Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Bernhard Tellenbach

# Visualisation of Client/Server Behaviour

Semester Thesis SA-2004.24
April 2004 to July 2004

Tutor: Thomas Dübendorfer
Co-Tutor: Arno Wagner
Supervisor: Prof. Bernhard Plattner

## Abstract

This thesis is part of the DDoSVax project. The goal of this project is to detect Distributed Denial of Service (DDoS) attacks and Internet worms and to develop and initiate countermeasures. In this thesis, a model to characterise the behaviour of Internet hosts as well as the host classificator algorithm that uses this model are presented. Furthermore the tool VISTOOL that was developed to visualise and evaluate the output of the host classificator algorithm is shown. The VISTOOL was implemented as Common Gateway Interface(CGI) program. It provides besides the known XY visualisation plot, new plot methods like the Venn diagram plot and the port map plot. They were necessary to display the huge amount of data in a meaningful way. Since the goal of the characterisation and visualisation was to be able to detect anomalous activity in the Internet traffic, a validation of the developed tools with normal and anomalous Internet traffic is also part of this thesis. For the validation the Cisco Netflow v5 data generated by the border gateway routers of the Swiss Education and Research Network SWITCH could be used. About 5% of the Internet traffic in Switzerland and crossing Switzerland is routed by these routers. It is shown that with the developed host classificator algorithm and the visualisation tool VISTOOL, significant changes can be seen when comparing the plots from normal traffic and from anomalous traffic.

# Contents

**1 Introduction** **5**

1.1 Motivation . . . 5

1.2 Task Description . . . 5

1.3 Overview . . . 6

1.4 Related Work . . . 6

**2 Characterisation of Client/Server Behaviour** **7**

2.1 Server Characteristics . . . 8

2.2 Client Characteristics . . . 8

2.3 How to Identify a Host as a Client and/or Server . . . 9

2.3.1 Choosing Relevant Parameters for a Client/Server Classification . . . 9

2.3.2 Mapping Hosts to Classes . . . 10

2.4 Port Activity . . . 11

**3 Host Classification Algorithm and Data Structures** **13**

3.1 Software Restrictions . . . 13

3.2 Hardware Restrictions . . . 13

3.2.1 Memory Limitation . . . 14

3.2.2 Processing Time Limitation . . . 14

3.3 Data Structures and their Memory Usage . . . 14

3.3.1 Traffic and Connector Class . . . 14

3.3.2 Responder Class . . . 14

3.4 Port Activity . . . 16

3.5 Memory usage calculation . . . 18

3.6 Host Classificator Algorithm . . . 18

3.6.1 Responder Classification . . . 22

3.6.2 Port Activity Analysis . . . 22

3.6.3 Possible Loss of Information . . . 24

3.7 Statistic File Format and the Directory Structure . . . 24

**4 Visualisation** **27**

4.1 Global Settings . . . 27

4.2 Host Classification . . . 28

4.2.1 Venn Diagram . . . 28

4.2.2 X-t Plot . . . 28

4.3 Port Activity . . . 28

**5 Validation** **33**

5.1 Memory Usage and Performance Analysis . . . 33

5.2 Normal Traffic . . . 34

5.3 W32.Blaster Traffic . . . 34

5.3.1 Comparison of the Results . . . 34

**6 Conclusion and Outlook** **47**
6.1 Conclusion . . . . . 47
6.2 Outlook . . . . . 47
6.2.1 Using the Host Classificator Plug-in Output for Anomaly Detection . . . . 47
6.2.2 Further Use of the Connection Tracking Capability . . . . . 48
6.3 Acknowledgements . . . . . 48

**A Appendix** **49**
A.1 The Converter Tool . . . . . 49
A.2 Host classificator Readme.txt . . . . . 49
A.3 VISTOOL Readme.txt . . . . . 50
A.4 Provided Files . . . . . 51

# Chapter 1

# Introduction

## 1.1 Motivation

In the year 2000, when some of the worlds leading dotcom companies suffered a severe economical damage caused by distributed denial of service (DDoS) attacks, the red light was seen by every businessman or businesswoman responsible for clients or servers connected to the Internet. Despite the fact that this problem was already known to some non-commercial sites like the ETH since they were attacked before, the private sector of the economy had almost no interest in basic research with the goal to develop detection algorithms and countermeasures. This interest was necessary to get the support to carry out projects like DDoSVax[1].

**DDoSVax project** A cooperation of the ETH with the operator of the Swiss Education and Research Network SWITCH[2], gave the researchers at ETH for the first time access to enough flow-level Internet traffic data to make a scientific analysis of if and how an identification and analysis of DDoS attacks is possible with the long-term objective to be able to initiate countermeasures. The traffic data is collected by four border gateway routers operated by SWITCH using Cisco Netflow V5 [1]. Unfortunately only about 5% of the Internet traffic inside or crossing Switzerland is routed by SWITCH, furthermore Cisco Netflow v5 records contain only few information about each detected connection[3] so that the detection of a DDoS attacks is non trivial. Altogether the data provided by SWITCH permitted the development of some sophisticated algorithms and applications to detect and visualise anomalous network behaviour.

**This thesis** Since Internet worms can be used to distribute software with which the infected hosts could be misused for a DDoS attack, it is obvious, that the detection of a worm, spreading over the Internet, is useful. What makes a detection difficult is the wide variety of distribution methods causing different anomalies when different worms are spreading. To detect and visualise these anomalies by characterising and evaluating server vs. client behaviour is therefore a challenging task and at the same time the motivation for this thesis. The focus of this thesis is laid on the characterisation and not on the detection.

## 1.2 Task Description

After characterising server vs. client behaviour of Internet hosts, a new type of graphical plot will be developed and implemented to show the current distribution of client and server behaviour in the Internet. A second type of graphical plot will be used for visualising how the behaviour of Internet hosts changes over time. For both plots, measurement parameters, efficient algorithms

[1]See http://www.tik.ee.ethz.ch/∼ddosvax/

[2]See http://www.switch.ch/

[3]a connection in the sense of Cisco Netflow v5 is identified by protocol, source IP address, destination IP address, source port and destination port and a few other parameters. It contains only data about the data flowing from the source to the destination. For more details see [1]

and visualisation plots will be conceived. The task is split in three major subtasks: Specification, Implementation, and Validation.

**Specification** Efficient aggregation algorithms and meaningful visualisation plots for the current state of observed hosts and for showing the change of host states over time have to be specified. A rough estimation on the needed processing power will have to be made. If performance will be a problem, a simple online and a more detailed but slower offline algorithm might be considered.

**Implementation** The goal is to generate the new visualisation plots for client/server behaviour by a plug-in for UPFrame[4] and to make the plots accessible on the web. In addition, characteristic parameter values should be made available for further processing by anomaly detection tools.

**Validation** As a validation of the usefulness of the plots and the efficiency of the algorithms, the archived NetFlow data prior and during the Blaster worm of August 2003 will be used. The plots should show significant visible changes during the outbreak of the worm.

## 1.3 Overview

The structure of the thesis is the following: First of all, in Chapter 2 criteria to identify when a host acts as a server and when it acts as a client are defined. The question, if the Cisco Netflow v5 data provides enough information to take an unambiguous decision, is discussed. Chapter 3 specifies the data structures and the host classificator algorithm that were developed to classify the host behaviour according to the criteria described in Chapter 2. Afterwards the visualisation plots are presented in Chapter 4 while the algorithm of the UPFrame plug-in and the visualisation software with the data prior and during the Blaster worm outbreak of August 2003 is validated in Chapter 5. Finally this document ends with Chapter 6 where proposals for the further development of the algorithm and the visualisation part are made.

## 1.4 Related Work

The paper 'Network Flow-Based Anomaly Detection of DDoS Attacks' [8] by Georgios Androulidakis et al. presented at the TERENA Networking Conference 2004 in Rhodes, Greece describes an algorithm that is used to detect DDoS attacks based on network traffic in the form of Cisco Netflow. Despite the fact that this thesis's topic is not the detection of DDoS attacks but of anomalous behaviour of Internet hosts, some similarities with the paper are obvious: Both define characteristics based on which an analysis could be made and both use Cisco Netflow data for the analysis.
The paper 'Correlation between NetFlow System and Network Views for Intrusion Detection' [9] was published in the time this thesis was in progress and therefore only later seen by the author. Parameters for port activity are similar to this thesis but the visualisation method is different. Another difference is, that the focus of that paper is laid onto inspection of every single host, but that of this thesis onto visualisation of the behaviour of the hosts of the entire network under observation. Nevertheless the paper could have been inspiring if published earlier.

[4]see http://www.tik.ee.ethz.ch/∼ddosvax/upframe/

# Chapter 2

# Characterisation of Client/Server Behaviour

This chapter describes client and server characteristics useful for deciding which class a host belongs to. To characterise a host, a definition of both terms, server and client, has to be given. In the World Wide Web (WWW) several definitions of the term 'server' can be found:

> A network device that provides service to the network users by managing shared resources. Note 1: The term is often used in the context of a client-server architecture on a Local Area Network(LAN). Note 2: Examples are a printer server and a file server. *'Server'. Telecom Glossary 2000. Vers.T1.523-2001. American National Standard for Telecommunications. 29.06.2004 <http://www.atis.org/tg2k>*
>
> A computer or device on a network that manages network resources. For example a file server is a computer or a storage device dedicated to storing files. Any user on the network can store files on the server. A print server is a computer that manages one or more printers, an a network server is a computer that manages network traffic. A database server is a computer system that processes database queries. Servers are often dedicated, meaning that they perform no other tasks besides their server tasks. On multiprocessing operating systems, however, a single computer can execute several programs at once. A server in this case could refer to the program that is managing resources rather than the the entire computer. *'Server'. Webopedia. 29.06.2004 <http://webopedia.com/TERM/server.html>*

Because the focus of this thesis is on Internet traffic, only servers using the Internet as communication medium are of interest. Since the first and second definition imply that there are many different use cases for servers, a characterisation for every use case seems impossible. An example of a use case is a FTP Server. In this case the server copies one or more files between two computers providing file organisation as well as transfer control. A characterisation on a higher abstraction level is a possible solution to that problem and is discussed in Section 2.1. For the technical term 'client' several definitions exist on the WWW:

> A client is:
>
> - a computer that needs access to a server to use applications or documents
> - a program that uses a server application to access a service Example: The client offers an interface for entering, manipulating and querying data but the processing of the request is done by a server application. In the Internet a client is a program that provides all rules and methods to access a specific service[1]. *'Client'. Net-Lexikon. Vers.Beta 0.71. 10.10.2003. 29.06.2004 <http://www.net-lexikon.de/Client.html>*

Due to the wide variety of services offered by servers, the variety of clients is enormous and can be characterised best on a higher abstraction level than the service level. Possible characteristics are discussed in Section 2.2 whereas in Section 2.3 it is explained which characteristics

[1]services like FTP,E-mail. . .

were selected and why they were selected for the use with the classification algorithm presented in Section 3.

## 2.1 Server Characteristics

Since Netflow v5 does not include the content of the IP packets in a flow and since a service can normally only be identified if their content is known, a characterisation based on the kind of service a server provides is impossible However, even if the content of the IP packets were known, such a characterisation is difficult. The problem is, that the services used over an encrypted link can not be detected and that there exist simply too many different services. Therefore other characteristics than the type of service a server provides are needed. Possible characteristics are:

A server. . .

- responds to connection requests.
- does not initiate connections to clients.
- receives a request with few data and sends a response with lots of data.
- is contacted by many different hosts.
- has a static IP address.
- offers a service on a well-known port which identifies at the same time the type of service
- uses a well-known port as source port
- generates traffic in more or less regular intervals

The list will be compared to the client characteristics in Section 2.3, where the resulting classification is presented too.

## 2.2 Client Characteristics

A list of client characteristics is provided here:

A client. . .

- initiates connections.
- does not respond to connection requests.
- sends a request with few data and gets a response with lots of data.
- contacts only few other hosts.
- often has a dynamic IP address.
- uses unprivileged ports as source port for its communication.
- communicates almost always with a well-known destination port
- generates few traffic in sporadic intervals

It will be compared to the list of server characteristics in Section 2.3

## 2.3 How to Identify a Host as a Client and/or Server

The lists of characteristics for a client and a server given in the two previous sections provide some information about how a host can be identified as a client or a server. Nevertheless the identification is not that easy, since the exception proves the rule and unfortunately some or even a lot of exceptions exist for every mentioned characteristic. One of the most important exceptions to the property that a client does only respond to connection requests, but does not answer them - a server acts vice versa - , is FTP in active mode[2]. If every TCP connection is observed for itself, the server seems to initiate connections without a previous request. Other examples, e.g. for the property of a server only to send data with a source port in the range of the well known ports identifying at the same time the service requested, exist. Passive FTP[3] where the request is not sent to the same port as the requested data will be received from later is such an example. Furthermore two special kinds of hosts have to be taken into consideration: the hosts of a P2P network and the hosts participating in an Internet Relay Chat(IRC). For those the characteristics listed in Section 2.2 and 2.1 are, if at all, only valid for some time and therefore do not qualify a host as a server or a client but rather as behaving server-like or client-like. But since research about P2P networks or IRC is the topic of other theses like [3] or [4], no special characteristics to identify such hosts were defined. A not yet mentioned problem is what the second definition of the term server in Section 2.1 implies, namely that a host can be a server, a client or both at the same time, only depending on the software running on it.

### 2.3.1 Choosing Relevant Parameters for a Client/Server Classification

The remarks in Section 2.3 imply, that a classification of a host as a server using the listed characteristics, can be ambiguous and therefore is not very useful. A more promising approach is to select characteristics from the lists in Section 2.1 and 2.2 and to decide for each of them if a host matches it or not. An overview of the behaviour of hosts in the Internet can then be given by observing how many hosts match a certain characteristic. The following parameters were deduced from the lists and will be used to characterise a host:

1. the number of connections
2. the amount of data sent and received
3. the number of bidirectional connections a host responds to
4. port activity of a host

It remains to justify the choice of the four characteristics, taken into consideration that they have to be useful for the DDoSVax project where anomaly detection has the highest priority, while gathering information about the distribution of e.g. file servers or http servers, is less important.
The *first* characteristic, the number of unidirectional connections a host opens to other hosts within a certain timespan, was chosen because a host with many of these connections is on the one hand probably a host providing service, but on the other hand, it could be a member of a peer-to-peer network or a host infected by a worm currently trying to infect other hosts.
The *second* characteristic can be used on the one hand to detect hosts that produce a lot of traffic, but on the other hand if the amount of data sent is greater than the amount of data received this host probably acts, as discussed in Section 2.3, as server.
The *third* item in the list is probably the most important characteristic for anomaly detection since most worms turn a host, that was beforehand only initiating connections, into a connection accepting host [5].
The *fourth* item is valuable for behaviour characterisation and anomaly detection if kept in mind that worms or zombies[4] communicate via certain ports or within certain portranges. If multiple hosts show suddenly activity on a port they didn't use before, it can be an indication for an anomalous behaviour and needs further investigation.
Finally, hosts that match the same characteristics are grouped and each group is identified by a class name. The mapping of hosts to classes, using the first three characteristics in the list,

[2]see http://slacksite.com/other/ftp.html,for details
[3]see http://slacksite.com/other/ftp.html#passive for details
[4]hosts that can be remote controlled e.g. to use them for spaming or DDoS attacks

is described in the following section. Why the fourth characteristic is not used as criteria for the classification, but is treated separately, is also explained there.

### 2.3.2 Mapping Hosts to Classes

Each host can match none, one or multiple of the first three characteristics listed in the preceding section, hence it is necessary to define a class for each possible combination. The class diagram in Figure 2.1 shows the resulting classes. The fourth characteristic, the port activity of a host, does not get it's own basic class since it would only be reasonable to do so if the basic class contains few sub classes. Unfortunately this would require a reduction of the whole port information to a few parameters, e.g. to a counter that counts each port on which a host showed for the first time activity, whose significance for anomaly detection were very questionable. As a consequence there is no basic class for the fourth characteristic but a separate analysis tool that is presented in detail in Section 4.3.
To continue, the requirements of the three basic classes are briefly explained.

**Traffic Class:** To be a member of this class a host has to fulfil the following requirement:

$$\frac{\text{data sent}}{\text{data received}} > \text{traffic class threshold} \tag{2.1}$$

Since Cisco Netflow v5 provides for each flow only information about the amount of network layer data sent by the source, the parameters 'data sent' and 'data received' represent the layer three data sent and received. It is possible that the actual amount of layer three data and the measured amount differ because the input filter described in Section 3.6 possibly[5] filters out some traffic.

**Connector Class:** To be a member of this class, a host has to fulfil the following requirement:

$$\text{number of outgoing connections} > \text{connector class threshold} \tag{2.2}$$

Only flows with the hosts IP address as source address are counted and their number is stored for each host. The reason for it is, that a host that scans certain IP address ranges would otherwise create many probably nonexisting connector class hosts. To count the flows where a host appears as source can be problematic too if IP address spoofing is present. IP address spoofing can falsify the result by counting connections for a host that is actually not responsible for it. But because IP address spoofing is not the rule but rather an exception and IP address spoofing can anyway not be detected except in special cases[6], it is neglected.

