TCPLOT - A NETWORK MANAGEMENT TOOL TO DETECT AND GRAPHICALLY DISPLAY FAULTY TCP CONVERSATIONS

DJ KOTZÉ

TCPlot - A network management tool to detect and graphically display faulty TCP conversations

DJ Kotzé

THESIS

Submitted in accordance with the requirements for the degree

MAGISTER SCIENTIAE

in the Faculty of Science

Department of Computer Science and
Business Data Processing

of the

University of the Orange Free State

Supervisor : Prof T McDonald

October, 1994

I would like to thank -

Prof T McDonald, my supervisor, for his dedicated support and guidence,

I Hatting, for the use of the menu system developed by him,

Technikon OFS, for the use of their network and equipment whilst developing and testing TCPlot,

and everyone involved in proof reading this thesis.

"Soli Deo Gloria."

Table of contents

CHAPTER 2 22

Chapter 1

Network Management in Perspective

1.1 Introduction

In the past 5 years a tremendous growth rate in local area networks have been experienced. This in turn has brought the whole concept of managing such networks sharply to the foreground. Furthermore, the demands for communications to remote users, resources and applications through local and remote network facilities are growing. The provision of systems and networks and the need to have them operational at all times, yet keeping them cost effective, requires effective and efficient network management.

As early as 1989 a study of American companies showed that a large portion of the real cost of a PC LAN went to Systems Management (Figure 1.1). Prinsloo [Prinsloo, 1991] feels that the cost curve will rise sharply as the time that a system has been implemented increases. It is also true that cutting corners on systems management causes an exponential rise in cost caused by unproductive downtime.

Effective management, however, requires information and the network manager must rely on the availability of tools to monitor traffic and diagnose faults on the network. This thesis will therefore describe the development of a PC based network monitor for an Ethernet network running TCP/IP. The monitor will apart from displaying network traffic on-line, also collect total or filtered packet traces for off-line processing. It will further describe an analyser to plot individual packets graphically. These graphs will enable the network manager to easily interpret traffic patterns and diagnose problematic conversations, without having to work through long packet traces.

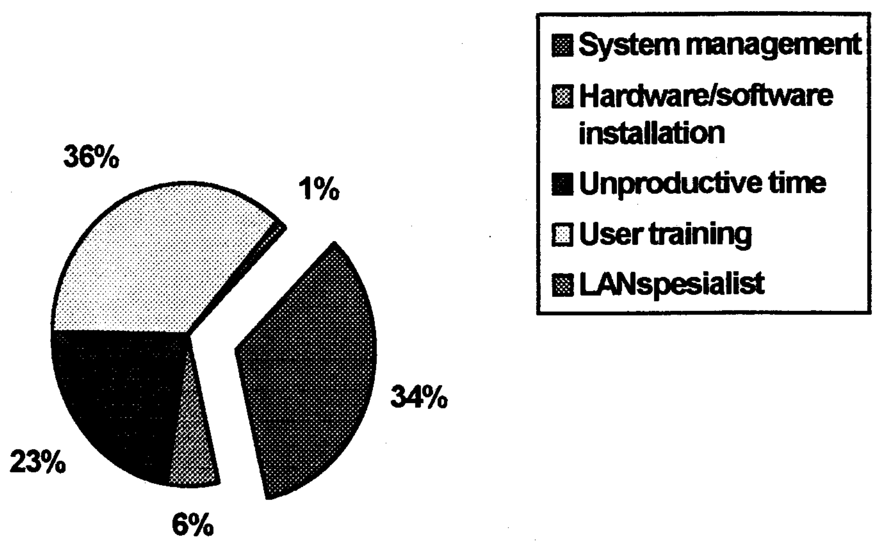


Figure 1.1 The Real Cost of PC LAN's
[Prinlsoo, 1991]

1.2 Management of a Computer Network

Defining network management as a process, like many common activities, poses a problem in that most people are fairly certain of what it is, but would be hard pressed to provide a definition.

It is commonly accepted today that network management involves the planning, organising, monitoring, accounting and controlling of network activities and resources [Black, 1992a]. However, the OSI management structure is focused mainly on monitoring, accounting and controlling of network activities and resources.

