Identifying short-term periodicities in Internet traffic

BSc thesis for Applied Mathematics & Telematics

University of Twente
Enschede - The Netherlands

Faculty of EEMCS

Author	:	Ivo Grondman
Date	:	December 15, 2006
Committee	:	dr. ir. P.T. de Boer dr. ir. G. Meinsma dr. ir. A. Pras

Abstract

Internet traffic shows periodicities over 24-hour and 7-day intervals, because of office hours, business days etc. This thesis tries to identify periodicities over shorter periods, in the order of seconds, tenths of seconds and so on. This is mainly done for UDP traffic on the Internet. Using Fourier analysis and available packet traces that were captured at several network locations with `tcpdump`, this thesis shows that these so-called "short-term periodicities" can indeed be identified. Furthermore, by looking at patterns (like peaks and notches) in the Fourier transforms of network traffic the behaviour in the time domain of this traffic is further analysed.

Acknowledgements

After quite some years of studying at the University of Twente, my mission of obtaining a Bachelor's degree in both Applied Mathematics and Telematics ends with this thesis. In these years, I have put in a lot of effort – especially in the last year – but I could not have done it without the help and support of others.

There are many people that I would like to thank, starting with my direct supervisors Pieter-Tjerk de Boer and Gjerrit Meinsma. Our weekly appointments were very helpful in the course of the assignment and a special thanks goes out to their immeasurable patience with me. In addition I would like to thank other people that were involved in the start up and approval of the assignment: Remco van de Meent, Aiko Pras, Jan Schut and Brigit Geveling.

Finally, I would like to thank my friends and family, for reading and correcting (the drafts of) this thesis and for their support in a year that was probably the toughest of all my years at the university.

Ivo Grondman

Contents

List of Symbols

Math symbol	C++ code	Description
t		Time
T		Period of a signal
Δt	`INTERVAL`	sampling period
n		Discrete time
s	`timestamp`	Time stamp of a packet
f		Frequency
ω		Angular frequency
v		Discrete frequency
N	`FFT_LENGTH`	Number of discrete time/frequency points
x		(Input) signal
y		(Output) signal
$H(\omega)$		Transfer function
$X(\omega)$		Fourier transform as a function of the angular frequency ω
$\tilde{X}(f)$		Fourier transform as a function of the frequency f
$X(v)$	`out`	Fourier transform as a function of the discretised frequency v

Chapter 1

Introduction

When designing data communication networks or devices for those networks, it is important to know the behavioural details of the traffic that will be carried by the network or passing through the devices. The first thing one might think of is, for example, the maximum possible traffic load that the network should be able to handle. Besides that, the variability of the traffic load is an interesting aspect. Figure 1.1 shows a typical pattern of traffic load on a network.

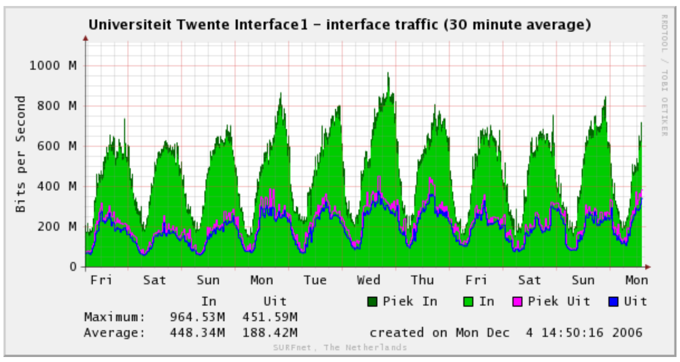

Figure 1.1: Traffic load on a network interface between the University of Twente and SURFnet.

Clearly the traffic load varies from time to time, but the same *pattern* of traffic load repeats day after day. This means that the traffic shows the same pattern over a 24-hour period. Moreover, the traffic load on weekends is on average somewhat lower than during business days: a 7-day period can therefore also be recognised in network traffic. The problem in this thesis is to identify periodicities in Internet traffic with periods that are much shorter than the two already mentioned, i.e. periods of seconds or tenths of seconds and shorter. These so-called short-term periodicities can play an important role in the evaluation of network (device) performance [6]: “Because the average of any periodic signal is lower than its peak, which means that a link with periodic traffic is under-utilized.”

1.1 Problem description

Researching short-term periodicities of Internet traffic, gives rise to the following main questions:

- How can short-term periodicities be detected?
- What kind of traffic is causing short-term periodicities?

If these questions are answered, answers to other, more simple, questions like:

- How much of the traffic is periodic?
- What periods are commonly found?

can be derived from the answers to the main questions.

1.2 Approach

Earlier work on short-term periodicities considers only specific types of traffic: [14] describes the traffic patterns (and periodicities) for a dynamic and heavily-accessed Web server environment, whereas [3] describes an application of using knowledge about short-term periodicities in TCP traffic in the defence against "Denial of Service" (DoS) attacks. Both [3] and [14] use statistical/stochastic methods as well as Fourier analysis to identify these periodicities. In this thesis, only Fourier analysis is adopted as a technique to identify periodicities. Fourier transforms are used to search a dominant frequency/periodicity in a packet trace, after which a binary search algorithm (also using Fourier transforms) is used to pinpoint the specific traffic that is likely responsible for this periodicity. It may seem odd to ignore the use of statistical/stochastic methods, but in the course of the research the sole use of Fourier analysis provided results that were good enough to come to some conclusions for the questions posed earlier. Another difference between this thesis and other work on short-term periodicities, is that it does not focus on one specific type of traffic (like traffic to a Web server or DoS attacks), but tries to identify periodicities caused by all kinds of protocols and services. The only limitation of this thesis is that there was not enough time to examine the traffic which belongs to protocols/services that operate on top of the Transmission Control Protocol (TCP). The protocols/services on top of the User Datagram Protocol (UDP) *are* examined and one trace is examined for periodicities in traffic which belongs to the Address Resolution Protocol (ARP). The Internet traffic that is used for the research are packet traces in the traffic repository [16], described in Section 3.2.

1.3 Thesis outline

The first few chapters of this thesis are of an introductory character. Chapter 2 gives a short introduction to Internet traffic: how applications generate traffic and how a packet of data on the Internet is assembled. This is necessary to show which information can be extracted from Internet traffic and how this information can be used in the analysis of short-term periodicities. Chapter 3 provides a rather elementary introduction to Fourier analysis, on which the rest of the thesis is heavily based. Furthermore, the technical details about the traffic repository that has been used are given, as well as an explanation about using Fourier analysis on the packet traces within the repository.
Chapters 4 and 5 show how ARP and UDP traffic are analysed and provide the results of this analysis. In the course of the research, a tool was developed to automate the process of identifying periodicities in packet traces. Chapter 5 provides a general introduction to this tool, whereas Chapter 6 discusses the implementation details of this tool.
After answering the questions stated above in Chapter 7, some recommendations for further research are given in Chapter 8.

Chapter 2

The Internet

When researching short-term periodicities in Internet traffic, one first has to understand what the Internet is and what sorts of traffic are sent over the Internet. This chapter will introduce the basics of the Internet and will give a short introduction to the protocols that play a role in this thesis.

2.1 Definition of the Internet

The Federal Networking Council (FNC) defines Internet to be the following:

> "*Internet* refers to the global information system that –
>
> 1. is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons;
> 2. is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and
> 3. provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein."

Although this definition was given by the FNC more than ten years ago, it is still accurate. A less formal definition of the Internet is that it is a global network that interconnects millions of devices (mainly computers) and that it enables these devices to communicate with each other, wherever they are in the world. The difference between the Internet of that time and the present time does not lie in the definition given above. It lies in the amount and type of machines and devices that are attached to this world wide network and the number of people that are using it. Those numbers have grown enormously in the past few years. Nowadays, many people have access to the Internet and they all use it in various ways. Computers still generate the most traffic on the Internet, although more and more services are facilitated by the Internet: digital television, Voice Over IP, WAP, GPRS, UMTS etc. are all techniques based on and making use of the Internet. With more people using the Internet and more services available on the Internet, the amount of traffic between devices on the Internet consequently also grows.
Traffic is generated when two hosts (which is a more general name for all the types of devices connected to the Internet) want to exchange information with each other (web pages, e-mails, multimedia streams, etc.). To be able to send and receive information between two hosts, *protocols* are needed. More information on the protocols observed in this thesis is given in the next section.

2.2 Protocols

Devices connected to the Internet are made by a large number of different manufacturers. To make communication between different devices possible, these manufacturers all have to implement the same kind of "language" in their devices to make sure that the devices understand each other. These languages are called *protocols*. In this section a few important protocols that form the basis of communication through the Internet are described. Only a description on *what* protocols do and how they interact with each other will be given. Information about the more technical side of *how* protocols do their job and about the functioning of routers, switches etc. is explained in associated RFC's[1] and books like [9], [11] and [15].

2.2.1 TCP/IP reference model

To start off with, consider the TCP/IP reference model in Figure 2.1, on which the functioning of the Internet (and its attached devices) is based. This model has four layers, which are based on the following

Application Layer
Transport Layer
Internet Layer
Host-To-Network Layer

Figure 2.1: The TCP/IP Reference Model

principles:

- When another level of abstraction is needed, a layer should be created.
- Every layer should have a well described function.
- The function of each layer should be chosen in such a way that certain protocols can be defined (and internationally standardised) to operate on that layer only.
- The amount of information that has to be passed through the interfaces between different layers should be as small as possible.
- The number of layers should be big enough so that different functions will not have to be in the same layer, but small enough so that the TCP/IP architecture becomes too extensive.

As said, each layer has its own function. To be able to carry out this particular function, a layer depends on the lower level layer. In this model, the shortest explanation of what the functions in each layer are and how they depend on each other is that the Application layer determines what *type* of information is exchanged (web page, mail, audio/video stream etc.). The Transport Layer then determines *how* this information should be exchanged (in practice this means a choice between reliable or unreliable transport). The Internet Layer determines *where* from and to it is exchanged and finally, the Host-To-Network Layer takes care of the physical sending of the information. This model is used to show what actions are undertaken to send information from one host to another in the next sections.

2.2.2 Application Protocols

When two hosts are exchanging information on the Internet, they make use of certain protocols in the Application Layer, of which some are given in Table 2.1. More application protocols exist of course, but these protocols are known by almost everyone that uses the Internet. Now suppose a web browser is used to request a web page from a web server of which the address `http://www.utwente.nl` is known. The type of information that is desired (a web page) and the application (a web browser) that is being used, already determine what protocol is used in the Application Layer, since this is the protocol that is implemented by the application. According to Table 2.1 the HyperText Transfer Protocol (HTTP) will be used to fetch web pages when using a web browser. The Application Layer takes care of constructing

[1]RFC stands for Request For Comments: a document containing (technical) details about a certain protocol or service.

Application	Protocol
Web Browser	HTTP
E-mail Client	POP / SMTP
File Transfer Client	FTP

Table 2.1: Examples of applications and their related protocols.

a correct HTTP request for the web server. This request then still has to be sent to the web server. This is *not* the task of the Application layer, so the information is passed on to the underlying layer together with the address information: the Transport Layer. In the Internet two protocols, TCP and UDP are defined on this layer. Whether TCP or UDP is chosen depends on the type of application used. The big difference between these two is reliability, which will be further explained in the following sections.

2.2.3 Transmission Control Protocol

The reliable transport protocol is called the Transmission Control Protocol (TCP). The protocol is connection oriented, which means that before hosts can send data to each other, a connection should be explicitly set up between the two. This process of setting up the connection is called “handshaking”. The reason for this handshake is that both hosts have to initialise many state variables, which are used to guarantee the reliability of this protocol, which will be shown later. When the connection is set up, both hosts can start sending data. The reliability of this protocol is achieved in the following way: when a host sends information to another host, the sending host will get notified by an ACK packet from the receiving host if the information was received. Figure 2.2 shows that when host A sends host B some

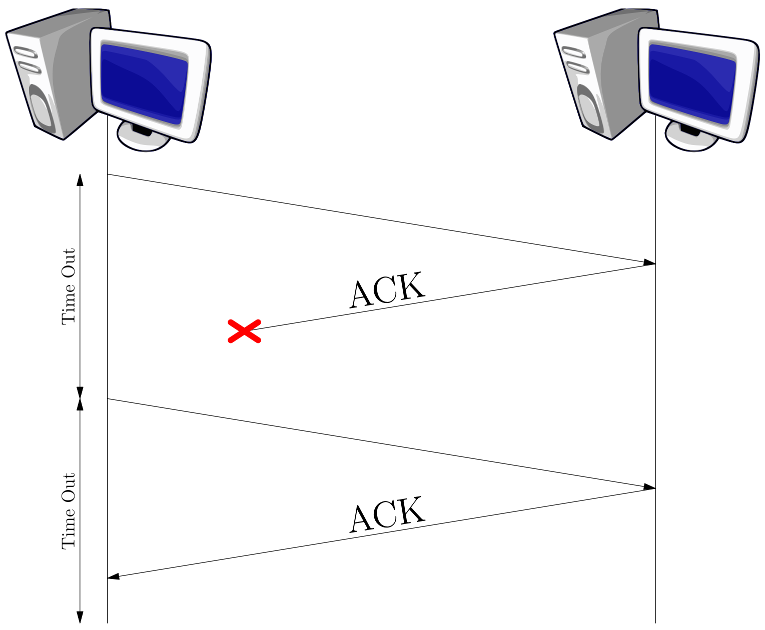

Figure 2.2: Example of the TCP protocol retransmitting a packet.

information, a timer is set at host A. If the ACK packet from host B is not received within a certain amount of time, host A will assume that host B did not receive the packet and will retransmit it (even if host B *did* receive the packet, because host A simply does not have proof of it). These (sometimes necessary) retransmissions result in both hosts having to do some bookkeeping to keep track of the

packets that are received, to be received, to be acknowledged etc., which explains the need for setting all those state variables with a handshake procedure. Typical applications that make use of the reliability property of the TCP protocol are web browsers (as in this example), mail clients and file transfer clients, since the information exchanged by those applications should be complete and without errors. A high transfer speed is always appreciated of course, but in applications that use the TCP protocol speed is not the most important criterion.
In Figure 2.3 the structure of a TCP packet is given. The *Destination Port* field is used to indicate

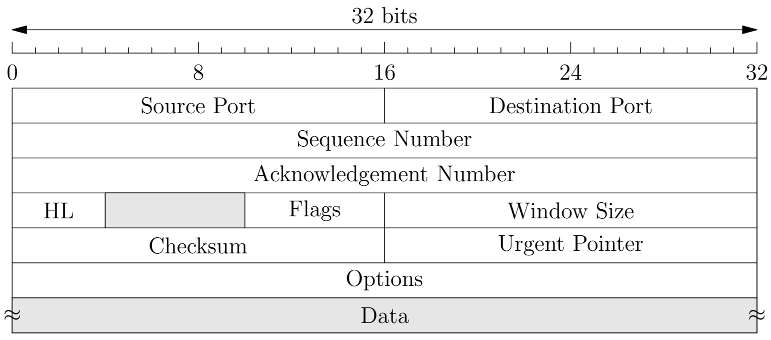

Figure 2.3: Structure of a TCP packet.

what upper layer application (protocol) should process the data contained in the TCP segment. For well known application protocols, standard port numbers are assigned by the Internet Assigned Numbers Authority (IANA). For this example, the typical destination port number would be 80, which is the standard port number on which web servers operate. The *Source Port* field gives information about the upper layer application (protocol) that sent this TCP packet. From the TCP header, only these port fields are the only two fields that are of interest in this thesis. The *Data* field contains the information that was passed on from the upper layer service, in this example an HTTP request. A description of the rest of the fields (which also includes fields which ensure the reliability of the TCP protocol) is given in RFC 793, which is the full specification for The Transmission Control Protocol.

2.2.4 User Datagram Protocol

The User Datagram Protocol (UDP) is a connection-less transport protocol, which means that applications can send information to each other at will, without having to do a handshake with the destination host. This handshake is not needed, since the UDP protocol is not a reliable protocol: information about whether or not the destination host received the information is not given, so no bookkeeping has to be done about this and therefore no state variables have to be initialised. The simplicity of the UDP protocol also simplifies the structure of a UDP header, as can be seen in Figure 2.4. Applications that make use of UDP, rather than TCP, are applications that want to profit from high transfer speeds in exchange for less reliability, e.g. real time and streaming applications like Internet telephony software.

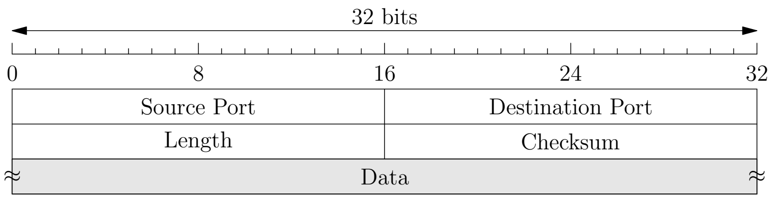

Figure 2.4: Structure of a UDP packet.

Using a less reliable protocol for multimedia streams even has an advantage over reliable protocols: a stream over a reliable connection through TCP would result in a stream with hick ups, because the playback will pause if packets are not received in the right order, due to lost and retransmitted packets. Less reliable protocols will just forget about the lost packets and the playback continues anyway. Furthermore, occasionally lost packets hardly have any disturbing effect on the quality of the playback. Like in TCP, the *Data* field contains the information from the upper layer service. The *Destination Port* and *Source Port* field will again be of interest in this thesis. A full description of the UDP protocol is given in RFC 768.

2.2.5 Internet Protocol

When the information that has to be sent is mapped on TCP or UDP packets, these segments/packets again have to be mapped to Internet Protocol (IP) packets. The Internet Protocol is the most important protocol used in the exchange of information on the Internet, since it is this protocol that makes the actual sending of information from a source to a destination anywhere on the Internet possible. Global unique addressing is one task of the Internet Protocol, as well as getting the packet from the source host to the destination host. Every host on the Internet has one or more IP addresses assigned to it, to which information in the form of IP packets (also called IP datagrams) can be sent. The information in an IP packet is given in Figure 2.5. In this thesis only the *Protocol*, *Source IP Address* and *Destination*

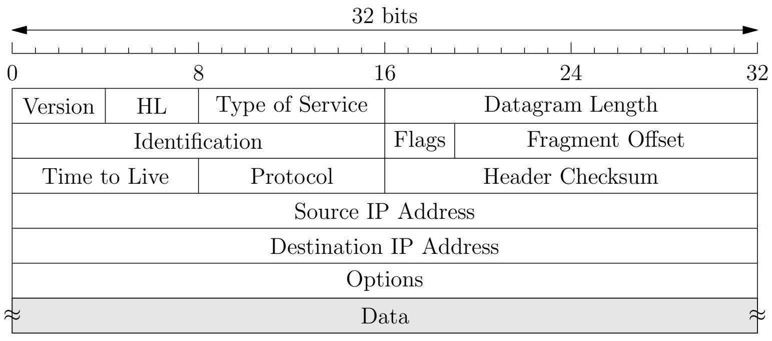

Figure 2.5: Structure of an IP packet.

IP Address fields from the IP header are used. The protocol field tells what upper layer protocol (TCP or UDP) should handle the further processing of this packet upon delivery at the destination host, i.e. it tells what kind of packet the *Data* field is holding. The source and destination IP address fields tell from which host the packet originates and to which host it should be delivered. Since only the address `http://www.utwente.nl` is known, another (application) protocol is called upon to resolve the hostname `www.utwente.nl` to an IP address. The protocol which resolves hostnames to IP addresses and the other way around is the Domain Name System (DNS) protocol (RFC 1034/1035). IP address information gathered with the help of this protocol is used in the IP header. A full description on all the fields in an IP packet is given in RFC 791.

The IP header can be thought of as an envelope and the data/payload as a letter you would put in that envelope, just as in regular mail: the information from the upper layer protocol is put into the payload, and the destination address, which was already known in the Application Layer, is “written” in the IP header. The IP protocol can then take care of the delivery of this packet. It could be that the two hosts that are exchanging information are not directly connected with each other. Figure 2.6 shows a simple example of a packet that has to be sent from one host to another, which are connected by one router. When the IP packet is sent out, the router will pick it up and forward the message to the correct host. To do this, the router does not have to know anything about the data that is sent, nor about the transport protocol that is used. It only has to look in the IP header to find out which host should receive this packet. This is why a router lacks the implementation of an Application Layer and

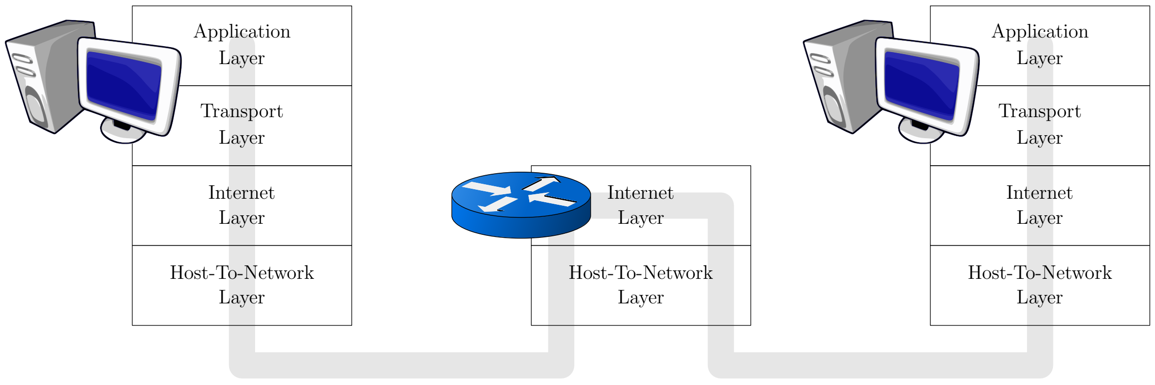

Figure 2.6: A router between two hosts only uses two layers.

a Transport Layer: the Internet Layer already gives the information needed for the router to do its job. So the Internet Protocol is not only taking care of the unique addressing of hosts, it also facilitates the correct forwarding of packets through network devices like routers, switches etc.
Now that the information is wrapped in the right packets and addressed to the appropriate destination host, it can still not be sent out on to the Internet *physically*. Devices on the Internet are all making use of IP addresses, but these are just virtual addresses mapped on the hardware that is actually connected to the Internet. Returning to the TCP/IP Reference Model in Figure 2.1, the IP addresses belong to the Internet Layer. In practise, though, devices communicate on the lower Host-To-Network Layer, by means of network adaptors which use another kind of addresses. In most modern day computers, these addresses are Ethernet addresses (also called MAC addresses). For the physical sending of information, the Ethernet address (which corresponds to the IP address the information should be sent to) should be known. This mapping from IP address to Ethernet address is taken care of by the Address Resolution Protocol, discussed in the next section.

2.2.6 Address Resolution Protocol

Suppose host A has to send out a packet to a certain host B on the same subnet or LAN (see Figure 2.7). For the physical sending of the packet, knowing the IP address is not enough, since the network interface which is sending out the packets, makes use of Ethernet addresses. An ARP module in the network interface can resolve IP addresses to Ethernet addresses by broadcasting a request to machines on the same subnet. In this example, host A would send out a request asking to which Ethernet address the IP address 192.168.0.2 is mapped. Host B would then send a reply to host A, mentioning its Ethernet address. Both hosts now have information on each others IP/Ethernet addresses stored in their ARP table. At that point, host A can finally start transmitting its information to host B by putting the IP packet into a so called Ethernet frame (see Figure 2.8), which results in Figure 2.9. This Ethernet frame is then what is actually sent out by the network adaptor.

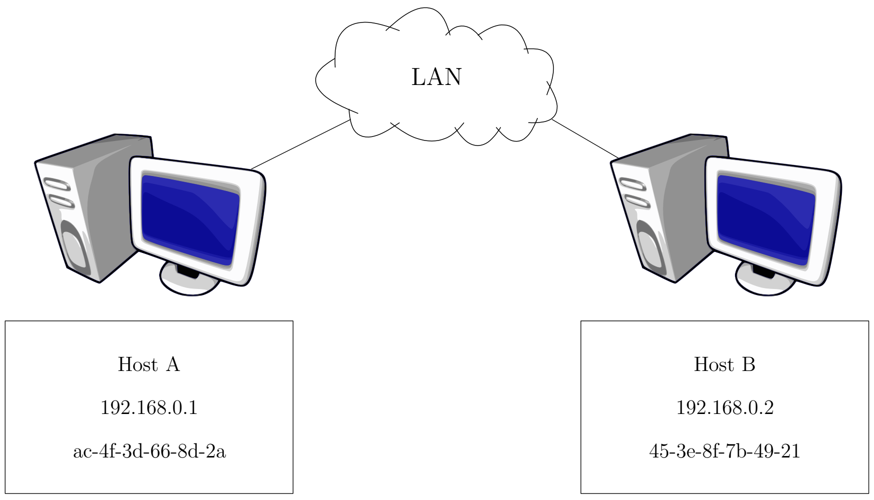

Figure 2.7: Two hosts on the same subnet

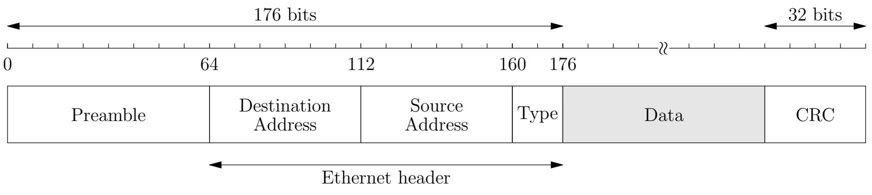

Figure 2.8: Structure of an Ethernet frame.

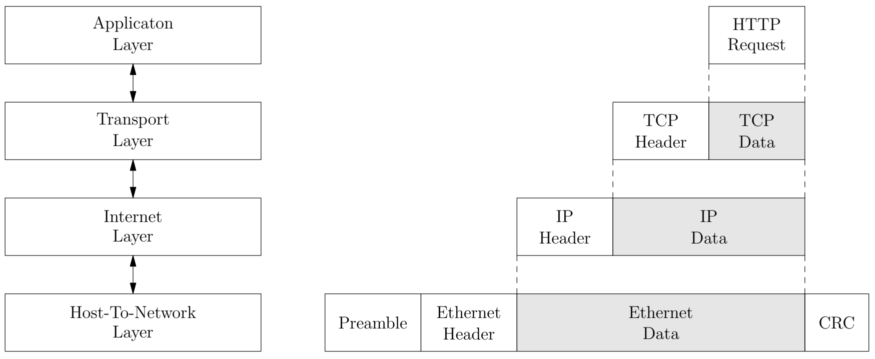

Figure 2.9: Ethernet frame containing an HTTP request.

Chapter 3

Technical Background

3.1 Fourier analysis

Since a lot of work done in this thesis involves Fourier transforms, this section will explain what Fourier transforms actually are. The first sections will discuss two kinds of representation (called domains) of a signal. Subsequently, Fourier transforms will be explained and the type of transform that is used in this thesis is explained more thoroughly.

3.1.1 Time domain

The time domain is the most natural representation of a signal. In this domain, the value of a signal is given with respect to time. As an example, take a simple sinusoid given by

$$x(t) = \cos(\pi t) \tag{3.1}$$

in Figure 3.1. From the figure the value of the signal at some point in time can be determined. At $t = 1$, the value is -1, at $t = 2$, the value is 1 and so on.