**Responder Class:** To be a member of this class a host has to fulfil the following requirement:

$$\text{number of bidirectional connections as responder} > \text{responder class threshold} \tag{2.3}$$

Since a Netflow v5 flow contains only information about one direction of a bidirectional connection, it is in a first step necessary to determine corresponding flows. To get in a second step the number of bidirectional connections as responder, it is necessary to count only those flow pairs, where the host is not the initiator[7]. A flow pair consists of a flow of which host H1[8] is the source and host H2 the destination and of a flow of which host H2 is the source and host H1 the destination.

[5]depending on the filter parameters
[6]e.g. if all existing hosts of a network are known and a packet with an IP address of a not existing host as source address appears, it has to be spoofed
[7]the criteria used to decide if a host is a responder is explained in Section 3.6.1
[8]identified by it's IP address and port number

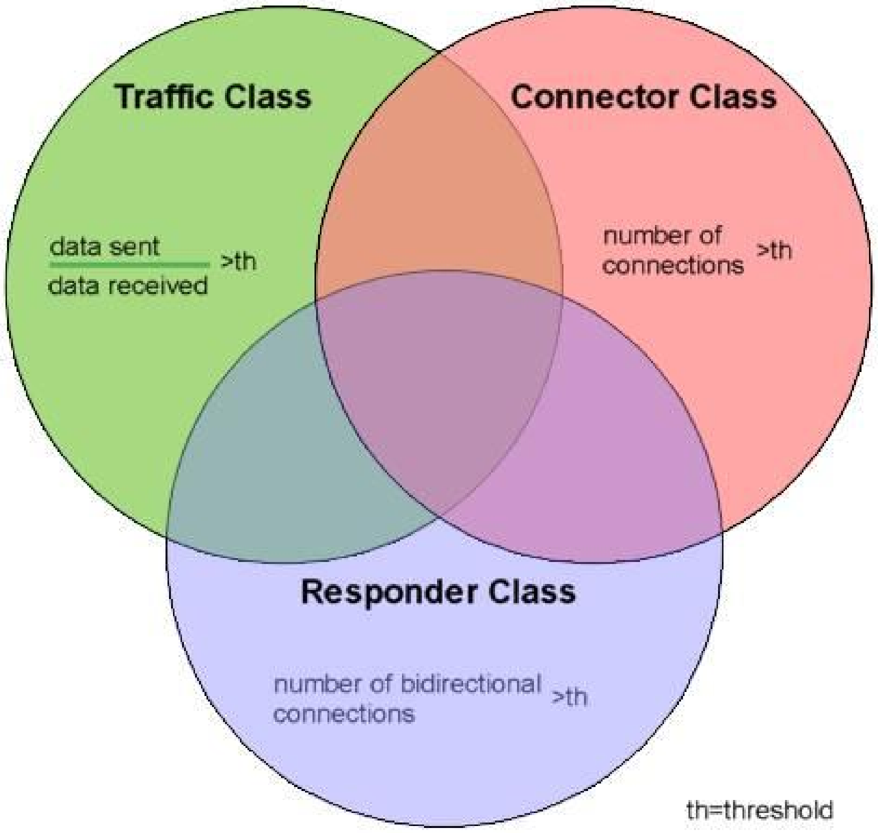

Traffic Class
data sent / data received >th
Connector Class
number of connections >th
Responder Class
number of bidirectional connections >th
th=threshold

Figure 2.1: Class diagramm with requirements for each class

**No Class:** A host that is not a member of any of the other classes, belongs to this class.

To sum it up it is not possible to characterise a host unambiguous as a server or a client but it is reasonable to decide if a host behaves rather server like or client like for a specific characteristic. To be a member of one or more of the specified classes, a host has to show rather server like behaviour for theses characteristics. As the comments in Section 2.3 showed, behaving server like does not imply, that the host is actually a server.

## 2.4 Port Activity

If the source and the destination ports used by a host are known and if only the well known ports are considered, the type of service a host uses or provides can be supposed. Furthermore the probability that the assumption is correct, is very high, since for each well known port the corresponding service is defined. It follows, that counting the hosts using a specific port would give a good overview of the behaviour of hosts in the Internet, but since it provides not much more information than already existing analysis tools that count all flows for a specific port, the following approach was chosen; Each port used by a host in the current interval is logged and afterwards compared to the ports used in the next interval. Those ports that were not used before are then reported. Hence a HTTP server showing continuous activity, would ideally only be reported once. Why the ideal case is unrealisable in a near real-time application and what the algorithm actually does, will be explained in Section 3.6.2.

# Chapter 3

# Host Classification Algorithm and Data Structures

After the characterisation of the client/server behaviour of Internet hosts in Section 2 it remains to extract the information needed for a characterisation from the Cisco Netflow v5 data. Since the characterisation has to be done in real time, it is necessary to specify memory saving data structures and an algorithm that processes the data on average faster than it is coming in. In it the term ’record’ is used for the first time. It stands for the Cisco Netflow v5 data structures contained in the UDP packets provided by UPFrame, representing the information about the flows. UPFrame[1] is an application framework that is able to receive and process incoming UDP packets at fast rates, buffer several megabytes of incoming data to smoothen out data bursts and feed the received packets to plugins that independently process the data in the packets.
The specification process was rather challenging since the amount of flows in a certain timespan can increase dramatically, e.g. during a worm outbreak. Only repeated testing and modification of the algorithm and data structures allowed to finally meet the memory and processing time limitations. The test results of the final version of the algorithm, using the flow data generated during the W32.Blaster worm outbreak in August 2003, can be found in Section 5.1.

## 3.1 Software Restrictions

The only software restriction is that UPFrame with it’s plug-ins can only be used on a computer running Linux or UNIX without adapting the code for the shared memory handling.

## 3.2 Hardware Restrictions

It is very important to meet the hardware restrictions since they assure that the plug-in does not need separate hardware for execution but that it runs together with UPFrame and other plug-ins on the same computer. The restrictions limit the memory available to plug-ins and processing time. It is important to know that several UPFrames running on different computers can be chained so that every UPFrame receives the same Netflow data. Unfortunately the header data with the IP address of the sender of the UDP packet is changed when the packet is transmitted from UPFrame to UPFrame. It follows, that to get the IP address of the router that sent a specific UDP packet, the computer running UPFrame has to be the first link in the chain of frameworks. A different solution would be to fake the sender IP address when transmitting it, but since to fake the source IP address, UPFrame has to be run as ’root’, this is not an option. To run an application as ’root’ is too risky. If e.g. it contains a security hole that allows the execution of some commands, these commands are executed with the rights of the administrator.
For the following considerations it is assumed, that UPFrame runs on a computer fast enough to avoid that flow data are lost and fast enough to allow at the same time the execution of the plug-in.

[1]see http://www.tik.ee.ethz.ch/∼ddosvax/upframe/ for details

### 3.2.1 Memory Limitation

The goal of a memory limit is to avoid failures due to a lack of memory and to be able to run UPFrame with several plug-ins on the same machine. The key to keep the memory usage low is to design data structures that use as little memory as possible and contain only information that can not be derived from already stored data. Another possibility to keep it low is to filter[2] data not adding to the significance of the analysis. To stay below the limit of 150 MBytes even while processing the W32.Blaster data of August 2003, the data structures and the filter method were adapted several times during the development process. It may be astonishing that not all adaptions reduced the memory usage, but if the processing time limitation discussed in the following section is taken into consideration too, the reason becomes clear.

### 3.2.2 Processing Time Limitation

To comply with the requirement to use not more than a certain amount of processor time, it is desirable to have an algorithm whose complexity is linear in the number of flows and unique IP addresses since the amount of flows can easily increase by a factor of ten and more when e.g. a new worm is spreading over the Internet. Hence if the complexity of the algorithm is $O(n^2)$ rather than $O(n)$ and if a processor load of 1% is assumed when processing normal traffic, the processor load is already 100% when the traffic increases only by a factor of ten. It follows that to use more main memory can be reasonable if with the additional memory, the complexity in the number of flows of the algorithm can be changed from $O(n^2)$ to $O(n)$.

## 3.3 Data Structures and their Memory Usage

First of all the data needed to classify each host according to the criteria in Section 2.3 has to be identified. But since Section 3.6.1 gives a detailed description of the classification algorithm and the reason for some data fields become obvious only there, the focus of this section is set on the used data structures and their memory usage.

### 3.3.1 Traffic and Connector Class

To check if a host fulfils the requirements given in Section 2.3 the first six fields of the hostinfo data structure in Table 3.5 are sufficient provided that the data for a host is updated immediately after a flow with the IP address of that host as source or destination, has arrived. Actually only the first three fields are necessary for the check but because a certain lookahead for the analysis of an interval is needed[3], a distinction between flows valid for the current and the next interval has to be possible. Hence another three fields are needed to store the information for the next interval. The memory usage of the hostinfo data structure is analysed in Section 3.4, after which the entire hostinfo structure has been discussed.

### 3.3.2 Responder Class

To determine the number of bidirectional connections a host responds to only the Netflow data is available. Since a flow represents only one direction of a bidirectional connection, this data should be organised in a way that corresponding unidirectional connections can be found with complexity 1. This is important because for each unidirectional connection a search for the corresponding connection in the other direction has to be performed and therefore the total complexity is already $O(n)$. If no hash table was used at all, the complexity of a search would be $O(n \times log(n))$ and doing the search n times would result in a complexity of $(n^2 \times log(n))$. To achieve a complexity of 1, the nested hash table structure shown in Figure 3.1 is used.
The structure of the keys and elements used with the nested hash table are described in the following tables:

[2]Section 3.6 describes the used filtering method
[3]the reason for this is explained in Section 3.6

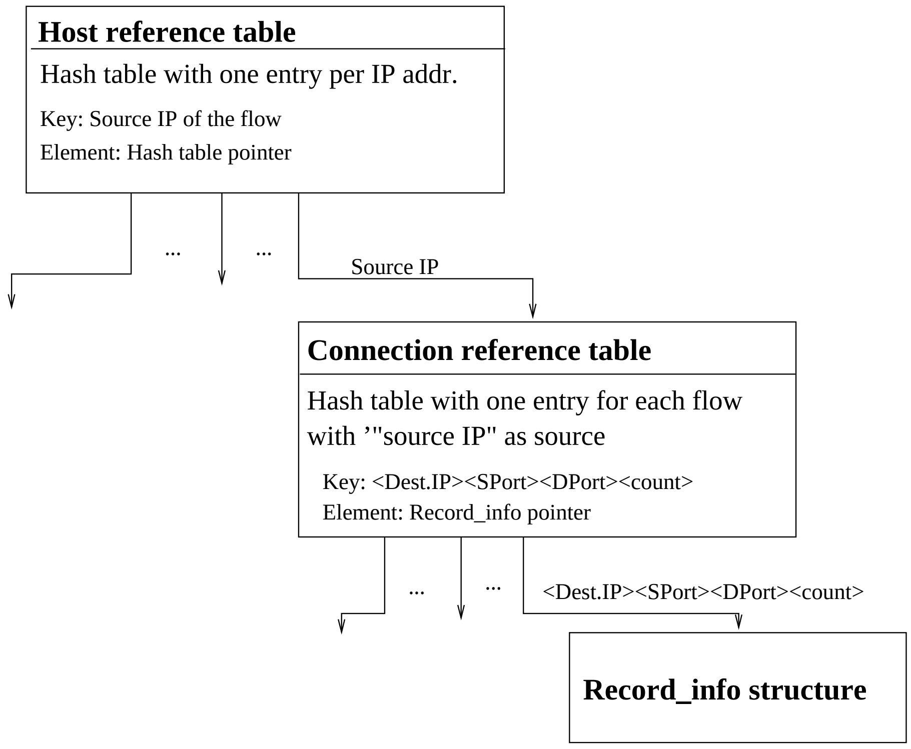

Host reference table
Hash table with one entry per IP addr.
Key: Source IP of the flow
Element: Hash table pointer
...
...
Source IP
Connection reference table
Hash table with one entry for each flow with ’"source IP" as source
Key: <Dest.IP><SPort><DPort><count>
Element: Record_info pointer
...
...
<Dest.IP><SPort><DPort><count>
Record_info structure

Figure 3.1: Data structure to store the required data of each record

| Host reference table entry | | |
|---|---|---|
| name | format | size [bit] |
| Element bucket | struct hashed_table_bucket | 128 |
| Key structure | uint32 | 32 |
| Data structure referenced by an element | struct hashed_table | 1120 |
| | | Total: 160 Bytes |

Table 3.1: Used memory for a host reference table entry

| Connection reference table entry | | |
|---|---|---|
| name | format | size [bit] |
| Element bucket | struct hashed_table_bucket | 128 |
| Key structure | (see table 3.3 | 80 |
| Data structure referenced by an element | record_info | 144 |
| | | Total: 44 Bytes |

Table 3.2: Used memory for a connection reference table entry

| Connection reference table key | | |
|---|---|---|
| name | format | size [bit] |
| Dest. IP | uint32 | 32 |
| Source port | uint16 | 16 |
| Dest. port | uint16 | 16 |
| Conn. counter | uint16 | 16 |
| | | Total: 10 Bytes |

Table 3.3: Structure of the keys used with the connection reference table

| struct record_info | | |
|---|---|---|
| name | format | size [bit] |
| protocol | uint16 | 16 |
| num_of_packets | uint32 | 32 |
| start_time | long long uint | 64 |
| duration | uint32 | 32 |
| | | Total: 18 Bytes |

Table 3.4: The record_info structure

The memory usage of the nested hash table structure can be calculated with the following equation:

$$\begin{aligned} m &= \text{number of records} \\ n &= \text{number of unique source IP's in the table} \\ e_{href} &= \text{size of a host reference table entry} \\ e_{cref} &= \text{size of a connection reference table entry} \end{aligned}$$

$$\begin{aligned} e_{href} \times n + e_{cref} \times m &= \text{used memory in Bytes} \\ 160 \times n + 44 \times m &= \end{aligned} \tag{3.1}$$

If taken into consideration that the used hash table has more slots for elements than are actually filled, and that every slot uses 16 Bytes of memory for a structure that is able to take up an element the Equation 3.1 has to be modified. To calculate the memory usage with the modified equation, the number of elements in each table as much as the load factor must be known. The resulting equation is:

$$\begin{aligned} l &= \text{load factor of host reference table ]0...1]} \\ l_j &= \text{load factor of connection reference table ]0...1]} \\ e_j &= \text{number of elements in table j} \\ m &= \sum_{j=1}^{n} e_j \end{aligned}$$

Used memory=

$$\begin{aligned} &\sum_{j=1}^{n}\left(e_{href} + \sum_{i=1}^{e_j}\left(e_{cref} + \left(\frac{e_j}{l_j} - e_j\right) \times 16\right)\right) + \left(\frac{n}{l} - n\right) \times 16 \\ &= \sum_{j=1}^{n}\left(160 + \sum_{i=1}^{e_j}\left(44 + \left(\frac{e_j}{l_j} - e_j\right) \times 16\right)\right) + \left(\frac{n}{l} - n\right) \times 16 \end{aligned} \tag{3.2}$$

## 3.4 Port Activity

The fundamental idea of the port activity analysis is to report source ports that a host did not use before but just started doing so. A host did not use a port before, if this port did not appear as source port in Netflow data, where the field source IP address contained the IP address of this host for some time. To perform the port activity analysis, all these ports have to be stored for each host. Furthermore a distinction between their usage as UDP or TCP port has to be made. Unfortunately the memory that the host classificator plug-in is allowed to use, is limited and not all but only a limited number of ports per host can be stored. The decision to provide eight slots to store port numbers for each of the categories privileged UDP ports/unprivileged UDP ports and privileged TCP ports/unprivileged TCP ports, proved to be well since at least for the privileged category almost in any case all the ports meeting the requirement could be

logged. Further details about the number of missed ports can be found in Chapter 5.
Please refer to Section 2.3 for an explanation why the first six fields of the hostinfo data structure are necessary to check if a host matches the requirement for the traffic and/or connector class. In addition to these six fields there is a field for storing the current class and activity code[4] and a pointer to a port_info data structure. The port_info structure is accessible via a pointer in the host_info data structure and contains the data necessary to analyse the port activity of a host. The reason for using a pointer to another data structure is, that by allocating memory for the port_info structure only in case a host appears as source[5] of a flow, a lot of memory can be saved.

| Host_info data structure | | |
|---|---|---|
| Data type | Name | Total size [bit] |
| unsignet int | data_sent | 32 |
| unsignet int | data_sent_next | 32 |
| unsignet int | data_received | 32 |
| unsignet int | data_received_next | 32 |
| unsignet int | num_connections | 32 |
| unsignet int | num_connections_next | 32 |
| port_info* | portinfo | 32 |
| char | activity_and_class | 8 |
| | | Total: 29 Bytes |

Table 3.5: Host_info data structure

| Port_info data structure | | |
|---|---|---|
| Data type | Name | Total size [bit] |
| char[8] | first_TCP_ports_priv | 8x16 |
| char[8] | first_TCP_ports_unpriv | 8x16 |
| char[8] | first_UDP_ports_priv | 8x16 |
| char[8] | first_UDP_ports_unpriv | 8x16 |
| unsigned short int | not_first_TCP_new_privcounter | 16 |
| unsigned short int | not_first_TCP_new_unprivcounter | 16 |
| unsigned short int | not_first_UDP_new_privcounter | 16 |
| unsigned short int | not_first_UDP_new_unprivcounter | 16 |
| | | Total: 72 Bytes |

Table 3.6: Portinfo data structure

The host_info structures are stored in a hash table using the IP as key. Since there is an administration overhead of 16 Bytes per entry, the total main memory used by the host information hash table is:

$$\begin{aligned} h_{tot} &= \text{number of unique IP's} \\ h_{act} &= \text{number of active hosts (appear as source of a flow)} \\ l &= \text{load factor of the hash table ]0...1]} \end{aligned}$$

$$\begin{aligned} \text{used memory} &= h_{tot} \times (29 + 16 + 4) + h_{act} \times (72) + \left(\frac{h_{tot}}{l} - h_{tot}\right) \times 16 \\ &= h_{tot} \times 49 + h_{act} \times (72) + \left(\frac{h_{tot}}{l} - h_{tot}\right) \times 16 \qquad (3.3) \end{aligned}$$

Equation 3.1 and 3.3 are important for the soft memory limit of the host classificator plug-in introduced in Section 3.5.