Held [Held, 1992 p35] provides the following definition for network management:

> 'Network management is the process of using hardware and software by trained personnel to monitor the status of network components and line facilities, question end-users and carrier personnel, and implement or recommend actions to alleviate outages and/or improve communications performance as well as conduct administrative tasks associated with the operation of the network.'

From the above definition it is clear that network management requires firstly human resources and secondly it requires hardware and software to examine network components. Kauffels [Kauffels, 1992], in his description of the classical responsibility areas of network management, divides the global management of networks into external and internal functions. The external functions are those which cannot be executed in full by the system itself. It must be executed by people such as administrators or technicians, while the management system plays at best a supporting role. The internal functions are executed by the system itself and guarantee its efficiency in terms of its functionality, reactions and reliability.

There are three factors critical to successful network management: methods, tools and human resources. The methods of network management differ widely with the structure of the network, yet it should not be influenced by the size of the network. With the rapid growth of networks, the trusted 'manage by hand' method is no longer appropriate. Fortunately network management tools are becoming more powerful and provide the network manager with an increasing amount of information. This overload of unfiltered information presently requires a highly trained network manager to interpret. As the complexity of networks increases, this will place an unbearable burden on network managers. Future development of network management tools should therefore be directed towards tools that either interpret information using an expert system or at least filter and present information in a way that it can easily be interpreted by network managers.

1.2.1 The OSI Management Framework

The OSI management framework (ISO DIS 7498/4, 1987), although applicable in the OSI environment, is described here to place this thesis in perspective. The OSI environment comprises all tools and services that are used to control and manage connection activities and managed objects. A managed object in terms of OSI is an object with an identifier and corresponding set of management information accessible to the OSI network manager.

The framework defines the structure of the OSI management in terms of three groups [Kauffels, 1992]:

- ***System management*** provides mechanisms for monitoring, controlling and co-ordinating all managed objects within an open system.

- ***Layer management*** provides mechanisms for monitoring, controlling and co-ordinating each of the seven layers of the OSI reference model.

- ***Protocol management*** provides mechanisms for monitoring, controlling and co-ordinating an individual communication transaction.

1.2.1.1 Systems Management Model

The definition of systems management describes three conceptual models [Kauffels, 1992]:

- Functional model.
- Organisational model.
- Information model.

The organisational model describes the distributed character of OSI and how network management tasks may be distributed through the network. The information model on the other hand, concerns itself more with managed objects together with the attributes, operations and reports of such objects.

This thesis, however, will concern itself more with the functional model that introduces the five *Specific Management Functional Areas* (SMFA's) [Black, 1992a, Kauffels, 1992]:

- Configuration management
- Fault management
- Performance management
- Security management
- Accounting management

It is worth noting that although the AT&T management structure [Black, 1992a] varies from the above in that it describes six major categories, these two management structures are the same in essence. This thesis will therefore only refer to the OSI network management standards in its discussion.

1.2.1.1.1 Configuration Management

A computer network is by its very nature a dynamic environment with changes occurring constantly. Resources such as bridges, hubs, nodes and servers may be added or removed from the network or their relationship with each other may vary at any time. For example a packet switch may be removed as a node due to software or hardware failures. If the problem is transient, the switch may be removed logically and the traffic routed around that node. In such a case a part of the network would be reconfigured to handle the problem.

Configuration management must therefore include the ability for managers to generate, observe and modify operational parameters and conditions that govern the mode of operation of components in the system. This includes:

- The existence and names of network components (adding/deleting).
- Relationship between network components.
- Addressing information.
- Operational characteristics of components (e.g. transmission speed).
- Information regarding the usability state of components (e.g. enabled, disabled, busy or active).
- Routing control.

With support from other network management areas, the planning and design of future extension to the network or the partitioning the network with bridges, can be seen as part of configuration management.