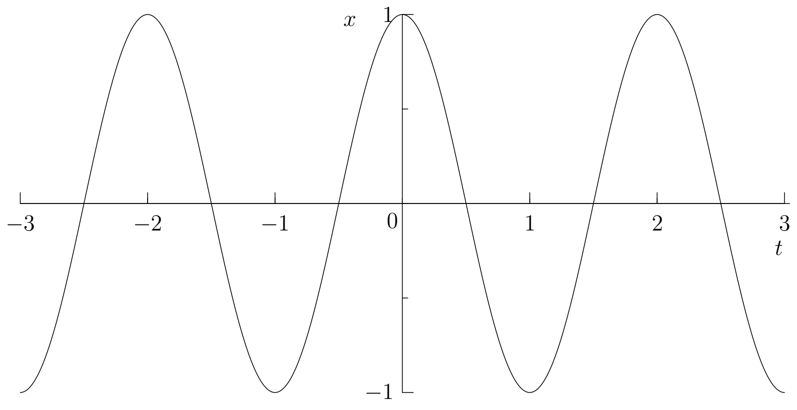

Figure 3.1: Signal (3.1) in the time domain

3.1.2 Frequency domain

While the time domain is the most natural and intuitive representation of a signal, it is not always the easiest domain to interpret when analysing a signal. This thesis is about gathering information on periodicities in a signal and therefore information about frequencies present in a signal. The frequency domain gives this information in an explicit way, whereas the time domain only gives it in an implicit way. In the example it is easy to see that the signal has a frequency of $\frac{1}{2}$ Hz, with a maximum amplitude of 1: a full period of the signal covers 2 seconds, which gives a frequency of

$$\frac{1 \text{ period}}{2 \text{ seconds}} = \frac{1}{2} \text{ Hz}$$

A plot of this signal in the frequency domain is given in Figure 3.2. It shows directly which frequency

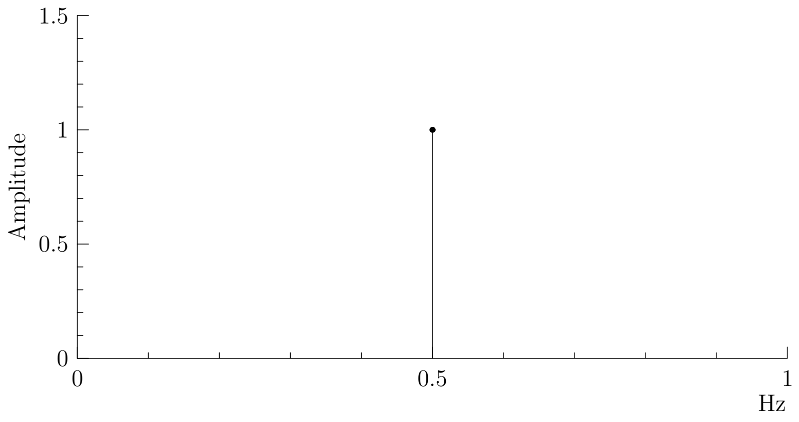

Figure 3.2: Signal (3.1) in the frequency domain

(together with its amplitude) is present in the signal, without the need of further calculations.

In the previous example one thing is overlooked: the signal was transferred from the time domain to the frequency domain only by looking at the frequency and the amplitude of the signal. But what happens if the signal is shifted (in the time domain) somewhat to the left or to the right? It would still be a signal with a frequency of 0.5 Hz and an amplitude of 1, so the amplitude plot in the frequency domain would look the same. Apparently, some extra information should be given in the frequency domain. The phase of a sinusoid gives information about its shifting to the left or right. To clarify this, another example is given in which a signal x is comprised of three different sinusoids:

$$\begin{aligned}
x_1(t) &= 2\cos(\pi(t + \tfrac{1}{2})) \quad &(3.2)\\
x_2(t) &= 5\cos(4\pi(t - \tfrac{1}{4})) \quad &(3.3)\\
x_3(t) &= 3\cos(\tfrac{4}{3}\pi(t + 1)) \quad &(3.4)\\
x(t) &= x_1(t) + x_2(t) + x_3(t) \quad &(3.5)
\end{aligned}$$

Figure 3.3 gives the plot in the time domain of signal x. To produce a plot in the frequency domain, three things about all the sinusoids that are in x should be known: amplitude, frequency and phase. To do this in the most easy way the decomposition of a signal should already be known, since the needed information is not easily extracted from Figure 3.3. In this example, Equations (3.2) to (3.4) give the decomposition of the signal x and the frequency domain information can be extracted from these equations. Rewriting these equations in the form

$$x_k(t) = a\cos(2\pi f t + \phi)$$

gives the information that is needed to reconstruct the signal in the frequency domain: for the frequency

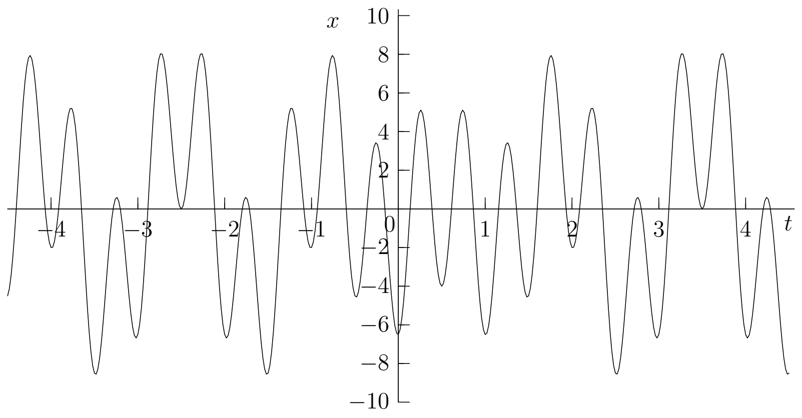

Figure 3.3: Signal x is built up from 3 different sinusoids

f, the amplitude is a and the phase is ϕ. Equations (3.2) to (3.4) become

$$x_1(t) = 2\cos(2\pi \cdot \tfrac{1}{2} \cdot t + \tfrac{1}{2}\pi) \tag{3.6}$$
$$x_2(t) = 5\cos(2\pi \cdot 2 \cdot t - \pi) \tag{3.7}$$
$$x_3(t) = 3\cos(2\pi \cdot \tfrac{2}{3} \cdot t + \tfrac{4}{3}\pi) \tag{3.8}$$
$$x(t) = x_1(t) + x_2(t) + x_3(t) \tag{3.9}$$

It is common use that the phase shift of a sinusoid lies in the interval $[-\pi, \pi)$. If the phase is not in this range, multiples of 2π should be added or subtracted to get it in that interval. This is justified by the fact that any 2π-multiple shift of a sinusoid, results in the same sinusoid. Since the phase of signal x_3 is beyond the interval $[-\pi, \pi)$, it should be reduced by 2π, which gives $-\frac{2}{3}\pi$ as the phase. The frequency domain information from the three signals is collected in Table 3.1. With the information available from this table, a amplitude and phase plot can be made (see Figures 3.4 and 3.5).

Signal	Amplitude	Frequency (Hz)	Phase
x_1	2	$\frac{1}{2}$	$\frac{1}{2}\pi$
x_2	5	2	$-\pi$
x_3	3	$\frac{2}{3}$	$-\frac{2}{3}\pi$

Table 3.1: Information about the three sinusoids used for the signal in Figure 3.3

3.1.3 Fourier transforms

So far, a signal can only be transferred from the time domain to the frequency domain in simple cases: either the signal only has one frequency present (Figure 3.1) or – if more frequencies are present (Figure 3.3) – the decomposition should be known in advance. When a signal in the time domain looks like Figure 3.6 it is virtually impossible to extract frequency domain information from the figure. Because the decomposition also is not known – which is the case in most real life situations – any periodic behaviour in the signal can not be determined, although there could be periodicities present. Apparently, better means are needed to discover the frequencies present in a signal like the one presented in Figure 3.6. Baron Jean Baptiste Joseph Fourier (1768-1830) provided such a mechanism. He introduced the idea

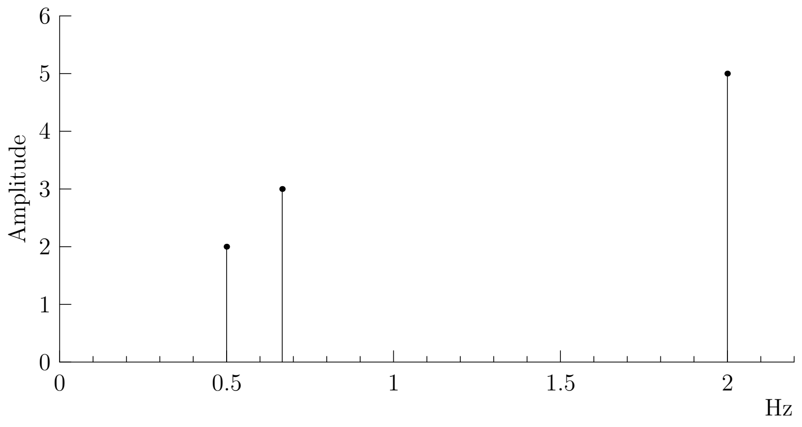

Figure 3.4: Amplitude plot for the signal in (3.5).

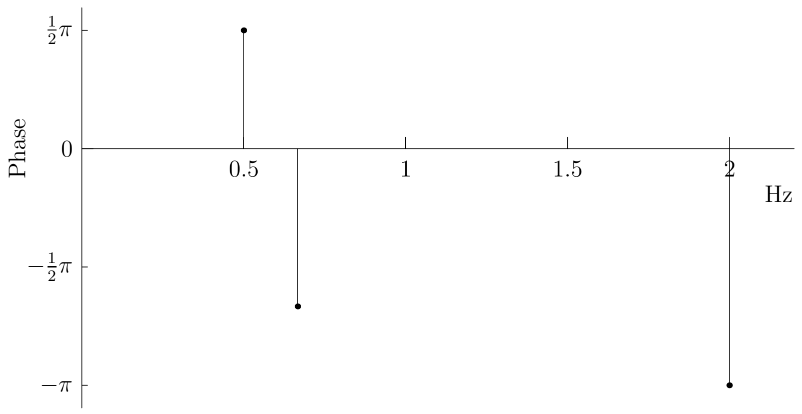

Figure 3.5: Phase plot for the signal in (3.5).

that any periodic function, or a function on a finite interval could be represented by a sum[1] of sines and cosines in the following way:

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\{a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)\}, \tag{3.10}$$

where

$$\begin{aligned} \omega_0 &= \frac{2\pi}{T} && (3.11)\\ a_k &= \frac{2}{T}\int_0^T x(t)\cos(k\omega_0 t)dt && (3.12)\\ b_k &= \frac{2}{T}\int_0^T x(t)\sin(k\omega_0 t)dt && (3.13)\\ & && (3.14) \end{aligned}$$

[1]In fact, Leonhard Euler had already published the expression

$$\frac{\pi}{2} - \frac{x}{2} = \sin(x) + \frac{1}{2}\sin(2x) + \frac{1}{3}\sin(3x) + \ldots$$

in 1744.

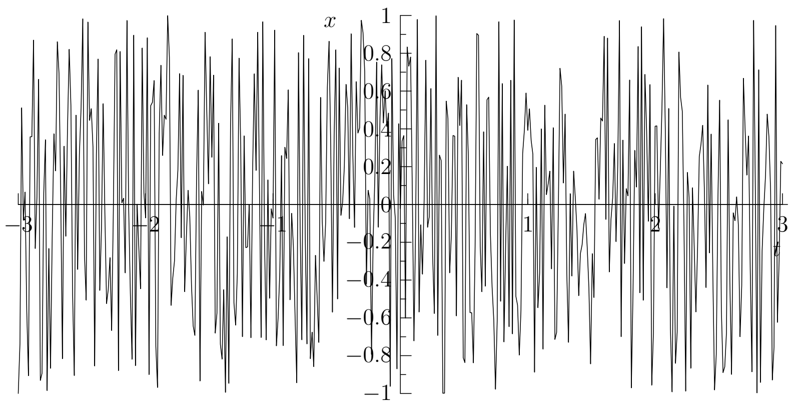

Figure 3.6: A signal in the time domain

In these equations, T is the period of the signal $x(t)$ and ω_0 is called the fundamental frequency of the signal. Integral multiples of ω_0 – here denoted with $k\omega_0$ – are called the harmonics of ω_0. The limits on both integrals go from 0 to T, but this is not necessary. This integration can also go from $-T$ to 0, or from $-\frac{1}{2}T$ to $\frac{1}{2}T$. As long as the integration takes place over one full period, the limits can be chosen freely. This can especially be helpful when integrating over a function with discontinuities. To

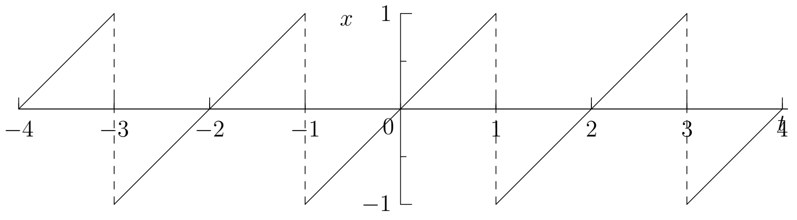

Figure 3.7: Sawtooth function

get a better feel for this, this is illustrated with the sawtooth function depicted in Figure 3.7. Although this function is *not* continuous at all points, it can be expressed by a sum of sines and cosines which *are* continuous. When looking at the sawtooth function, it is clear that the period T of the signal is 2 seconds. This immediately gives the fundamental angular frequency ω_0, since:

$$\omega_0 = \frac{2\pi}{T} = \frac{2\pi}{2} = \pi.$$

Equations (3.10)-(3.13) now become:

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\{a_k \cos(k\pi t) + b_k \sin(k\pi t)\} \quad (3.15)$$

$$a_k = \int_0^2 x(t)\cos(k\pi t)dt \quad (3.16)$$

$$b_k = \int_0^2 x(t)\sin(k\pi t)dt \quad (3.17)$$

The next step is to calculate the coefficients a_k and b_k. Notice that the coefficients a_k do not have to be calculated at all: the sawtooth function is an odd function, just like sines are. Adding cosines to the expression would disturb this property of the sawtooth function. Hence,

$$a_k = 0 \qquad (\forall k \in \mathbb{Z}).$$

Calculating the coefficients b_k gives:

$$\begin{aligned} b_k &= \int_0^2 x(t)\sin(k\pi t)dt \\ &= \frac{2\cdot(-1)^{k+1}}{k\pi} \qquad (\forall k > 0) \end{aligned}$$

Now that the coefficients are calculated, Equation 3.15 can be filled in and the expression of the sawtooth function becomes:

$$x(t) = \frac{2}{\pi}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}\sin(k\pi t)}{k} \tag{3.18}$$

On page 17 the Figures 3.10 to 3.14 show the result of this approximation for respectively 1, 2, 3, 4 and 10 sine terms. It is clear that Equation (3.18) approaches the sawtooth function when more harmonic frequencies are added. The above is actually an example of a Fourier *series*: starting off with the fundamental angular frequency ω_0 of the signal $x(t)$, the signal can be expressed in terms of sines and cosines with this fundamental frequency and its harmonics. Subsequently the frequency domain information of the sawtooth function is known (see Figures 3.8 and 3.9).

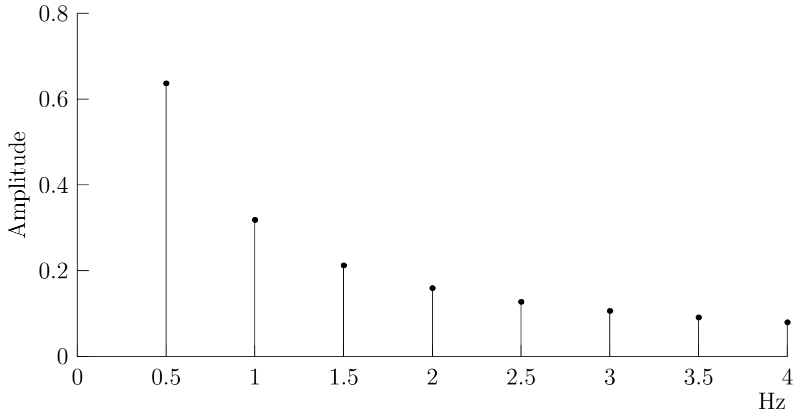

Figure 3.8: Amplitude plot for the sawtooth function

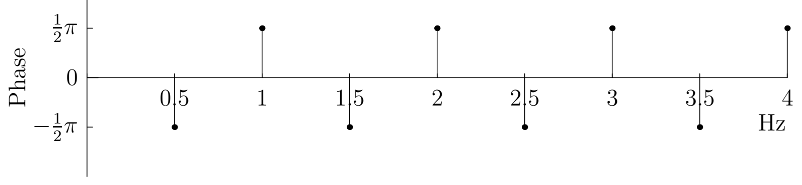

Figure 3.9: Phase plot for the sawtooth function

Although this is another way to map a signal from the time domain to the frequency domain, this is still not a solution to the problem of determining periodicities in *all* possible signals: to develop the Fourier series the fundamental frequency should be determined, but this can be an impossible job for functions

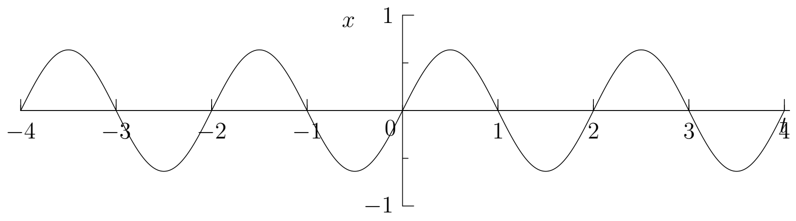

Figure 3.10: Approximation of the sawtooth function with only the fundamental frequency

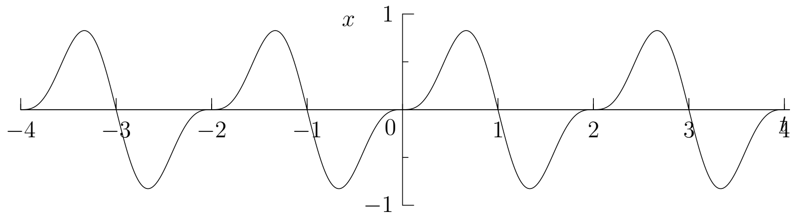

Figure 3.11: Approximation of the sawtooth function with 2 harmonics

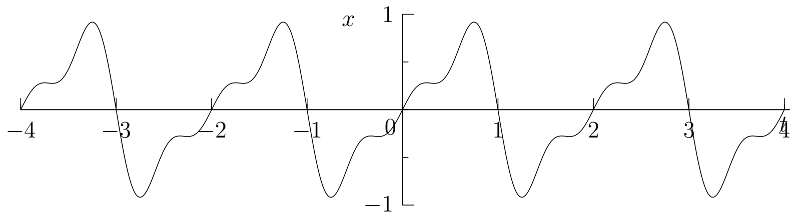

Figure 3.12: Approximation of the sawtooth function with 3 harmonics

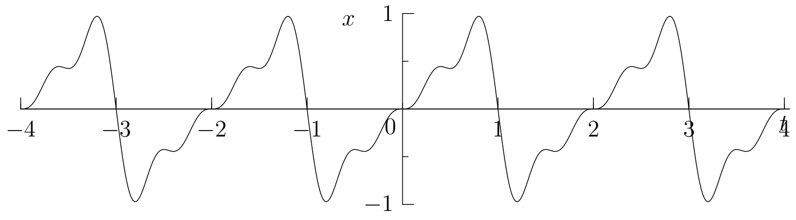

Figure 3.13: Approximation of the sawtooth function with 4 harmonics

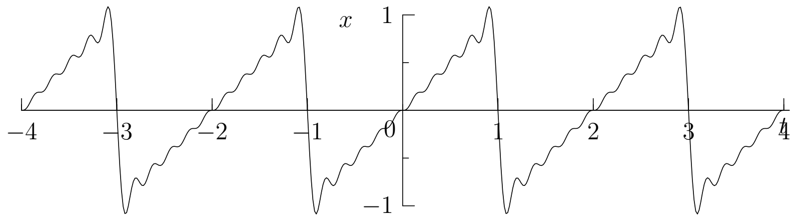

Figure 3.14: Approximation of the sawtooth function with 10 harmonics

where this fundamental frequency is not easily seen (consider Figure 3.6 again). A better approach would be some kind of function that maps a signal from the time domain to the frequency domain in a more explicit way. This is achieved by the (continuous) Fourier *transform*:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt \tag{3.19}$$

If the continuous signal in the time domain $x(t)$ is known, the frequency domain representation of this signal can be determined by solving the integral in Equation (3.19). Again this seems a good solution to the problem, but to solve this equation the function $x(t)$ should be known. In practise, only a sampled discrete function $x[n]$ of the actual function $x(t)$ is known. Sampling is done by measuring the value of the signal $x(t)$ at discrete-time points $t_n = n\Delta t$, where $n = \ldots, -2, -1, 0, 1, 2, \ldots$. The sampling period is denoted by Δt. A formal definition of sampling then is:

$$x[n] = x(t)|_{t=n\Delta t} = x(n\Delta t) \qquad (\forall n \in \mathbb{Z})$$

In Table 3.2 an example is given for the $\sin(t)$ function, which is sampled over one period of 2π, with a sampling period T of $\frac{1}{10}\pi$ seconds. This gives 20 sampled values. Notice that the last sample does not

Sample	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$	$x[8]$	$x[9]$
Value	0	0.3090	0.5878	0.8090	0.9511	1	0.9511	0.8090	0.5878	0.3090
Sample	$x[10]$	$x[11]$	$x[12]$	$x[13]$	$x[14]$	$x[15]$	$x[16]$	$x[17]$	$x[18]$	$x[19]$
Value	0	-0.3090	-0.5878	-0.8090	-0.9511	-1	-0.9511	-0.8090	-0.5878	-0.3090

Table 3.2: The function $\sin(t)$ samples over one period with 10 samples.

have the value zero, since this is a value of the *next* period of the function. In Figure 3.15, the discrete

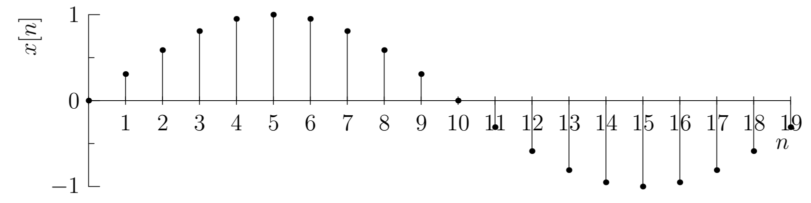

Figure 3.15: A plot of the sampled $\sin(t)$ function

signal is plotted. The shape of a sine function is clearly visible. These samples can then be used in a Discrete Fourier Transform (DFT), which is discussed in the next section. This enables the transforming of a (continuous) signal, of which the exact function $x(t)$ is unknown. A disadvantage of sampling is the loss of information: the value of the signal is not known for all moments in time. The consequence of this is that the accuracy of the DFT depends strongly on the chosen sampling period. Of course, the DFT can also be used on signals that are already discrete by their nature.

3.1.4 Discrete Fourier Transform

For finite discrete signals the Discrete Fourier Transform, or DFT in short, is available. The Discrete Fourier Transform is given by:

$$X[v] = \sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi}{N}vn} \qquad (v = 0, 1, \ldots, N-1) \tag{3.20}$$

In this equation, v stand for the discretised frequency. The actual frequency f depends on the sampling period Δt that was used to generate the discrete signal or time series and the total number of samples N:

$$f = \frac{v}{\Delta t N} \tag{3.21}$$

If $X[v]$ is the N-point DFT of $x[n]$, then $x[n]$ can be obtained from $X[v]$ by applying the inverse DFT given by

$$x[n] = \frac{1}{N}\sum_{v=0}^{N-1} X[v] e^{i\frac{2\pi}{N}vn} \qquad (n = 0, 1, \ldots, N-1) \tag{3.22}$$

Using Euler's identity

$$e^{i\phi} = \cos(\phi) + i\sin(\phi)$$

Equation 3.22 expands to

$$x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[v]\left\{\cos(\frac{2\pi}{N}vn) + i\sin(\frac{2\pi}{N}vn)\right\} \qquad (v = 0, 1, \ldots, N-1) \tag{3.23}$$

and again a sum of sines and cosines shows up, as seen before. Now suppose the DFT for the signal described by Table 3.2 and Figure 3.15 would be calculated. A quick look at Equation (3.20) shows that for the full DFT N elements will have to be calculated. Each element itself needs the full vector of samples, which has length N. Therefore, the calculation of a DFT will roughly take something in the order of N^2 multiplications. Calculating the DFT for the example in Figure 3.15 would already cost about four hundred multiplications to come up with the information shown in Figure 3.16.

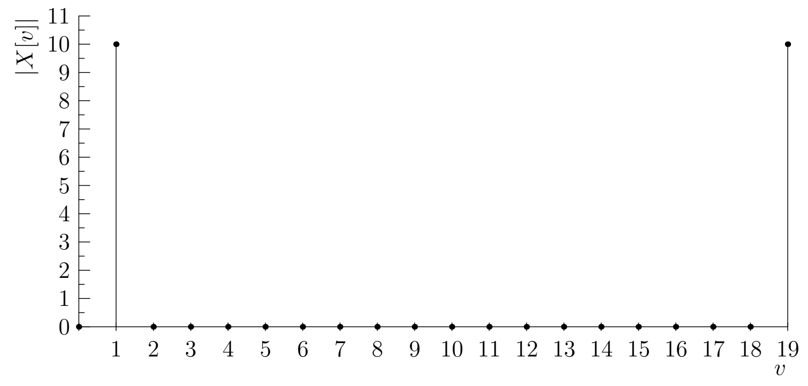

Figure 3.16: Absolute value of the DFT of the signal in Figure 3.15.

Two peaks show up at the discretised frequencies $v = 1$ and $v = 19$. Calculating the associated frequencies gives:

$$\begin{aligned} f_1 &= \frac{1}{\frac{1}{10}\pi \cdot 20} = \frac{1}{2\pi}\text{Hz} \\ f_{19} &= \frac{19}{\frac{1}{10}\pi \cdot 20} = \frac{19}{2\pi}\text{Hz} \end{aligned}$$

The frequency of the original sinusoid is $\frac{1}{2\pi}$ Hz, which is exactly equal to the frequency indicated by the peak at $v = 1$ in the DFT of the sampled signal. Remarkable is that the peak at frequency $v = 19$ is exactly the same as the peak at frequency $v = 1$. A check with Equation (3.20) shows that calculating $X[19]$ results in the complex conjugate of $X[1]$. When plotting the absolute value (or only the real part) of the DFT complex conjugates show the same value, so $X[19]$ corresponds to the same frequency as

$X[1]$. When sampling a signal with a sampling period Δt, the highest frequency that can be measured in that signal is $\frac{1}{2T}$ Hz, which is called the folding frequency, or *Nyquist* frequency. The absolute values of a DFT (of real data) are symmetric about this Nyquist frequency. This is because of the same relationship that tied $X[1]$ and $X[19]$ together: $X[v]$ is the complex conjugate of $X[N-v]$. Paying attention to frequency components higher than the Nyquist frequency is therefore not useful, since it corresponds to information that is already given by lower frequencies. So when making a plot of the absolute values of a DFT, taking frequencies up to the Nyquist frequency is sufficient, when the original data is real valued.

