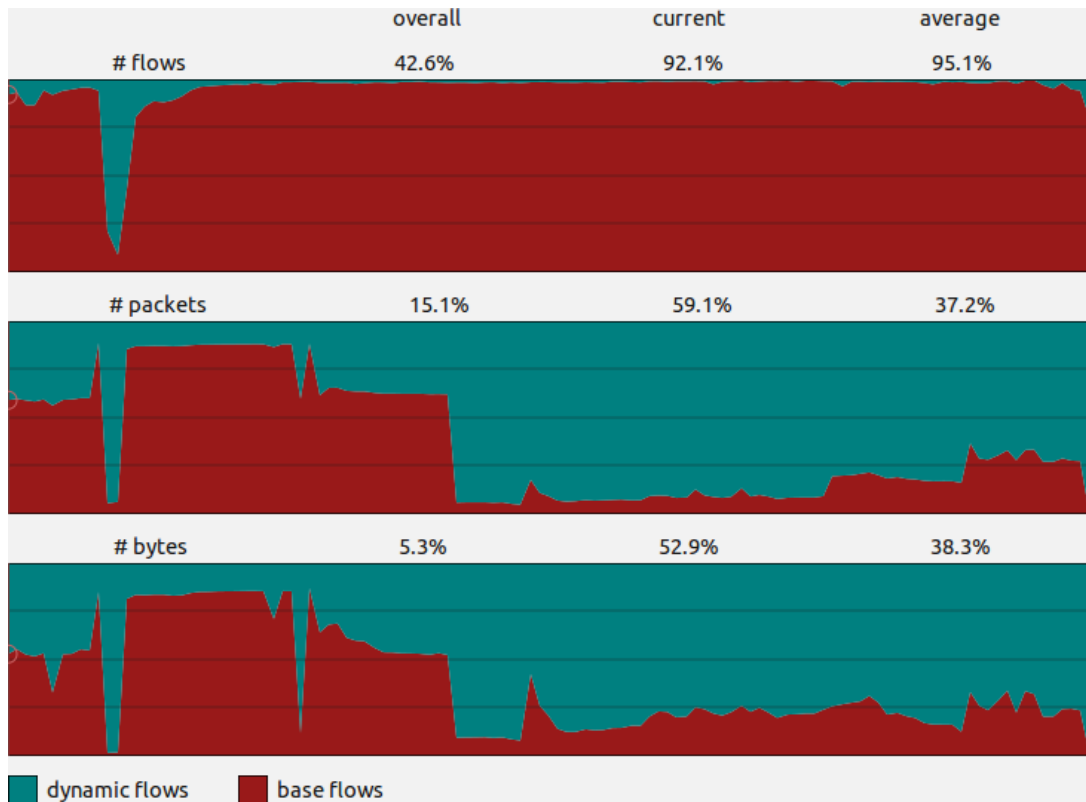


Figure D.5: *Base Flow Percentage of Dataset PD1 Without Reverse DNS Lookup*

Figure D.6: *Base Flow* Percentage of Dataset *PD2* Without Reverse DNS LookupFigure D.7: *Base Flow* Percentage of Dataset *GS1* Without Reverse DNS Lookup

# Appendix E

## Contents of the CD

This thesis in PDF and PS format and a short abstract in English and in German as a text file are located in the root directory of the appended CD.

In addition, the CD contains several folders, their content is described in the following sections.

### E.1 Source Code

The folder `hapviewer` contains the source code for the project. This includes files for `cmake`, for creating the documentation, as well as for the tool itself. The actual C++ files containing the classes described in appendix C together with the source files from the pre-thesis part of HAPviewer are located in the folder `hapviewer/src/`.

### E.2 Installation Files

Both scripts mentioned and described in appendix A are located in the folder `install`. In addition to the two scripts, the - partly modified - source of the required libraries `libfixbux`, `libpcap++`, and `GeoIP` are located in this folder.

For instructions on how to use the scripts located here and how installation should be done, please read appendix A.

### E.3 Report

This thesis has been written using  $\text{\LaTeX}$ <sup>1</sup>, using the `latex` command for compilation. All source files out of which this report is generated, as well as the figures included in this report in full resolution in PNG format can be found in the folder `report`.

---

<sup>1</sup><http://www.latex-project.org/>

## E.4 Data

The folder `data` contains the netflow files used for the evaluation in chapter 4. Each of the three netflow files contains the netflow data for a complete observation period.

In addition, the IP to domain name mappings acquired at a point in time close to the time of recording the netflow data itself is available in the folder. This mapping can be imported using the reverse DNS lookup feature of HAPviewer. Not only does importing the mapping guarantee equal results as given in chapter 4, as IP to domain name mapping may be different later in time, it also greatly increases the speed of calculation, as looking up all the mappings for days of netflow data can take up to hours.

File	Description
<code>nfcapd.gs1</code>	Netflow data collected from a server from April 21 to April 26, 2011
<code>gs1_mapping.csv</code>	IP to domain name mapping of all IP addresses in <code>nfcapd.gs1</code>
<code>nfcapd.pd1</code>	Netflow data collected from a server from April 21 to May 3, 2011
<code>pd1_mapping.csv</code>	IP to domain name mapping of all IP addresses in <code>nfcapd.pd1</code>
<code>nfcapd.pd2</code>	Netflow data collected from a server from August 6 to August 11, 2011
<code>pd2_mapping.csv</code>	IP to domain name mapping of all IP addresses in <code>nfcapd.pd2</code>

Table E.1: Files in Data Folder

Table E.1 lists the files provided in the folder `data` on the attached CD.

The files `nfcapd.pd1` and `nfcapd.pd2` have been recorded on the host with the IP 192.168.1.99, while `nfcapd.gs1` has been recorded on the host with the IP 192.168.1.97.

Please note that the file `nfcapd.pd2` contains two port scans conducted from a computer placed in the local network towards the end of the file and `nfcapd.gs1` has a bittorrent client running for some time early in the file.

## E.5 Papers

For further study, a digital copy of all the papers mentioned in this thesis can be found in the folder `papers`.

## **E.6 Documentation**

The folder `documentation` contains the doxygen documentation of the HAPviewer project, including all the extensions written for this thesis. The file `documentation/index.html` contains the main page from which the user can browse through the documentation in HTML format.