1.2.1.1.2 Fault Management

Fault management includes the detecting, diagnosing and isolating of conditions in the network that would prevent normal operation, as well as taking the appropriate steps to correct these faults. The three main areas here being:

Fault detection. Detecting faults on a network requires monitoring the traffic on the network and sounding an alarm if pre-set thresholds are exceeded. Statistics of the number of frames transmitted, number of collisions detected on the system and the number of framing errors detected, must also be kept. From these statistics deductions can be made to enable the network manager to deal with and/or anticipate problems before they cause deterioration of network services.

Diagnosing faults. Further analysis of the traffic on the network may be required to diagnose and isolate the faults or the resources causing them. For example, an analysis of duplicate or retransmitted frames on the network may point to a resource not functioning as expected for various reasons. Another diagnosing method may be the running of a diagnostic program to catch a network component in a particular act.

Finally, the process of **error correction** involves a combination of measures that may include replacing the hardware. This process is supported by configuration management.

Michalski [Michalski, 1991] sees the fault management area as an excellent candidate for the application of an expert system. With a well-defined set of fault management functions, a knowledge base and a set of algorithms to analyse faults, such an expert system could prove invaluable to network managers.

1.2.1.1.3 Performance Management

Performance management can be used to measure several critical characteristics and operations of the network in order to evaluate the efficiency of communication activities. This management area requires a regular provision of collected statistical data in order to analyse performance and can therefore also be used to predict trends in the network. In doing this, the medium-to-long term functionality of the network can be guaranteed by predicting and preventing bottle-necks and thus supporting the design of extension to the network.

The OSI standard defines the following measurements:

- Throughput.
- Workload.
- Propagation delay.
- Wait time (Connection establishment/release delay).
- Response time.
- Quality of service (QOS).

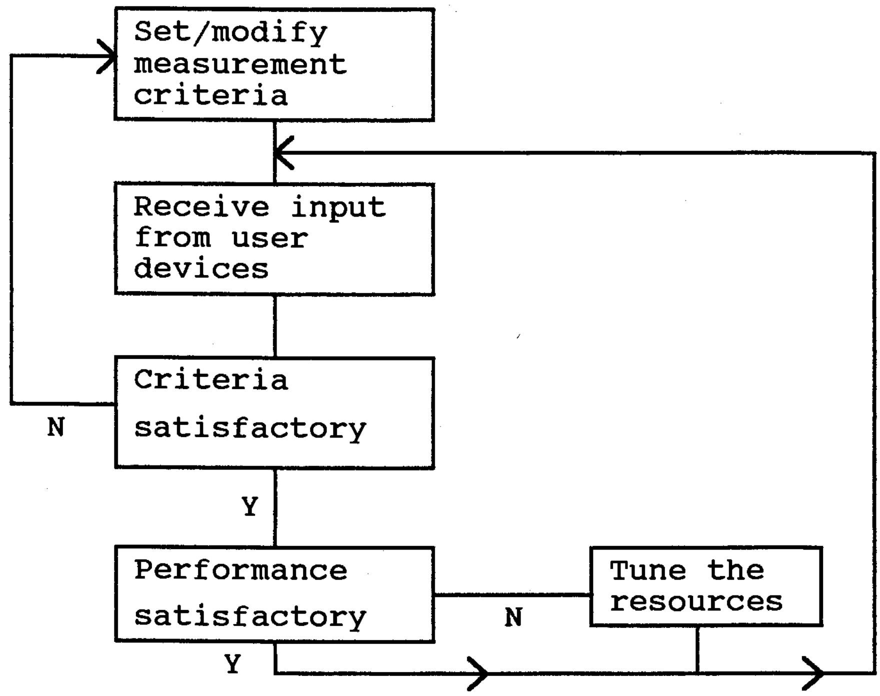


Figure 1.2 The Organisation of Performance Management Functions.
[Black, 1992a, p 217]

Essentially performance management is centred around monitoring, analysing and tuning functions. Figure 1.2 shows the organisation of performance management functions.

Although it is not explicitly mentioned in the above, performance degradation may be due to faulty conditions in the network and tuning must in such cases include support from the fault management area.

1.2.1.1.4 Security Management

Security management varies in importance depending on the specific network environment. It is concerned with protection against random unauthorised access by normal users in a relatively unimportant network, but in a military or bank network, it may mean protection against highly specialised attacks. Although network security is a field on its own, it is the network manager's responsibility to ensure that proper measures are in place and that security breaches are detected.