3.1.5 Fast Fourier Transform

In the previous section it was already stated that the calculation of an N-point DFT will take as much as N^2 multiplications. This becomes a major problem when calculating DFT's for large time series. In this thesis, DFT's will be calculated from time series with around 900,000 elements, which results in 810,000,000,000 multiplications when computing it by Equation (3.20). Even on the fast machines that are available today, this will take quite a while. This old fashioned way of computing a DFT proves to be very inefficient. The Fast Fourier Transform overcomes this problem. The FFT is an algorithm which became widely known in 1965 after a publication of J.W. Cooley and J.W. Tukey ([4]), though the idea for such an algorithm was not new at that time. Cooley and Tukey independently had re-invented an algorithm which was already described by Carl Friedrich Gauss in 1805 and also by Carle Runge in 1903 ([13]). The Fast Fourier Transform can be explained in the following way. When the number of samples in a time series is even, Equation (3.20) can be rewritten in two separate equations. One for even indices $v = 2m$

$$X[2m] = \sum_{n=0}^{N/2-1} (x[n] + x[n+N/2])e^{-i\frac{2\pi}{N/2}mn} \tag{3.24}$$

and one for the odd indices $v = 2m+1$

$$X[2m+1] = \sum_{n=0}^{N/2-1} e^{-i\frac{2\pi}{N}n}(x[n] - x[n+N/2])e^{-i\frac{2\pi}{N/2}mn} \tag{3.25}$$

If a new time series $y_{\text{even}}[n]$ is defined to be

$$y_{\text{even}}[n] = x[n] + x[n+N/2], \qquad n = 0,1,\ldots,N/2-1 \tag{3.26}$$

and another new time series $y_{\text{odd}}[n]$ to be

$$y_{\text{odd}}[n] = e^{-i\frac{2\pi}{N}n}(x[n] - x[n+N/2]), \qquad n = 0,1,\ldots,N/2-1 \tag{3.27}$$

the Equations (3.24) and (3.25) are actually the $N/2$-point DFT's of $y_{\text{even}}[n]$ and $y_{\text{odd}}[n]$. Creating $y_{\text{even}}[n]$ from $x[n]$ requires $N/2$ additions. Creating $y_{\text{odd}}[n]$ requires not only $N/2$ additions, but also $N/2$ multiplications. The computation time for a multiplication is about the same as the computation time for an addition. If the computation time for those operations is ε, the total computation time to create both $y_{\text{even}}[n]$ and $y_{\text{odd}}[n]$ is

$$dN, \qquad \text{where } d = \frac{3\varepsilon}{2}$$

If $C(N)$ is the computation time needed to compute an N-point DFT, the previous calculations show that

$$C(N) = dN + 2C(N/2)$$

Since the calculation of a 1-point DFT requires no computations (the only frequency component $X[0]$ will be equal to the only time component $x[0]$), $C(1) = 0$, which gives

$$C(N) = dN\log_2 N$$

With the help of the FFT algorithm, only $\mathcal{O}(N\log_2 N)$ operations are needed, instead of $\mathcal{O}(N^2)$ operations. The savings of the FFT algorithm is apparent in Figure 3.17. The FFT algorithm is most efficient when the number of samples that is transformed is an exact power of two, since the FFT algorithm can then break the original transform down in exactly $\log_2 N$ transforms of length 1.

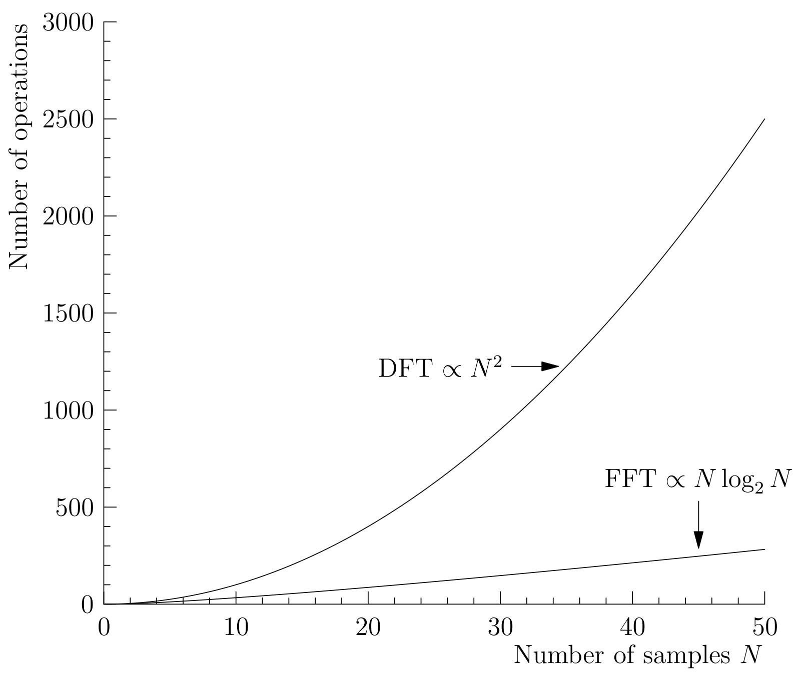

Figure 3.17: The number of operations needed for calculating an FFT and a direct DFT.

3.1.6 Windowing

Windowing is a technique used with Fourier transforms for extracting and/or smoothing data. A window typically is a positive, smooth symmetric function that has a value of one at its maximum and approaches (or is) zero at the sides. A window is applied by multiplying it with the time series to be transformed. When using a window, one should be aware that the input data is altered. Several well known windows exist, including the Hamming, Hann and Blackman window, which all have their own characteristics.
In Section 3.1.4, a signal was sampled over exactly one period. In practise, this kind of ideal sampling is impossible since the frequencies present in the signal are unknown, which means that the length of one period of the signal is also unknown. Moreover, a signal could contain periodicities (which can be shown by Fourier analysis), but does not have to be periodic itself. An ideal sampling period then does not exist. This is illustrated by the following example: a $\frac{1}{2}$ Hz signal, given by

$$x(t) = \cos(\pi t) \tag{3.28}$$

is sampled with a sampling period of 0.22 and 32 samples are taken. Taking the absolute value of the FFT, results in Figure 3.18. Notice that only 17 samples are plotted, since the absolute value of the other 15 samples are copies of the samples 1 to 15. First of all, the actual frequency of the signal has no FFT value associated with it. The information about this frequency is spread out to adjacent frequencies. This phenomenon is called *leakage*. The amount and visibility of leakage depends strongly on the chosen sampling period, the length of the time series and the actual signal that is transformed. Using a window can reduce the effect of leakage, which results in a DFT that can be interpreted in a better way. In Figure 3.19, a Hamming window is used on the sampled data before taking the FFT. A vector that represents the 32 sample Hamming window is given by

$$H_i = 0.54 - 0.46 \cos\left(\frac{2\pi i}{32}\right)$$

with i being the index in the vector. The result of using the Hamming window is that leakage into adjacent frequencies is strongly reduced, which gives a plot that can be better interpreted.

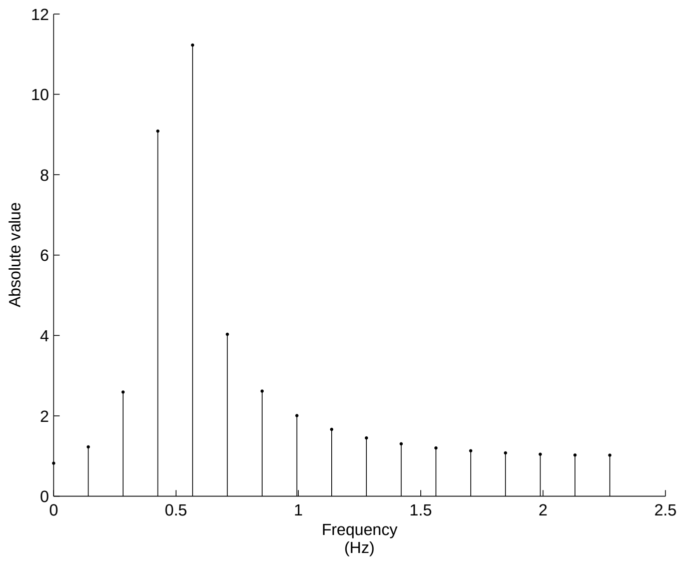

Figure 3.18: Absolute value of the FFT of the sampled (3.28) signal.

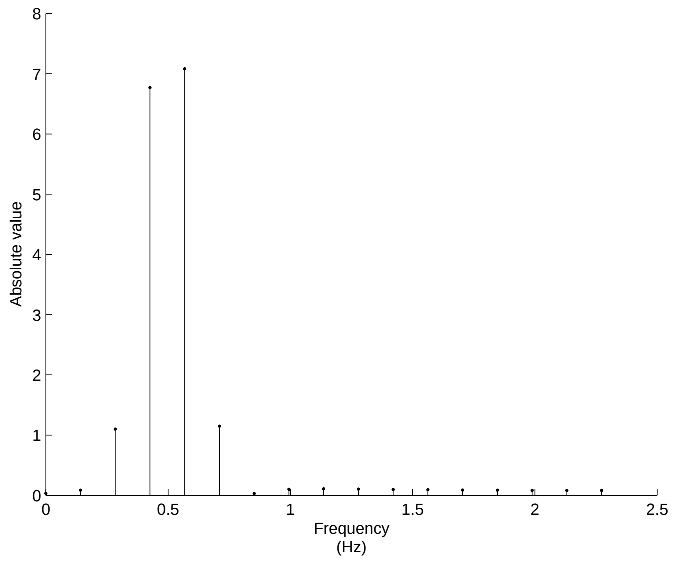

Figure 3.19: Absolute value of the FFT of the sampled signal (3.28) with the Hamming window

3.1.7 Zero padding

Zero padding is commonly used with DFT's and especially FFT's. Zero padding means appending zeros to the time series to be transformed. These zeros can be added either at the beginning, at the end or even somewhere in the middle of the time series, as long as they are added together in one place. The main advantage of zero padding is that more resolution in the frequency domain is gained, since the spacing in the frequency domain is inversely proportional to the number of samples in the time domain. Adding more samples in the time domain, even zero-valued samples, will result in a more detailed frequency spectrum. As an example, the sum of a 1 Hz and a 1.35 Hz sinusoid is sampled over 32 samples with a sampling period of 0.125:

$$x(t) = \cos(2\pi t) + \cos(2.70\pi t) \tag{3.29}$$

Taking the 32-point FFT results in Figure 3.20. Although it is for sure that two different frequencies are present in the signal, these two distinct frequencies do not show up in Figure 3.20. When determining short-term periodicities in a signal, this is undesirable behaviour. If 96 zeros are appended and the 128-point FFT is taken, Figure 3.21 is the result. It may look like a bigger mess (which are the effect of leakage), but there are clearly two peaks visible in this figure, which are the result of more resolution in the frequency domain, made possible by zero padding. Another use of zero padding is to speed up the calculation of the FFT (see Section 3.1.5).

3.2 Traffic repository

The available packet traces within the traffic repository are binary files in PCAP-format[2], containing fifteen minutes of network traffic. All packets sent within those fifteen minutes are stored without their payload/data from the Application Layer. The header information of each packet (discussed in Section 2.2) *is* still available. Furthermore, the `tcpdump` tool that was used to create these packet traces added a time stamp to each packet, which indicates when the packet was captured.
The following information per packet in a packet trace is therefore available:

- Time stamp
- Protocol
- Source IP address (scrambled) and port number
- Destination IP address (scrambled) and port number
- Ethernet addresses

To protect user's privacy, the IP addresses in the packet traces are scrambled, using the `tcpdpriv` utility. According to [16], addresses within one packet trace are scrambled in such a way that if two of the original addresses are equal in the most significant n bits, then these two addresses will map to scrambled addresses that are similarly equal in the most significant n bits. Therefore it can safely be assumed that the mapping between original and scrambled addresses is a bijective mapping. As a consequence, it is possible to determine whether or not packets were sent or received by the same host, which is also of importance for the analysis of periodicities.
More information from the packet trace is available (like packet size), but not used for the research. The information that is *not* available is:

- Actual IP-addresses
- Payload / data within the packets

[2]Typical applications that can read and write PCAP-files are `tcpdump`, `ethereal` and a periodicity analysis tool (`pat`) that was developed during the research, which will be commented on in Chapter 6

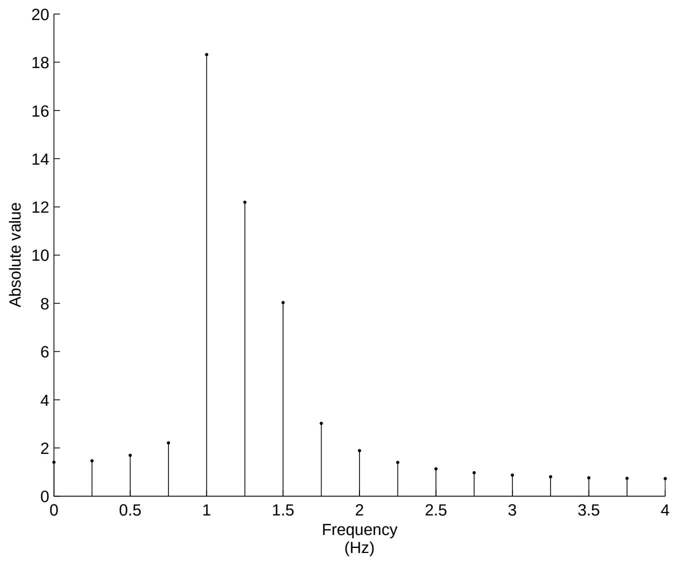

Figure 3.20: The absolute value of the FFT of signal (3.29) without zero padding.

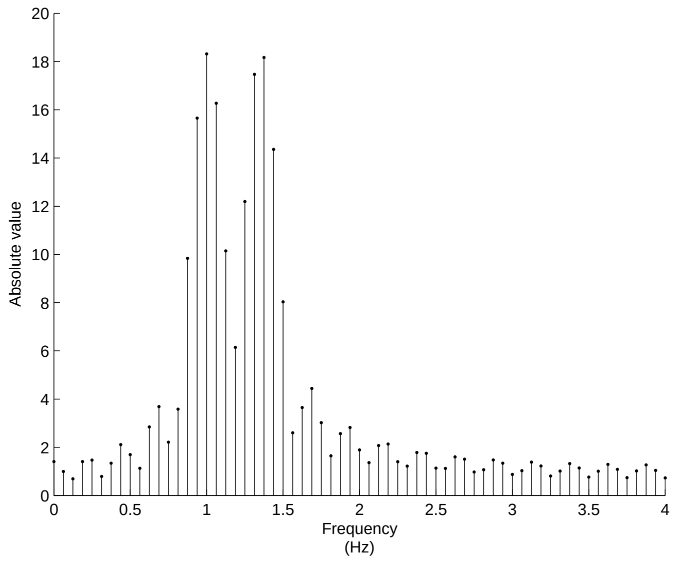

Figure 3.21: The absolute value of the FFT of signal (3.29) with zero padding.

So in principle, for determining which traffic is responsible for certain periodicities there is enough information available: the time stamps are necessary for information about the periodicities because time domain information is needed to enable the use of Fourier analysis, while protocol information and port numbers will give a hint on the sending/receiving application that could cause these periodicities. The rest of the available information can be used for fine tuning the cause of periodicities.

3.3 Plots of transforms

To determine periodicities in the available packet traces, frequency domain information should be extracted from those packet traces. To do this, the techniques that are described in the previous sections are used. This section explains how Fourier analysis can be applied to a packet trace. The first step in this process is to create a time series from the packet trace. After that, the FFT of this time series discloses frequency information about this packet trace. This is further clarified by the following example. In Table 3.3, some information about the first 10 packets in packet trace `loc1-20020626-0415` is given. If

Time stamp	Protocol	Source Address	Source Port	Dest. Address	Dest. Port
04:15:01.263888	IP	135.216.170.24	3610	135.216.205.97	139
04:15:01.266938	IP	135.216.172.50	4887	135.216.205.190	139
04:15:01.267628	IP	135.216.205.190	139	135.216.172.50	4887
04:15:01.267775	IP	135.216.205.190	139	135.216.172.50	4887
04:15:01.269928	IP	135.216.168.239	1451	135.216.206.97	139
04:15:01.270131	IP	135.216.206.150	3648	135.216.128.214	53
04:15:01.270273	IP	135.216.172.79	1037	135.216.204.22	139
04:15:01.270667	IP	135.216.128.214	53	135.216.206.150	3648
04:15:01.271137	IP	135.216.204.22	139	135.216.172.79	1037
04:15:01.271272	IP	135.216.171.66	1046	135.216.205.213	139

Table 3.3: Information about the first 10 packets in `loc1-20020626-0415`.

the first packet's time stamp is chosen as a reference for $t = 0$, the traffic can be modelled as a function of time, as shown in Figure 3.22. To create the time series, some kind of sampling has to be done on

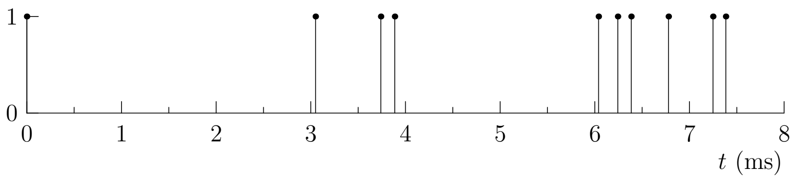

Figure 3.22: First 10 packets in `loc1-20020626-0415` plotted as a function of time.

this function. The samples should represent the amount of traffic at a certain time. The approach here is to model every packet as a unit impulse (with value 1) at the point in time that is equal to its time stamp. Subsequently, the number of packets within a certain intervals $[\Delta t \cdot i, \Delta t(i+1))$ are counted, with Δt being the chosen sampling period and i the index in the time series that is being created. If, for the example above, a sampling period of $\Delta t = 0.5$ ms is chosen, the following time series would be generated:

$$x = [\, 1\ 0\ 0\ 0\ 0\ 0\ 1\ 2\ 0\ 0\ 0\ 0\ 3\ 1\ 2\ 0 \,]$$

which can be Fourier transformed, possibly after windowing and/or appending zeros. This short example gives rise to the following questions:

- What length should the sampling period be?
- Should windowing and/or zero padding be applied?

In Section 3.1.4 the term "sampling period" has already been explained. It also showed the importance of this first question, since the sampling period determines the highest frequency that can be detected (the Nyquist frequency). A trade off should be made here: a small sampling period will result in a higher Nyquist frequency and therefore a broader frequency spectrum. On the other hand, a large sampling period results in smaller time series and faster computations. After some trying, a sampling period of 1 ms seemed attractive: the computations of Fourier transforms were done in an acceptable amount of time and the Nyquist frequency was at 500 Hz, which should be high enough for the first explorations in the periodicity of Internet traffic, since this corresponds to a period of 0.002 seconds. This should be enough because, for example, round trip times (the time between sending a packet and getting the acknowledgement that it has been received, see Section 2.2.3) in network traffic are usually longer than this period.
The packet traces in the repository each contain fifteen minutes of data. Sampling a packet trace with a period of 1 ms, results in a time series with about 900,000 elements. Calculating the FFT from this time series takes a long time. Appending zeros to get a time series that has a length equal to a power of two will speed up this process significantly. Even if the computation of the FFT would be fast enough on 900,000 elements, zeros would still need to be appended to get enough resolution in the frequency domain. The amount of zeros that should be appended depends on the spacing in the frequency domain that is needed for good results. As said before, more zeros will reduce the spacing in the frequency domain and thus the error when determining at which frequency a peak in the FFT is present. This choice was especially important in the development of the tool, described in Chapter 6. Returning to the first question, why not take a sampling period that will immediately give a time series with a length equal to a power of two? Solving the sampling period Δt in

$$\frac{900}{\Delta t} = 2^n$$

that is needed when sampling 900 seconds (15 minutes) of data with 2^n samples is easily done. This however, will not work. First of all, the packet traces contain *about* 15 minutes of data, which means that for every separate packet trace the above equation should be solved. An option is, of course, to just truncate the data at 15 minutes, ignoring the rest. Then a problem would occur in the case of a packet trace that is actually *shorter* than 15 minutes and zeros would still have to be appended to the time series. Having a fixed sampling period has the advantage of working directly for every possible packet trace length. Secondly, there is no real qualitative advantage of "stretching" the time series to a power of 2 over the padding of zeros to a power of 2. So the effort of calculating appropriate sampling periods, truncating time series etc. might just as well be saved. Another – though weaker – reason for not using this method is that for basic calculations, it is a lot easier to work with a sampling period that is "a nice number" instead of a number that is inversely proportional to a power of 2.
Windowing showed little useful effects on the outcome of the transforms. Zeros are appended, which means that the data is already somewhat smoothed without the help of windowing. Moreover, applying a window mostly effects the sides of a time series; in this case the right side of the time series is comprised of zeros, on which windowing will not have any effect. For this reason, windowing is not used with time series that represent the full 15 minutes of a packet trace.
When the FFT is calculated, the absolute value of it is plotted against a properly defined vector of frequency values (calculated with Equation 3.21), which gives the Fourier transform plots used in this thesis.

Chapter 4

Address Resolution Protocol

Traffic caused by the Address Resolution Protocol (ARP) is the first kind of traffic to be discussed. Although the choice of starting with ARP is rather arbitrary, it proves to be a good exercise for the rest of the traffic to be researched. This is mainly because the amount of ARP traffic on a (local) network is relatively small compared to TCP- and UDP-traffic: when some pattern is discovered in the frequency spectra of the traffic, it is fairly easy to resolve which packets caused this pattern to occur. The knowledge obtained from this ARP exercise then can subsequently be used in the further analysis of other traffic. For example: if some pattern in the frequency spectrum of ARP traffic can be explained, a similar pattern in the frequency spectrum of UDP traffic might be explained by the same phenomenon.

In the next sections a closer look is taken at periodicities in ARP traffic. After starting off with a small discussion about what kind of periodicities could be found in ARP traffic, a first frequency spectrum is shown and analysed.

4.1 Possible periodicities in ARP

This section is about ARP traffic that could influence the periodicity of network traffic. So why may someone expect periodic behaviour in ARP traffic in the first place? One reason could be unanswered requests. Many protocols use time outs when waiting for an answer. If an answer is not received within this time-out, another request will be sent and again a time-out is set. Several attempts for obtaining an answer can be made in this way, although in ARP only a few attempts will be made at a rate of one per second. If there is still no reply after the last request, the network interface will not try again until a host tries to send new data (for which again ARP requests have to be sent). A quick conclusion would be that these repeated retransmissions at a rate of one per second could, for now, be interpreted as an occasionally appearing periodic signal at a frequency of 1 Hz.

An entry in a host's ARP table is only kept there for limited time. A typical expiration time for an entry is 20 minutes. So suppose that in the example from Figure 2.7, host A has nothing to send to host for at least 20 minutes. The ARP table at host A will then drop the entry for host B. If host A then wants to send information to host B, one or more new ARP requests must be sent. So if a session between host A and host B interrupts frequently for at least 20 minutes, again a repeating retransmissions of ARP requests would be seen, but at a much slower rate. Furthermore, it is likely that these "interruption intervals" are not of the exact same length: one could be 30 minutes, another could be 50 minutes and yet another could be 3 hours. So the rate at which requests are retransmitted in this case are most certainly not fixed. A periodicity caused by expiring entries in ARP tables is therefore unlikely to occur. Even if it would occur, the matching frequencies would be far lower than the frequencies which are of interest in this thesis, which are the frequencies belonging to *short-term* periodicity. Moreover, these intervals are larger than the 15 minutes that the packet traces are in length, which makes it even impossible to figure out whether a frequency should be associated with an expired entry in an ARP table.

4.2 A first packet trace examined

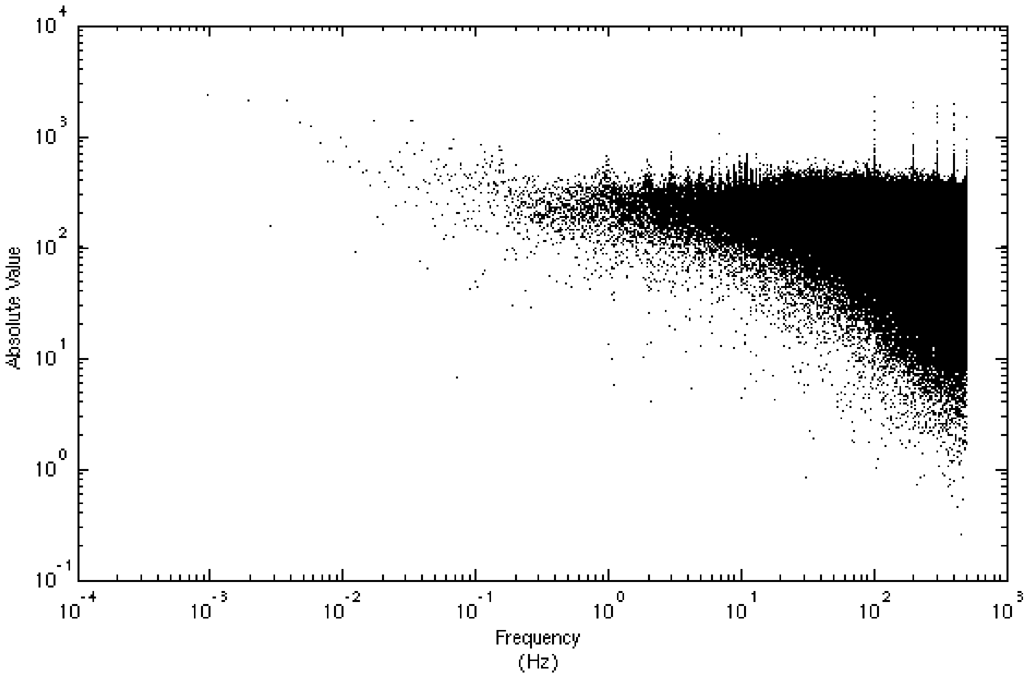

Figure 4.1: ARP frequency spectrum for `locl-20020625-0415`

If the ARP traffic in packet trace `locl-20020625-0415` is Fourier transformed and plotted, Figure 4.1 is the result. The choice for discussing this packet trace is based on the remarkable peaks seen at the frequency of 100 Hz and its harmonics. These peaks clearly indicate some kind of periodicity in this packet trace which can be further investigated.

Intuitively, one could already say that there is something interesting happening with a (repeating) time interval of 0.01 second, since this is the period corresponding to a frequency of 100 Hz. A first move to retrieve the source of this frequency is to pick those packets out of the packet trace that have something to do with this 0.01 second interval. Although it is not likely that a frequency of 100 Hz is caused by two *successive* packets that are sent/received a hundredth of a second apart from each other, it is worth the try to plot the time intervals between two successive packets. In Figure 4.2 this is done for all the packets within the earlier mentioned packet trace. The horizontal axis represents the packet's position within the trace. The vertical axis represents the time interval between that packet and its predecessor. Of course, for the first packet there is no predecessor and hence no plotted point. Surprisingly enough, at a time interval of 0.01 second, a horizontal line can be identified, which indicates that a reasonable amount of packets are this interval away from their predecessor. The next step is to isolate these packets from the trace and see if there is something special happening. This is simply done by taking the original packet trace and dropping those packets that are more than 0.0105 or less than 0.0095 seconds away from their predecessor. The 5% margin is rather arbitrary but some margin should be used, since most packets will not match the exact time interval. Furthermore, it is not sure that the frequency that is being investigated is *exactly* 100 Hz. A big amount of packets in this selection were sent from Ethernet address 00:90:27:4d:61:7e. Taking a closer look at these packets shows that the 0.01 second intervals are caused by retransmitted ARP requests sent from this Ethernet address. A reasonably large amount of ARP requests, for many different addresses, is done by this address so it could be assumed that it is the address of a router. Another possibility could be that some virus is trying to spread itself to other

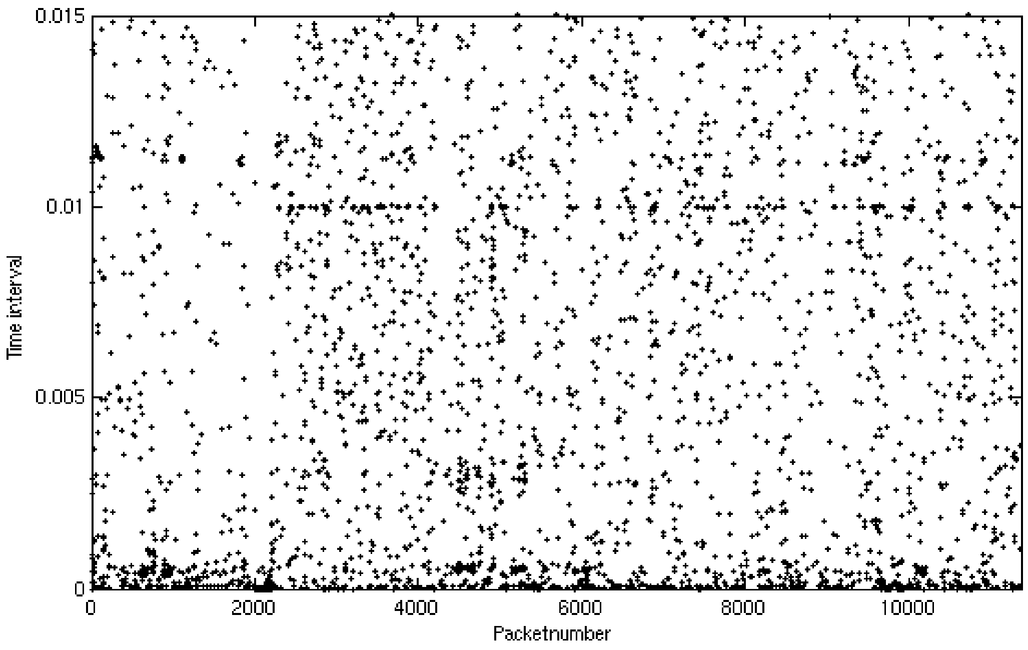

Figure 4.2: Time intervals between packet n and packet $n-1$

machines on the network, which also results in a large amount of ARP requests. Taking a closer look at the ARP requests that are sent out by this network adaptor with `Ethereal`, showed that these requests are always at least 0.01 seconds apart from each other and that an ARP request for the *same* address is never repeated within one second. A conclusion so far is that the 100 Hz peaks found earlier are mainly caused by this network adaptor (a router or some virus infected machine), sending out (retransmissions of) ARP requests at a rate of 100 per second, most probably due to the internal clock of the router or machine.