[4]see Section 3.6 for details
[5]Only the source ports in flows where the host appears as source are stored. See Section 3.6.2 for details.

## 3.5 Memory usage calculation

Now that equations to calculate the memory usage of the different data structures were presented, it is shown why they are not suitable for realising a fast algorithm, and what functionality to limit the memory is used instead.

To limit the memory usage of the algorithm a soft limit was included in the algorithm. It is called a soft limit because the memory currently used by the plug-in is calculated and not checked via an operating system command. The reason for calculating the amount of used memory is, that a simple calculation is faster than reading the used memory e.g. from the /proc filesystem. Even though it would be possible to check the used memory less frequent, what would allow to read the amount of used memory form the /proc filesystem, this is no real option since the used memory can easily increase by 10 MBytes or more per half a second.
The algorithm that limits the used memory works like that: The used memory is calculated after fetching a new flow and if the limit is reached no more flows are accepted. Additionally the status bit for reaching the memory limit is set. Afterwards it is still checked if an advancement to the next interval and an hence an evaluation of the current interval is necessary. Because after the evaluation some memory is freed, the status bit for reaching the memory limit is reset and flows are accepted again.
To calculate the memory needed by the different data structures, the in Section 3.3 provided equations could be used. Unfortunately on the one hand not only the data structures use memory but also their administration and on the other hand some parameters of Equation 3.2 have to be calculated also. In addition some memory is only allocated during the evaluation phase and it is not known in the acquisition phase how much that will be. To avoid using more memory than specified, it is assumed that all memory that could be allocated, will be allocated. Considering all this, the calculations in Section 3.3 can only serve as start point of search for an equation approximating the actually used memory. The equation was adapted until the result of the calculation corresponded well enough to the value reported by the /proc system. Well enough means, that though the result of the calculation can indicate more memory than is actually used, the opposite should never be the case. The equation used is the following:

$$\begin{aligned} n &= \text{total number of unique source IP addresses in the flows} \\ m &= \text{total number of acquired flows for the current and the next interval} \\ h &= \text{total number of hosts in the host information hash table} \end{aligned}$$

$$\text{used memory in MBytes} = (n * 160 + m * 84 + hostcounter * 68 + h * 72)/1048576 + 6 \quad (3.4)$$

The additional 6 MBytes in Equation 3.4 stand for the amount of memory used by the host classificator plug-in when no data is processed. If Equation 3.1 and 3.3 are summed up without considering the memory used for empty elements and assumed that for all host information structures a port information structure is allocated, the following equation is the result:

$$\text{used memory in MBytes} = (n * 160 + m * 44 + hostcounter * 49 + h * 72)/1048576 \quad (3.5)$$

## 3.6 Host Classificator Algorithm

The host classificator algorithm, one of the core pieces of this thesis, is now presented with the focus on data processing. Furthermore it is explicitly shown where, depending on the parameters, eventually some Netflow data is discarded. Figure 3.2 shows the relevant stages of the algorithm. It follows a description of each block's functionality.

**Reading Record/Header** The first step of the acquisition phase is to check the UDP packet received from UPFrame. The source IP address listed in the header of the packet is extracted

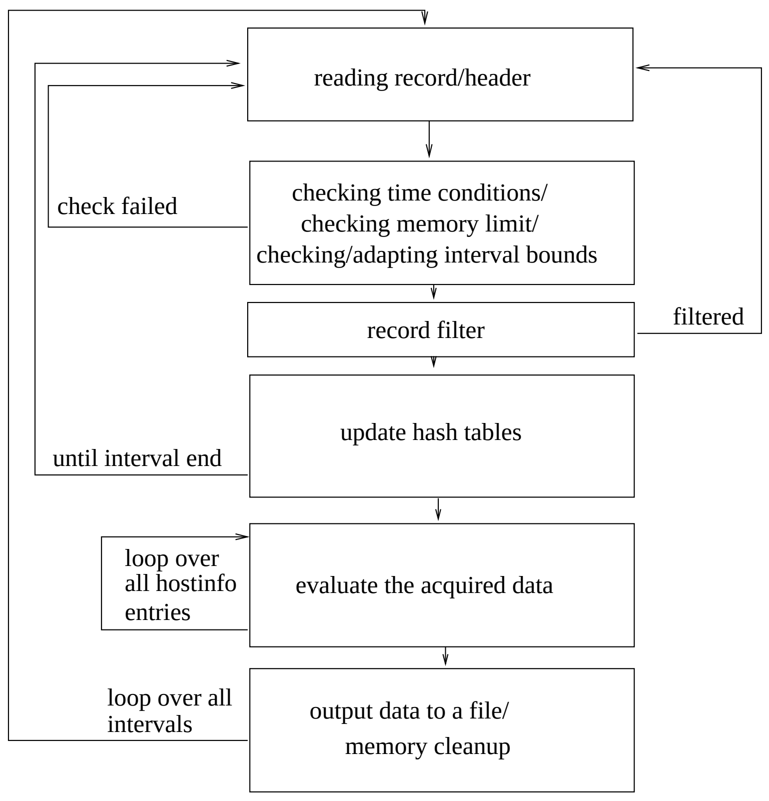


Figure 3.2: Structure of the host classificator algorithm

and the engine ID in the Netflow v5 header is stored for further use. Currently only packets with engine ID zero are processed because only those are created by the hardware engine of the router. Packets with another engine ID are discarded.

**Perform checks** In this stage the following checks are performed:

- Checks if the interval bounds are still valid. If not, the status bit for adapting the interval bounds is set and the new bounds are determined. Assuming that e.g. a router fails and there is no incoming data for some time, the next received packet, when the router is working again, has a time stamp much later than the end of the interval where the router failed. Due to the check described in the previous item, the algorithm evaluates the current interval and advances to the next by adding the interval length to it's bounds. But if the router was down for an hour 60 such advancements are necessary until the host classificator again observes a valid interval. That's why some checks to adapt the interval bounds in theses cases are made.

- Calculate the memory used by the host classificator plug-in (approximation) as it is shown in Section 3.5. If a memory limit was set and it is reached, the check fails. From now on, information about incoming flows is not stored anymore and it is only checked if the end of the current interval is reached. If it is reached, an evaluation based on the already acquired data is done. Since not all data in the interval is analysed, it should be possible to identify these intervals. Therefore the status byte value that is written to the statistic file is set accordingly for this interval.

- Check if the condition to advance to the next interval is fulfilled. Figure 3.3 illustrates that an advancement is initiated if a flow with a start time later than the end of the current interval plus a certain read ahead is detected. The used interval and read ahead lengths are one minute.

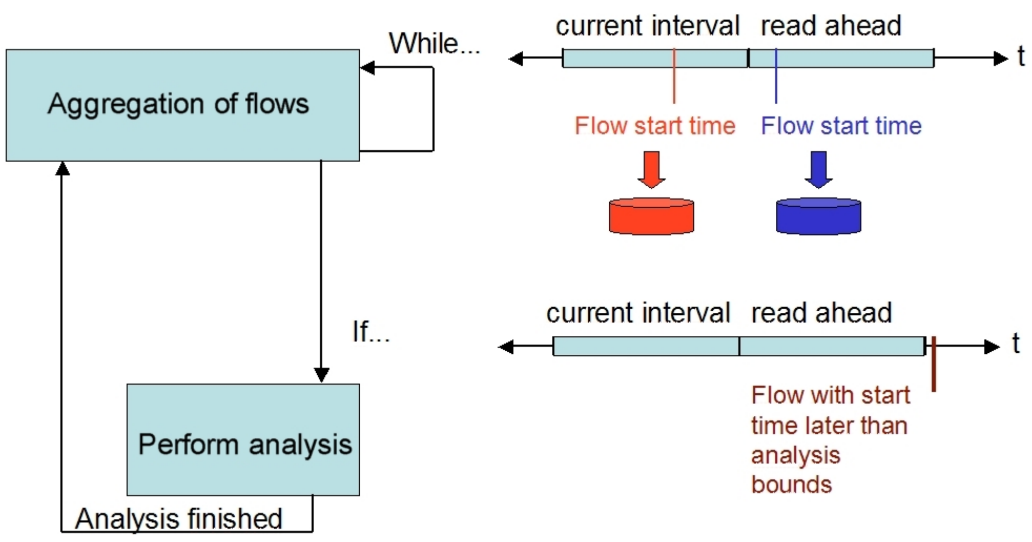


Figure 3.3: Advance to the next interval

**Record filter** To reduce the amount of data to process, all flows with another protocol identifier than TCP or UDP are filtered. Additionally, a minimum number of packets that a flow must have to not to be filtered can be specified. For the validation in Chapter 5 the minimal packet number was set to two[6]. With this setting, the memory usage can be reduced from 106 MBytes to 39 MBytes when analysing the Netflow data of the hour starting at 12.08.2003 12:21 (UTC)[7].

**Update hash tables** Generate hash table keys and store the needed fields of the currently analysed record. Furthermore it is checked to which interval the record belongs so that the activity bit can be set accordingly.

**Evaluate the acquired data** Figure 3.4 shows the stages of the evaluation process. Before it is tested if a host meets the requirements for the traffic class, it is checked if the data for the host is evaluated at all. It is not whenever the IP address of the host does not appear as source IP address of a flow. Since this is an indicator that this host does not exist, no classification relevant data is lost. Afterwards it is checked if the host is active in the current interval and if it is, all classification relevant criteria are checked and the host is assigned to the corresponding class(es). If it is not active in the current interval only the activity and class field is updated. The check if a host is a member of the responder class is a bit more complicated than the check for the traffic or connector class and is therefore explained in detail in the following section. Finally the port activity is evaluated. Details about this evaluation can be found in Section 3.6.2.

[6]This configuration eliminates a lot of port scan traffic
[7]Over 680'000 flows per minute due to W32.Blaster activity are registered

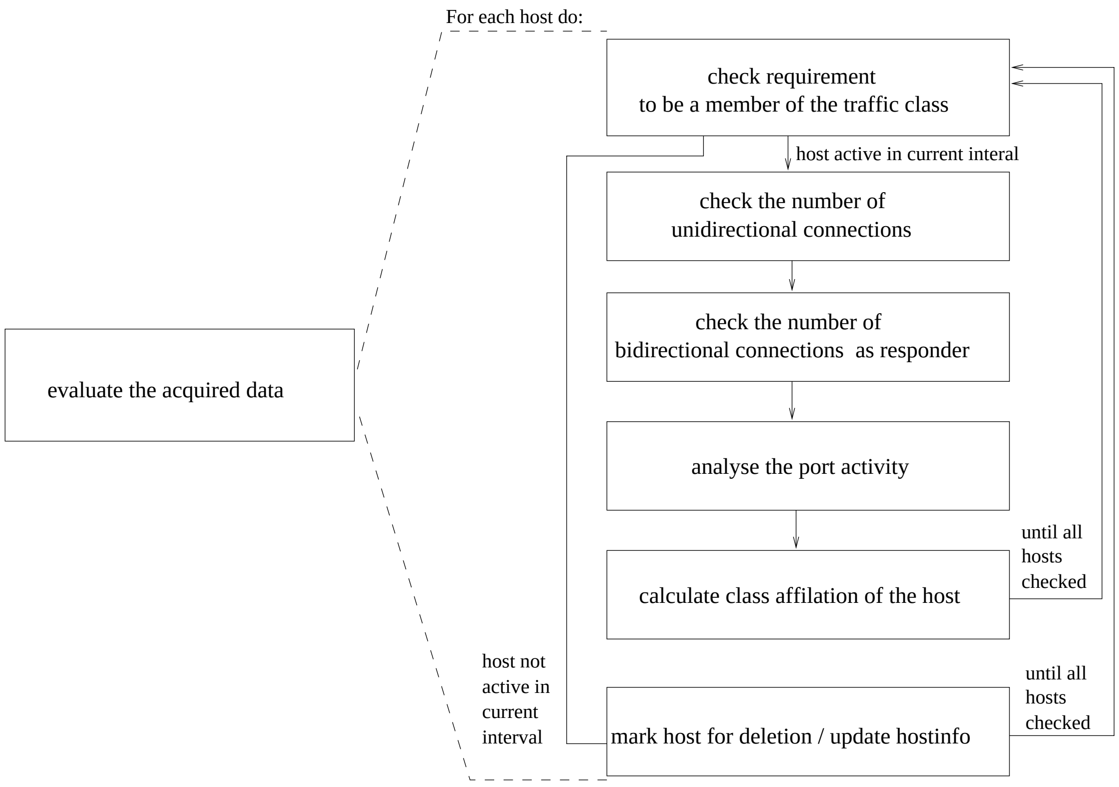

For each host do:
evaluate the acquired data
check requirement to be a member of the traffic class
host active in current interal
check the number of unidirectional connections
check the number of bidirectional connections as responder
analyse the port activity
calculate class affilation of the host
until all hosts checked
host not active in current interval
mark host for deletion / update hostinfo
until all hosts checked

Figure 3.4: Stages of the evaluation process

### 3.6.1 Responder Classification

To be a member of the responder class a host has to fulfil the following condition:

number of connections as responder > threshold

In Section 2.3.2 it was not yet specified when a host responds to a connection and when it initiates one or when even no decision can be made. The criteria used is the following: Host H1 is responder for a connection to host H2 identified by H1:h1/H2:h2[8] if a connection in the other direction identified by H2:h2/H1:h1 can be found where

$$\text{abs}\left(\text{flow start time}\left(\text{H1}\right) - \text{flow start time}\left(\text{H2}\right)\right) > \text{time threshold} \tag{3.6}$$

holds. The time threshold is set to 50ms, the flow start times are in UTC time.
The problem of this check is, that if e.g. the data form H1 to H2 is routed by a router R1 and the data from H2 to H1 by a router R2, it is not save to assume that the result of the check is correct. The problem is, that the clocks of the two routers may not be synchronous. Nevertheless this scenario was unaccounted for because to test the behaviour of the plug-in, it has to run on a server directly connected with the boarder routers[9]. Furthermore the clocks in the routers are synchronised so that this problem may not exist. Nevertheless preparations for a flow handling with respect to the routers who reported it are made. Only the code for handling the data in the different cases has to be inserted. Another problem is the duplication of flows caused e.g. by a connection from host H1 to host H2 via router R1 and router R2. Both router report a flow form H1 to H2 and from H2 to H1. To eliminate these duplicated flows the following check is performed before putting the information about the flow into the hash table:

$$\text{abs}\left(\text{flow start time}\left(\text{current}\right) - \text{flow start time}\left(\text{all flows with (H1:h1/H2:h2)}\right)\right) < \text{time threshold} \tag{3.7}$$

If this condition holds, the flow is already in the hash table and is therefore not inserted. The plug-in version provided with this thesis uses a time threshold of 5ms.

### 3.6.2 Port Activity Analysis

In Section 2.4 it is proposed to store ports used by a host and report for each interval only those ports not used before. But if the goal is to keep memory consumption low, it is not possible to store each port a host uses. Another problem would be, that no data structure with a fixed size can be used, what again complicates the calculation of the utilised memory. Hence the algorithm stores for each category [10] only eight port numbers. Another restriction is that the information about the used ports is only available of the last interval. The consequences are, that if a host was not active in the last interval but in the one preceding it, every[11] port it uses is reported anew. Figure 3.5 shows the structure of the port evaluation algorithm. It is shown that only information about the source ports used by a host is stored. This is sufficient since a successfull attack, at least when using TCP, needs communication in both directions so that the destination port of a request will appear as the source port of the reply. Hence no relevant information is lost since the destination port of a request will be logged at the destination host. The only destination ports not logged are those of hosts not answering to the requests. But if they don't answer on that port, it is not interesting for the evaluation of the port activity as it is done here. Nevertheless for scan detection it could be indeed of interest. The two described communication scenarios are shown in Figure 3.6.