1.2.1.1.5 Accounting Management

The main purpose of accounting management is to monitor and control information and resources that concern individual users in the environment. This enables network administrators to monitor the use (and if necessary restrict the use) of resources by users and where costs are incurred, to calculate and allocate them appropriately.

1.2.1.2 OSI Layer Management

Layer management concerns all activities needed to monitor all OSI resources belonging to a certain layer (for example, routing in layer 3). Layer management is supported by layer management entities which permit the observation of layer-specific information such as protocol operations, events and parameters for performance analysis. This model, however, requires special protocols to recover from broken connections in the network.

1.2.1.3 OSI Protocol Management

Protocol management concerns those protocol internal mechanisms within one of the seven layers which are used to monitor a specific instance of the communication link. As protocol management is described in the specification of each protocol, it is not described in the OSI management framework.

1.3 Towards standardization

Due to the variety of user needs, LANs were constructed more and more of heterogeneous network products. The TCP/IP protocol suite took care of the communication between the different equipment, but the management of such equipment posed a problem. Most vendors provided management functions for their own products with little regard to other vendors' products. IBM was the first major vendor to define a strategy for integrated management of a heterogenous network in 1986 (Open Network Management Strategy) [Herman, 1990].

The need for a standard management platform soon became clear. From the Internet side the **Simple Network Management Protocol** (***SNMP***), designed specifically for TCP/IP, was proposed, while the OSI/Network Management Forum (OSI/NMF) proposed a **Common Management Information Protocol** (***CMIP***). To accommodate TCP/IP they also endorsed ***CMOT*** (CMIP-over-TCP/IP) [Sekkaki, 1991]. The Internet Activities Board (IAB), the internet's governing body for operational as well as research and development issues, has developed both long- and short-term solutions to help bridge the gap between the current network needs and those for the future [Ben-Artzi, 1990]. In Request for Comment (RFC) 1067 [Case, 1988], the board approved SNMP as a short-term solution for a standard platform, while ISO's CMOT was seen as the long-term solution [Lew, 1989]. Late in 1989 , however, Fisher describes a swing towards SNMP [Fisher, 1989], while Herman [Herman,1990] feels that IMB's SNA based management stategy has lost against SNMP. Yet Perkins [Perkins, 1990] still saw a future for CMIP, expecting local availability by 1992.

Towards the end of 1991 SNMP, which was supposed to be a short-term solution only, has emerged as a powerful standard in its own right while CMOT was losing ground [Jander, 1991]. Black [Black, 1992a], does not even consider CMOT as a contender for network management. SNMP on the other hand is now supported by most vendor's [Greenfield, 1991] and will therefore be the only management protocol discussed here.

1.3.1 Simple Network Management Protocol (SNMP)

The focus of this thesis is not the management of network objects (or information obtainable from them), but rather the analyses of TCP traffic (or conversations) between two elements. Packets are collected directly from Ethernet and SNMP is not used at all. It is, however, for the sake of completeness, necessary to describe SNMP briefly.

The purpose of SNMP is to allow a managing entity, typically a network management station, to control and retrieve information from managed objects. In an Internet context these objects are referred to as *network elements*. The network station communicates with network elements through agents located in the network element. Where a network element is not sophisticated enough to run SNMP software (e.g. modems) or where the network element is not directly reachable by the managing station, proxy agents are used. Proxy agents are therefore network elements which can be reached by the managing station and which will on its behalf, through convergence functions such as protocol conversion, communicate with the unreachable element and perform network functions.

A network element contains a Management Information Base (MIB), that is a database containing numbers and flags reflecting both network and node performance. Each value in the database is contained by an object that can be updated, read or written to across the network [White, 1989].

The management strategy of SNMP is through polling and traps. The managing station polls network elements in its Management Information Base (MIB) at certain times requesting information. The network element on the other hand may, in case of an urgent event, send an interrupt (called a trap) to the managing station. The station will then respond with an appropriate command or control message[Black, 1992a].