This can be verified by taking these particular packets apart from the rest of the traffic, making a new Fourier transform and plot the results (see Figure 4.3). This shows some quite interesting results: not only do the peaks at 100 Hz and its harmonics show up again, on both sides of these peaks some smaller peaks made their way into the plot. A possible explanation for these so called sidebands could be that some kind of modulation is applied to a signal. Unfortunately, it is not clear where this modulation-like behaviour comes from. Another interesting feature in Figure 4.3 is that an arch pattern is showing repeatedly from the left to the right of the plot. This arch pattern will be further explored in the next section.

Now the disadvantage of this method is that periodic traffic does not necessarily have to be caused by successive packets, which was assumed earlier. Consider Figure 4.4: the red packets are sent periodically and the blue packets are randomly placed. Sometimes the red packets are in successive order, but a random number of blue packets can be in between two reds. This means that for the method to work, the "red stream" should first be taken apart from the rest of the traffic. Then the time intervals between two packets can be measured to see if some periodicity is caused by this traffic. To do this for all the streams in a trace would take a lot of time. Another solution for this problem could be to plot not only the time interval between to successive packets (packet n and packet $n-1$), but also the time intervals between packets with other indices, so n and $n-2$, n and $n-3$ etc., since these intervals can also give a hint for the presence of a periodicity.

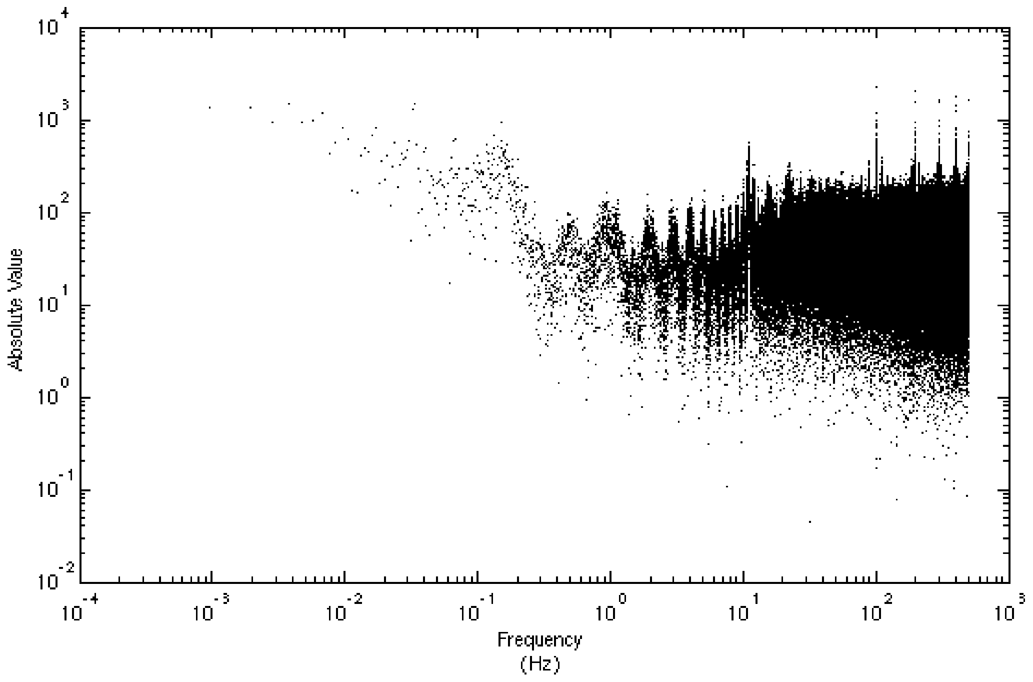

Figure 4.3: Fourier transform of ARP requests sent from 00:90:27:4d:61:7e

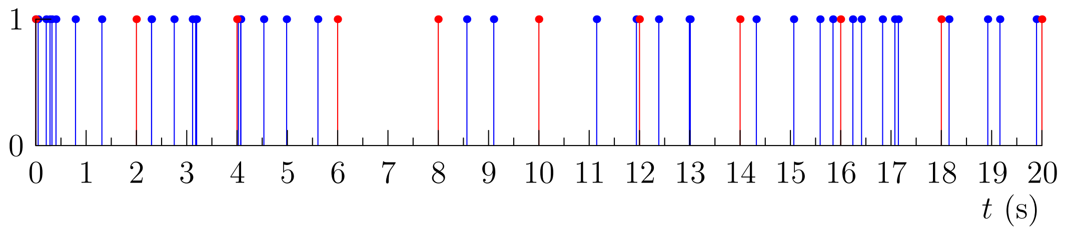

Figure 4.4: Counter example for the method described earlier: periodic traffic does not have to be caused by successive packets.

4.3 Repetitive behaviour in the time domain

As discussed in Section 4.1, ARP requests are retransmitted if no answer is received. In the packet trace used, a retransmission for an unanswered ARP request was retransmitted at most two times. So when a host is offline and ARP requests are done for this host, three ARP requests are done in total. This can be modelled as a system with a (discrete) input signal $x[n]$ and an output signal $y[n]$, which repeats the input signal three times with a delay of k seconds. The following holds for this system:

$$y[n] = x[n] + x[n-k] + x[n-2k] \tag{4.1}$$

A visual representation of this is given in Figure 4.5. The point in discussing this is to explain the

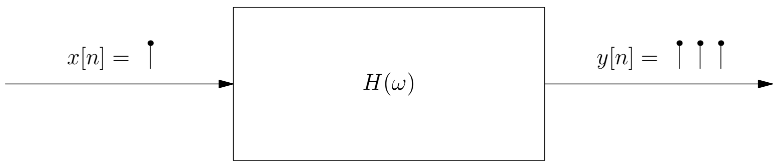

Figure 4.5: The system with transfer function $H(\omega)$ causes a repetition of the input signal

arch pattern in Figure 4.3. The transfer function $H(\omega)$ of this system can be easily constructed from Equation (4.1), since it is a superposition of delayed signals. For a system delaying the input signal with k time steps the function for the output signal is

$$y[n] = x[n-k]$$

with a corresponding transfer function

$$H(\omega) = e^{-ik\omega}$$

For the system described the transfer function therefore is

$$H(\omega) = 1 + e^{-ik\omega} + e^{-i2k\omega},$$

which can be rewritten as

$$\begin{aligned} H(\omega) &= e^{-ik\omega}\left(e^{ik\omega} + 1 + e^{-ik\omega}\right) \\ &= e^{-ik\omega}\left(2\cos(k\omega) + 1\right). \end{aligned}$$

Taking the absolute value of $H(\omega)$ yields

$$\begin{aligned} |H(\omega)| &= \left|e^{-ik\omega}\left(2\cos(k\omega) + 1\right)\right| \\ &= \left|e^{-ik\omega}\right| \left|2\cos(k\omega) + 1\right| \\ &= \left|2\cos(k\omega) + 1\right| \end{aligned} \tag{4.2}$$

Plotting the function (4.2) results in Figure 4.6. Repetitive behaviour in the time domain clearly gives arch patterns in the frequency domain. This could explain the arch patterns in Figure 4.3: they could be the result of the repeatedly sending of ARP requests. The notches seen in Figure 4.6 give information about the frequency at which packets are retransmitted in the case of two retransmissions (3 packets in total), as depicted in Figure 4.5. The places at which these notches occur can be calculated by solving

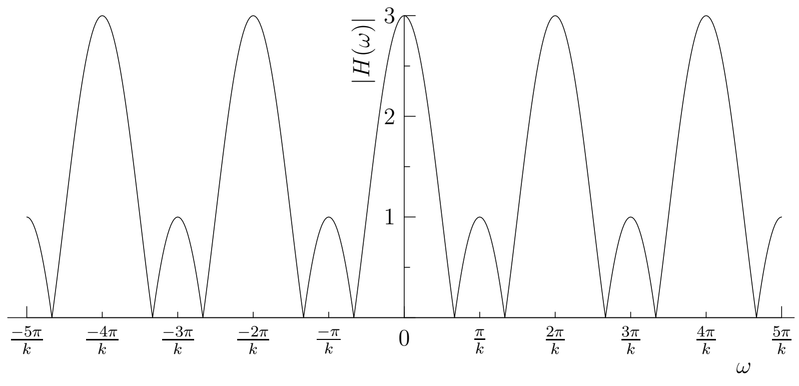

Figure 4.6: The absolute value of the transfer function at different angular frequencies

Equation 4.2. To identify the related frequency, only the place of the first notch has to be solved:

$$\begin{aligned}
2\cos(k\omega) + 1 &= 0 \\
\cos(k\omega) &= -\frac{1}{2} \\
k\omega &= \frac{2\pi}{3} \\
\omega &= \frac{2\pi}{3k} \\
2\pi f &= \frac{2\pi}{3k} \qquad \text{with } f = \frac{\omega}{2\pi} \\
f &= \frac{1}{3k}
\end{aligned}$$

When looking at Figure 4.3, it seems that the first notch appears at the frequency of 0.3 Hz. Substituting this frequency in the equation above gives

$$\begin{aligned}
0.3 &= \frac{1}{3k} \\
k &= \frac{1}{0.9} \approx 1.1 \text{ seconds}
\end{aligned}$$

The Fourier transform therefore confirms what `ethereal` also showed: packets are retransmitted twice spaced by an interval of about 1.1 seconds, which is confirmed by the slight “hump” in the transform at the frequency of 1 Hz. The found arch – or rather notch – patterns clearly indicate a finite number of retransmissions.

4.4 Result of ARP analysis

While analysing one specific packet trace with ARP traffic, first a way to investigate the periodicities present was developed. Although it was just a wild attempt, plotting the time intervals between the arrivals of two successive packets did show useful results. The plot of the Fourier transform showed a frequency of 100 Hz (and harmonics) and plotting the time intervals between two packets showed a line at 0.01 seconds, which coincides with the frequency that was found. Filtering out the packets which were responsible for this line eventually gave information about the machine that was responsible for the presence of a periodicity with a frequency of 100 Hz. Still, why was it 100 Hz? Since the documentation about the ARP protocol does not say anything at all about packets being sent every 0.01 second (or even on some other interval), the 100 Hz should be a feature of the machine itself and has nothing to do with the ARP protocol. The best guess then is, that it has something to do with the internal clock of the router/machine or with the characteristics of a possible virus on the machine that could be causing this periodicity.

Making a Fourier transform of the traffic caused by this machine resulted in a figure with interesting characteristics. The arch pattern could be giving some more insight into how the machine actually behaves. It was shown that the pattern is there because of the finite repeated sending of ARP requests and that a retransmission period can be deducted from it, or – in a more easy way – can be confirmed by it. In Section 4.1, the presence of a 1 Hz signal due to these retransmissions was already predicted. When looking at Figure 4.3, a hump at 1 Hz (and its harmonics) does show up. This again confirms the idea of a periodicity due to retransmission of ARP requests.

The analysis of ARP traffic gave some answers towards the research of periodicities in network traffic in general, but it also posed new questions (like the “sidebands” in Figure 4.3). Unfortunately, after quite some brainstorming about these sidebands, it is not clear what caused these sidebands to appear.

Chapter 5

User Datagram Protocol

The second and last type of traffic that is researched in this thesis is User Datagram Protocol (UDP) traffic, of which a short description was given in Section 2.2.4. As in the previous chapter, some predictions on what traffic could cause periodicities are made first. Secondly, differences between UDP and ARP traffic, which call for another research approach, are discussed. The different approach that has been used is described as well. After that, one packet trace from the repository is analysed in full detail which gives more insight in how the results from Section 5.5, which cover several more packet traces, are gathered. Finally, some conclusions are drawn with respect to periodicities in UDP traffic and the use of a developed tool is evaluated in the last section of this chapter.

5.1 Possible periodicities in UDP

Although UDP traffic is *not* connection oriented and there is no connection that is being set up between two hosts, it is still imaginable that two hosts interact with each other on a periodic basis. Earlier it was mentioned that multimedia streams use the UDP protocol as the transport protocol. This is a typical kind of traffic that is causing periodic behaviour in Internet traffic, since these streams consist of chunks of data being sent at some specific rate. Protocols that facilitate the streaming of audio and/or video should therefore cause periodicities. The Domain Name System (DNS) protocol that was mentioned briefly in Section 2.2.5, is also a candidate cause for periodicities, since it has a task that is similar to that of the Address Resolution Protocol (discussed in the previous chapter): mapping one kind of address to another. In the case of DNS, this is a mapping from hostnames (e.g. `www.utwente.nl`) to IP addresses and vice versa. Again, requests to resolve an address might not be answered in time, which causes retransmissions of these requests, which in turn would cause periodicities. As a consequence, DNS servers that have the task to respond to these requests will probably be sending out responses in a periodic pattern, which makes them a cause of periodic traffic as well. These two possible causes (the streams and the DNS traffic) are already enough reason to take on UDP traffic in this chapter, but other protocols may of course also cause periodic behaviour within UDP traffic. A selection of well known protocols that were found during the research as possible causes for periodicities are listed in Section 5.5. Before going into detail about the causes of periodicities in UDP traffic, a definition of what such a "cause" is should be given. In the rest of this thesis, the cause of a periodicity is defined as follows:

> The cause of a periodicity is a collection of packets which show peaks (or other periodicity related phenomena, like notches) in their frequency domain and have one or more of the following header fields set to the same value:
>
> - Source port
> - Destination port
> - Source IP address

- Destination IP address

Returning to the example of DNS traffic: if a client periodically sends requests to a DNS server, these packets share the same source IP address, destination IP address and destination port number and are designated as a cause of periodic traffic. The responses from the DNS server are also packets that have some header fields set to the same value and these responses will be *separately* considered as a cause of a periodicity, although this traffic is induced by the original request traffic.

5.2 How to analyse UDP frequency spectra

One of the tough problems in analysing frequency spectra is finding a way to relate peaks and other phenomena to specific traffic in the original (time domain) data. In the previous chapter, the rather simple approach was:

1. Determine the most dominant frequency f.
2. Plot the time intervals between two successive packets in a packet trace, to see if some pattern (such as the line in Figure 4.2) occurs at the time interval of $\frac{1}{f}$ seconds.
3. Isolate the packets that constitute this pattern: these packets are a likely cause of the peak that was seen in the frequency spectrum.

In this section, the differences between ARP traffic and UDP traffic are discussed, which give even more reason to search for another way of identifying traffic that is responsible for peaks in the frequency spectra of packet traces.

5.2.1 Differences with ARP traffic

Apart from the problems that the above mentioned approach already has as described in Section 4.2, the differences between the ARP protocol and UDP protocol give reason to think about another approach towards identifying periodicities. The foremost difference between ARP traffic and UDP traffic is the quantity of it that is travelling the Internet. In the packet traces that were used for the research, the traces that contained both ARP traffic *and* UDP traffic showed that quantities of both types of traffic is in the proportion of 1 to 15 during the night and in the proportion of 1 to 250 during office hours. This large amount of traffic immediately results in the plots with interval times between successive packets to become hard to analyse with the human eye. Of course this does *not* mean that the method has to be discarded immediately, but it does give a hint that an automated process should do the work.
Another difference is the purpose that both protocols serve. In Chapter 2, the operational difference was already explained: the UDP protocol resides on the Transport Layer and the ARP protocol resides on the Host-To-Network Layer. This is no problem in principle: traffic on different layers could well be analysed in the same way. What *does* cause problems is that the functionality of UDP traffic is more diverse: ARP traffic is all about requesting Ethernet addresses and sending/receiving a reply for that request, whereas UDP traffic is used by other protocols with totally different functions. Plots that indicate time intervals between two successive packets make reasonable sense if those packets are of the same type or application protocol, but those plots are practically useless if those packets probably haven't got anything to do with each other. Even if packets *are* of the same type or application protocol, it could still be that one packet belongs to a complete other process than the other.
The problems with the already used approach and the differences between the two researched protocols plead for a new approach with at least the following two properties:

- Human interaction in the analysis of traffic should be reduced as much as possible. This will result in more accurate results, as well as a speed advantage.
- The used approach should be able to handle the different types of UDP traffic, so that periodicities can be assigned to particular services and protocols that are present on the Internet.

In the next section, the method that was developed and used for the research of UDP traffic is described in a quite general sense. Chapter 6 describes the technical details on how this method is implemented in a "Periodicity Analysis Tool" (`pat`) that was used during the research.

5.2.2 Binary search algorithm

The method to find a possible cause for a periodicity utilises an algorithm that is widely used in all kinds of fields: the binary search algorithm. A useful property of this algorithm is that with each step it takes, the number of possible causes for a periodicity is reduced by half. To clarify this, a short example is worked out here. A more detailed (and technical) description of this algorithm is given in Section 6.2.4. To identify a likely cause of a periodicity, the following steps are taken.

1. Take a Fourier transform of all UDP traffic in a trace.
2. Determine the dominant frequency.
3. Perform a binary search algorithm:
   (a) Divide the possible causes in two ranges/halves (further explained in the example below).
   (b) Take a Fourier transform of both halves.
   (c) Determine which half still contains the dominant frequency that was determined in step 2.
   (d) Discard the other half and go back to step 3a with this half.

With packet trace `loc3-20031002-1700`, step 1 of the method results in Figure 5.1, which shows that the dominant frequency (step 2) is 100 Hz. In the next step, the traffic is divided in two halves. But there are several options to cut the traffic in half:

- Port based: starting off with a range of port numbers, one half will contain the first half of this range and the other half will contain the second half of this range.
- Address based: starting off with a range of IP addresses, one half will contain the first half of this range and the other half will contain the second half of this range.

In this example it is assumed that some destination port number (e.g. all traffic to DNS servers, which corresponds to destination port number 53) is causing periodic traffic with a frequency of 100 Hz. Possible port numbers lie in the range of 0 through 65535. Dividing this in two halves will result in a part with port numbers in the range of 0 through 32767 and a part with port numbers in the range of 32768 through 65535. Taking the Fourier transform of both parts results in Figure 5.2, where the upper half is plotted in blue, and the lower half in black.
The figure clearly shows that the dominant frequency is still present in the lower half of port numbers and not seen in the upper half. Therefore the upper half of port numbers can be thrown away and another step in the binary search algorithm can be taken with the lower half. This means that the lower half is again split in two ranges of port numbers: 0 through 16383 and 16384 through 32767. Again two Fourier transforms are calculated, compared and a "winning" half is chosen in the same way as before. Making the choice for such a winning half can be easy, but Figure 5.3 shows what could happen after more reduction steps.
Although the choice for a winner would clearly be the lower (black) side of the remaining plot number range, it can not be denied that the higher (blue) side of the range is also contributing to the dominant frequency of 100 Hz that was found earlier. This does not have to be a problem: proceeding with the lower half will ultimately give a destination port number that is likely causing the periodicity. Traffic with this destination port number can then be removed from the original packet trace, after which another run of the tool can be done on the remaining traffic. This again includes a new search for a dominant frequency. With one cause already cancelled out now, it is likely that the "blue peak" could be the winner in one of the next runs. This process of determining a cause, removing this cause from the original traffic and running the tool again on the remaining traffic can be repeated until all possible

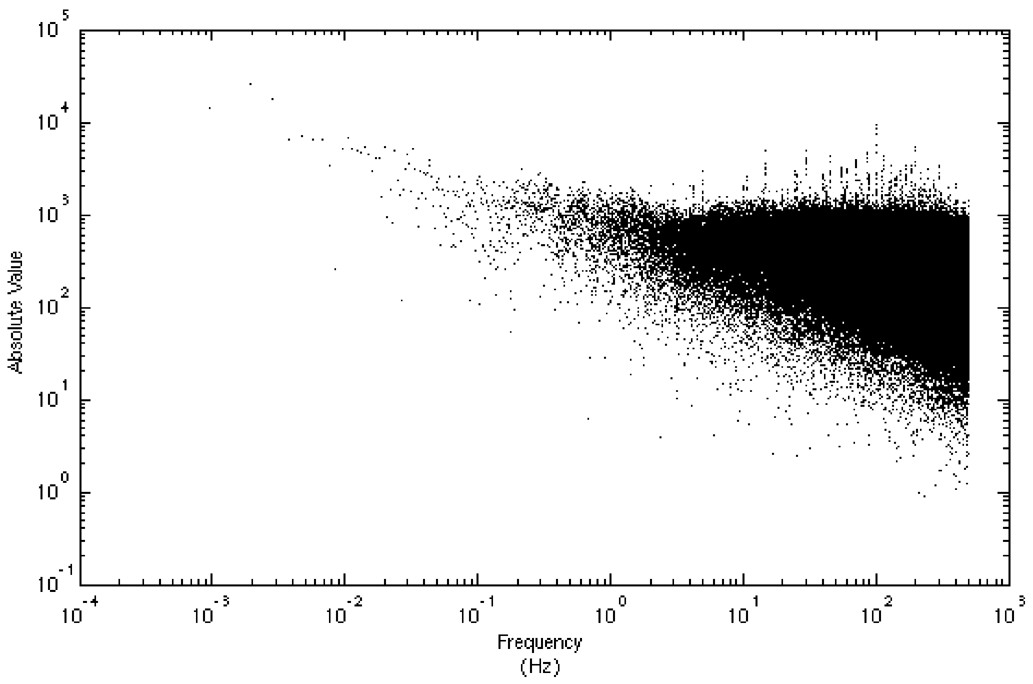

Figure 5.1: Fourier transform of the UDP traffic in `loc3-20031002-1700`.

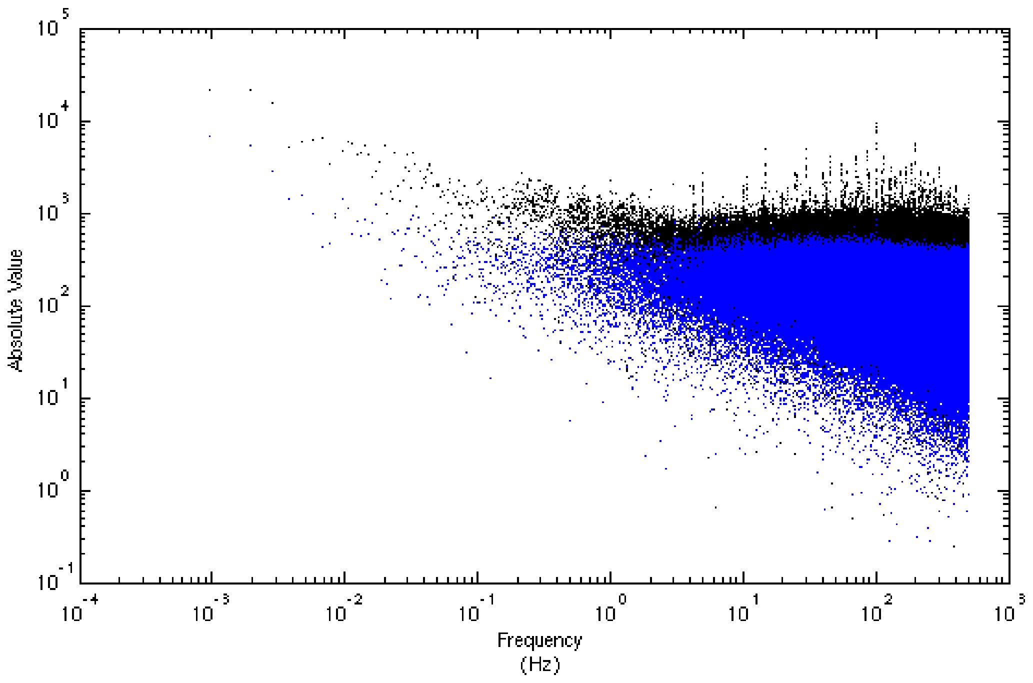

Figure 5.2: First step in the binary search algorithm for a destination port number.

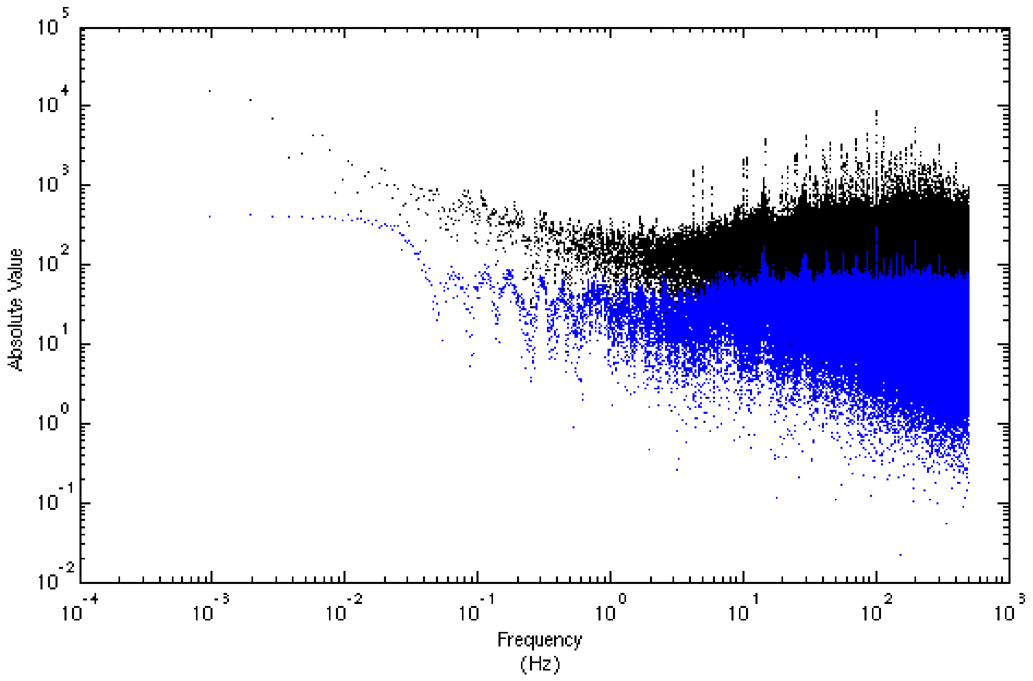

Figure 5.3: Tenth step in the binary search algorithm for a destination port number.

causes that are contributing to a periodicity are found; also the ones that were thrown away in previous runs of the algorithm.
The log file (created by the tool) gives the following information about the step in Figure 5.3:

```
Processing bit 10...
Low peak (17408 packets) : 9012.32523
High peak (582 packets) : 302.89457
Low side wins with a relative difference of    2875.40%
```

It shows that the lower side contained about 30 times as many packets as the high side and that the absolute value of its Fourier transform is also about 30 times bigger, which makes it the winner in this step. This reveals a negative aspect of the tool: the choice for a winner is highly influenced by the *number of packets* within the ranges that are examined. This has an immediate negative effect on the ability to find periodicities. Suppose that packets causing some periodicity end up in one range, and *many* more packets which show no sign of periodic behaviour at all (e.g. white noise) in the other. The tool would probably designate the white noise as the cause of the periodicity, since its Fourier transform may have a higher absolute value than the Fourier transform of the traffic that actually *is* causing the periodicity. Further steps in the algorithm will be taken with the side containing the white noise, which results in useless guesses about possible causes of the periodicity that was found in the first place. This is not only a theoretical problem: the tool sometimes did get fooled by this effect.
It should also be kept in mind that the search for a responsible *destination port* was just based on an assumption. The phenomenon seen in Figure 5.3 could also be caused by some source host that is contacting several other machines with different port numbers on the Internet. If it is this source host that is the actual cause of the periodicity, the binary search algorithm will of course work for a while, but eventually the destination machines involved (or better: the destination port numbers) will end up in the different halves that are examined in one step of the algorithm and a situation like the one in Figure 5.3 is created. Therefore, it is important not to stick with one possible cause of a periodicity,

but to try and see on which of the four possible causes mentioned earlier the binary search algorithm gets its best results. The tool makes this possible by outputting PCAP files containing the traffic that is believed to cause the periodicity. Fourier transforms of these packet traces can then show which packet trace is the most likely cause.