[8]format: IP address of host X:port used by host X/IP address of host Y:port used by host Y

[9]since the router IP address is not contained in the Netflow data it can currently only be identified by the sender IP address of the UDP packets containing the Netflow records

[10]these are: TCP privileged, TCP unprivileged, UDP privileged, UDP unprivileged

[11]the first eight of each category, afterwards only the missed port counter for this category is increased

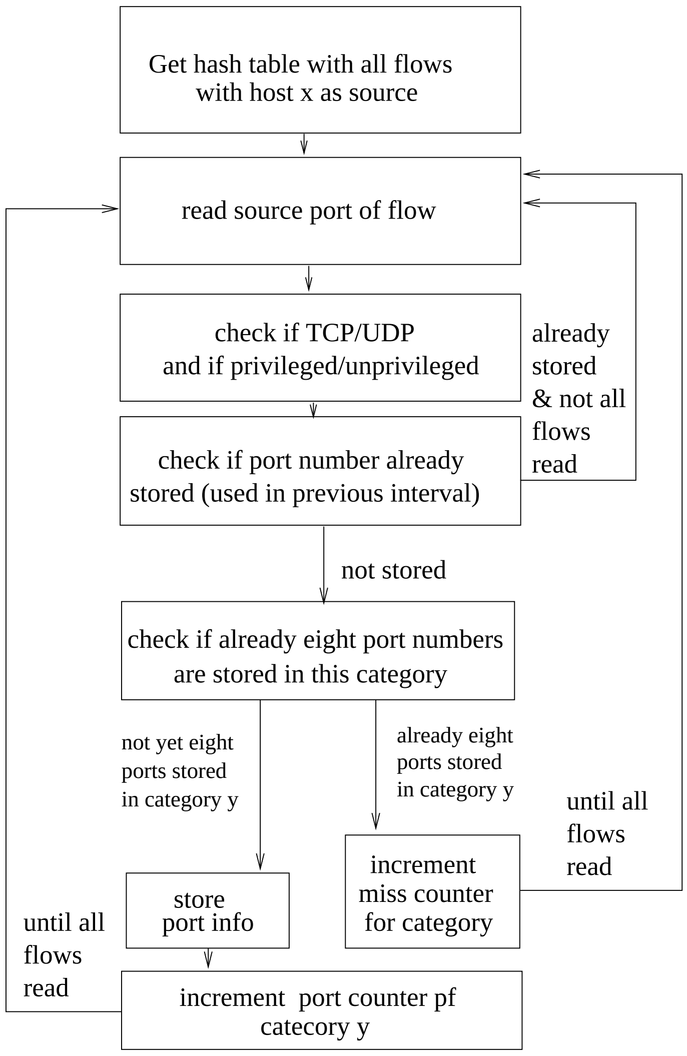

Get hash table with all flows with host x as source
read source port of flow
check if TCP/UDP and if privileged/unprivileged
check if port number already stored (used in previous interval)
already stored & not all flows read
not stored
check if already eight port numbers are stored in this category
not yet eight ports stored in category y
already eight ports stored in category y
until all flows read
store port info
increment miss counter for category
until all flows read
increment port counter pf catecory y

Figure 3.5: Port activity evaluation

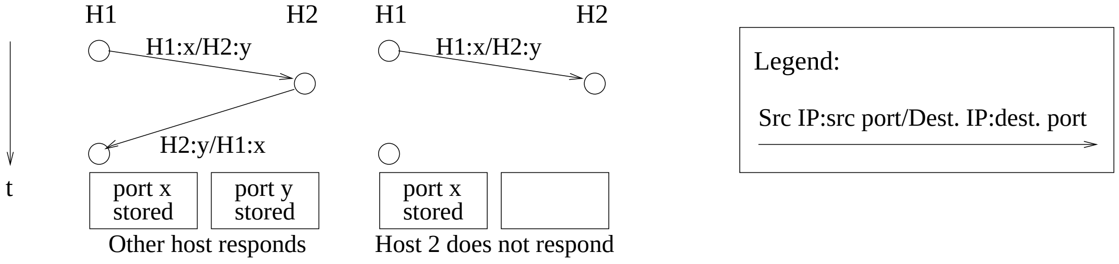


Figure 3.6: Communication scenarios

### 3.6.3 Possible Loss of Information

The following list sums up where data can be discarded or is lost:

- Reading record/header: Netflow v5 packets with engine ID other than zero are discarded
- If a specified memory limit is reached, no more data is acquired for this interval
- Record filter: Only TCP and UDP flows pass the filter
- Record filter: Depending on the setup, flows with less than x packets are filtered

## 3.7 Statistic File Format and the Directory Structure

What data the algorithm has to make available is determined by the purpose of use. Since that is the topic of Chapter 4, this section discusses only how the data is stored. Even though there are different methods to provide data to other applications, like message passing or shared memory, only by saving the data to disk the main memory usage can be kept low. To easily find the data corresponding to a specific date and time, the following directory structure and file naming is used:

*<base_dir>/<year><month>/<day><hour>.stat<number>*
<base_dir> defined in the host_classificator.cfg file
<year>,. . . UTC date and time of the data stored in this file
<number> is only added if a file with the same name already exists

Though it is important to reduce the data written to disk for each interval as much as possible, it is desirable to have a lot of information about the traffic. Since the answer to the question, what data should be saved to disk, depends on what data is needed by the visualisation software, there is little scope for optimisation of disk space usage here. It remains to choose the smallest possible representation of that data, where basically two possibilities exist: binary or ASCII. The ASCII representation has the advantage that the file content is human readable, but it's disadvantage is the bigger filesize. To reduce the filesize, compression could be used but because that would need valuable processor power, the statistic files of the host classificator are saved using the binary format. To compensate the disadvantage that the file content is not human readable, a binary to ASCII converter[12] has been implemented. An additional advantage of this approach is, that e.g. if only the amount of hosts in each class is of interest, the converter can write only this information into the ASCII file. If considered that else one had to find these numbers in a file that contains 130'000 numbers per stored interval, this is a great advantage. Now that the format of the statistic file is known, the structure of the file can be described. The host classificator saves for each interval the data shown in Table 3.8 to the corresponding statistic file.
The status information in bit 0..7 of each entry allows the identification of faulty or, e.g. due to the memory limitation[16], not entirely aggregated intervals. The allowed values for the status information are listed in Table 3.9.
The entire file consists of a format identifier and the entries for each interval. Table 3.7 sums this up.
The resulting file size of a type 0 file containing all data for an hour with one entry per minute is:

$$\begin{aligned} size &= 1 \times sizeof(char) + 60 \times ((2 \times 65536 + 17) \times sizeof(int) + 1 \times sizeof(char)) \\ &= 60 \times (131'089 \times 4 + 1) = 1 + 60 \times (524'357) \\ &= 31'461'421 Bytes (\cong 30MB) \end{aligned}$$

and for a type 1 file it is:

$$\begin{aligned} size &= 1 \times sizeof(char) + 60 \times ((14) \times sizeof(int) + 1 \times sizeof(char)) \\ &= 1 + 60 \times (14 \times 4 + 1) \\ &= 1 + 60 \times (57) \\ &= 3421 Bytes (\cong 3.34kB) \end{aligned}$$

[12]some details about the converter can be found in Appendix A.1
[16]see Section 3.2.1 for details

| statistic file format | | |
|---|---|---|
| bits | type | description |
| 0..7 | char | statistic file type (0 or 1) |
| 8+n*(entry)..(n+1)*entry+8 | entry | entry type specified in Table 3.8 |

Table 3.7: Format of the statistic file

| bits | data type | description |
|---|---|---|
| 0..7 | char | status information |
| 8..39 | unsigned int | number of active hosts[13] |
| 40..71 | unsigned int | number of hosts not member of a class |
| 72..103 | unsigned int | number of hosts only in the traffic class |
| 104..135 | unsigned int | number of hosts only in the connector class |
| 136..167 | unsigned int | number of hosts only in the responder class |
| 168..199 | unsigned int | number of hosts in the traffic and connector class |
| 200..231 | unsigned int | number of hosts in the traffic and responder class |
| 232..263 | unsigned int | number of hosts in the connector and responder class |
| 264..295 | unsigned int | number of hosts that are a member of all three classes |
| 296..327 | unsigned int | number of hosts that are newly in the traffic class |
| 328..359 | unsigned int | number of hosts that are newly in the connector class |
| 360..391 | unsigned int | number of hosts that are newly in the responder class |
| if file format indicator is 1 | | |
| 392..423 | int | number of hosts with activity on the specified tcp port |
| if file format indicator is 0 | | |
| 392..423 | unsigned int | number of missed privileged tcp ports |
| 424..455 | unsigned int | number of missed unprivileged tcp ports |
| 456..487 | unsigned int | number of missed privileged udp ports |
| 488..519 | unsigned int | number of missed unprivileged udp ports |
| 520..524’807 | 65536*uint | number of hosts with first activity on tcp port x[14] |
| 524’808..1’049’095 | 65536*int | number of hosts with activity on port x for the first time[15] |

Table 3.8: Format of the entries in the statistic file

| status information of an interval | |
|---|---|
| 0 | valid entry |
| 32 | no data available |
| 64 | memory limit reached |
| 128 | interval bounds adapted during acquisition |
| 192 | code 64 and 128 together |

Table 3.9: Status info values

Despite the fact that the files are as small as they can be if no compression is used, the size of the type 0 file is not suitable for long term archiving. It is therefore recommended to compress a whole subdirectory containing the statistic files for an entire month, if the acquisition of the data for this month is finished. Several tests with bzip2 showed, that compression rates up to 1:10 can be achieved while a rate of 1:3 or 1:4 can be expected. Assuming a compression rate of 1:4, approximately 5.5 GBytes[17] of disk space are needed to store the output of the host classificator plug-in of a full month. To generate the output, the plug-in processes the Netflow v5 data sent to a computer running UPFrame. This data is the Netflow v5 data of about 5% of the Internet traffic in Switzerland and crossing Switzerland.

[17]assumed that the month has 31 days

Chapter 4

# Visualisation

After the specification and implementation of the algorithm, the focus can be set onto how to visualise the output of the algorithm. Since the task was to make the plots accessible over the web, the visualisation software VISTOOL was implemented as Common Getaway Interface(CGI). Since the post-processing of the data and the generation of the plots as much as the handling of binary files should be as fast as possible, not Perl or PHP were used to implement VISTOOL but C. Since the presentation of all the data for each of the up to 80’000 hosts per interval in a single plot is unlikely to be clear, separate visualisation plots to show the port activity and the other characteristics were designed. Nevertheless it would be still difficult to find a visualisation method that shows for each host it’s classification and that presents the information in a way that the detection of anomalies is easy. At first a similar approach as A. Weisskopf used in its semester thesis "Plug-ins for DDoS Attack Detection in Realtime" for the visualisation of the IP activity in the Internet [6], seemed to be the solution to the visualisation problem. Further analysis showed that with the host classificator output and some colour coded pixels per host it is difficult to get a good overview about each hosts behaviour and to realise significant changes in the plot in case of an anomaly. After checking some methods to further reduce the information to display, it became clear that basically only the total number of hosts in each class is necessary to show significant changes in the behaviour of the hosts. Since that meant a reduction of the data to nine numbers and was therefore ideal to keep the disk space usage of the statistic files low, finally only the total number of hosts and the number of hosts in each class were saved per interval[1]. The plots based on these nine numbers are presented in Section 4.2.
It remains to display the port information for the hosts in a meaningful way what requires without doubt again a reduction of the information to display. The following approach reduces the amount of data for the visualisation of the port activity of the hosts considerably; It is not stored which host used which ports for the first time but only how many hosts used a specific port for the first time[2]. To allow a separate analysis of TCP and UDP ports, the total number of hosts using a certain port for the first time is saved to disk separately for the UDP and TCP ports[3]. The graphical representation, which is named "port map", is discussed in Section 4.3.

## 4.1 Global Settings

Despite the fact that different analysis methods need different parameter fields, some fields are common to all methods. These are the radio buttons to specify the analysis method, the input fields for the date and time in offline mode and finally the refresh field in online mode. Although the data input for most of the fields should be self-explaining just as the various analysis modes, a short description of the difference between the online and offline mode shall be given. Both modes operate on the data generated by the host classificator algorithm. To analyse data not

[1]see Section 3.7 for details of the statistic file format
[2]see Section 3.6.2 for details
[3]details about the file format are in Section 3.7

yet processed by the host classificator plug-in, it has to be fed to the plug-in first. This can be done e.g. by replaying it with the netflow_replay tool developed by Arno Wagner. In online mode, the statistic file name is generated out of the current UTC time of the computer running the CGI. In offline mode any data and time can be specified. It is then searched in the corresponding file for the data belonging to the current or the specified interval. For plots needing more than the data of one interval, the specified time in offline mode or the current time in online mode represents the end time of the data to present. When using the online mode due to the delay of the output to the input, no data for the current time is available. Therefore it is searched for data of the preceding intervals, up to a maximum of a four intervals look-back. In offline mode the look-back is too active, but since there the data is normally available at the time specified by the user, no look-back is necessary. In online mode the time in UTC of the computer running VISTOOL may not differ too much from the time in UTC of the NetFlow data generating routers since the host classificator plug-in writes the statistic file entries using the time stamps set by the router. A feature of the online mode is, that a refresh rate can be set at which the plot is updated.

## 4.2 Host Classification

There exist a lot of different variants such as bar, line or pie charts to visualise the amount of hosts in each class over time. Unfortunately none of theses variants allow both, to compare the number of hosts in a class to the number in all others and to follow the change in the number of hosts over time in a single plot. Therefore two separate plots are generated, a kind of a Venn diagram to visualise the proportions between the classes and a x-t plot to show the number of hosts in one or more class(es) over time.

### 4.2.1 Venn Diagram

Figure 4.2 shows the Venn diagram based on the data of 11.08.2003 21:20(UTC). The Venn diagram shows only the number of hosts in each class for a specific interval and not the change of proportions over time. The number of hosts in a class determines the diameter of the circle as much as the colour by using a logarithmic scale where the maximum can be specified. The status information[4] about the interval is displayed in the lower right corner.

### 4.2.2 X-t Plot

Figure 4.3 shows the x-t plot of the hour before 11.08.2003 21:50 (UTC) for the traffic, connector and responder class. The plot below the x-t plot, the error code plot, shows for each interval the status information. The plots are both created with the round robin database tool RRDTool [7] and can display up to 60 hours of data. It would be possible to display more data by modifying the CGI code, but since displaying 48 hours of data already takes approximately 2 minutes on a Intel Pentium 4 2.5 GHz with 512 MB RAM, the limit was set to 60 hours.
The advantage of this plot is, that it allows the identification of anomalies in the number of hosts per class but it's disadvantage is that the proportions between the number of hosts in each class can hardly be seen. The disadvantage is traced back to the fact that if the results for multiple classes are plotted for, either the lines are too close together and crossing each other or the used scale is bad because one class has much more hosts in it, so that the amplitude variations of the others can hardly be seen. Using a logarithmic scale for the number of hosts could help.

## 4.3 Port Activity

As already mentioned, the statistic file contains not the data about the used UDP and TCP ports per host, but only the total number of hosts that use a specific port for the first time. Nevertheless there are 65'536 values for the TCP ports and also for the UDP ports that have to be visualised. Since this is still too much data to present it clearly arranged, a further reduction

[4]see Table 3.9 for absolute values

of the data is unavoidable. Displaying only the results for a certain port range is therefore an option. Fortunately two port ranges are already defined; The privileged ports from 0 to 1023 and the unprivileged ports from 1024 to 65535. Since almost all popular services[5] are offered via a privileged port, and because the goal of an attack is normally to do harm to as many hosts as possible, only exploiting a security hole of a popular service is interesting. Therefore the data for the 1024 privileged ports has to be presented in a way that the number of hosts using a specific port for the first time can be identified at a glance. To identify fluctuations not only the results for one interval but for several are shown. The developed plot displays the results for the 9 last intervals. An example of the plot that is called port map is shown in Figure 4.4. If the privileged ports are displayed, each port has it's own rectangular. Else the results for 64 ports are added up and presented in a rectangular. The port number, in case of the unprivileged ports it is the number of the first of the 64 ports, that correspond to a specific rectangular can be calculated by adding the numbers displayed above the column and besides the row in which the rectangular is located. Figure 4.1 illustrates what exactly is displayed in the rectangular.