To prevent the unauthorised control of network elements, SNMP makes provision for SNMP communities each with a unique Internet name. Access privileges to entities and the MIB in a community are called a community profile and is normally stored as a configuration file within the system. SNMP further provides an authentication scheme to ensure the authenticity of SNMP messages originating in a SNMP community.

1.4 Why Another Network Management Tool?

While information required to do planning and organising is readily available with tools known as network monitors or traffic analysers, problems on the network are not so readily detected and diagnosed. This is mainly due to the multi-layer approach of TCP/IP. High level protocols shield the user from activities occurring at lower levels. This is generally a good idea as users do not want to be burdened with topology, access methods, bandwidth and reliability of the network. Unfortunately in this scenario one also effectively shields the user from problems in the network.

Due to these robust protocols, problems such as bad connections, malfunctioning cards causing excessive collisions and partial failures may manifest themselves not as overt failures but as performance degradations indistinguishable from each other. Although traffic analysers mentioned in section 1.4.2 go a long way in helping the network manager to trace these problems, they are typically only able to produce traces of packets. Reports normally include source and destination addresses, protocol type, sequence numbers, packet length, etc.

Another major factor to be considered when evaluating the performance in a network, is the fact that protocols like TCP (Transport Control Protocol) were developed to work over a variety of networks and provide a variety of services (see Chapter 2). The specifications of such protocols thus leave some of the details such as window size, how quickly segments should be sent and whether to try and batch acknowledgements by dallying, almost entirely to the designer of a particular implementation of the protocol [Shepard, 1991].

The performance of a connection between two different implementations of TCP, is the collective result of the performance of both implementations on their own and together, and can affect data throughput, efficient use of bandwidth and timely recovery from lost packets. Although it is rare today to find two different TCP implementations that are unable to connect and carry data between them, it is not rare to find TCP connections performing poorly. The cause of this can be traced to the assumptions made by the implementor which might not match the network being used or the assumptions made by the implementor of the other TCP implementation. Performance problems as described above, can be pinpointed only by working through packet traces of the connection and studying the behaviour of both sides of the connection to determine the reason for poor performance.

Tools in the **fault** and **performance management areas** can be categorised into two main classes, namely network monitors that are in essence only packet counters [Derfler, 1990] and protocol analysers that concern itself with the contents of packets and the protocols generating them.

1.4.1 Network monitors

Development of tools in the monitor class was started in the early 1980's by the Xerox Alto Research Centre with products such as ***Ether Watch***, that could display packets in octal as they were received, ***PeekPup***, an off-line program that could capture packets in a text file with headers in human-readable form for later analyses and ***Etherload*** that displayed the average load on an Ethernet as a bar graph [Mogul, 1990]. MIT followed with an IBM-PC style package, later to be known as ***LANwatch,*** that had the capabilities of PeekPub and EtherWatch. The commercial product ***LANWatch*** stores the first 300 bytes (including the packet header) of packets in a trace file as default. The source code of parser programs is supplied to enable users to write their own parser programs to analyse TCP/IP packets in depth.

At Stanford development was done on a generation of monitors on the original Sun workstation and the V-System. At least one of these monitors could graphically display a matrix showing communicating hosts [Mogul, 1990].

A monitoring tool described by Mogul [Mogul, 1990] went further in aiding the network manager in the configuration management area by, among other, giving a graphical display of traffic on the network. This was done by depicting all communicating systems as nodes on a graph with the edges weighted according to traffic. The development platform was a Sun workstation and packets were captured passively by using the NIT (Network Interface Tap) of the Sun Microsystems Inc. Operating System.

During the last few years a host of other network monitors have been developed, some with added functions such as datalink fault detection and management or packet traces with decoded header information for protocol debugging [Sudama, 1990].

1.4.2 Protocol Analyzers

Nutcracker developed by Exelan Inc in 1984 can be seen as the first of the protocol analysers [Spanier, 1988]. Packets were timestamped to enable performance calculation to be performed on traces. The fact that it was CP/M based was a major disadvantage and it was quickly followed by a second generation product, ***EX5000***. Although this product was DOS based it used only the EX5000 intelligent network interface. It produced trace files as output and the user could write his own parser programs as the file formats of both statistical and trace files were published [Spanier, 1988].