5.3 A first packet trace examined

To get an idea on how the Periodicity Analysis Tool works, this section shows how one packet trace is examined from start to end. The results given by the program are discussed, together with Fourier transforms of the traffic that seemed to cause a periodicity.
Starting off with all the UDP traffic in packet trace `loc1-20020626-0415`, Figure 5.4 on page 41 is generated by taking a Fourier transform. The transform shows some high absolute values at the frequency of 100 Hz and harmonics of this frequency, which clearly indicates a periodicity present in the traffic. On the following pages, this particular packet trace is examined for periodicities with help of the Periodicity Analysis Tool. The frequency reported by the tool is given, a Fourier transform of the traffic that is believed to cause the periodicity and a part of the traffic is printed as `tcpdump` output. These three sources of information should give a clue about the actual cause of some periodicity.

A first run of the tool, shows that the dominant frequency in the trace is 99.996 Hz and that DNS traffic (discussed in the next section) might be causing this periodicity. Looking at the output PCAP files of the program, the likely cause of this periodicity proves to be traffic from a host with the (scrambled) IP address `135.216.206.244` with port number 1025, which is sending packets to several DNS servers on port 53. Taking this traffic apart from the trace and generating its Fourier transform results in Figure 5.5. The most dominant (fundamental) frequencies are approximately 0.2, 2 and 100 Hz. A part of the trace that was transformed in Figure 5.5 is given here:

```
03:19:55.645137 IP 135.216.206.244.1025 > 201.198.86.137.53:  43459[|domain]
03:19:55.647696 IP 135.216.206.244.1025 > 204.120.103.25.53:  63793[|domain]
03:19:55.662305 IP 135.216.206.244.1025 > 201.198.86.137.53:  38827[|domain]
03:19:55.683275 IP 135.216.206.244.1025 > 201.198.86.137.53:  16128[|domain]
03:19:55.712405 IP 135.216.206.244.1025 > 201.198.86.137.53:  8064[|domain]
03:19:57.112675 IP 135.216.206.244.1025 > 209.41.168.247.53:  51087[|domain]
03:19:59.663072 IP 135.216.206.244.1025 > 209.52.65.20.53:  29815[|domain]
03:19:59.664180 IP 135.216.206.244.1025 > 209.52.65.20.53:  22705[|domain]
03:19:59.665262 IP 135.216.206.244.1025 > 209.52.65.20.53:  46230[|domain]
03:19:59.666344 IP 135.216.206.244.1025 > 209.52.65.20.53:  27947[|domain]
03:19:59.702567 IP 135.216.206.244.1025 > 209.52.65.20.53:  64251[|domain]
03:19:59.722470 IP 135.216.206.244.1025 > 209.52.65.20.53:  60878[|domain]
03:20:01.132956 IP 135.216.206.244.1025 > 132.41.193.87.53:  63207[|domain]
03:20:03.663191 IP 135.216.206.244.1025 > 221.207.238.203.53:  16264[|domain]
03:20:03.664438 IP 135.216.206.244.1025 > 209.0.129.195.53:  40900[|domain]
```

Several DNS servers are getting queries from one host. Some queries are transmitted only 1 ms apart from each other, but some intervals are as large as 1.4 seconds. The information from this packet trace is hard to relate to the peaks in Figure 5.5. In Section 5.5.2, an attempt is made to relate the found periodicities to the specification of DNS, given in RFC 1035.
When some periodicity has been identified, the traffic that is believed to cause this periodicity is removed from the original packet trace. On the traffic that remains after the filtering of the found periodicity, another run of `pat` is done and other periodicities may be identified.

After filtering the packet trace in this example from all DNS traffic, the next dominant frequency found is that of 16.000 Hz, which is caused by traffic from IP address `135.216.205.122` with port number 67, sent to `135.216.202.135` with port number 68. These are the port numbers that are associated with the Bootstrap Protocol (BOOTP), explained in Section 5.5.1. A Fourier transform of this traffic is shown in

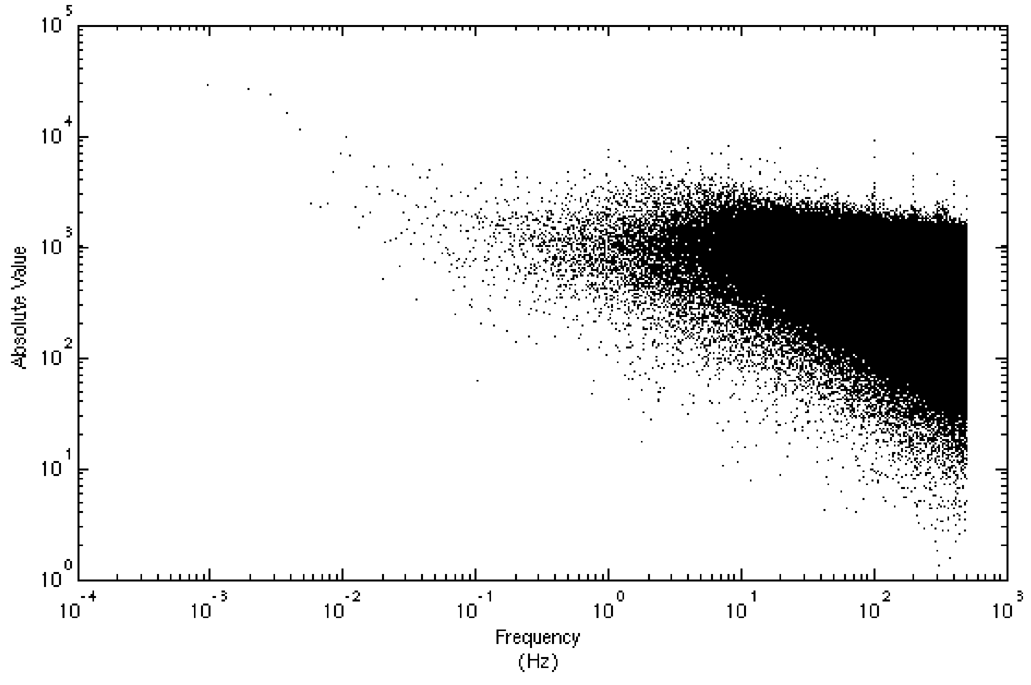

Figure 5.4: Fourier transform of all the UDP traffic in `loc1-20020626-0415` (183, 251 packets).

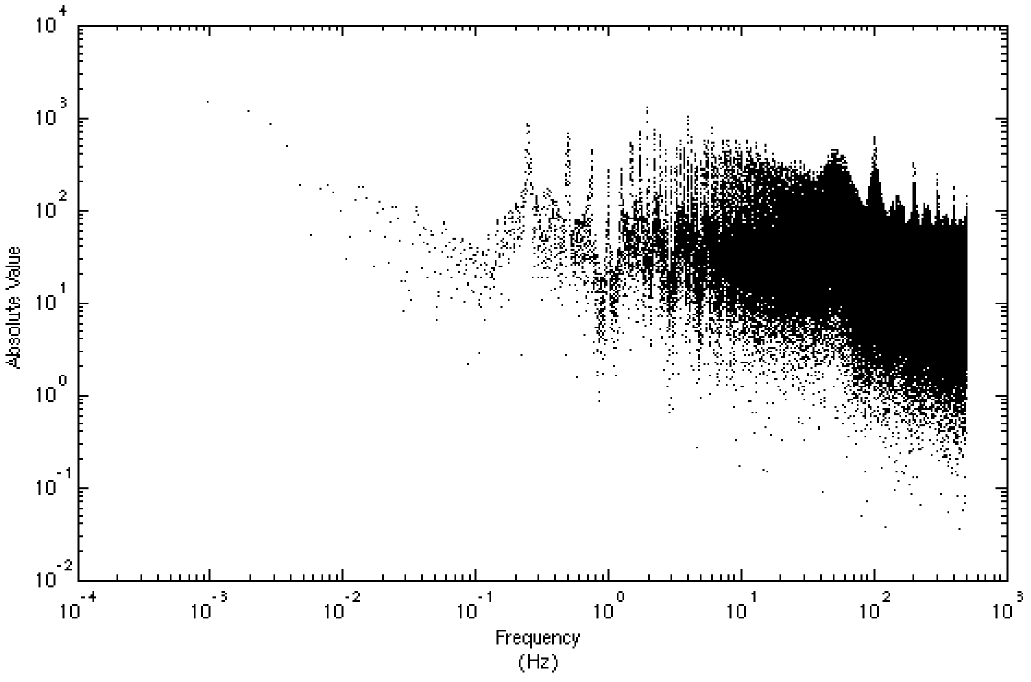

Figure 5.5: Fourier transform of DNS traffic from a specific host to several DNS servers (1, 561 packets).

Figure 5.6 The first peak seen in the figure is at the frequency of 2 Hz. The dominant frequency (about 16 Hz) that was determined by the tool was the frequency with the highest absolute value in the Fourier transform, but it is a harmonic frequency of the actual frequency of the periodic traffic. This means that the tool does not necessarily return the fundamental frequency of the periodicity, but may also report on a more dominant harmonic frequency that is present. The pattern seen in the figure is further described in Section 5.4.

The next dominant frequency in the packet trace is that of 1.640 Hz, and was caused by traffic between to hosts with 1028 as the source port number and 1900 as the destination port number. Port number 1028 is not associated with any service or protocol and port number 1900 is associated with the Simple Service Directory Protocol. The limited documentation available for this protocol made it impossible to determine whether it could be this protocol that was actually running on that port, causing the periodicity. Figure 5.7 shows the Fourier transform of this traffic. This plot shows a pattern that is similar to the arch pattern found in Chapter 4. A peak is present around 1.6 Hz, but another, even higher, peak can be seen at 2 Hz. Using this traffic as the input for `tcpdump`, the output has the following pattern:

```
03:24:48.496588 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 132
03:24:48.496731 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 133
03:24:48.999182 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 132
03:24:48.999284 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 133
03:25:13.496892 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 132
03:25:13.496994 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 133
03:25:13.999464 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 132
03:25:13.999561 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 133
03:25:38.498163 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 132
03:25:38.498273 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 133
03:25:39.000187 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 132
03:25:39.000285 IP 135.216.204.196.1028 > 135.216.205.122.1900: UDP, length: 133
```

Every 25 seconds, the following event occurs: two packets are sent 0.0001 seconds apart from each other and 0.5 seconds later another two packets are transmitted 0.0001 seconds apart from each other. The frequency associated with the 0.0001 seconds interval is way beyond the Nyquist frequency and therefore not visible in the picture: these two packets are just too close to each other to be separately noticed when using a sampling period of 1 ms. As a result, only the frequencies associated with 0.5 seconds (2 Hz) and 25 seconds (0.04 Hz) are visible in the figure. The 1.6 Hz frequency that was found earlier is a harmonic frequency of the latter frequency. The notches at 1 Hz, 3 Hz, 5 Hz etc. are also caused by the finite recurrence of two packets at an interval of 0.5 seconds, which will be explained in Section 5.4.

The next type of periodic traffic found in the trace is again BOOTP traffic, but this time it is broadcast traffic to destination port 67, which is the port number of a BOOTP server. The frequency is now 16.000 Hz, which lies very close to the frequency that was found earlier for BOOTP traffic from a server to a client. Plotting this broadcast traffic results in Figure 5.8, which is quite similar to Figure 5.6, where the traffic from server to client was transformed. That traffic and this broadcast traffic are directly related to each other: the packet traces showed that both periodicities were caused by the same IP addresses. The first peak is seen at 2 Hz, so the tool again reported a harmonic frequency instead of the fundamental frequency.

Traffic sent from the SNMP (Simple Network Management Protocol, see Section 5.5.7) port of one host to the SNMP trap port of another is the next kind of traffic, that was found as a likely cause of a periodicity. The analysis tool reported a frequency of 1 Hz, and taking the Fourier transform of the responsible traffic resulted in Figure 5.9. Although this figure seems to have some none random pattern in its Fourier transform, it is not as easily analysed as the other transforms seen so far. One possibility is this traffic *is* random and that the noise beyond the Nyquist frequency is “folded back” (aliased),

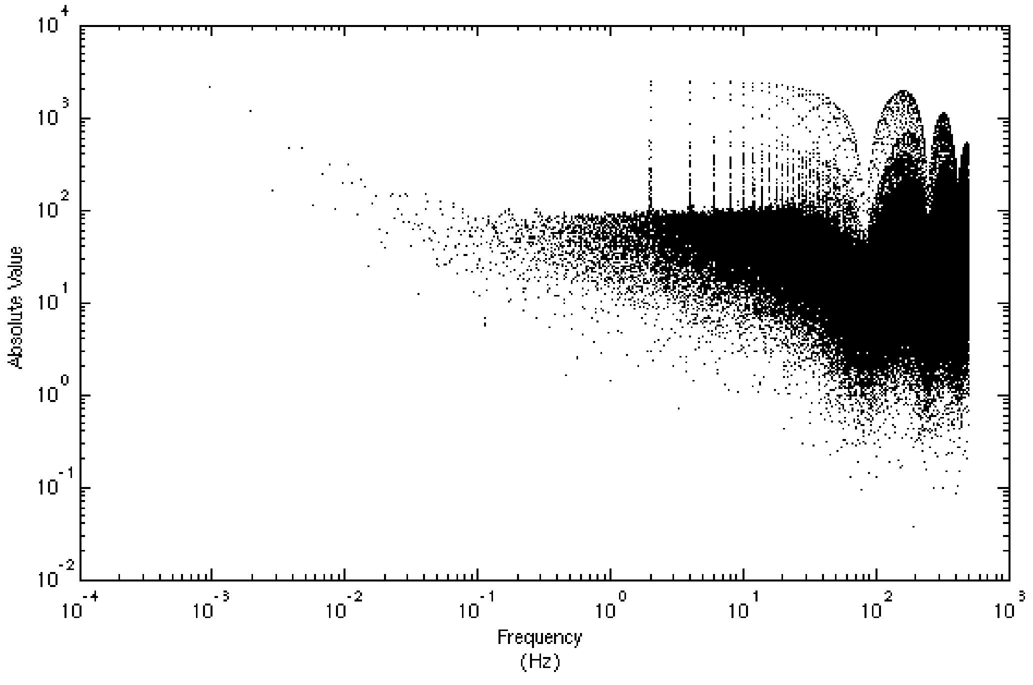

Figure 5.6: Fourier transform of BOOTP traffic from one server to one client (2587 packets).

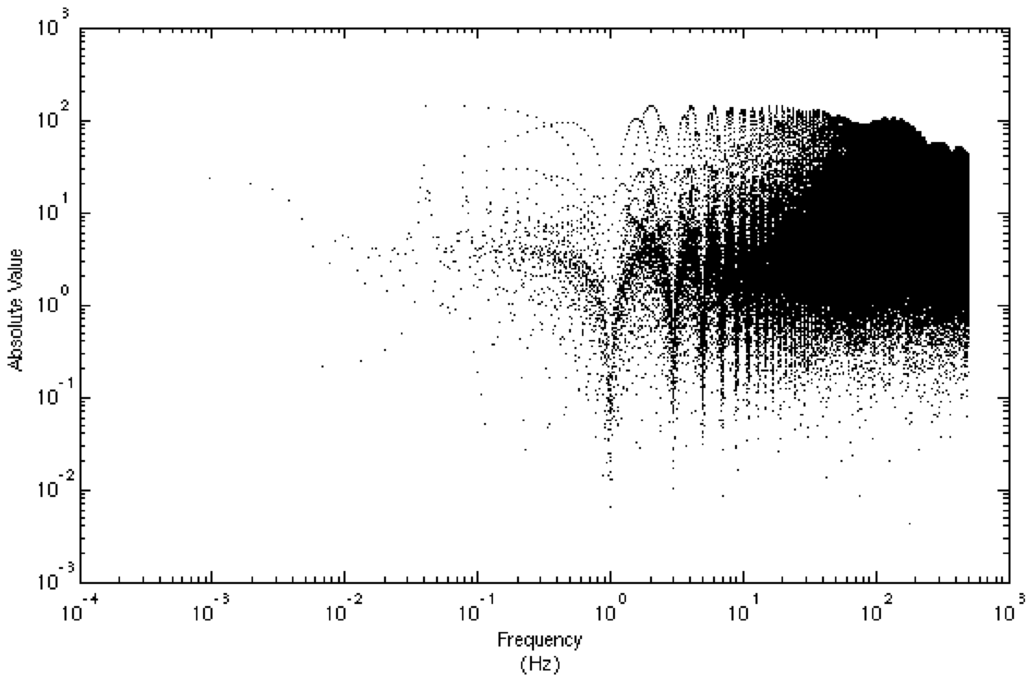

Figure 5.7: Fourier transform of traffic to port number 1900 (144 packets).

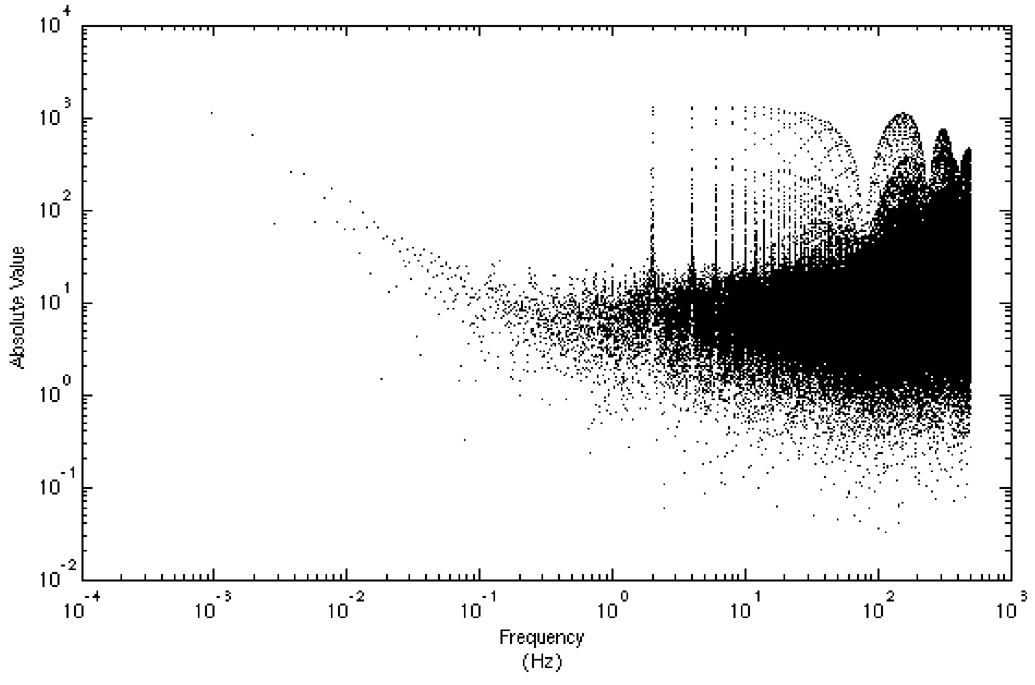

Figure 5.8: Fourier transform of BOOTP broadcast traffic (1377 packets).

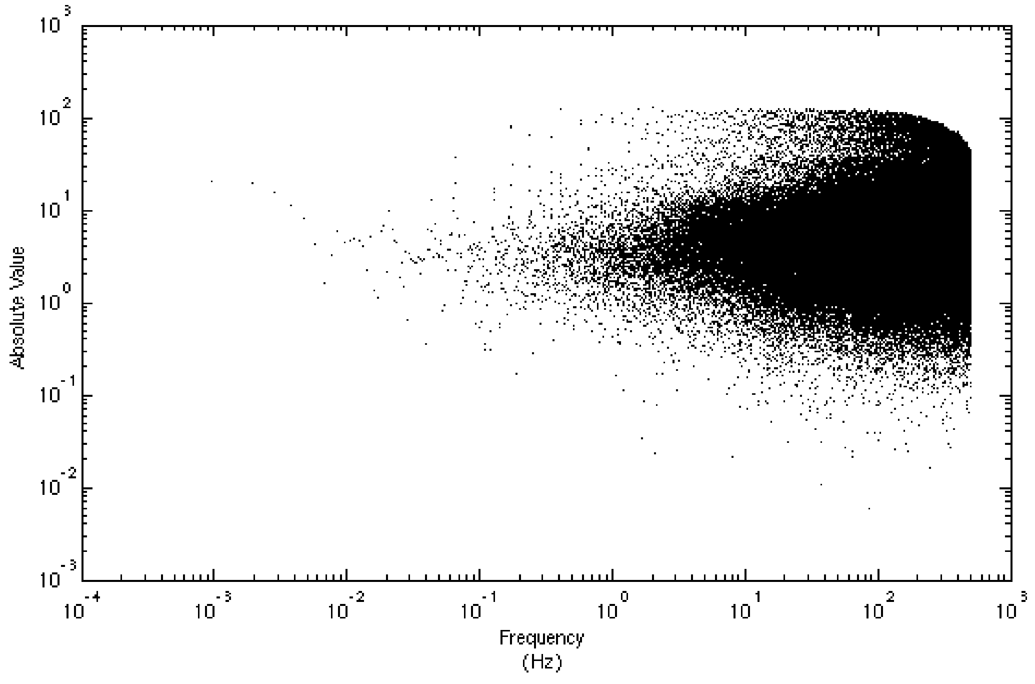

Figure 5.9: Fourier transform of SNMP trap traffic (130 packets).

which seems to create a pattern. Noise effects could be reduced by implementing a low pass filter so that frequencies beyond the Nyquist frequency are filtered out in advance.

The Samba protocol, operating on ports 137 to 139, also causes periodic behaviour. In Figure 5.10, the Fourier transform of traffic from one host's port 137 to another host's port 137 is shown.
The packet trace that was transformed shows an interesting pattern:

```
03:29:17.625347 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:19.127770 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:20.629490 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:26.638486 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:28.140552 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:29.642542 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:35.651276 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:37.153658 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:38.655573 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:44.664301 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:46.166656 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:47.669103 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:53.678279 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:55.180606 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
03:29:56.681734 IP 135.216.4.148.137 > 135.216.205.36.137: [|SMB]
```

After the first packet, the second is sent after 1.5 seconds. The interval between the second and the third packet is 1.5 seconds. The interval between the third and fourth packet is 6 seconds. From the fourth packet on, this pattern repeats itself: two intervals of 1.5 seconds are followed by one of 6 seconds. The length of one pattern is 9 seconds. The corresponding frequency for this is $1/9 \approx 0.11$ Hz, which explains the peak seen just to the right of 10^{-1} Hz. The period of 1.5 seconds corresponds to a frequency of 0.67 Hz, which is also showing up in the figure. Notches are also present in the figure, which will be discussed in Section 5.4.

Figure 5.11 shows the transform of the traffic that is responsible for the next found periodicity of 100.000 Hz. It is all traffic originating from one host, with source port 1864. According to the IANA list of well known port numbers, this port number is used by "Paradym 31"[1]. For this particular program, no standards or specifications are freely available so the found periodicity can not be clearly related to this program. Furthermore, the fact that IANA has registered this port number for this program, does not mean that all traffic sent to or from this port must have something to do with that program.
Remarkable is that the peaks in this figure are relatively strong compared to the peaks that were found in the earlier stages of the analysis but did not show up as dominant frequencies earlier. This might be another shortcoming of the algorithms used in the tool or it might just be inherent to the data that has been used.

The Time protocol, which operates on port 37, is the next cause of a periodicity at the reported frequency of 1.613 Hz. The tool reported that the possible cause of this periodicity could be traffic that is sent to one specific Time server. Figure 5.12 shows the Fourier transform of that traffic. The first peak is seen around 0.3 Hz. The following peaks all seem to be harmonic frequencies. The output PCAP file which was generated by the tool, with all the traffic to port 37 on the (scrambled) IP address `135.216.4.228`, shows nothing special at first:

```
03:15:01.300364 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.318269 IP 135.216.181.27.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.462261 IP 135.216.181.28.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.463355 IP 135.216.181.30.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.464054 IP 135.216.181.17.1024 > 135.216.4.228.37: UDP, length: 0
```

[1] `http://www.synapsesystem.com/paradym31.html`

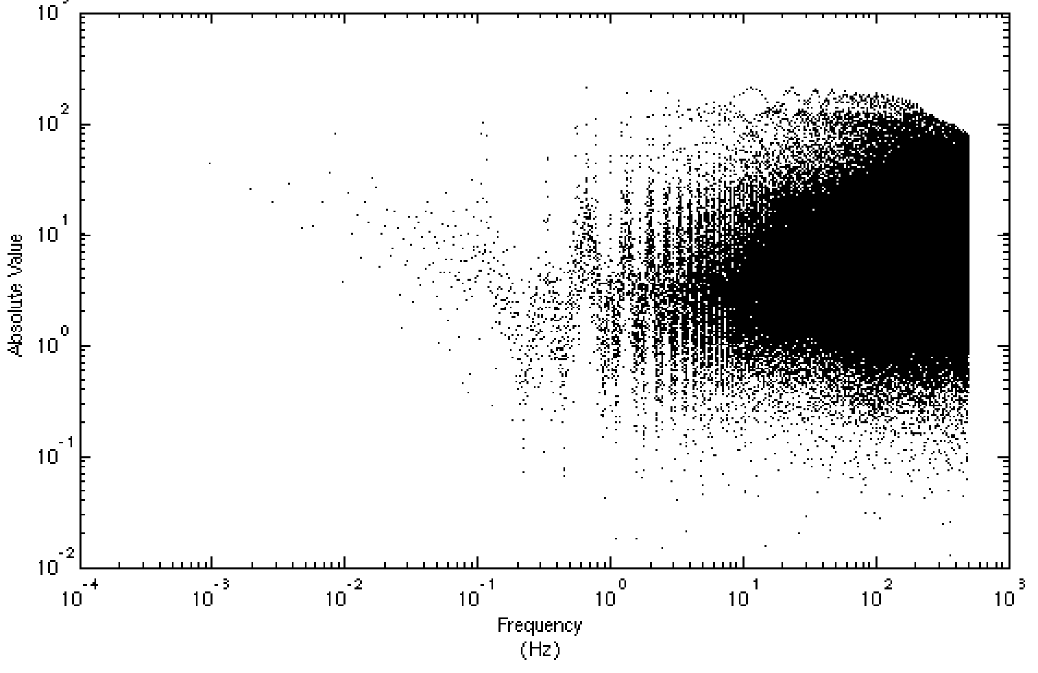

Figure 5.10: Fourier transform of Samba traffic on port 137 (222 packets).