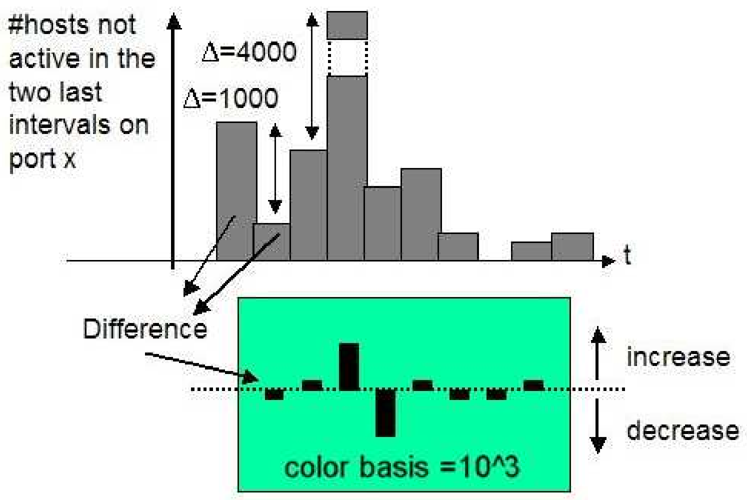
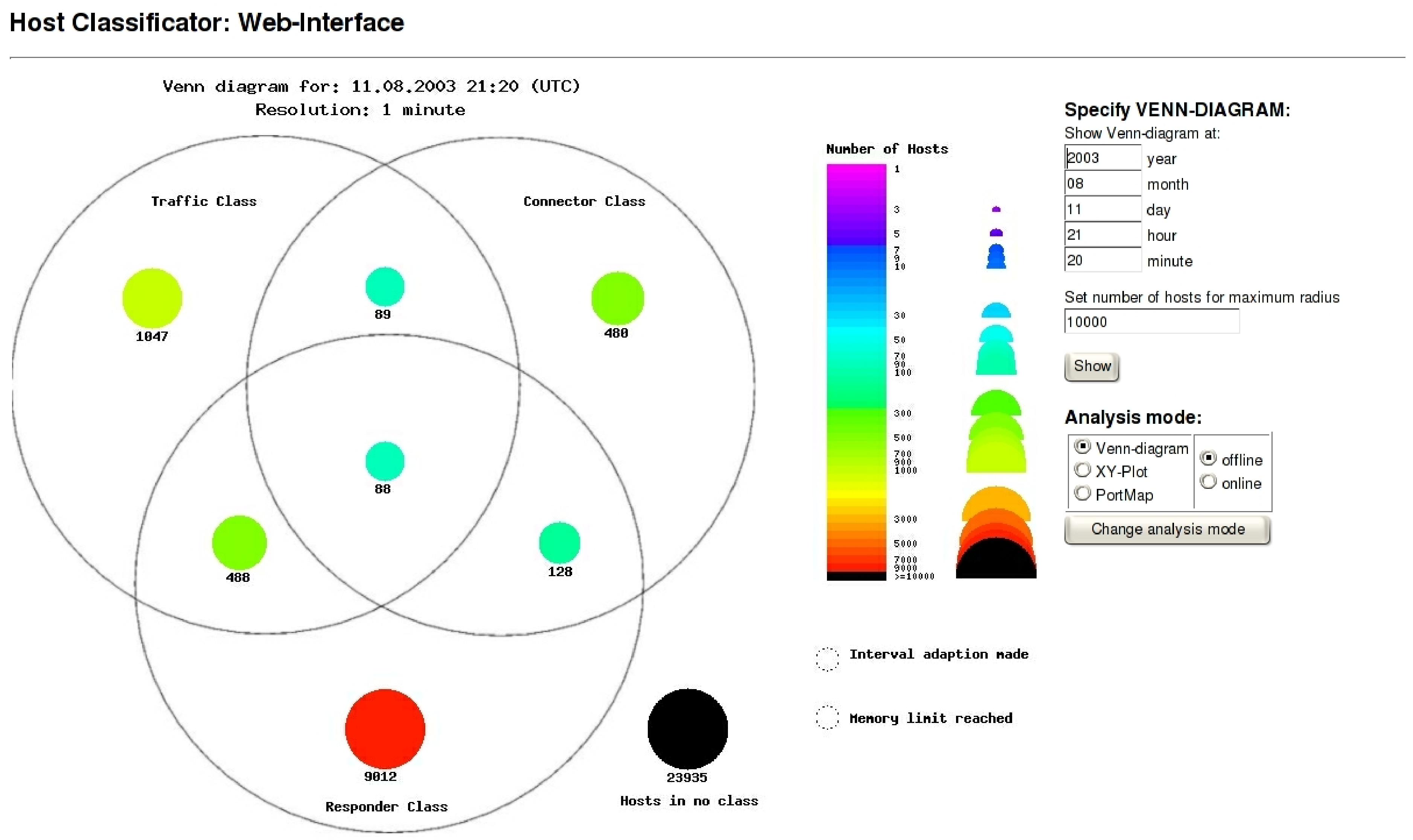


Figure 4.1: Detailed view of a cell of the port map

It emanates from that Figure, that not the total number of hosts active for the first time on a specific port in a specific interval is shown, but the difference of the total number from the current and the last interval. If the difference from one interval to the other is big enough, the background colour indicating the basis may change. An increase or decrease from interval to interval is shown with an amplitude in the corresponding direction. If the amplitude of a bar corresponds to the smallest possible amplitude it means that the absolute value of the change is smaller or equal the basis value. If no bar at all is displayed the difference was zero. If the status information value does not indicate that an interval is alright, the bars have special colours. A colour legend is available by clicking the status information link on the port map analysis page.

[5]Exception: P2P. P2P services are the topic of various other theses and are not treated here.



Host Classificator: Web-Interface

Venn diagram for: 11.08.2003 21:20 (UTC)
Resolution: 1 minute

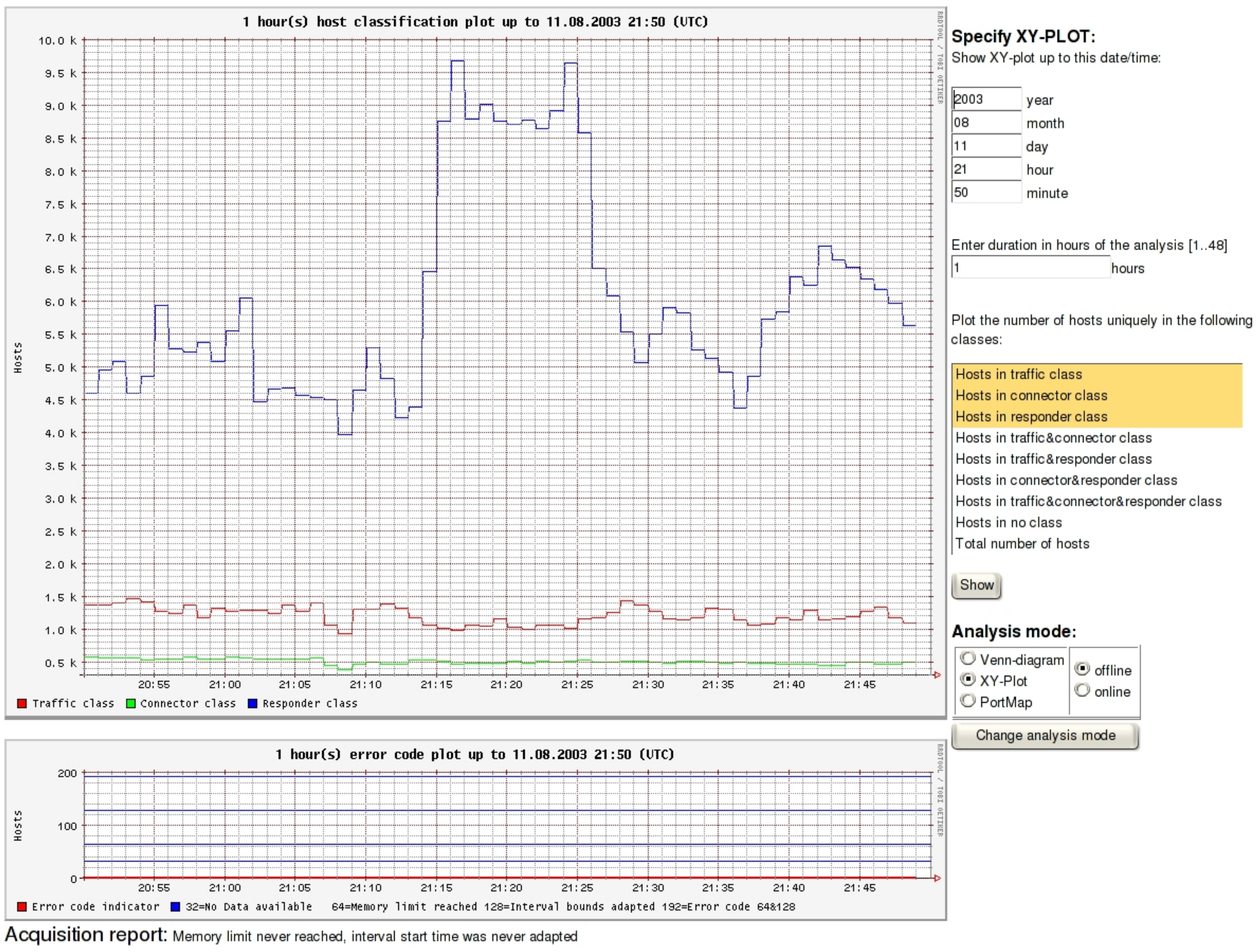
Specify VENN-DIAGRAM:
Show Venn-diagram at:

| | |
|---|---|
| 2003 | year |
| 08 | month |
| 11 | day |
| 21 | hour |
| 20 | minute |

Set number of hosts for maximum radius
10000

Analysis mode:
[✓ Venn-diagram] XY-Plot PortMap
[✓ offline] online

Figure 4.2: Venn diagram


1 hour(s) host classification plot up to 11.08.2003 21:50 (UTC)
Hosts
10.0 k
9.5 k
9.0 k
8.5 k
8.0 k
7.5 k
7.0 k
6.5 k
6.0 k
5.5 k
5.0 k
4.5 k
4.0 k
3.5 k
3.0 k
2.5 k
2.0 k
1.5 k
1.0 k
0.5 k
20:55
21:00
21:05
21:10
21:15
21:20
21:25
21:30
21:35
21:40
21:45
🟥 Traffic class 🟩 Connector class 🟦 Responder class
RRDTOOL / TOBI OETIKER
1 hour(s) error code plot up to 11.08.2003 21:50 (UTC)
Hosts
200
100
0
🟥 Error code indicator 🟦 32=No Data available 64=Memory limit reached 128=Interval bounds adapted 192=Error code 64&128
Show
Change analysis mode

**Specify XY-PLOT:**
Show XY-plot up to this date/time:

|  |  |
|---|---|
| 2003 | year |
| 08 | month |
| 11 | day |
| 21 | hour |
| 50 | minute |

Enter duration in hours of the analysis [1..48]
1 hours

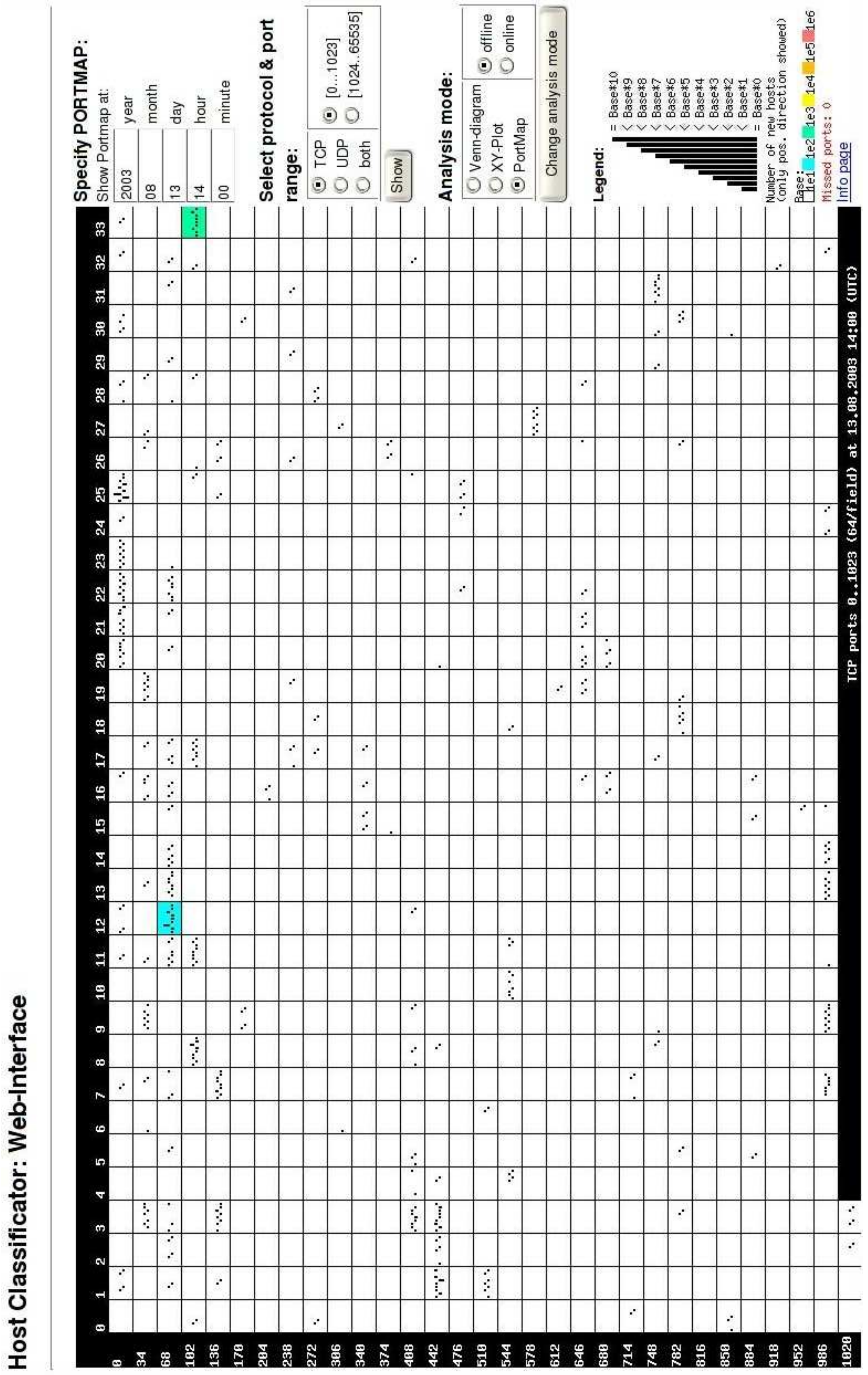
Plot the number of hosts uniquely in the following classes:

- Hosts in traffic class
- Hosts in connector class
- Hosts in responder class
- Hosts in traffic&connector class
- Hosts in traffic&responder class
- Hosts in connector&responder class
- Hosts in traffic&connector&responder class
- Hosts in no class
- Total number of hosts

**Analysis mode:**
Venn-diagram [✓ XY-Plot] PortMap
[✓ offline] online

Acquisition report: Memory limit never reached, interval start time was never adapted

Figure 4.3: x-t plot



Host Classificator: Web-Interface

Specify PORTMAP:

Show Portmap at:

| | |
|---|---|
| 2003 | year |
| 08 | month |
| 13 | day |
| 14 | hour |
| 00 | minute |

Select protocol & port range:

[✓ TCP] UDP both

[✓ [0...1023]] [1024..65535]

Show

Analysis mode:

Venn-diagram XY-Plot [✓ PortMap]

[✓ offline] online

Change analysis mode

Legend:

= Base*10
< Base*9
< Base*8
< Base*7
< Base*6
< Base*5
< Base*4
< Base*3
< Base*2
< Base*1
= Base*0

Number of new hosts (only pos. direction showed)

Base: 1e1 🟦1e2 🟩1e3 🟨1e4 🟧1e5 🟥1e6

Missed ports: 0

Info page

TCP ports 0..1023 (64/field) at 13.08.2003 14:00 (UTC)

Figure 4.4: Port map

# Chapter 5

# Validation

In this section it is verified if the output of the host classificator plug-in shows significant changes during the outbreak of the W32.Blaster worm. Furthermore the performance and memory usage for two use cases is measured to verify if the plug-in can be used for real time processing of Netflow data.

## 5.1 Memory Usage and Performance Analysis

To test the performance and the memory usage of the plug-in, an hour of Netflow data from before the outbreak of the W32.Blaster worm and an hour from the ongoing outbreak was replayed. The name of the files containing the replayed Netflow data and the corresponding start time are:

1. File 1:
   19993_00004791_1060690791.dat.bz2, start date=12.08.2003, time=14:19:51(UTC)

2. File 2:
   19993_00004740_1060507174.dat.bz2, start date=10.08.2003, time=11:19:34(UTC)

Since different filter[1] settings lead to different results, the test was made twice; once filtering no flows and once filtering flows with a packet count greater than one.
The hardware used for the test was a Intel Pentium 4 with a clock rate of 2.5 GHz and 512 MB RAM. The measurement of the maximum memory usage was made with the process table of the KDE System Guard using a refresh rate of half a second. The time needed to process the data of a file was acquired using the process information command ps. The results in Table 5.1 show, that even during the outbreak of the W32.Blaster worm, an additional filtering of flows with less than $x$ packets[2], is not necessary, as long as enough memory is available. Nevertheless a memory limit set to 150 MBytes or more is recommended if no additional filtering is used. The average processor load of 16.25% when replaying the file where the W32.Blaster worm was active, seems to be acceptable. Furthermore an optional filter that filters all flows reporting only one packet, reduces the processor load to 4.8%.

| File | Filter | Max. used memory [MB] | Processing time [sec] | $\phi$ processor load [%] |
|---|---|---|---|---|
| File 1 | - | 103.5 | 585 | 16.25 |
| File 1 | >1 | 47.7 | 170 | 4.8 |
| File 2 | - | 53.3 | 238 | 6.7 |
| File 2 | >1 | 27.3 | 118 | 3.28 |

Table 5.1: Performance measurements

[1]see Section 3.6 for details
[2]$x$ can be any 32-bit number

## 5.2 Normal Traffic

Figure 5.1 shows an 48 hour x-t plot of the number of hosts in the traffic, connector and responder class, Figure 5.2 shows an 48 hour x-t plot of the number of hosts in the traffic&connector, traffic&responder, connector&responder and traffic&connector&responder class. Both show flow data acquired before the outbreak of the W32.Blaster worm. The output was generated with no memory limitation and with flows with less than 2 packets filtered. The normal daily fluctuation can be seen.
Figure 5.3 and Figure 5.4 are port maps generated at 19:00 and 12:00 of the 07.08.2003. Since the W32.Blaster Internet worm uses the destination port 135 to infect other computers, special attention shall be given to port 135 when examinating the port maps.

## 5.3 W32.Blaster Traffic

Figure 5.5 shows the plot of the two first days of the W32.Blaster worm spreading for the traffic, connector and responder class. Figure 5.6 does this for the traffic&connector, traffic&responder, connector&responder and traffic&connector&responder class. The output was generated with no memory limitation and with flows with less than 2 packets filtered.
Figure 5.7 and Figure 5.8 are port maps generated at 19:00 and 12:00 of the 11.08.2003.