A traffic analyser described by Protogeros [Protogeros, 1990] could be used in real time or off-line mode, gathering statistics over a longer period. In both these modes packet headers could be stored for later examination. This product could further display statistics per station in a table or in histogram format, give statistics on all network traffic and perform Time Domain Reflectometry tests to determine the location of cable breaks.

Towards the end of 1990, ***Sniffer*** a product from Network General Corp., had become the most popular protocol analyser. Although Sniffer is not a software only solution and must be purchased with a network interface card, it can decode various protocols, including TCP/IP. The protocol decoders translate binary data in packets to English words with the main focus being on decoding the seven layers of communication [Derfler, 1990, Miller, 1992]. Packets can be filtered and captured for later analysis. The way that Sniffer reports and displays this on screen, however, is difficult to understand. In the middle of 1993 upgraded Sniffer software was used in Network General's Internetwork Analyser. Again hardware and interface came with the software, but the upgrade now had the ability to analyse bandwidth usage according to the protocol being used [Jander, 1993].

1.4.3 Graphical Trace Representation

Unfortunately all of the above concentrated primarily on network traffic and considered the trace files with decoded packet headers as sufficient. Even Halsall

[Halsall, 1990], while recognising the need to reduce the level of detail presented to the user, did not manage to address the problem completely in his protocol analyser. The tedious task of manually working through these trace reports, trying to identify duplicate packets and interpret their significance or detecting erratic protocol behaviour, were still left to the network manager.

As an enhancement of the above, Shepard [Shepard, 1991], in his research on the behaviour of the TCP protocol, proposed a graphic representation of the above traces to aid network managers in analysing traces.

The essence of Shepard's work was to analyse packet traces collected on a MicroVAX-III computer running 4.3BSD UNIX by using an adapted version of the network interface tap (NIT) found on Sun systems. In order to analyse the TCP connection, the UNIX tools **grep** and **awk** were used to extract only packets belonging to that connection from the trace. These packets were then plotted as a time-sequence plot where the horizontal axis is indexed by time and the vertical axis is indexed by sequence number. On this same plot, Shepard also plotted the window and acknowledgement numbers as a line.

1.5 Objective of this Thesis

This thesis will concern itself with the fault and performance management areas of the OSI Systems Management Model. The development of a network monitoring tool to supply network managers with the necessary information and statistics to detect and diagnose faults on the network, will be described.

Such a monitor will be able to supply statistical information in real time as well as collect packet traces for off-line analyses.

As an enhancement of previous work, this thesis will describe the development of an analyser capable of analysing packet traces collected with the monitor, *automatically identifying problematic connections* and displaying them graphically. The graphical display is based on a method proposed by Shepard [Shepard, 1991]. He, however, used the method to research the behaviour of the TCP protocol suite, and did not implement it as a network management tool.

1.6 Definition of the Problem

The solution to the problem as described above, consists of the following basic steps:

- Developing a network monitor capable of operating in promiscuous mode to monitor all traffic on the Ethernet.

- Developing a timer for a PC with a high enough resolution to timestamp packets on arrival without taking up too much of the PC's resources.

- Developing an analysing tool to analyse and graphically display selected TCP traffic and thereby enabling network managers to interpret traffic and trace faults easily.

- Developing a strategy and means to identify problematic conversations automatically.

1.7 Development Environment

The management tool described in this thesis was developed on a 386DX 40Mhz DOS machine using an SMC Ultra Ethernet Card. Tests were done on the Technikon OFS campus educational Ethernet network. This network consists of 170 PC workstations (DOS) in the academic building, sharing programs and data on a HP 9837 UNIX fileserver. The fileserver is situated in the computer centre and ***LAN Manager*** with TCP/IP as protocol is used to access DOS-applications on the fileserver.

While the workstations are cabled with 10Base-T, the connection between buildings is fibre optic. These workstations account for traffic peaks with large packets, typically when a class of 60 students simultaneously load an application at the start of a class.

Apart from the workstations, there are a further 36 terminals in the academic building. All terminals are connected to the HP server using two terminal servers with TCP/IP as protocol. This in turn provides a steady stream of small telnet packets.