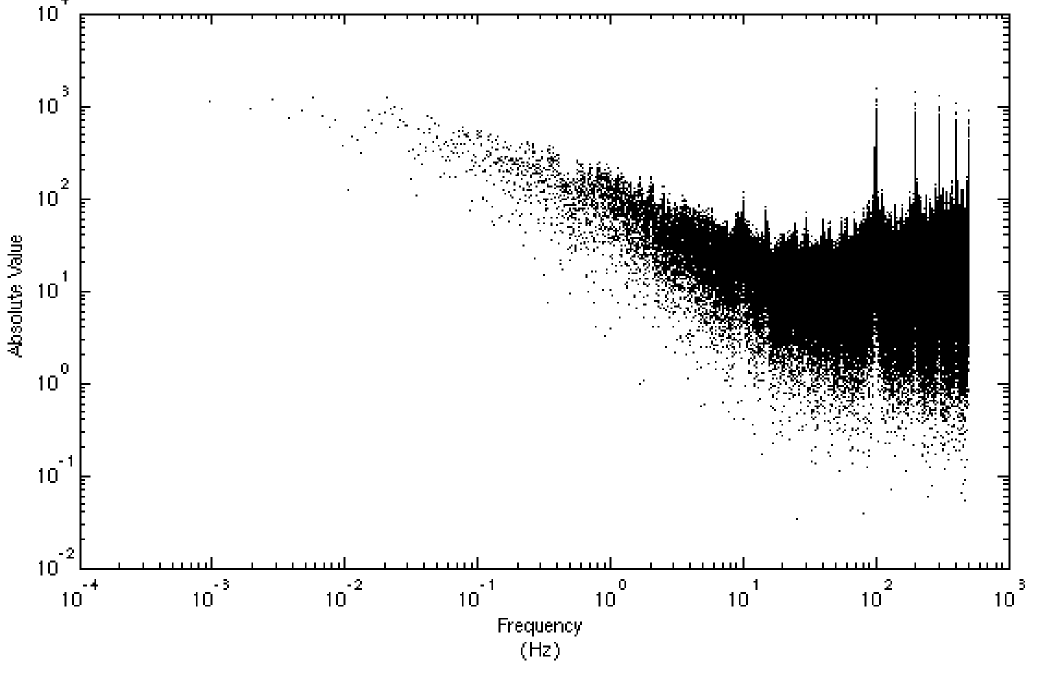

Figure 5.11: Fourier transform of traffic from port number 1864 (1612 packets).

```
03:15:01.532930 IP 135.216.181.3.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.678443 IP 135.216.181.22.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.828883 IP 135.216.181.26.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.865782 IP 135.216.181.50.1024 > 135.216.4.228.37: UDP, length: 0
03:15:01.912787 IP 135.216.181.46.1024 > 135.216.4.228.37: UDP, length: 0
03:15:02.081956 IP 135.216.181.35.1024 > 135.216.4.228.37: UDP, length: 0
03:15:02.116289 IP 135.216.181.37.1024 > 135.216.4.228.37: UDP, length: 0
03:15:02.143719 IP 135.216.181.20.1024 > 135.216.4.228.37: UDP, length: 0
```

These are all different hosts sending packets to the same Time server, but with completely unrelated time stamps. However, when this trace is divided per source host, the packet traces per host all show the same pattern:

```
03:15:01.300364 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:04.446701 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:07.501419 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:10.600876 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:13.747398 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:16.802028 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:19.902252 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:23.049271 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:26.129488 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:29.230975 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
03:15:32.349692 IP 135.216.181.21.1024 > 135.216.4.228.37: UDP, length: 0
```

This clearly indicates that every host is sending packets to the Time server, at a rate of 1 packet per 3 seconds. This immediately explains the frequency of 0.33 Hz.

The next frequency that is reported by the tool as a dominant frequency is 32.000 Hz. The output PCAP file showed that the most of the time, the traffic is sent in the following pattern:

```
03:15:17.603375 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:19.603517 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:21.603432 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:23.603467 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:25.603495 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:27.603528 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:29.603554 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:31.603586 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:33.603652 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:35.603636 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
03:15:37.603678 IP 135.216.201.137.3334 > 39.152.218.84.1025: UDP, length: 58
```

At a quite accurate rate of two packets per second, packets are sent from host to another, corresponding to a frequency of 0.5 Hz, which actually is the first peak seen in Figure 5.13. The port number 3334 corresponds to the web casting service of DirecTV. In Section 5.1, the presence of periodicities as a result of multimedia streams was already seen as as a possible outcome. If the traffic above is indeed originating from a web casting service, the assumption made about multimedia streams is now confirmed.

The previous causes of periodicities were all related to a well known protocol, or at least to some fixed port number. The following traffic that was found by the analysis tool (at 6.545 Hz), *does* show peaks in its Fourier transform which confirms the periodic activity, but the traffic is sent from different port numbers on the same host:

```
03:18:43.863520 IP 135.216.206.199.1136 > 135.216.185.14.39213: UDP, length: 268
03:18:43.864060 IP 135.216.206.199.1137 > 135.216.185.14.39213: UDP, length: 112
03:18:45.085559 IP 135.216.206.199.1138 > 135.216.185.14.39213: UDP, length: 824
03:18:46.307050 IP 135.216.206.199.1139 > 135.216.185.14.39213: UDP, length: 268
```

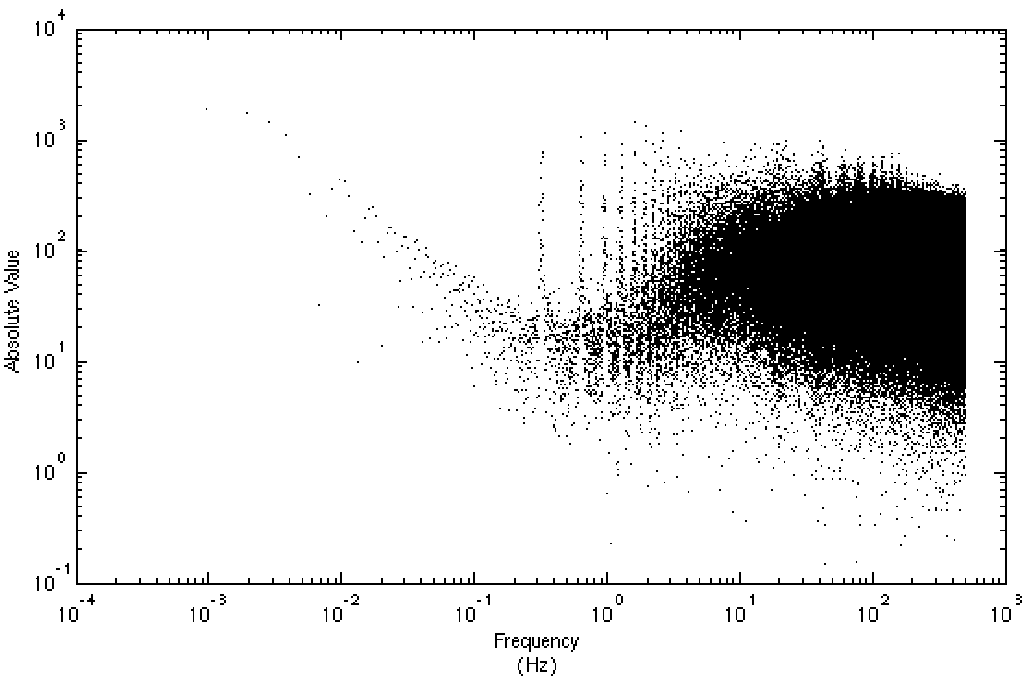

Figure 5.12: Fourier transform of Time traffic (11983 packets).

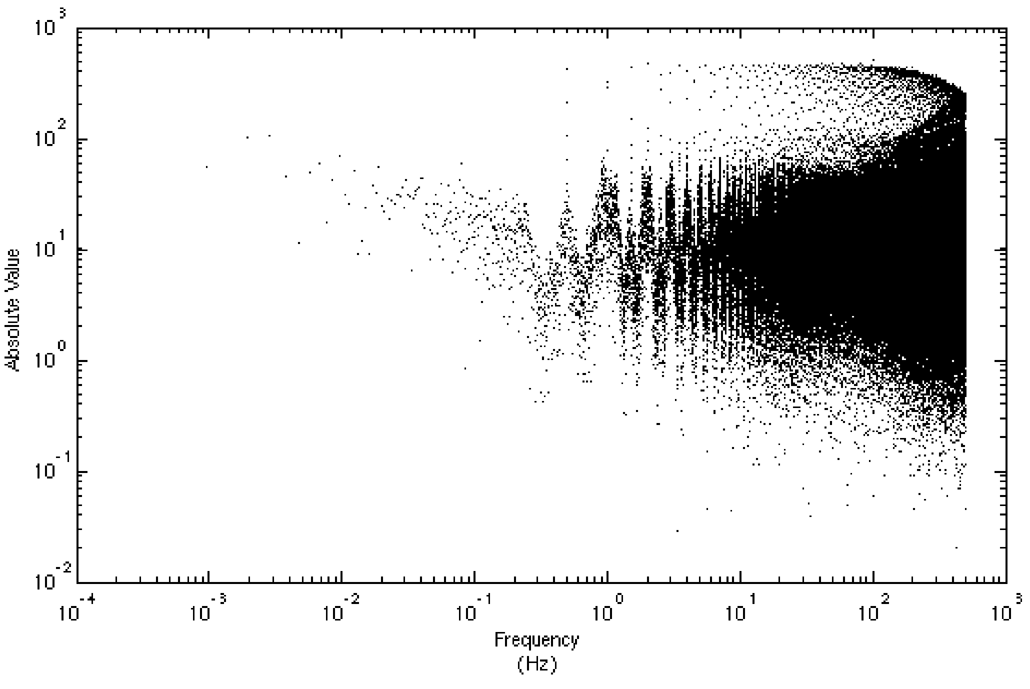

Figure 5.13: Fourier transform of DirecTV traffic (582 packets).

```
03:18:46.307593 IP 135.216.206.199.1140 > 135.216.185.14.39213: UDP, length: 112
03:18:48.139957 IP 135.216.206.199.1141 > 135.216.185.14.39213: UDP, length: 824
03:18:48.750559 IP 135.216.206.199.1142 > 135.216.185.14.39213: UDP, length: 268
03:18:48.751113 IP 135.216.206.199.1143 > 135.216.185.14.39213: UDP, length: 112
03:18:51.194069 IP 135.216.206.199.1144 > 135.216.185.14.39213: UDP, length: 268
03:18:51.195032 IP 135.216.206.199.1145 > 135.216.185.14.39213: UDP, length: 824
03:18:51.195301 IP 135.216.206.199.1146 > 135.216.185.14.39213: UDP, length: 112
```

The host with scrambled IP address `135.216.206.199` is firing away packets from successive port numbers. The reason for this traffic is unclear: packets are sent at a rate that is not constant and the destination port (39213) is not related to any well known protocol or service. If it were the other way around, one could suspect some kind of port scanning activity going on, but examining the original packet trace for traffic *from* `135.216.185.14` *to* `135.216.206.199` shows no packets at all. Perhaps this machine is infected with some kind of virus or worm which causes it to send out rubbish. Figure 5.14 shows the Fourier transform of this "scan"-like traffic. From the complete packet trace (of which a small part is shown above) it seems that packets are sent in groups of two or three packets at a time, with an interval of 2.5 seconds. This corresponds to a frequency of 0.4 Hz, which is shown in the figure. Because the source of this traffic is using successive port numbers, no specific protocol or service can be assigned to this periodicity.

The next frequency found by the tool is 32.010 Hz, which is confirmed by Figure 5.15: it shows a peak next to 30 Hz. The related packet trace shows traffic originating from port 800, and sent to machines with port 2049. The part which seems responsible for the mentioned frequency is:

```
03:28:13.114675 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.115027 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.116613 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.116967 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.117317 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.117669 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.144947 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.145236 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.145651 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.146003 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.147590 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.147942 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.148293 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.148647 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.174644 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.174996 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.175348 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.175699 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.177218 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.177571 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
03:28:13.177922 IP 135.216.194.68.800 > 135.216.207.22.2049: UDP, length: 4216
```

Groups of around 8 packets are sent with an interval of around 0.03 seconds, which corresponds to the found frequency. The port number 800 is associated with the "MDBS" protocol or service and the port number 2049 is associated with the Networked File System (NFS). The MDBS protocol seemed to have no documentation available and it is therefore impossible to explain the periodicity with respect to that protocol. The Networked File System protocol is a widely used protocol and could theoretically be the cause of this periodicity (see details in Section 5.5.3).
The traffic that still remained in the packet trace after the removal of these periodicities, showed no more significant periodic activity (see Figure 5.16) and the analysis of this packet trace is considered done. From the original 183251 packets in the trace, 9431 remained.

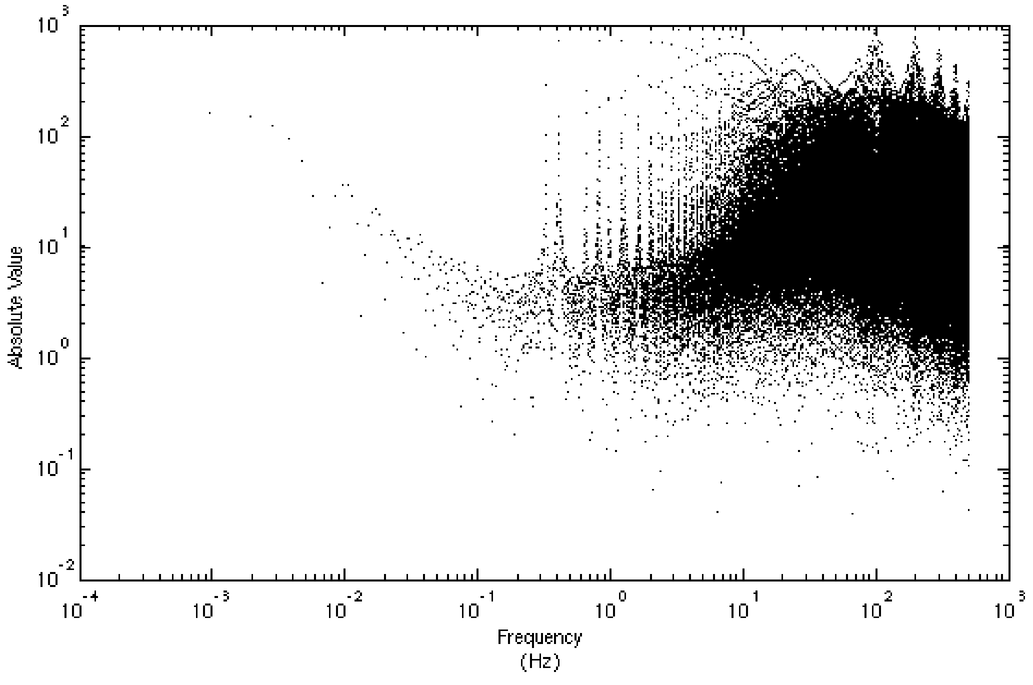

Figure 5.14: Fourier transform of “scan”-like traffic (1034 packets).

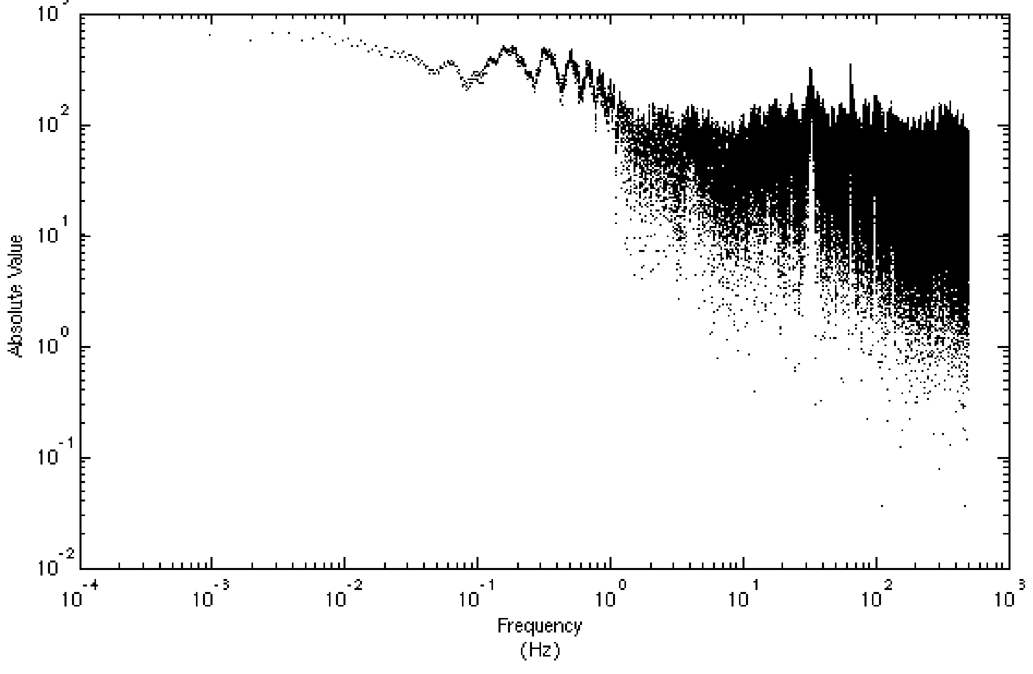

Figure 5.15: Fourier transform of traffic from port 800 to port 2049 (724 packets).

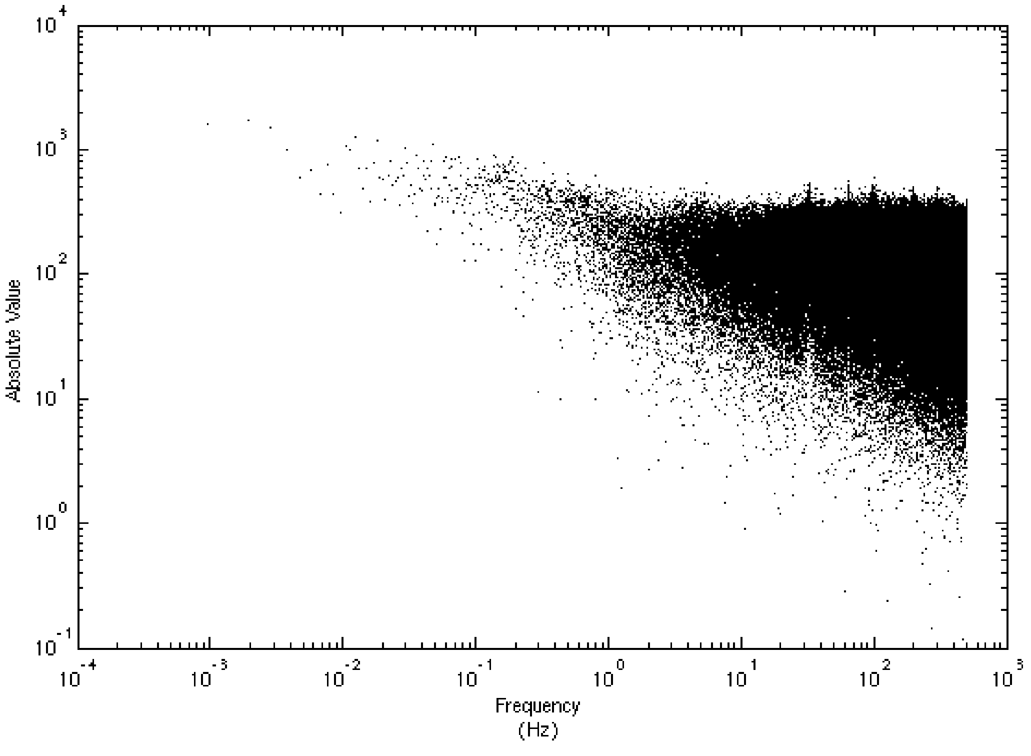

Figure 5.16: Fourier transform of the traffic remaining after the removal of periodicities (9431 packets).

5.4 Common patterns in UDP frequency spectra

Most of the patterns that showed in the frequency spectra of traffic that seemed responsible for the presence of a periodicity, were series of notches and/or a series of sharp peaks. When examining the traffic data in the time domain, it became clear that most of the time, the traffic has the (time domain) form depicted in Figure 5.17. A group op n packets is sent every T_g seconds. The packets within each

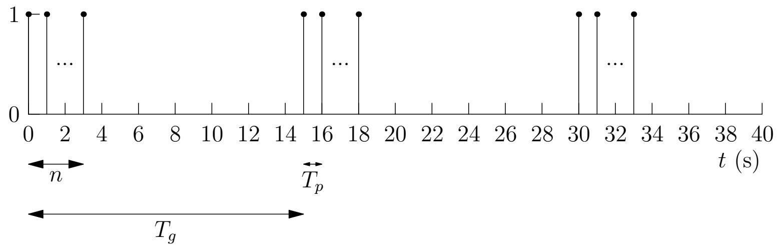

Figure 5.17: Common time domain pattern of periodic traffic.

group are sent T_p seconds apart from each other. This behaviour can be simulated in MATLAB with the code from Listing 5.1. The total time of the simulation is 900 seconds, since this corresponds with the 15 minute length of the packet traces. The rest of the variables can be set to any (realistic) value. Taking the Fourier transform of the signal built in the example gives Figure 5.18.
This shows that a repeating signal of 1 packet at a time at a frequency of 1 Hz causes a peak to show up at this frequency and all the harmonics of this frequency. The Fourier transforms shown in the previous sections – as well as in Chapter 4 – are not noise free. Adding phase noise to the signal in this example gives Figure 5.19. The same pattern is still showing, but this time the peaks more to the right of the plot are decreasing in their absolute value. Whether there is noise present or not: a series of peaks (at some fundamental frequency and harmonic frequencies) indicates a that a strong infinitely repeating signal is present.
The presence of notchesin a Fourier transform was already discussed in Section 4.3: a finite number of retransmissions causes these notches to appear. Combining these two phenomena (both the infinite repeating of a signal and a finite number of retransmissions) results in a signal seen earlier in Figure 5.17. groups of a finite amount of packets get sent at a fixed mutual interval. In turn these groups are infinitely repeated at a fixed interval.
Simulating this behaviour with a number of 2 packets sent at a time (spaced with 0.01 seconds) at an interval of 1 second, results in Figure 5.20. Now both peaks and notches show up in a clear way. The figure also shows resemblance to the BOOTP figures seen in Section 5.3. This shows that the repeatedly sending of groups of packets, which are very close to eachother results in a Fourier transform showing peaks at the frequency of the sending of groups of packets and that notches are created by the frequency within such a group.
The difference between the last simulation and the BOOTP figures are the arches which are running through the peaks. In all three simulations so far, $T_g = 1$ was chosen, which is a divider of the whole time period of 900 seconds. If $T_g = 1.007$ is chosen, the total time period can not be divided in a round number of intervals. Leaving $n = 2$ and $T_p = 0.01$, Figure 5.21 is the result, which shows the abovementioned effect.

5.5 Analysis of UDP spectra

In Section 5.3 one packet trace was thoroughly analysed. This section shows the results that were gathered by examining other packet traces with the tool. Per protocol or service that was believed to cause a periodicity, the frequency that was associated with it is given. If necessary, a short functional

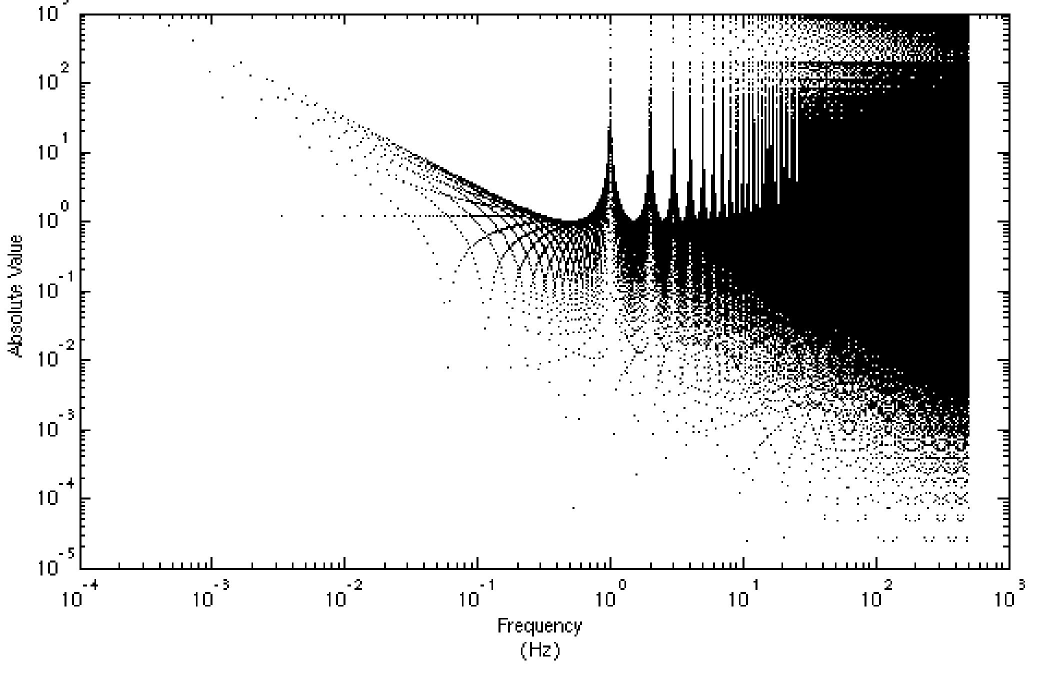

Figure 5.18: Fourier transform of a signal with $n = 1$ and $T_g = 1$ (noise free).

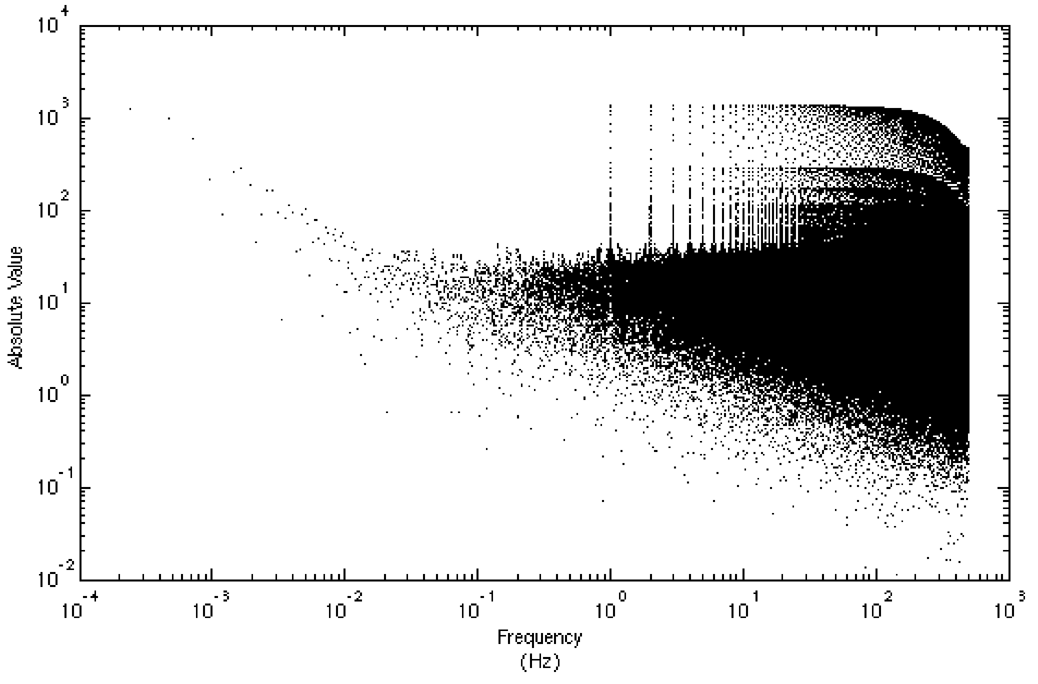

Figure 5.19: Fourier transform of a signal with $n = 1$ and $T_g = 1$ (with noise).