### 5.3.1 Comparison of the Results

**X-t plot** The requested significant change in the plots during the spreading of the W32.Blaster worm can be seen best in Figure 5.5. There are around 4000 hosts and more in the responder class whereas Figure 5.1 illustrates, that before the outbreak only around 2000 hosts could be counted. Responsible for the increase in this class is most likely the W32.Blaster worm trying to contact other hosts that were before only initiating connections on port 135. These hosts then eventually respond and appear as member of the responder class. The increase of hosts in the traffic&responder class in Figure 5.6 is anomalous too when compared to the number of hosts in this class before the W32.Blaster activity. The following perhaps a bit bold statement could serve as explanation; Inactive hosts that are contacted by a infected host respond while transmitting at least data of the size of three times the request data. Since the size of the data transmitted during a W32.Blaster contact or infection is very small, only otherwise inactive hosts or hosts, where the traffic requirement is almost met, can become traffic class members.
Another interesting fluctuation in Figure 5.2 is the increase in the traffic&responder class around 19:00, 21:30 and 23:30. If the port maps in Figure 5.3 and 5.9 are consulted it stands out that many ports above port 600 show exactly the same behaviour. Most likely this is the result of a large scale port scan since all ports are scanned in a single interval. All in all a significant change during the W32.Blaster spreading can be seen as much as other interesting fluctuations that could be further investigated.

**Port map** The port map too shows significant changes during the outbreak of the W32.Blaster worm. If the two port maps before and the two port maps during the activity of the worm are compared, it stands out, that there are big fluctuations in the number of hosts using port 135 for the first time[3]. Since the port map allows to see any fluctuation in the number of hosts using a specific port as source port and since within normal use only the server side avails itself of source port numbers in the range of the well known ports, these are with little doubt fluctuations in the number of active hosts behaving like servers. It follows that any worm that causes an infected host to act server-like on a specific port, can be detected if it's spreading is wide enough.

**Example: Combined analysis** It is clear that if the x-t plot and the port map are used in combination, more information about an anomaly or in general, more information about the behaviour of the hosts in the Internet can be extracted. Figure 5.10 illustrates how such an analysis could look like. For the analysis data from the 11th of August was used. In the x-t plot

[3]see Section 3.6.2 for the definition of 'first time'

showing the number of hosts in the responder class for the hour before 22:00, a significant decrease can be seen around 21:26. But what sort of traffic is responsible for the decrease? The two extracts of the port map from 21:26 and 21:27 illustrate, that at 21:27 between 2000 and 3000 hosts less than at 21:26 use port 135 for the first time. Since the decrease in responder class hosts is a bit more than 2000 hosts and considering that the observed peak is as much anomalous as an orange coloured port 135 field, the decrease in responder class host can be ascribed to the decrease of hosts using port 135 for the first time.

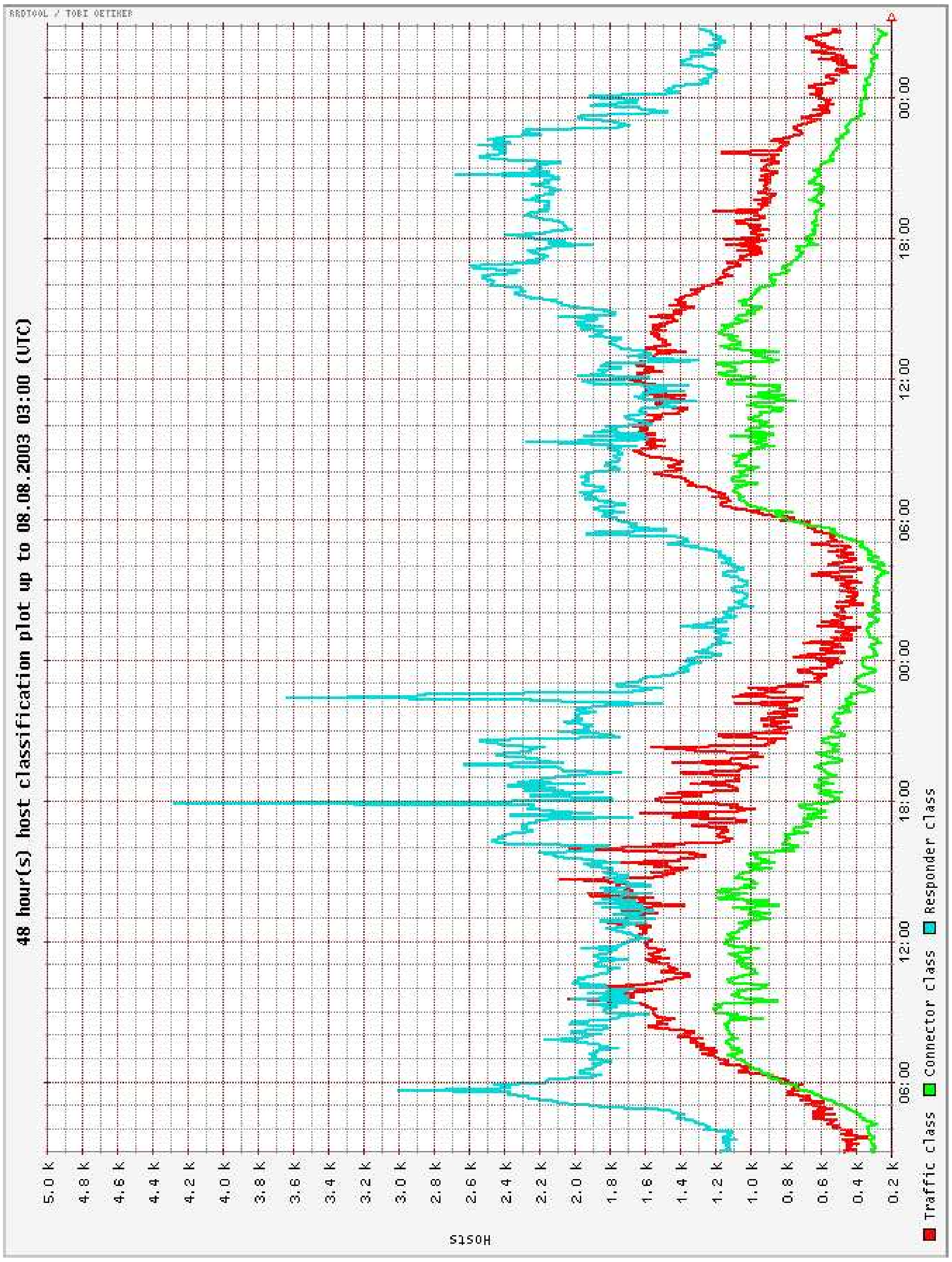


Figure 5.1: x-t plot: generated using data from before the outbreak of the W32.Blaster worm

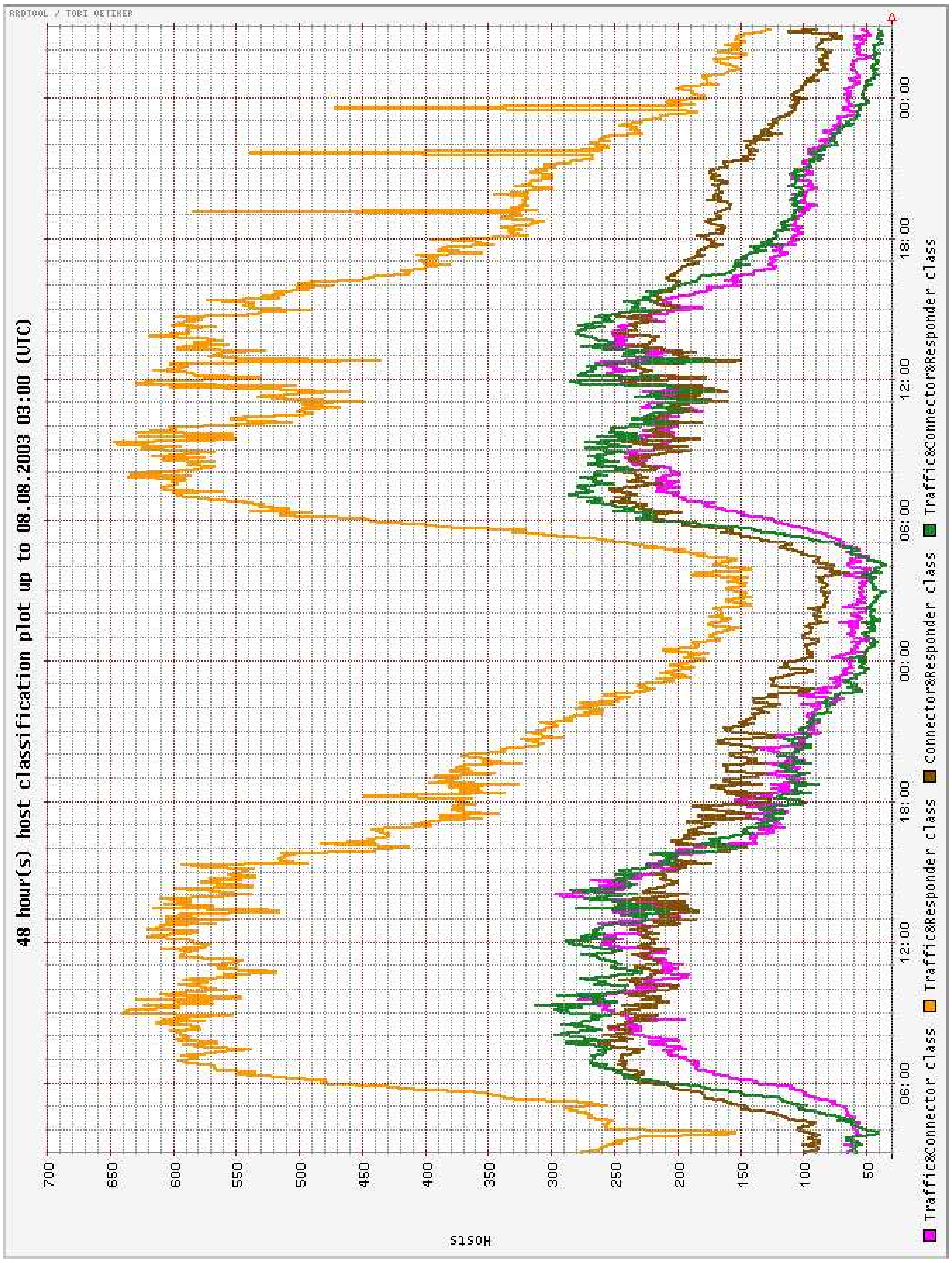


Figure 5.2: x-t plot: generated using data from before the outbreak of the W32.Blaster worm