The development machine, together with seven other DOS workstations, using TCP/IP, were situated on a 10Base2 Ethernet segment in the academic building and therefore close to the terminal servers. As far as the time stamps of packets captured in promiscuous mode are concerned, it might therefore be assumed accurate when the sender is one of the DOS workstations or a terminal server. Packets sent from the file server however, will have travelled some distance and a propagation delay might be expected. This should, however, have no serious effect as the propagation delay for a signal to transverse a cable of a 1000 meters, will only be 5μs [Falaki, 1992].

The whole network is connected to the Internet by a SLIP-link, using PC-Route as software, situated in the computer centre.

1.8 Organization of this thesis

Because the primary focus is on TCP/IP, this protocol suite will be discussed in more detail in Chapter 2. This is followed by a discussion of packet drivers in Chapter 3. The development of a PC-based network monitoring tool that was used to obtain packet traces for analysis as well as the implementation of a high resolution timer on the PC will be discussed in Chapter 4. In Chapter 5 the graphical representation of a TCP/IP connection, as well as the significance of certain occurrences, will be discussed. The tool developed to produce these plots will also be described.

Chapter 6 is in the form of a technical reference where procedures used in the tools described in the previous chapters, are discussed individually. The final chapter, Chapter 7 concerns itself with the testing and evaluation of TCPlot.

Chapter 2

TCP/IP an Overview

2.1 Introduction

The widely used DARPA Internet Protocols are a set of protocols originally designed to allow various network topologies to be interconnected into one large internetwork, the DARPA Internet.

Although this protocol suite includes a selection of protocols (some of which are described below), TCP (Transmission Control Protocol) and IP (Internet Protocol) are the best known and thus the whole family of protocols became known as TCP/IP. This may sometimes lead to confusion, for example, NFS (Network File System) is sometimes described as being TCP/IP based, yet it does not use TCP. It does use IP, but instead of TCP it uses UDP (User Datagram Protocol).

TCP/IP is truly an Open System in that the definition of the protocol suite and many of its implementations are available at little or no cost [Stevens, 1993]. Where the OSI Reference Model divides the network protocol in seven layers, TCP/IP is essentially a four-layer system. The functions of the four layers are:

The **link** layer (datalink) normally includes the device driver and interface card of the computer [Stevens, 1993]. Its function is to handle all the details of physically interfacing the hosts.

The **network** layer handles the movement of packets around the network. Although routing of packet and routing protocols are used in this layer, IP (Internet Protocol) is the protocol that is used to transport TCP packets and is therefore of interest.

The **transport layer** services the application layer above and provides a flow of data between two hosts. The TCP/IP protocol suite has two transport protocols: TCP (Transmission Control Protocol) which provides a reliable flow of data between two hosts. It does this by dividing data in segments of the correct size, acknowledging received packets and checking with the help of timers that dispatched data is received and acknowledged. UDP (User Datagram Protocol) on the other hand, just sends data from one host to another providing no guarantee of receipt.

The **application** layer handles the details of applications such as Telnet or FTP (File Transfer Protocol).

2.1.1 Some services of TCP

- **File Transfer.** The File Transfer Protocol (FTP) allows any networked computer to get files from or send files to another computer.
- **Remote login.** The Network Terminal Protocol (Telnet), allows users to log into another remote computer on the network. Using this protocol the user's computer will send any character typed to the remote computer and display any character received (except control characters) on screen, thus acting as a terminal connected to the remote computer.

- **Electronic mail.** The Simple Mail Transfer Protocol (SMTP) allows mail messages to be sent between users on different computers within a network.

- **Network File System** (NFS). This protocol allows a user to access a file system on another computer with the illusion that the file system is a local device.

- **Terminal servers.** Terminals can be connected to a terminal server instead of directly to a computer. A terminal server is a device (small computer) that runs Telnet and can connect any of these terminals to any computer on the network using Telnet.

Although some of the above are not protocols in the TCP/IP suite, they are implemented using TCP/IP.