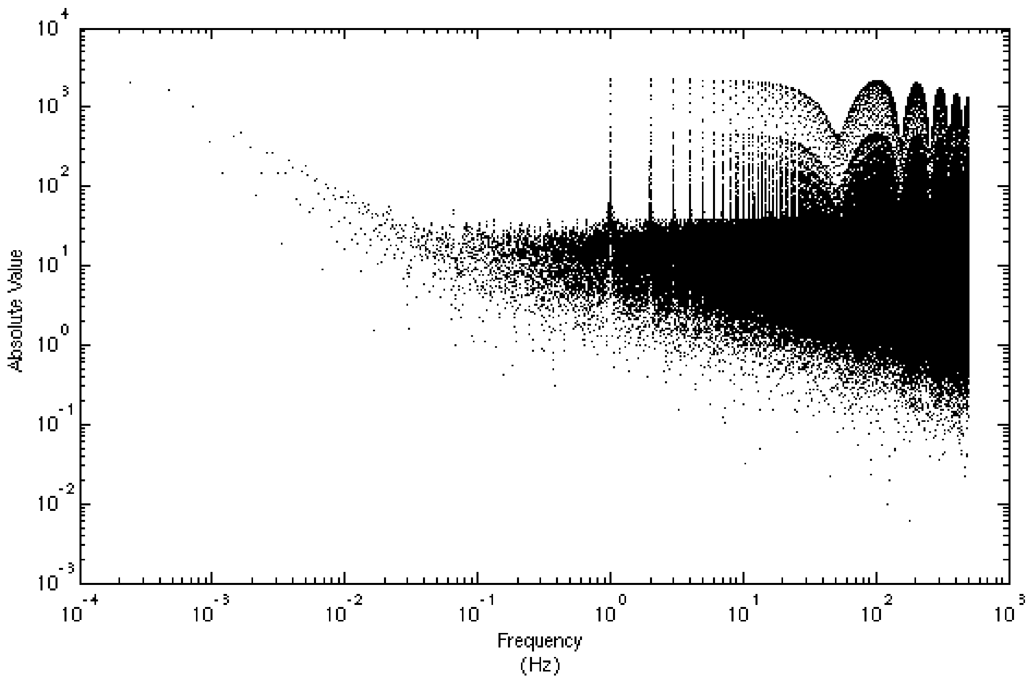

Figure 5.20: Fourier transform of a signal with $n = 2$, $T_g = 1$ and $T_p = 0.01$.

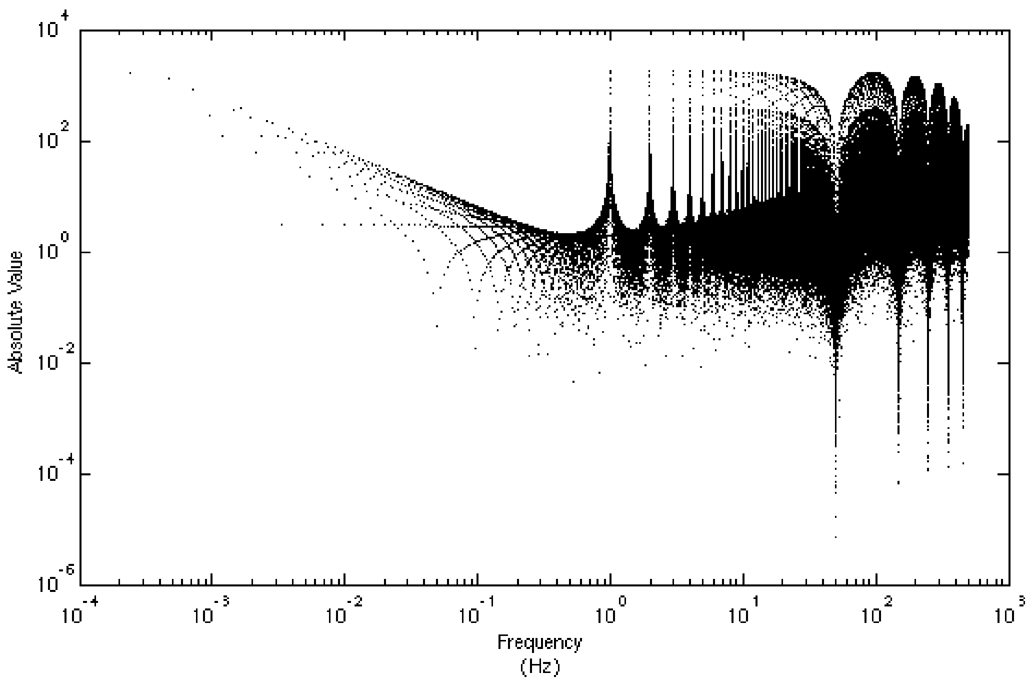

Figure 5.21: Fourier transform of a signal with $n = 2$, $T_g = 1.007$ and $T_p = 0.01$.

```
% Total time
Tend = 900;

% Sampling interval (seconds)
t = 0.001;

% Period of time between groups of packets (seconds)
Tg = 1;

% Period of time between two packets of one group (seconds)
Tp = 1;

% Number of packet within one group
n = 1;

% Add phase noise (1 for yes, 0 for no);
noise = 0;

% Calculation of the signal
ind=1:Tg/t:Tend/t;
f=zeros(1,2^22);

% Add phase noise to groups
ind = ind + noise*rand(1,length(ind))*2;

% Create packets in groups
for i = 0:n-1
    f(floor(ind+i*Tp/t))=1;
end

% Build the signal
f(floor(ind)) = 1;
```

Listing 5.1: Matlab code to recreate common patterns of periodic traffic.

description of the protocol is given. Finally, an effort is done to explain this exact frequency, based on the standard (e.g. an RFC) in which this particular protocol is described.

5.5.1 Bootstrap Protocol

The Bootstrap Protocol (BOOTP), which is defined in RFC 951, makes it possible for booting hosts to configure their network settings themselves without user intervention. When a host boots, it will send out (broadcast) a BOOTREQUEST message to port 67, which is the standard port for a BOOTP server. If such a server exists on the network, it will reply with a BOOTREPLY message, which contains the configuration information for the client. The RFC of the BOOTP protocols mentions that a successful “BOOTP procedure” only takes a single packet exchange. Since a single packet exchange can not be the cause of a periodicity, the periodic behaviour caused by the BOOTP protocol should be the result of some error. The only error that can occur is that a host sends out its BOOTREQUEST message, but does not receive a BOOTREPLY message. According to the RFC, timeouts are used to retransmit requests until a reply is received. This is exactly what will cause periodic traffic, if requests keep getting unanswered. The definition of the BOOTP protocol does not give any information about the time interval that should be chosen for the timeout, so no specific frequency can be assigned to this protocol.

5.5.2 Domain Name System

In Section 2.2.5, the Domain Name System was already mentioned as the translator between hostnames and IP addresses. In this chapter the focus is more on *how* protocols and services do their work and how this could result in period traffic. A lot of protocols and services work with a request and reply system. For DNS, this is true as well. When a host needs a translation from a hostname to an IP address or the other way around, it sends a query to a DNS server (port 53), which then replies with the correct translation. Since this discussion is about DNS over UDP, which can result in the loss of packets, a retransmission strategy should again be used. RFC 1035 does not give a specification on the retransmission strategy that should be used, but does give the following recommendations for it:

- The client should try other servers and server addresses before repeating a query to a specific address of a server.
- The retransmission interval should be based on prior statistics if possible. Too aggressive retransmission can easily slow responses for the community at large. Depending on how well connected the client is to its expected servers, the minimum retransmission interval should be 2-5 seconds.

If these recommendations are implemented, the Domain Name System will cause periodicities with frequencies ranging from 0.2 to 0.5 Hz. Returning to the previous section were DNS was causing a periodicity, these two recommendations seem to be implemented by the client which was sending the queries: multiple DNS servers are contacted and the frequency of 0.2 Hz is showing up in the Fourier transform. It is, however, still not clear what is causing the higher frequencies to show up in the transform on page 41.

5.5.3 Network File System

The Network File System (NFS), described in RFC 3035, is a distributed file system protocol. The main idea behind this protocol is to make files available for other hosts (called clients) on the network. These clients can then “mount” these file systems and perform operations on it as if it were a local file system. The whole operation of the protocol relies on commands (e.g. OPEN and CLOSE commands) sent from a client to a server over either TCP or UDP. The possible cause of periodicities within the NFS protocol is again the retransmission of these commands if no reply is received upon them. The protocol specification does not give a (recommendation for) a fixed retransmission period, so the frequency that was found in the previous section can not be confirmed by the specification. Another reason for periodicities caused by the NFS protocol could be the streaming of some file from a mounted Network File System.

5.5.4 (Network) Time Protocol

The Time protocol (port 37, RFC 868) and the more sophisticated Network Time Protocol (port 123, RFC 1305) both have the same function: supplying the possibility to synchronise internal clocks of multiple hosts. The client sends a request to a time server and receives a reply with information which enables the client to synchronise its clock to the server’s clock. Again one might expect periodicities because of the following reasons

- A client sends a request but does not receive a reply and will retransmit requests until it does receive a reply.
- A client regularly wants to synchronise its clock with the time server, which causes requests to be sent every few minutes, hours or days. In this case periods of hours and days can not be detected: the packet traces only contain 15 minutes of traffic.

The client again decides the retransmission interval, so no fixed frequency can be associated with this protocol.

5.5.5 Real-time Transport Protocol

In sections 2.2.4 and 5.1, the streaming of multimedia was already considered as a popular application on top of the UDP protocol, which could also cause periodic behaviour. Some packet traces indeed showed the RTP protocol as a cause of periodicities. The Fourier transforms of RTP traffic shows a lot of peaks, which do not seem related to each other (see Figure 5.22). In this section, an effort is done to explain why there is no clear pattern between these peaks.

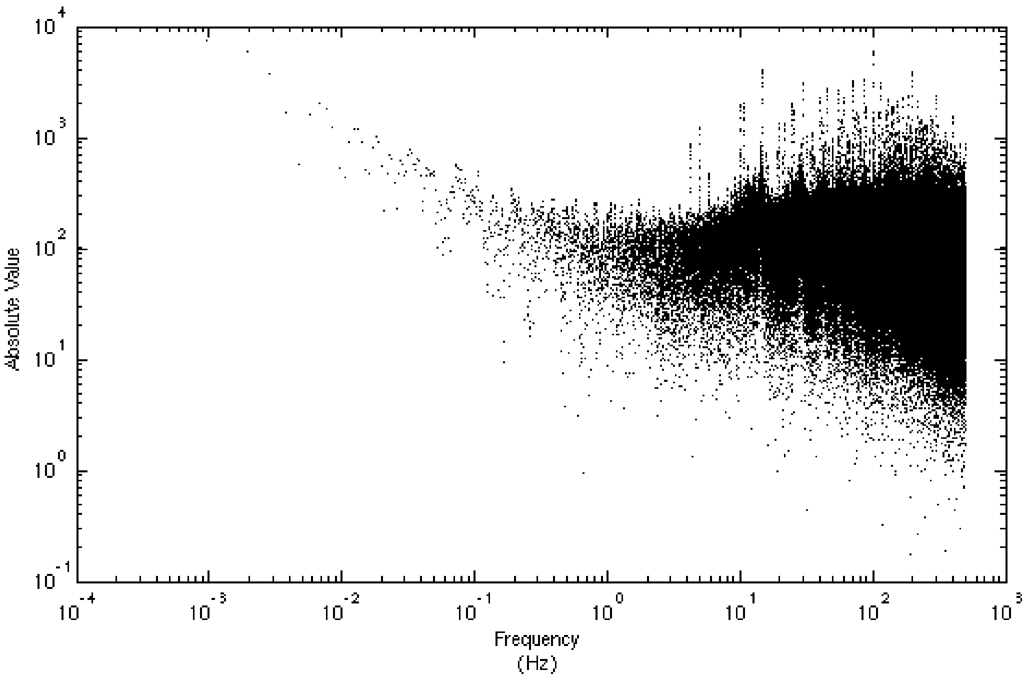

Figure 5.22: Fourier transform of RTP traffic (multiple streams).

When streaming multimedia files, a client requests a file from a certain server. The server then sends the file to the client by segmenting the file. These segments are then encapsulated with special headers. The Real-time Transport Protocol (RTP) is a standard protocol capable of encapsulating such segments and is used by applications like QuickTime and RealPlayer. RFC 3550, which gives a specification of the RTP protocol and the related RTCP protocol (Real-time Transport Control Protocol), has the following information about the periodic traffic that these two protocols are causing:

- Several clients can request the same multimedia file at the same time. To keep track of the amount of clients that are still present and the reception quality of those clients, these clients periodically send out so-called reception reports through the RTCP protocol.
- RTP-packets (which contain the actual stream) encapsulate the segments of data that should be sent to the client(s). It is possible that these segments are sent out periodically as well.

So in real-time streaming applications, the RTCP messages are more likely to cause periodic traffic than the RTP messages containing the stream itself. The RFC specifies how the RTCP transmission interval should be calculated in sections 6.2 and 6.3.1. Unfortunately, this transmission interval (and therefore also the frequency) is not fixed: if reception reports from clients would be sent out at a constant rate, the amount of control traffic would grow linearly with the amount of clients. The RFC mentions more factors that influence the RTCP transmission interval, which make it even more difficult to state something about a specific frequency (range) that could be related to RT(C)P traffic. This also explains the amount of (unrelated) peaks in Figure 5.22.

5.5.6 Samba

Samba[2] is a software suite that runs on Unix systems. Its goal is to integrate Unix systems in a Windows environment. Probably the most used feature of the Samba suite is the sharing and accessing files on a network. Although "Samba" is the name of Unix software, it is more widely used as a synonym for the protocol it implements: the Server Message Block (SMB) protocol, which is the protocol that is also used between two Windows machines for sharing files and printers. The SMB protocol also works with a request and reply paradigm: a client that wants to access files shared by a server, communicates with that server with certain commands (e.g. `open` and `close`) to which a server responds. In that sense, the idea of Samba is quite the same as that of the Network File System, described in Section 5.5.3. For this reason, one might expect the same type of periodicities in Samba traffic as in NFS traffic.

5.5.7 Simple Network Management Protocol

The Simple Network Management Protocol also showed periodic behaviour in some packet traces. The protocol is described in RFC 1157 and was developed to facilitate management of network devices (e.g. routers, switches). SNMP is another request/reply protocol that uses two types of request messages. One type is a request to get status information about a device: the traffic load on that specific device for example. The other type is a request message to set certain parameters in the device. These messages should not be causing periodicities at all, but in the previous section SNMP traffic *did* show periodic activity. The destination port number that was associated with the traffic, already showed that it were so-called SNMP "traps" that were causing the periodic behaviour. Traps have nothing to do with the above mentioned "get" and "set" requests that are sent by a managing device, but get sent by the managed device unsolicited in case of an error or some other important event. So the periodicity found earlier was probably caused by a device that was repeatedly reporting errors to a managing device. Unfortunately, the payload of packets in the packet trace is not available, so it can not be confirmed what the actual trap message was. Neither can a specific frequency be assigned to SNMP trap traffic, since the transmission interval of the traps depend on the origin of the trap.

5.5.8 Online games

The last type of traffic that is worth mentioning is caused by (real-time) online games. A lot of traffic that showed periodic behaviour was traffic sent from/to ports beyond port number 25000. When searching for applications related to those port numbers, mostly titles of popular games that can be played online show up (e.g. Quake III, HalfLife). Typical fundamental frequencies that were associated with this traffic lie in the range of 10 Hz to 100 Hz, which means that these hosts sent information to each other with transmission intervals between 10 and 100 ms. It is imaginable that this is information about the status of the players in the game: their location within the game, the weapon they're carrying etc. This is all information that has to be updated regularly in order to keep the game real-time. Online games are therefore a very plausible cause of periodicities.

5.6 Result of UDP analysis

5.6.1 Causes of periodicities

The most natural cause of periodic traffic is real-time traffic, like the streaming of multimedia and the exchange of information by online games. The synchronising of clocks through time protocols also caused periodicities to show up. These are types of traffic that show periodic behaviour simply because

[2]`http://www.samba.org`

of their nature and function. However, periodicities are also caused by traffic that does not have these characteristics.
Many UDP protocols implement the request/reply paradigm: a client sends a request which should be replied upon by a server. If no reply is received, the client sends it request again until it receives a reply. These retransmissions of requests result in periodic traffic, which was also the case with the DNS traffic. Another cause of periodic behaviour was SNMP trap traffic, probably caused by a failing network device which repeatedly reported the failure. Abnormal behaviour of certain machines, like the "scan"-like traffic seen in Section 5.3, was also detected as periodic behaviour by the periodicity analysis tool. One conclusion therefore is that many periodicities in UDP traffic are caused by error-like behaviour: replies not being received (in time), error-reporting machines and hosts sending out packets that probably have no function at all (perhaps caused by viruses or worms).

5.6.2 Evaluation of the analysis tool

The analysis of UDP traffic relied heavily on the Periodicity Analysis Tool (Chapter 6). In the course of the research on UDP traffic, some advantages and disadvantages of the tool became clear. The advantages of the tool:

- Dominant frequencies were discovered accurately most of the time. In some cases, the reported frequency was not the fundamental frequency of the periodicity, or the reported frequency disappeared (or became non dominant) in the Fourier transform of the traffic that was believed to be responsible for the periodicity.
- The binary search algorithm proved to be quite a fast and accurate algorithm to relate a certain frequency to specific traffic, although the influence of the number of packets on the calculated Fourier transforms' absolute values could cause disturbances.
- Performing binary searches on both source *and* destination addresses/ports sometimes made it more easy to pinpoint the traffic that was responsible for a periodicity: sometimes the output PCAP file created by the binary search on a responsible source address was exactly the same as the output PCAP file created by the binary search on a responsible destination address, which confirmed quite strongly that these two hosts were responsible for the found periodicity.
- Output PCAP files with likely causes of a periodicity were written automatically by the tool. These PCAP files could then be used as a good starting point for further examination.

Disadvantages of the tool - one of which is already discussed in Section 5.2.2 - are:

- Strong periodicities caused by small amounts of traffic probably did not get discovered if there was much more "white noise" traffic disguising it.
- After one run (i.e. the identification of *one* dominant frequency and its possible causes) of the tool, user intervention was still needed to pick out the most likely cause of the periodicity by manually examining the output PCAP files.
- The dominant frequencies that are reported by the tool are always correct, but it would be more convenient if the tool could decide whether or not it is a fundamental or a harmonic frequency.

The first disadvantage can be overcome by considering another possible algorithm instead of the binary search algorithm to find the traffic that is causing some periodicity or by defining another calculation to compare two Fourier transforms so that the influence by the number of packets is cancelled out. The second disadvantage can be overcome by extending the tool: Fourier transforms of all output files can be made and compared to each other to determine the most likely cause of the periodicity. The winning output file can then be further examined by the tool to get more detailed information about the cause of the periodicity. For example: in Section 5.3, a periodicity was found caused by traffic sent to one specific time server. The output file associated with it did not show any obvious periodic behaviour until the output file was further split per source host. If the tool could do this automatically that would be a big advantage. The problem with fundamental/harmonic frequencies is a lot harder to solve: every time a dominant frequency is determined, the tool should have some kind of means to check whether it is an integer multiple of a lower (less dominant) frequency. To do this, the tool could inventory the first couple of dominant frequencies to see if they are related in some way and if this is the case, choose the lowest frequency as the fundamental frequency.

Chapter 6

Periodicity Analysis Tool

The approach to identify the traffic causing a certain periodicity that was described in Section 4.2 was no longer helpful in the research on periodicities in UDP traffic. The amount of UDP traffic in a trace is much bigger, which causes figures like Figure 4.2 to be much more crowded with dots. In that figure, it was possible to see a line at an interval of 0.01 seconds with the naked eye. In a figure with a higher density of dots, these lines are not easily seen, or not seen at all. Subsequently, it is not possible to determine which packets are associated with this line and the hunt for the cause of a periodicity is forced to stop. Moreover, plotting several figures and drawing conclusions about them just by looking takes a lot of time.
To overcome these problems, a tool was developed in C++ to identify the traffic causing a certain periodicity in UDP (and TCP[1]) traffic, without human intervention. This way, more detailed information can be extracted from a packet trace and time is saved. This "Periodicity Analysis Tool", which carries the name `pat`, is capable of the following:

- Accepting a packet trace from the traffic repository as input.
- Determining a dominant periodicity with a certain frequency in that packet trace.
- Associating traffic with a common property as the cause of this periodicity. Possible common properties are: the same source port, destination port, source IP address or destination IP address.
- Generating output files which split the traffic that seems responsible for the periodicity from the other traffic.

The main challenge with the development of this tool was to make it as efficient and fast as possible. Only useful information must be extracted from the packet trace and Fourier transforms should be carried out as fast as possible. In this chapter, the Periodicity Analysis Tool is described together with some code listings, to show some implementation details.

6.1 Program usage

The program is designed as a pure command line application and should be used with the following arguments:

```
pat udp|tcp type trace_file
```

The first argument tells `pat` whether it should handle UDP or TCP traffic. The `type` argument should be a number ranging from 1 to 15, indicating the desired output. This number should be the sum of one or more of the following values:

[1]The analysis of TCP traffic is not discussed in this thesis, but the tool can handle this.

8	Source port
4	Destination port
2	Source IP address
1	Destination IP address

Notice that each of these values is an exact power of 2, which means that every number from 1 to 15 has its own unique sum of these values associated with it. The given number will in fact be decomposed by `pat` as a sum of these values. For each value that is in the decomposition, a binary search algorithm (discussed in Section 6.2.4 is performed on the field that is represented by that value. Output files created by the program, which are discussed in Section 6.2.5, also depend on this second argument.
The final argument is the packet trace file (in PCAP format) that is to be analysed. To clarify the use of the periodicity analysis tool, consider the following example: suppose the trace file `trace.pcap` should be analysed to find periodic UDP traffic, caused by packets with the same destination port or packets originating from the same source IP address. The following command should be executed to give the desired output:

```
pat udp 6 trace.pcap
```

The first and third argument speak for themselves, but the second argument probably needs further explanation. When the number 6 is decomposed as a sum of the values given above, the only possible sum is 4 + 2, which means that a binary search on “destination port” and ”source IP address” will be done.

6.2 Program structure

After the parsing of the command line arguments, the program will follow the flow diagram from Figure 6.1. The example given in the previous section is illustrated in the diagram with a grey line. In the following sections, the functionality of each (rectangular) block and its implementation is explained.

6.2.1 Inventorying the packets

First, the *relevant* packets from the packet trace that was given at the command line are inventoried. This means that only the packets within the packet trace that are of importance in the execution of the program end up in the inventory, e.g. if TCP traffic should be searched for periodic activity, UDP packets will *not* end up in the repository.
Reading the packet trace is done with the help of the PCAP library[2] `libpcap`, which is also used by programs like `tcpdump` and `ethereal`. This library provides functions to read and write packet traces. Together with the C-`struct` definitions for protocol headers and accompanying functions and variables that are provided by C++[3], it is possible to extract the header information that was discussed in Chapter 2 from the packets in the packet trace. Per packet, the following information is stored:

- Time stamp
- Source IP address
- Destination IP address
- Source port
- Destination port
- Contribution to the real part of a Fourier transform

[2]The PCAP library is available at `http://www.tcpdump.org/`

[3]To see which C++ libraries are used in the program, the reader is encouraged to take a look in the source of `pat`, which is available together with this thesis.

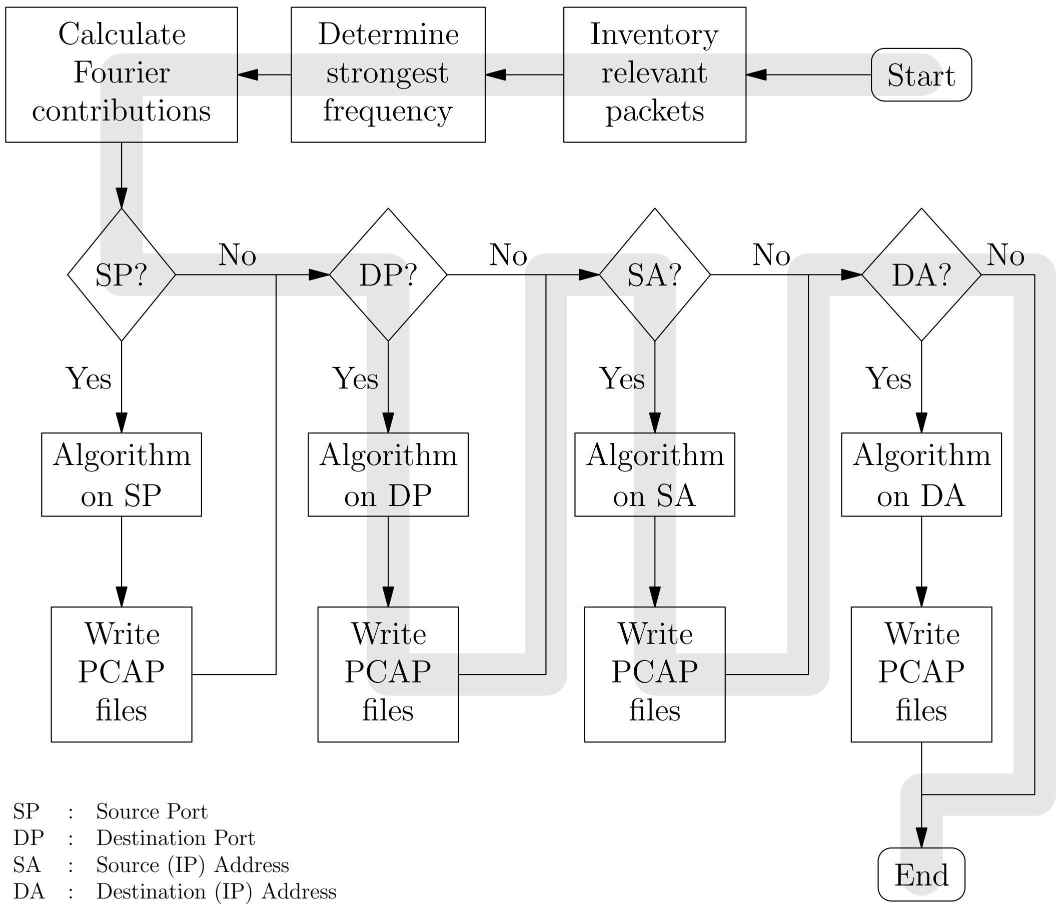

Figure 6.1: Flow diagram for the `pat` program.

- Contribution to the imaginary part of a Fourier transform

To store this information, a new C-`struct` is introduced in the program to represent a packet in the packet trace (see Listing 6.1).
The first field in the struct is a time stamp, measured in seconds with a precision up to a microsecond[4]. From Figure 2.5 it is clear that IP addresses take up 32 bits. To store these values, the struct should make use of a data type that also uses (at least) 32 bits. On the machine that was used for the development of `pat`, unsigned long integers in C++ also take up 32 bits, which is enough for the storage of IP addresses. Both UDP and TCP packets (Figures 2.3 and 2.4) have port number fields that take up 16 bits. For this, the struct should of course make use of a data type that uses (at least) 16 bits. In this case, unsigned short integers take up the same amount of bits and therefore can represent port numbers. Finally, there are fields which store the real and imaginary contributions to a Fourier transform for a certain frequency. This is explained later on.
Since the tool is only designed to handle UDP and TCP traffic, which are protocols that both use the IP protocol as the lower level protocol, the *Type* field of the Ethernet header should be set to the IP protocol. In Listing 6.2, this field is checked for its value. The variable `data` is a pointer to the raw packet data, provided by the `libpcap` library. By casting this data to a C-`struct` that represents an Ethernet header, it is possible to extract the value of the *Type* field in the actual Ethernet header of

[4]When creating a packet trace with the `libpcap` library, this is the maximum precision that is given to the time stamp, but it might not be that accurate.

```
struct Packet {
        double timestamp;
        unsigned long int src_addr;
        unsigned long int dst_addr;
        unsigned short int src_port;
        unsigned short int dst_port;
        double real;
        double imag;
};
```

Listing 6.1: Packet C-`struct` used for the packet inventory.

```
// Define the Ethernet header in the packet
struct ether_header * ether_hdr = (struct ether_header *) data;

// Determine the type of Ethernet packet
unsigned int ethertype = ntohs(ehter_hdr->ether_type);

// Only IP packets are of interest
if (ethertype == ETHERTYPE_IP) {
        ...
}
```

Listing 6.2: Checking whether the packet is an IP packet.

the packet. If the value is equal to the standard protocol type code for IP packets (which is predefined in `ETHERTYPE_IP`), the next step in the inventorying process is taken. If this is not the case, the next packet from the given packet trace is processed.
At the command line, the user specified whether UDP or TCP traffic should be examined. This means that either UDP or TCP packets should end up in the inventory. Listing 6.3 shows how this is accomplished. Now that it is known that the packet is in fact an IP packet, the raw data that is "behind" the Ethernet header can be cast to a struct that defines an IP header. Line number 3 in the listing shows that this done by casting the raw data that starts after `ETHER_HDR_LEN` bits to the IP header struct. Now the source and destination address can easily be extracted from the IP header and stored in the inventory. The `ip_p` field in the IP header struct represents the *Protocol* field in Figure 2.3. The Internet Assigned Numbers Authority keeps a list with possible values for this field. A value of 6 indicates that the upper layer protocol was the Transmission Control Protocol and a value of 17 the User Datagram Protocol. In both cases, the data which is behind both the Ethernet and IP header is cast to the appropriate header struct and the source and destination port numbers can be extracted and stored in the inventory. When a packet meets all the requirements to be stored in the inventory, the time stamp of the packet gets set by using the data provided by the `libpcap` library.