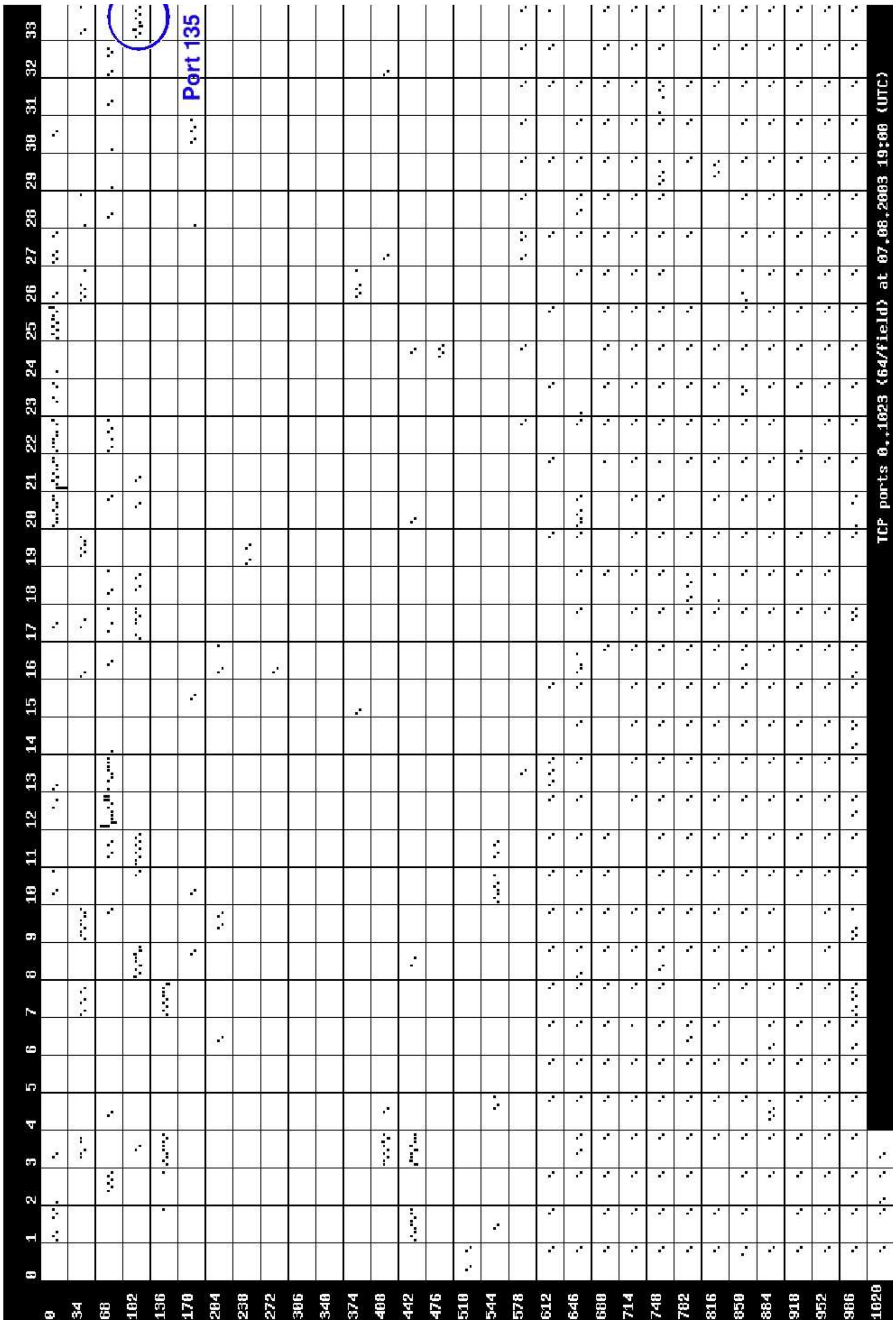


Figure 5.3: Port map using data from before the outbreak of the W32.Blaster worm

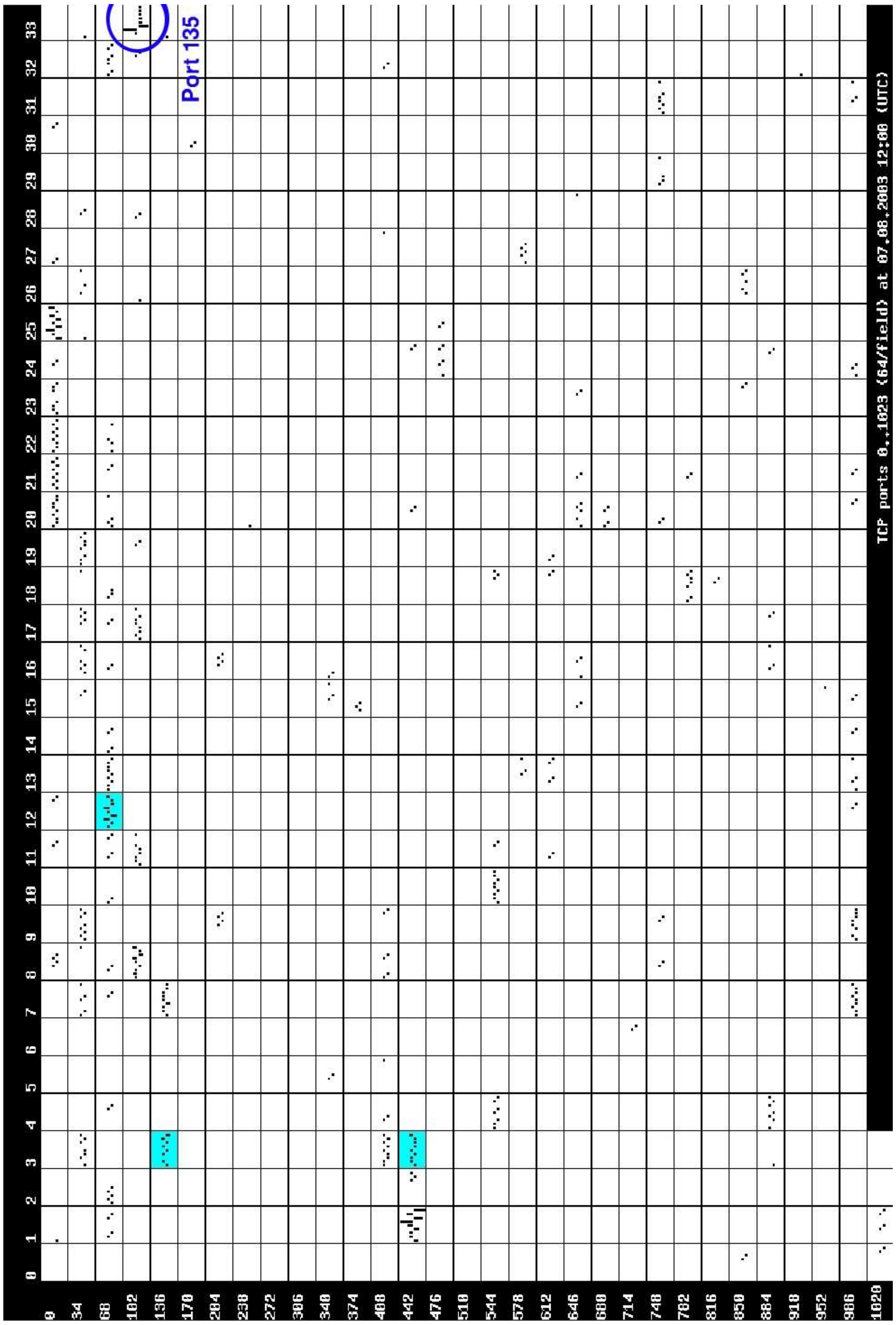


Figure 5.4: Port map using data from before the outbreak of the W32.Blaster worm

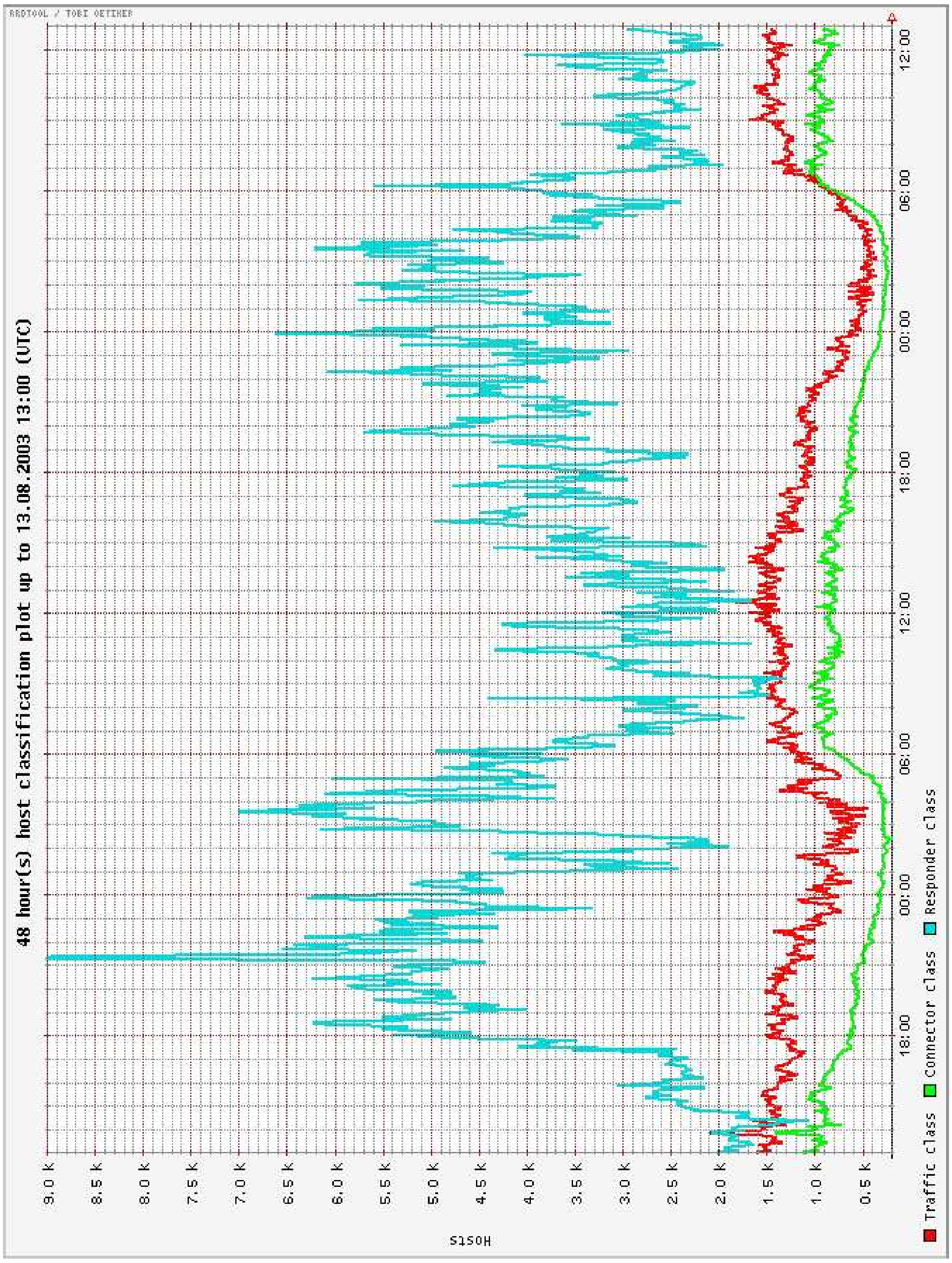


Figure 5.5: x-t plot: generated using data acquired during the outbreak of the W32.Blaster worm

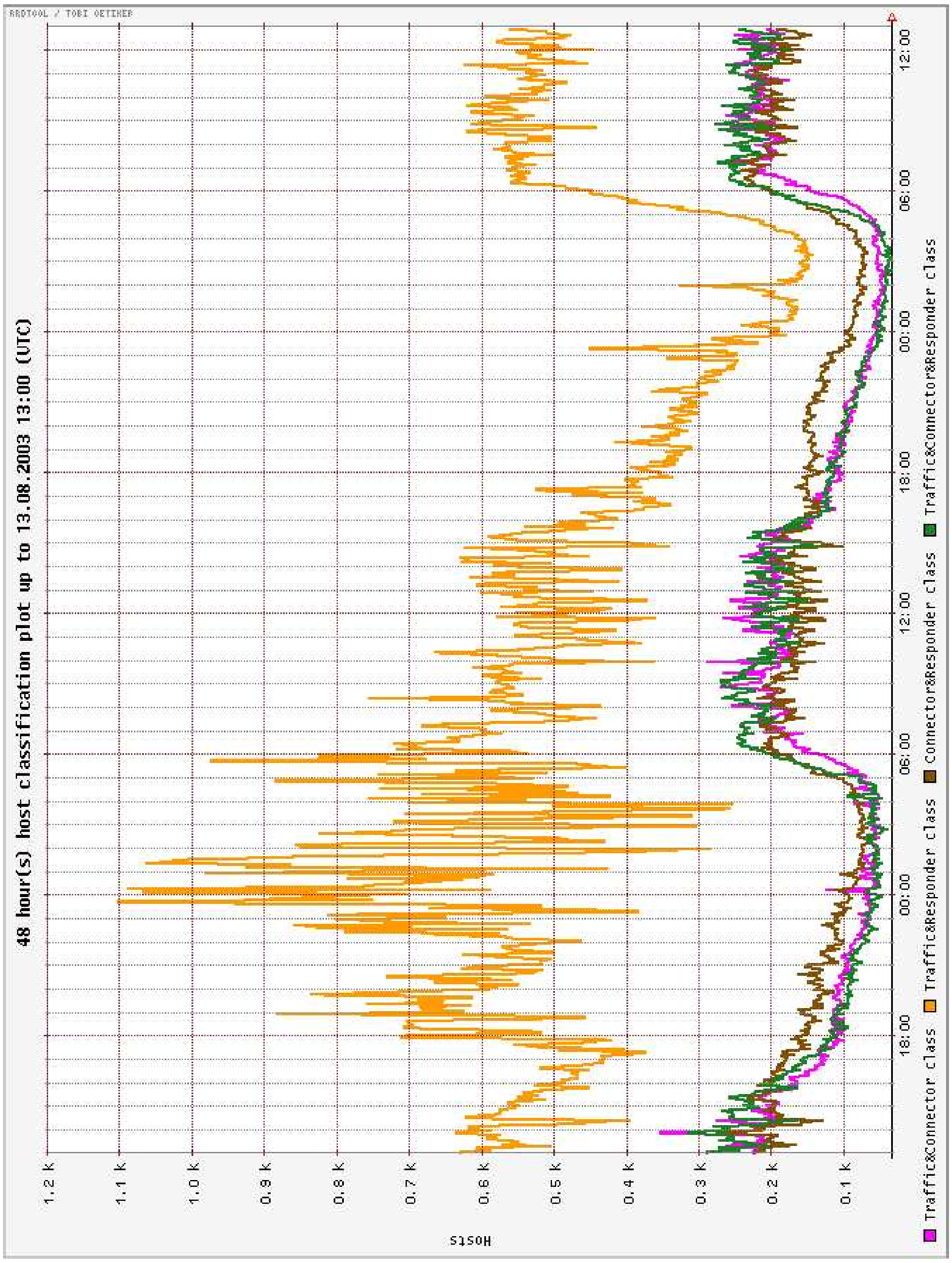


Figure 5.6: x-t plot: generated using data acquired during the outbreak of the W32.Blaster worm