6.2.2 Finding the dominant frequency

After all the (relevant) packets have been inventoried, the frequency spectrum of this data set can be calculated. The first step in this process is converting the data in the inventory to a time series. How this is done is already explained in Section 3.3. Up next is the actual discrete Fourier transform. The `fftw3` library[5] provides functionality to perform Fourier transforms with the Fast Fourier Transform algorithm. In Listing 6.4, a piece of code from the function that is responsible for finding the dominant frequency is shown. The variable `out` is a two dimensional array that contains the complex values of

[5]The `fftw3` library is available at `http://www.fftw.org/`.

```
// Define the IP header in the packet and assign values to the
// address fields in the inventory.
struct ip * ip_hdr = (struct ip *) (data + ETHER_HDR_LEN);

// Set the source and destination IP address in the
// inventory.
packet_inv[packetcount].src_addr = ip_hdr->ip_src.s_addr;
packet_inv[packetcount].dst_addr = ip_hdr->ip_dst.s_addr;

// The packet is a TCP packet
if (ip_hdr->ip_p == 6 && proto == 6) {
        ...
        // Define the TCP header in the packet and assign values
        // to the port number fields in the inventory
        struct tcphdr * tcp_hdr = (struct tcphdr *) (data + ETHER_HDR_LEN +
            (ip_hdr->ip_hl*4);
        packet_inv[packetcount].src_port = tcp_hdr->th_sport;
        packet_inv[packetcount].src_port = tcp_hdr->th_sport;
        packetcount++;
}
// The packet is a UDP packet
else if (ip_hdr->ip_p == 17 && proto == 17) {
        ...
        // Define the UDP header in the packet and assign values
        // to the port number fields in the inventory
        struct udphdr * udp_hdr = (struct udphdr *) (data + ETHER_HDR_LEN +
            (ip_hdr->ip_hl*4);
        packet_inv[packetcount].src_port = udp_hdr->uh_sport;
        packet_inv[packetcount].src_port = udp_hdr->uh_sport;
        packetcount++;
}
```

Listing 6.3: Determining the transport protocol.

the Fourier transform of the time series that was created from the data in the inventory. The variable `out[i][0]` contains the *real* part of the transform at index `i` and `out[i][1]` contains the *imaginary* part. Equation (3.21) showed how to calculate the frequency corresponding to a certain index v in a Fourier transform. Rewriting this equation gives

$$v = f \Delta t N \tag{6.1}$$

where f is the frequency, Δt the sampling period and N the number of samples. This is used for determining the index which represents a frequency of 1 Hz on line 6. Adding 0.5 on this line is for rounding purposes: casting a value to an integer automatically "floors" a value. By adding 0.5 before the casting to an integer, the value will be rounded as one would expect: to the nearest integer. Starting from this index, the rest of the spectrum (up to the Nyquist frequency) is searched for a maximum. In the end, the variable `peakindex` will contain the value of the index that represents the dominant frequency. In other words, if the Fourier transform is defined as $X[v]$, the following should hold for the calculated "peak index" p:

$$p = \underset{v \in [v_s, v_e]}{\arg\max} |X[v]| = \underset{v \in [v_s, v_e]}{\arg\max} \sqrt{\Re(X[v])^2 + \Im(X[v])^2}$$

```
// Initialize the peak value to be zero
double peakvalue = 0;

// The search is started at the peak index which is associated with a
// frequency of 1 Hz.
int peakindex = (int)(1.0*FFT_LENGTH*INTERVAL + 0.5);

// The search can be stopped at the index which represents the Nyquist
// frequency, since the rest is an exact copy of the values that were
// seen so far.
for (int i = peakindex; i < FFT_LENGTH/2+1; i++) {
        double abs = sqrt(out[i][0]*out[i][0]+out[i][1]*out[i][1]);
        if (abs > peakvalue) {
                peakvalue = abs;
                peakindex = i;
        }
}

...

return 1.0*peakindex/FFT_LENGTH/INTERVAL;
```

Listing 6.4: Determining the index in the Fourier vector with the highest absolute value.

in which v_s is the index corresponding to a frequency of 1 Hz and v_e the index corresponding to the Nyquist frequency. The choice for the “minimal search frequency” of 1 Hz was made because of the tail running upwards from 1 Hz to the left in most Fourier transforms of network traffic (see Figure 5.1 for example). Not using this minimal frequency would cause some point in that tail to be appointed as the dominant frequency, which hardly ever makes any sense. Subsequently, the corresponding frequency (using Equation (3.21)) is calculated, returned and used in the following block of the flow diagram: calculating the contributions to the discrete Fourier transform per packet. The precision of this calculated frequency depends on both the chosen sampling period Δt and the number of samples N used for the transform. When considering the equation

$$f = \frac{v}{\Delta t N}$$

again, the frequencies associated with a certain index v lie $\frac{1}{\Delta t N}$ apart from each other. The tool uses a sampling period of 1 millisecond and the number of samples N is 2^{22}. This means that the calculated frequencies are spaced at intervals of

$$\frac{1}{10^{-3} \cdot 2^{22}} \approx 0.00023842\text{Hz}$$

which makes the reported frequencies accurate to the first three decimals.

6.2.3 Calculating the Fourier contributions

In sections 3.1.3 and 3.1.4 two types of Fourier transforms were stated. The continuous transform

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt \tag{6.2}$$

and the discrete transform

$$X[v] = \sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi}{N}vn} \qquad (v = 0, 1, \ldots, N-1) \tag{6.3}$$

The only part of the Fourier transform that is of concern now, is the value of the transform at the frequency which was found in the previous section. So for the further calculations of the program, it is not necessary to calculate the full Fourier transform. Although it seems attractive to use Equation (6.3) for this reason, since it can give the Fourier transform for only one index v by just summing terms instead of integrating (which can be a tough job numerically), it will be shown below that the integral in Equation (6.2) can be reduced to something that can be calculated easily without numerically integrating. The information available now is the (dominant) frequency f from the previous section and the packet inventory (with a time stamp for each packet). In Section 3.3, traffic was modelled as a function of time in the following way:

$$x(t) = \begin{cases} 1 & \text{if } t \text{ is the time stamp of a packet} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

A better notation for Equation (6.4) is to define a delta function for each packet in the traffic. A packet with time stamp s would contribute $\delta(t-s)$ to the "traffic function" $x(t)$, since

$$\delta(t-s) = \begin{cases} 1 & \text{if } t = s \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

and this is exactly what is meant in Equation (6.4). Summing these delta functions for all packets gives a good definition for the (continuous) traffic function $x(t)$:

$$x(t) = \sum_{n=0}^{N-1} \delta(t - s(n)) \tag{6.6}$$

where $s(n)$ is the time stamp s belonging to the packet in the inventory with index n. Substituting Equation (6.6) in Equation (6.2) yields

$$\begin{aligned} X(\omega) &= \int_{-\infty}^{\infty} \sum_{n=0}^{N-1} \delta(t - s(n)) e^{-i\omega t} dt \\ &= \sum_{n=0}^{N-1} e^{-i\omega s(n)} \\ &= \sum_{n=0}^{N-1} \cos(\omega s(n)) - i \sin(\omega s(n)) \end{aligned} \tag{6.7}$$

This shows that finding the Fourier transform for one point ω only needs ω itself and the time stamps of all the packets. Using the relation between the angular frequency ω and the frequency f in hertz ($\omega = 2\pi f$), Equation (6.7) can be written in a form that gives the Fourier transform as a function of the frequency f, denoted by $\tilde{X}(f)$:

$$\tilde{X}(f) = \sum_{n=0}^{N-1} \cos(2\pi f s(n)) - i \sin(2\pi f s(n)) \tag{6.8}$$

The big advantage of using the continuous transform, which resulted in Equation (6.8) over using the discrete transform is that the time stamp of the packet can be used directly. No discretisation in time has to be done in advance of the calculations. The binary search algorithm described in Section 6.2.4, makes use of this one point Fourier transform. To save calculation time, the real part

$$\cos(2\pi f s(n))$$

and imaginary part

$$\sin(2\pi f s(n))$$

of the Fourier transform per packet are stored in the inventory (see Listing 6.5).

```
for (int i = 0; i < packetcount; i++) {
        packet_inv[i].real = cos(2.0*M_PI*freq*packet_inv[i].timestamp);
        packet_inv[i].imag = sin(2.0*M_PI*freq*packet_inv[i].timestamp);
}
```

Listing 6.5: Assigning the real and imaginary part for a Fourier transform to a packet.

6.2.4 Binary search algorithm

The main goal of the periodicity analysis tool is to ascribe the presence of a certain periodicity in a packet trace to traffic with either the same source or destination port number or the same source or destination IP address. A rather straightforward procedure to do this is calculating Fourier transforms for all possible source ports/addresses and destination ports/addresses and see for which port or address the (absolute) value of the Fourier transform at a certain frequency is the highest. That port or address would then be the most apparent cause of that periodicity. This would work, but searching for a responsible port would take 2^{16} (a port number is a 16 bit number) Fourier transforms and another $2^{16} - 1$ comparison operations. For a responsible address this would be 2^{32} Fourier transforms and $2^{32} - 1$ comparison operations. This is far from efficient and calls for a more intelligent approach. Instead of discarding possible causes (ports/addresses) one by one, it would be convenient if groups of possible causes could be discarded at the same time. In this section, such a method is described.
Using the fact that source ports and IP addresses are sequences of bits, a "binary search algorithm" can be easily implemented. As an example, suppose that in some packet trace, a dominant frequency is found and the question is whether it is traffic with a specific source port that caused this periodic behaviour. A binary search algorithm is then started to find a responsible source port. The main idea of this algorithm is to *reconstruct* the port number that is the most likely cause of the periodic behaviour. In other words, a 16 bit number has to be constructed in some way.
Starting with the most left bit, a choice has to be made between a 0 and a 1, based on a certain condition or calculation. When the first bit is set, the second bit has to be chosen, again based on a certain condition or calculation. This process is continued until a 0 or 1 is assigned to all 16 bits. For each choice made, the range of possible port numbers decreases by half (see Figure 6.2). The algorithm

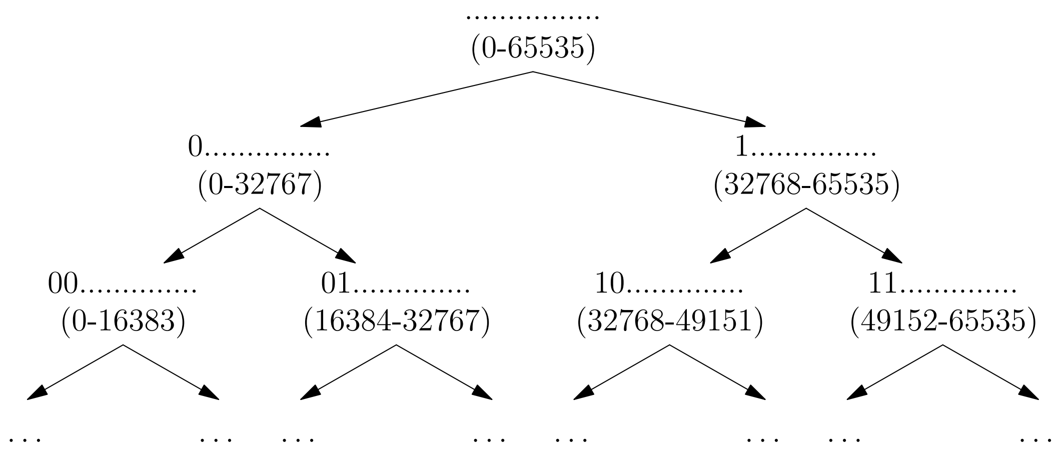

Figure 6.2: Tree with choices when constructing a 16 bit number.

starts off with the complete inventory that was built up in the previous sections. At this point, all source port numbers are still considered as a possible cause for the dominant frequency that was found before. The algorithm then divides the packets in two groups: one group where the source port numbers are

in the lower half (0-32767) of the range and a group where the source port numbers are on the upper half (32768-65535). For both "traffic groups", the absolute value of Equation (6.8) is then calculated. The group for which the highest absolute value is calculated will most likely contain the port number responsible for the periodicity and will be further examined. The other group is no longer relevant. E.g., if the lower side has the highest absolute value, the bit that is considered is set to 0 and a choice for the next bit will be made in the same way: the "winning" group is divided in two traffic groups (source ports 0-16383 and 16384-32767), Fourier transforms are calculated and the side with the highest absolute value wins. After 16 iterations, all bits for the source port number are set, which means that only one possible source port is left. This port number is then assigned to be the port number that is most likely causing the periodic behaviour. Determining a responsible IP address (either source or destination) makes use of the same algorithm, only then 32 steps are done. With the use of this algorithm, the number of Fourier transforms that has to be calculated while determining a responsible port is only 32 (sixteen steps, with each step calculating two Fourier transforms) instead of 2^{16} and for a responsible address this is only 64 instead of 2^{32}. Listing 6.6 on page 71 shows the most important part of the implementation of this algorithm.
The variable `step` is the index of the bit in the number that is constructed and `totalsteps` is the total number of bits for which a choice should be made, i.e. its value is 16 for port numbers and 32 for IP addresses. The variables `lowpoint`, `splitpoint` and `highpoint` keep track of the range that is still "active" in the search. They store the first possible value, the last possible value and the value at which the range is split in half. Their initial values are assigned earlier in the code (not shown) and depend on whether a search is done for a port number or an IP address. At the place of `...`, `packetvalue` gets its value, which is extracted from the C-`struct` which represents the packet which is then considered. This depends on which of the four possible causes mentioned before is examined. The rest of the code should be fairly self explanatory.

6.2.5 Writing PCAP files

After the binary search algorithm finishes its job, two output files are generated. One file will be packet trace which contains traffic from the repository which is probably causing the periodicity. It should be noted that not all of the traffic in this trace file is part of the cause. Suppose that in the example above a certain host is causing a periodicity with its traffic that is originating from port 139. If `pat` indeed finds source port number 139 as the most likely cause of a periodicity, it writes *all* traffic in the inventory with source port 139 to this first output file. So not only the traffic from that certain host is written, but also some "noise traffic" from other hosts. Of course, to filter out this noise, the first output file can be examined for a periodicity caused by some source IP address with `pat`, which would probably find the host that was responsible for the periodicity in the first place.
The other output file contains the traffic that was not deemed responsible for the periodic behaviour with the frequency found earlier. Obviously, this does not mean that this remaining traffic does not contain any periodicities at all. Running the tool over this output file can again find periodicities and their possible causes.

6.3 Program output

Besides the PCAP files that are written during the execution of the program, some other output is generated as well. While the programming is running, it keeps the user informed with on-screen output on what it is doing at that moment and (intermediate) results. A typical on-screen output of `pat` (using the example mentioned earlier) is:

```
$ ./pat udp 6 loc1-20020626-0415
Finding dominant frequency...done

Dominant frequency: 99.996090 Hz
Total number of packets: 183118
```

```
Finding a possible responsible destination port...done
Destination port: 53
Writing PCAP-files...done

Finding a possible responsible source address...done
Source address: 135.216.206.10
Writing PCAP-files...done

Log written to pat.log
```

This output clearly shows that the program follows the diagram in Figure 6.1. One thing is still to be discussed. During the execution, a log file is kept under the name `pat.log`. This file contains detailed information which gives more insight in the decisions that were made during the binary search algorithm. For every step that is made in the binary search algorithm, the following information is written to the log file:

- Bit position of the port number or address that is processed
- Number of packets in the lower half
- Number of packets in the upper half
- Absolute value of the Fourier transform of the lower half
- Absolute value of the Fourier transform of the upper half
- Relative difference between these two absolute values

This makes it possible to track back decisions that were made by the program that could have been somewhat unfortunate. For example, it could happen that the relative difference between two calculated absolute values is fairly small. The program would still choose the half with the highest value, although the port or address that is the dominant cause of the periodicity might just as well be in the other half. The other "noise" traffic present in the inventory disturbs the absolute values of both halves and this disturbance could be in favour of the wrong half. Subsequently, the program will only process this wrong half and never find the most dominant cause of the periodicity. With the help of the log file however, it is possible to see whether such an event occurred. If it did occur, another run of the program (on one the output PCAP file with the traffic that was marked *not* responsible for the periodicity) can still reveal the originally dominant cause that was thrown away at first.

```
// The binary search
while (step < totalsteps) {
        // The totals of real and imaginary contributions
        // are declared and set to 0
        double totalreallow = 0, totalimaglow = 0, totalrealhigh = 0,
            totalimaghigh = 0;

        ...

        for (int i = 0; i < packetcount; i++) {

                // Split the packet to the appropriate side
                if (packetvalue > splitpoint && packetvalue <= highpoint) {
                        totalrealhigh += packet_inv[i].real;
                        totalimaghigh += packet_inv[i].imag;
                }
                else if (packetvalue >= lowpoint && packetvalue <= splitpoint) {
                        totalreallow += packet_inv[i].real;
                        totalimaglow += packet_inv[i].imag;
                }
        }

        // Calculate the absolute values of the 1-point DFT of the low
        // and high side.
        double lowpeak =
            sqrt(totalreallow*totalreallow+totalimaglow*totalimaglow);
        double highpeak =
            sqrt(totalrealhigh*totalrealhigh+totalimaghigh*totalimaghigh);

        // Determine which side wins and adjust the start, end, and
        // splitpoint.
        if (lowpeak > highpeak) {
                highpoint = splitpoint;
                splitpoint = splitpoint - (splitpoint - lowpoint + 1)/2;
        }
        else {
                lowpoint = splitpoint+1;
                splitpoint= splitpoint + (highpoint-splitpoint)/2;
        }
        step++;
}
```

Listing 6.6: Binary search algorithm.

Chapter 7

Conclusions

In Chapters 4 and 5, the last sections already dealt with the results of the analysis of ARP and UDP traffic. In this chapter, these are briefly summarised to answer the questions posed in Section 1.1.

7.1 How can short-term periodicities be detected?

The first approach to identify periodicities, described in Section 4.2, which made use of the information about time intervals between two packets in a trace did give some results. It is not a very safe way to relate specific traffic to a found periodicity: it only makes sense when these time intervals are measured for packets that are in a same stream of traffic. This means that for a proper result with this method, all streams in a trace should be considered separately, which takes too much time.
Using Fourier transforms in combination with a binary search algorithm worked much better and faster: a complete packet trace could be examined at once without having to examine every seperate stream and the algorithm was easier implemented than the earlier mentioned method. The developed tool also was a helpful addition in the research of short-term periodicities. It was shown that the following could be deducted from a plot of a Fourier transform:

- Peaks at some fundamental frequency and harmonic frequencies indicate a strong infinitely repeating signal throughout a packet trace.
- Notches indicate that a finite number of retransmissions is done at a non-fixed interval. The mutual interval between retransmission *is* fixed.
- The presence of both peaks and notches indicate that groups of packets are sent at some fixed interval (see Figure 5.17).

7.2 What kind of traffic is causing short-term periodicities

In the analysis of UDP traffic, the following causes for short-term periodicities were found.

- The streaming of multimedia files: the Real-time Transport (Control) Protocol and some file system protocols (SMB, NFS) indicated this.
- Regular transmission of packets to exchange or synchronise information: the updating of clocks with the Network Time Protocol and the exchange of information during online gaming are good examples for this.
- Retransmission of (unanswered) packets: the DNS and BOOTP traffic discussed in Section 5.3 showed such behaviour.

- Erroneous behaviour of a host: a worm infected machine for example. The machine that was sending out packets from successive ports (see page 47) could be such a machine.

For ARP traffic, only the last two bullets apply as possible causes for short-term periodicities.

7.3 How much of the traffic is periodic?

In Section 5.3 9431 of the originally present 183251 packets remained after the removal of periodic traffic[1], which would indicate a "periodicity percentage" of 95%. This seems to justify the conclusion that 95% of UDP traffic is periodic, but it does not: other traces show completeley different percentages, ranging from 7% to 97%. First of all, these percentages are not that accurate: when a periodicity was found in some protocol or service, *all* traffic from this protocol or service was filtered from the original packet trace and not only the part that was causing the periodicity. DNS traffic, for example, is not periodic by itself, but if some periodicity was found in a part of DNS traffic *all* DNS traffic is filtered out and seen as periodic traffic. This has a huge impact on the accuracy of periodicity percentages. Sometimes the tool reported traffic that was causing periodic behaviour, which was not confirmed by the Fourier transform of that traffic (see Figure 5.9): non-periodic traffic then would also be counted as periodic traffic. Furthermore, the examination of packet traces began by filtering out traffic that was already noticed as periodic traffic in several other traces, to avoid running into results that were already known before encountering new results. The length of the used packet traces (15 minutes) also makes it hard (or even impossible) to say something about the amount of periodic traffic in general, especially because the packet traces were captured at fixed times and did not cover a complete 24-hour cycle of a day. Moreover, these packet traces were captured at four different locations, each having its own special characteristic traffic. A sensible percentage of periodic traffic in Internet traffic in general can only be given by examining *many* packet traces without filtering out the traffic that was found to be periodic in other traces. Also, 24-hour periods or even 7-day periods should be considered to get an accurate view of the average amount of periodic traffic. Finally, more locations should be considered.

7.4 What periods are commonly found?

Many protocols and applications that operate on top of the UDP protocol should not show any *specific* periodic pattern by nature. For the protocols that *do* exhibit periodic behaviour it can be quite easily explained where this behaviour comes from, but the frequencies associated with this behaviour are not fixed most of the time. The only convincing fixed frequency that was discovered during the research was the frequency range that could be associated with DNS: 0.2 Hz to 0.5 Hz, since this is specified by the RFC of this protocol. Other periods that where found are probably due to things like internal clocks of devices (like the 100 Hz peaks that were found regularly) and because the retransmission intervals of packets are mostly determined autonomously by devices (like the synchronisation interval with NTP servers).

[1]The tool can be run over and over till the complete packet trace is empty, which would make every trace 100% periodic. The moment at which the tool should stop finding periodicities – since it no longer makes sense after some runs – can not be clearly defined. This also has an influence on the percentage of traffic that was found to be periodic.

Chapter 8

Recommendations

Some problems that occurred during the research could not be dealt with immediately. This is the case with both the analysis of traffic and the analysis tool that was developed. Therefore, this final chapter summarises what can still be done to achieve more and better results, which in turn should give a better answer to the questions stated in Chapter 1.

8.1 Traffic analysis

The analysis of ARP and UDP traffic already gave some information about short-term periodicities in Internet traffic, but there is still a lot left to research and explain:

- Traffic on top of the connection oriented transport protocol was not examined for periodicities, although this should give some nice results. Since TCP is connection oriented and reliable, the sessions that two hosts set up will probably cause a high amount of periodic traffic on the Internet, due to retransmissions of lost packets and round trip times (i.e. the time between sending a packet and receiving the acknowledgement that it has been received, see Section 2.2.3).
- In some frequency spectra, dominant frequencies (either fundamental or harmonic) were accompanied by sidebands, which hint towards modulation-like behaviour. Perhaps the explanation of these sidebands helps with the identification and explanation of periodicities.
- A packet trace was converted to a time series by counting the number of packets within a certain time interval. Another approach that could be tried out is to count the *size* of these packets within a certain time interval and use that as a basis for a time series.

8.2 Periodicity Analysis Tool

Although the tool helped a lot in determining causes of periodicities, its functionality can be extended and/or improved by considering the following:

- The tool now only identifies the possible cause of *one* periodicity per run. The user still has to examine the output files of one run manually to determine the actual cause of a periodicity. After that, the user can use on of these output files as the new input for another run of the tool. It would be better if the tool could make these decisions on its own, so that its output after one run would be information about *all* possible periodicities in a packet trace.
- Only already stored packet traces can be processed by the tool. Perhaps it is possible to extend this to live traffic. This could be done by using a sliding window algorithm, i.e. by constantly buffering the last several minutes of traffic and determining periodicities in these minutes.

- The binary search method did quite a good job, but the results could get much better if the amount of traffic in both halves that are considered during a step of the algorithm is taken into account. This would partly overcome the problem seen in Section 5.2.2.
- In addition to the previous point, the "white noise problem" is also caused by something else: a dominant frequency found by the tool is now based on only one criterion: the Fourier transform at that frequency should have the highest absolute value of the complete frequency spectrum. However, the Fourier transform of white noise (which has no dominant frequency at all) will also have a maximum absolute value somewhere. So at least one additional criterion should be considered when searching for a dominant frequency. This could be, for example, that the relative difference between this "peak" and the rest of the frequency spectrum should be bigger than some predefined value.
- The influence of noise could also be reduced by implementing a low pass filter, which prevents frequencies higher than the (chosen) Nyquist frequency to show up aliased in the transforms.

Bibliography

[1] R. N. Bracewell. *The Fourier Transform And Its Applications*. McGraw-Hill Companies, Inc., third edition, 2000.

[2] E. O. Brigham. *The Fast Fourier Transform*. Prentice-Hall, Inc., 1974.

[3] C. Cheng, H. Kung, and K. Tan. Use of spectral analysis in defense against dos attacks. In *IEEE GLOBECOM Global Telecommunications Conference*, pages 2143–2148, 2002.

[4] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, April 1965.

[5] D. Donnelly and B. Rust. The Fast Fourier Transform for Experimentalists, Part I: Concepts. *Computing in Science & Engineering*, 7:80–88, March/April 2005.

[6] A. C. Gilbert, Y. Joo, and N. McKeown. Congestion control and periodic behavior. In *IEEE LANMAN Workshop*, 2001.

[7] J. F. James. *A Student's Guide to Fourier Transforms*. Cambridge University Press, second edition, 2002.

[8] E. W. Kamen and B. S. Heck. *Fundamentals of Signals and Systems*. Prentice-Hall, Inc., second edition, 2000.

[9] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring The Internet*. Addison Wesley Longman, Inc., 2001.

[10] H. Kwakernaak and G. Meinsma. Time Series Analysis and System Identification, 2004.

[11] L. L. Peterson and B. S. Davis. *Computer Networks: A Systems Approach*. Morgan Kaufmann, second edition, 2000.

[12] R. W. Ramirez. *The FFT, Fundamentals and Concepts*. Prentice-Hall, Inc., 1985.

[13] C. Runge. *Zeit für Math und Physik*, 48:433, 1903.

[14] M. S. Squillante, D. D. Yao, and L. Zhang. Internet traffic: Periodicity, tail behavior and performance implications. In *System Performance Evaluation: Methodologies and Applications*. CRC Press, 1999.

[15] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Inc., third edition, 1996.

[16] R. van de Meent. M2C Measurement Data Repository, December 2003. M2C Deliverable D1.5.

Index