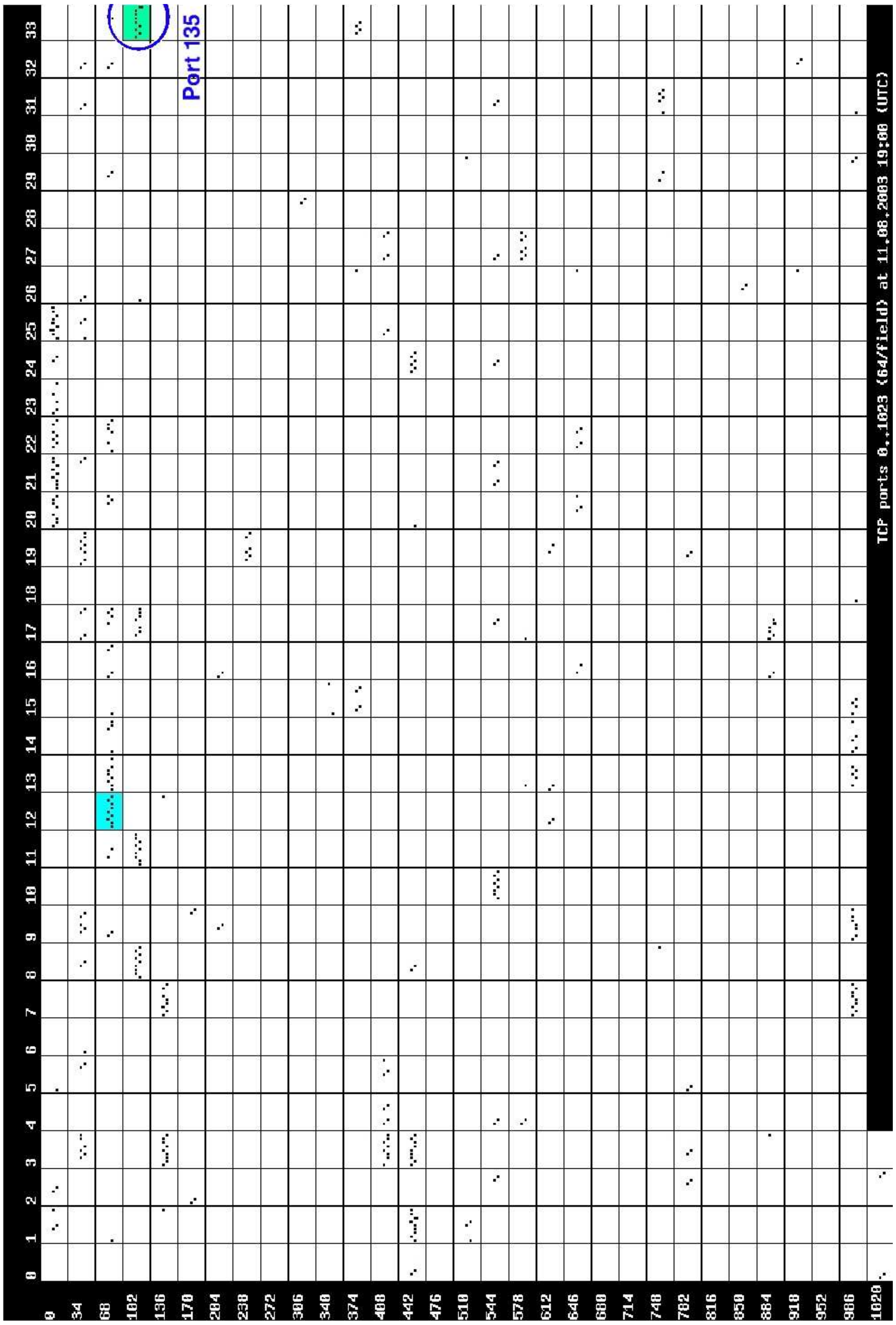


Figure 5.7: Port map using data from before the outbreak of the W32.Blaster worm

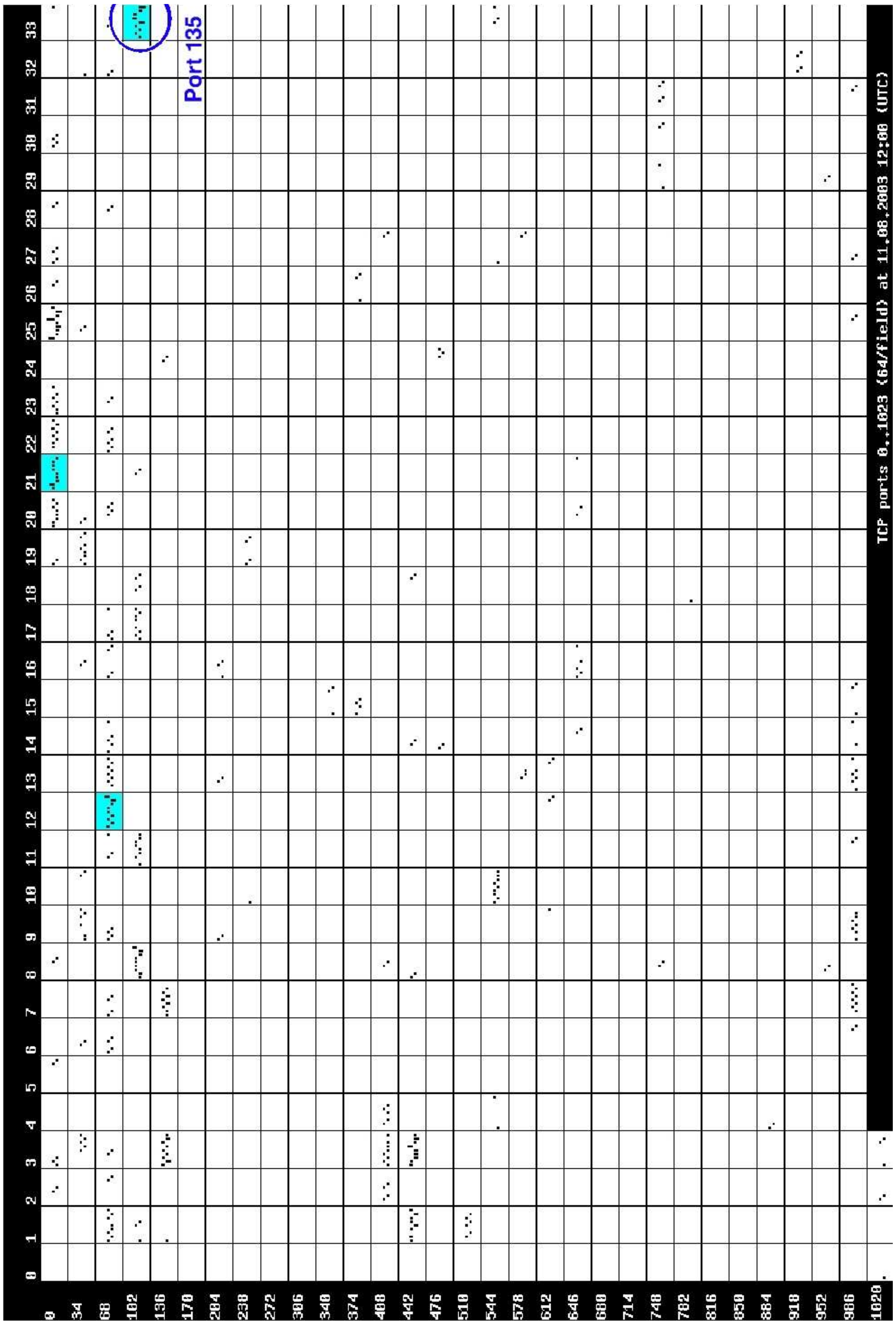


Figure 5.8: Port map using data from before the outbreak of the W32.Blaster worm

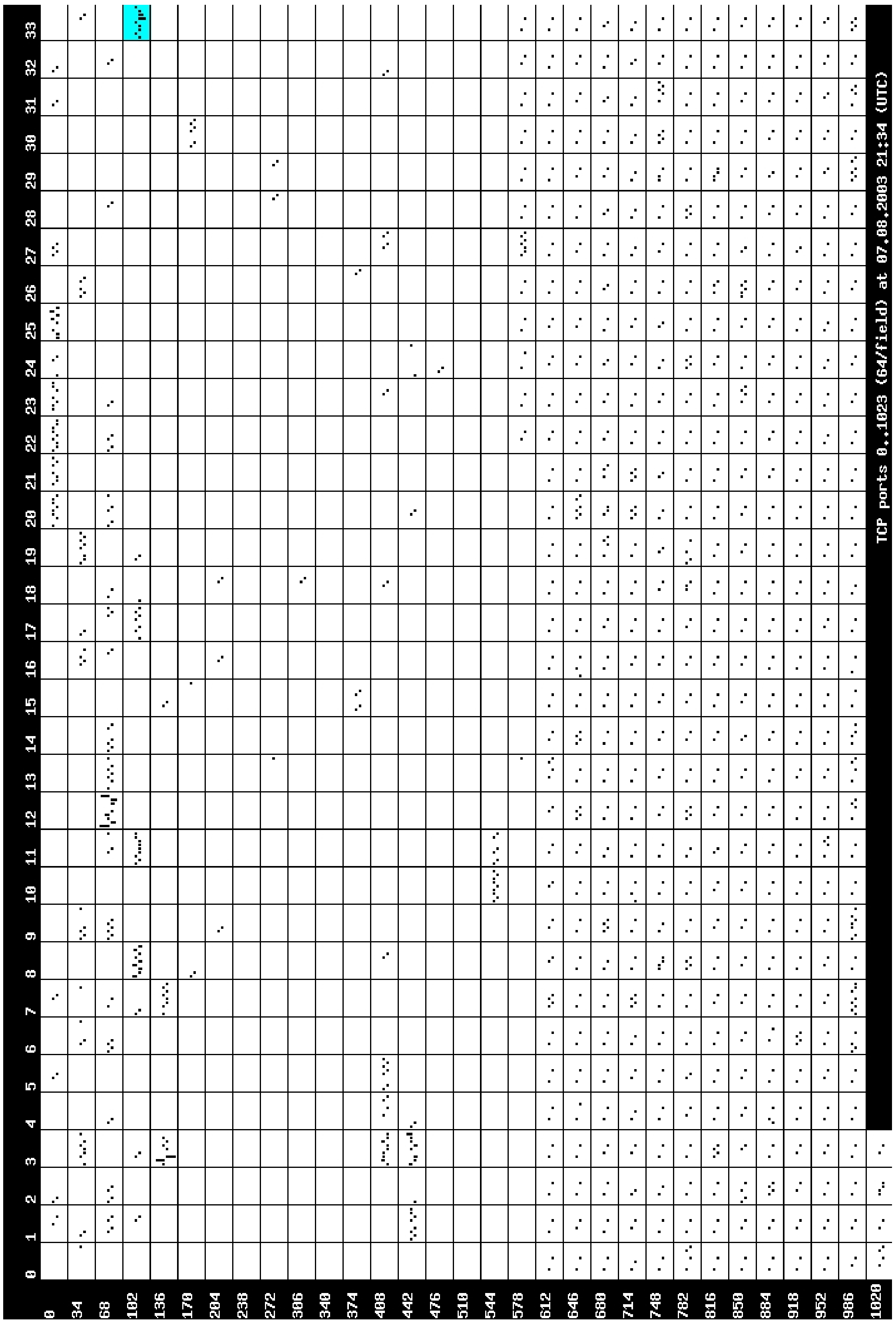


Figure 5.9: Port map probably showing a large scale port scan

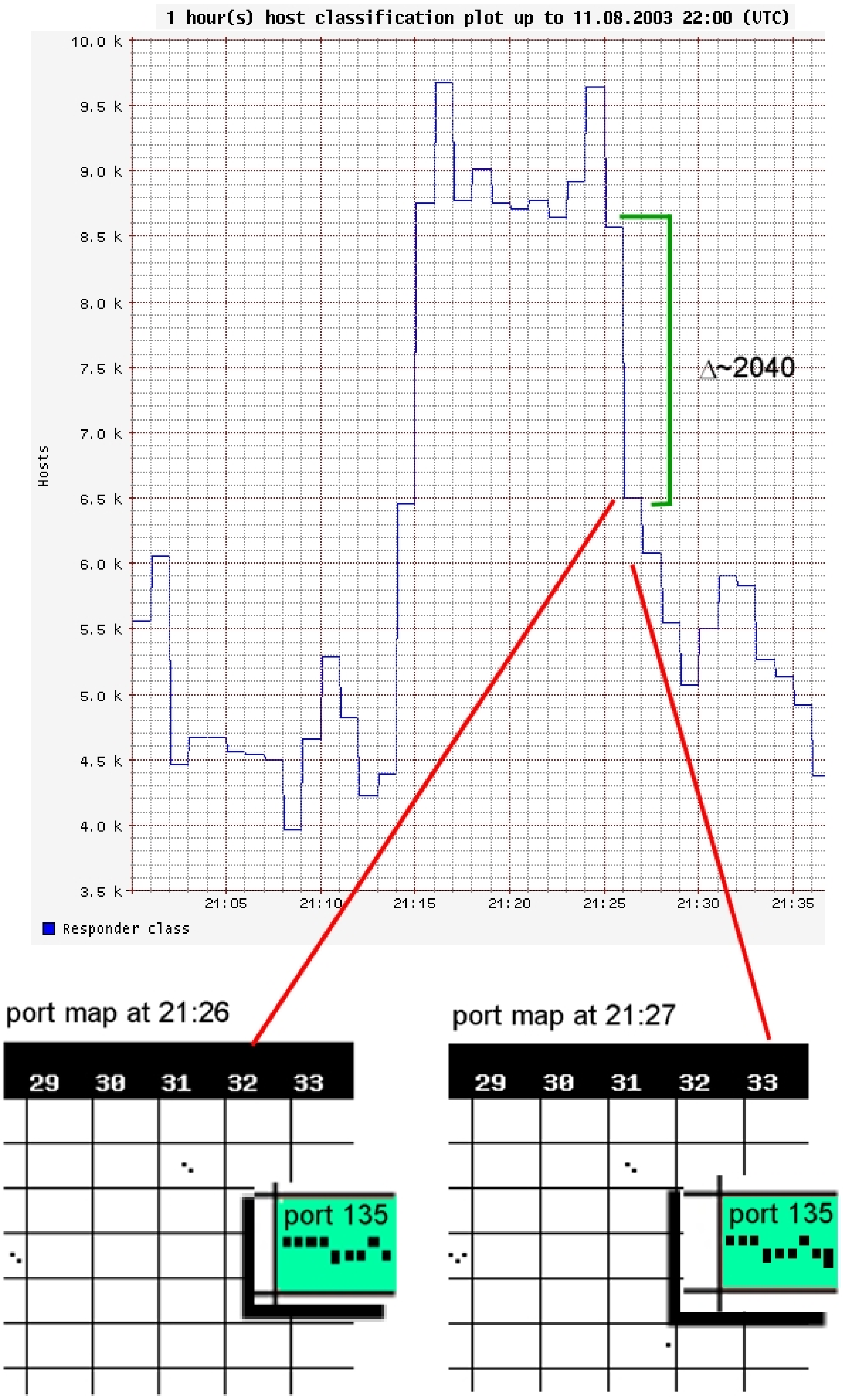


Figure 5.10: Combining the port map with the x-t plot

# Chapter 6

# Conclusion and Outlook

## 6.1 Conclusion

It could be shown that with appropriate data structures it is possible to implement an algorithm that is fast enough to do a near real time search to find for every acquired flow the corresponding flow in the other direction, even if the complexity of this task without the use of nested hash tables was $O\left(n^2 \times log\left(n\right)\right)$. Furthermore it could be shown that the algorithm can be used for near real time traffic data analysis even with the traffic generated during the W32.Blaster worm spreading. The moderate use of main memory and processing time would allow to analyse even more traffic while to ensure that not more memory than specified is used, a memory limit can be set.
The developed visualisation methods show significant changes during the W32.Blaster spreading whereby it could be shown that the defined characteristics provide significant information about the behaviour of hosts in the Internet. But still a lot could be done to improve the algorithm and the visualisation of the behaviour of the hosts. The following section explicates some of these improvements and proposes two applications for the host classificator plug-in.

## 6.2 Outlook

Since the task was to develop an analysis method that provides results in near-real-time, using only a limited amount of main memory and processing ressources, the tracking of the host behaviour over more than minutes was not applicable. But for a more detailed analysis of their behaviour, the tracking of every class change could be interesting. Only hosts that changed their class from interval to interval could be reported. Nevertheless it seems not to be very useful for the near real time algorithm because if a host is not active for more than two intervals[1] its class information is lost. If it becomes active again, it is reported as new host and a change in class is not detected. If near-real-time results are not required and a lot more resources were available, the interval length could be increased to improve the class change analysis. A first impression of an analysis based on reporting only the hosts that change their class can be gained easily. With slight changes in the source of the host classificator plug-in, only the number of hosts that changed their class are reported. These changes consist of a replacement of the counters written into the statistic file by the counters counting the hosts that were in the previous interval not in the same class as in the current interval.

### 6.2.1 Using the Host Classificator Plug-in Output for Anomaly Detection

Since the output of the host classificator plug-in is written to a file, it can be accessed by any other application. As it is already specified in Section 3.7, the file provides for each interval the number of hosts per class as much as the number of hosts per port that use this specific port for the first time[2]. Some of these parameters show significant changes when the data collected

[1]see Section 3.6 and Figure 3.3 for details
[2]for limitations and description see Section 3.6.2

during the outbreak of the W32.Blaster worm is replayed and analysed. Though the output could be used for anomaly detection, one should not neglect that the choice of the threshold value for the assignment of hosts to a class, can be critical for a successful detection. So if the output of the plug-in will be used for an automated anomaly detection, the evaluation of traffic with different thresholds should be an important part of the development process. Another topic in relation with the analysis of parameters could be to find a filter that reduces the amount of data to process, while eliminating none or only very few data that is critical to anomaly detection.

### 6.2.2 Further Use of the Connection Tracking Capability

The host classificator plug-in provides algorithms and data structures to search very efficient for special ’communication’ patterns[3] . Based on this algorithm a sort of virus scanner for the Internet could be developed. The scanner does not search, like a conventional virus scanner, for patterns in the data, but for special communication patterns.

## 6.3 Acknowledgements

Various helpful comments and ideas were received from Thomas Dübendorfer and Arno Wagner. Their support was always competent and swift. Furthermore, discussions with colleagues and fair comments of them created input to the work.

[3]a communication pattern is a specific sequence of connections which themselves are identified by the quadruple source IP address /destination IP address//source port/destination port

# Appendix A

# Appendix

## A.1 The Converter Tool

The converter tool converts the binary statistic files into human readable ASCII files so that an analysis without the visualisation software VISTOOL is possible. Since the information needed for an analysis of the port activity is part of one of the two possible file formats and because the port information consists of more than 130’000 numbers per interval, the option to ignore these information seems useful. Otherwise the ASCII files with more than 60 times 130’000 numbers are not much more human readable than the binary files. The other file format has only the information about one TCP port. It is used, if the host classificator plug-in is started with the -p parameter and a valid port number.
The following syntax is used to convert a file:

> *converter <filename> <Options>*
> *Options:* -p converts port information too (!large file if statistic file contains port information for all ports!)

The resulting file is names XXXX.conv where XXXX is the name of the binary file. It is stored in the directory where the converter tool is located. If already a file with that name exists, no output is generated but a warning.

## A.2 Host classificator Readme.txt

```
File: Readme.txt
Author: B.Tellenbach (bernhard.tellenbach@airmail.ch)
Requirements: see Installation.txt file.

Host classificator plug-in

This plug-in analyses the Netflow v5 data provided by the UPFrame
framework of Thomas Schlegel. The goal of the analysis is to characterise
each host that appears in the Netflow v5 data. This is done by assigning it
to one of the following eight classes:

   -Traffic class
   -Connector class
   -Responder class
   -Traffic and connector class
   -Traffic and responder class
   -Connector and responder class
   -Traffic, connector and responder class
   -no class
```

```
Additionally the port activity of each host is observed. Each port a host
did not use in the previous interval but does so in the current interval is
logged if the eight slots for the corresponding category (UDP/TCP
privileged/unprivileged) are not yet filled.

To save disk space, only the number of hosts in each class and the total
number of hosts using a specific port for the first time is stored in the
statistic file that contains one entry per interval. The interval length
is currently one minute. For details about the statistic file format see
http://www.tik.ee.ethz.ch/pub/students/2004-So/SA-2004-24.pdf.

The settings are defined in the host_classificator.cfg file. The
parameters are described there. Nevertheless a parameter can be
overwritten by providing it as argument when the host classificator
plug-in is started. The following parameters are possible:

     -f <number>   Filter flows with fewer packets than <number>
                   <number> must lie between 0..1000

     -h            Prints this message.

     -p <number>   Port mode. Saves only info for port <number>
                   in statistic file. <number> is 0..65535

     -r <path>     update rrd tool data, rrdtool at <path>
                   in a browser)

     -m <number>   set memory limit for application in  MBytes
                   ignores data received after limit is reached
                   but evaluates interval with data received up
                   to when the limit was reached.

     -w <command>  enable watchdog, restart command script <scr_name>

     -t <time>     watchdog timeout in seconds

     -b <path>     path to fifo

     -d     log to stdout

     -l <file>     log to file <file>

     -s            log to syslog
```

## A.3 VISTOOL Readme.txt

```
File: readme.txt
Author: B.Tellenbach (bernhard.tellenbach@airmail.ch)

Vistool

-----------------------------------------------------------
Installation instructions can be found in the file
installation.txt.
```

```
Since this tool depends on the CGIC library and the
GD library of Thomas Boutell, you have to accept the terms
and conditions set by him too. See the files license.txt and
support.txt for terms of use and support information.
------------------------------------------------------------

To use the vistool, only a web browser is needed and the
cookie set by the tool has to be accepted.
The vistool visualises the output of the host classificator
plug-in. Hence to analyse data with the vistool the Netflow
data has to be fed first to the host classificator plug-in
for processing. The plug-in classifies each host according to
its characteristics and stores how many hosts use a
specific port in the current interval they did not use in
the last interval. For more details see:
www.tik.ee.ethz.ch/pub/students/2004-So/SA-2004-24.pdf

The following analysis modes are available:

 -Venn diagram: Shows the number of hosts in each class
                for the current (in online mode) or
                the specified interval (in offline mode)
                as Venn diagram with logarithmic scaling.

 -XY-Plot: Plot the number of host in each class for
           the last x hours ending at the current time
           in online mode or at the time specified
           by the user.

 -Port map: Draw a port map that shows the port activity
            of the hosts for each port. For each port the
            difference of the number hosts of the current
            and the last interval, that sent for
            the first time packets with this port as source
            port, is plotted. For the first time means that
            a host did not use this port in the previous
            interval.

 All the three described modes can either work in online or
 offline mode.

 -online mode: The current time in UTC is used to fetch
               the data from the statistic files and a
               refresh rate can be set.

 -offline mode: The time used to fetch the data from the
                statistic files can be specified by the
                user.
```

## A.4 Provided Files

Directory structure and files provided with the host classificator plug-in and the VISTOOL.

The VISTOOL files:

```
./cgi:
vistool.c
```

```
./cgi/www:
cgi-bin  index.html  pmaplegend.html  pmaplegend.jpg

./cgi/www/cgi-bin:
cgi_config.cfg  venn.jpg
```

Other libraries needed by the vistool:

- cgic library
- gd library (requires: jpeg-6b, libpng and zlib libraries)

The host classificator files:

Base directory: UPFrame base directory

```
./include:
adminpage.h     getenvs.h                 libwrite.h    rawsendconfig.h
buffer.h        hashed_table.h            log.h         shmemblk.h
bufferpacket.h  hostinfo_hashed_table.h  message.h     sysvipc.h
cfile_tools.h   libnetflowutil.h          mysignal.h    udppacket.h
collection.h    libread.h                 netflow_v5.h
fifo.h          libwatchdog.h             rawsend.h

./plugins:
Makefile.hostcl  hashed_table.c              hostinfo_hashed_table.c
Readme.txt       host_classificator.c        starthostcl.sh_dist
converter.c      host_classificator.cfg_dist
```

In the include directory only the hostinfo_hashed_table.h and the hashed_table.h file were edited by the author. The other files are provided with the UPFrame distribution.

# Bibliography

[1] NetFlow FlowCollector Installation and User's Guide, 3.5
http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/...
...products_installation_guide_book09186a00800ed0ac.html
29.06.2004

[2] NetFlow Export Datagram Format. CISCO CNS NETFLOW COLLECTION ENGINE.
http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/...
...products_installation_guide_chapter09186a00800ed343.html#wp1006108
29.06.2004

[3] Jardas, P. (12.08.2003).
P2P Filesharing Systems: Real World NetFlow Traffic Characterization
*http://www.tik.ee.ethz.ch/∼ddosvax/sada/*
29.06.2004

[4] Racine, S. (03.11.2003).
Analysis of Internet Relay Chat Usage by DDoS Zombies
*http://www.tik.ee.ethz.ch/∼ddosvax/sada/*
29.06.2004

[5] Blaster Worm Analysis (11.08.2003)
eEye Digital Security
*http://www.eeye.com/html/Research/Advisories/AL20030811.html*
29.06.2004

[6] Weisskopf, A. (05.2004) Plug-ins for DDoS Attack Detection in Realtime
*http://www.tik.ee.ethz.ch/∼ddosvax/sada/*
10.07.2004

[7] Oetiker, T. (05.09.2003)
RRD Tool
*http://people.ee.ethz.ch/∼oetiker/webtools/rrdtool*
14.07.2004

[8] Georgios Androulidakis, Vasilis Chatzigiannakis et al. ( June 2004)
Network Flow-Based Anomaly Detection of DDoS Attacks
*www.terena.nl/conferences/ tnc2004/core_getfile.php?file_id=219*
20.07.2004

[9] Cristina Abad,Yifan Li,Kiran Lakkaraju,Xiaoxin Yin,William Yurcik (April 2004)
Correlation between NetFlow System and Network Views for Intrusion Detection
*www.ncassr.org/projects/sift/papers/icdm04.pdf*