2.2 Description of the TCP/IP Protocols

In order to understand the purpose and design of the monitoring tool developed here, it may be necessary to give some attention to the protocols involved and the format of the traffic that they generate.

TCP/IP is a layered set of protocols where one protocol uses the services of another. Typically an application such as mail (SMTP) will hand a message to TCP for delivery. TCP provides for an end-to-end reliable byte stream network connection over a datagram network. Any TCP connection actually provides a pair of byte streams, one in each direction. Large messages are broken down by TCP into smaller segments which are carried between the TCP modules at the end point of the connection by IP (Internet Protocol). The TCP modules ensure a reliable byte stream by arranging for transmission, flow control, sequencing and acknowledgement of bytes. Using the acknowledgements as well as timers to detect lost packets, the TCP modules arrange for retransmissions to recover.

IP on the other hand is a connectionless protocol that uses a Datalink protocol such as Ethernet to transfer datagrams from source to destination.

As mentioned before, TCP splits large messages into manageable segments at the source and reassembles the messages at the destination. To be able to do this, there must be some indication in the TCP segment as to which message a certain segment belongs and where in that message it fits. This is done by placing a header containing the necessary information in front of the TCP data segment at the source and removing it at the destination. As the segment is handed down through the layers, each protocol places its header in front of the Protocol Data Unit (PDU) of the previous layer in this way. A datagram on the physical layer in an Ethernet network would thus have an Ethernet header, an IP header and a TCP header (if TCP was used).

2.3 The TCP Segment Header

Figure 2.1 is a layout of the fields in a TCP header with the number of bits comprising a field in brackets.

<table>
<tr><td colspan="4">Source Port (16)</td><td colspan="5">Destination Port (16)</td></tr>
<tr><td colspan="9">Sequence Number (32)</td></tr>
<tr><td colspan="9">Acknowledgement Number (32)</td></tr>
<tr><td>Data
OffS
(4)</td><td>Reserved
(6)</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td>Window (16)</td></tr>
<tr><td colspan="6">Checksum (16)</td><td colspan="3">Urgent Pointer (16)</td></tr>
<tr><td colspan="8">Options (Variable)</td><td>Padding</td></tr>
<tr><td colspan="9">Data (Variable)</td></tr>
<tr><td colspan="9"></td></tr>
</table>

Figure 2.1 TCP Segment header
[Black (1992b), p 165]

2.3.1 Fields in the Header

2.3.1.1 Port Numbers

An eight bit unit in the header is referred to as an octet and not as a byte. The reason for this is that some systems do not work with 8 bit bytes. The first two octets in the header are used for the source port number and the next two for the destination port number. These port numbers are assigned by TCP to keep track of conversations going on between computers. If three users at host A have TCP connections to a host B, TCP on host A might allocate source port numbers 1201,1202 and 1203 to the three connections. At the destination (host B) the TCP module will allocate its own port numbers which will be put in the destination port number field after opening the connection. In datagrams returned from host B to A the source and destination port numbers will be reversed.

2.3.1.2 Sequence Number

Each TCP segment in a datagram has a sequence number to enable the TCP module at the destination to rebuild the message. TCP does not number the datagrams but the octets. The number placed in this field will be the number of the first byte in the user data field. If there is no data to be sent in a segment, the sequence number is set to the first byte not yet sent. Packets with no data are sent when the sender needs to convey the fact that there is no data, to the other end of the connection. The sequence number is also used during a connection establishment operation. If a connection request is used between two TCP entities, the sequence number specifies the initial send sequence (ISS) to be used for subsequent numbering of user data.

2.3.1.3 Acknowledgement Number

The acknowledgement number is set to a value which acknowledges data previously received. The value in this field is that of the next expected octet from the transmitter. Since this value is set to the next expected octet it provides an inclusive acknowledgement capability in that it acknowledges all octets up and including this number, minus 1.

2.3.1.4 Window

The window field is used as a flow control mechanism. This allows the sender to send more than one datagram without awaiting the acknowledgement of the previous one, yet prevents the sender to send so many datagrams that the receiver can not cope. As the receiver receives more data the amount in the window decreases and if it reaches zero the sender must stop sending until some data has been processed and the window increases again. The value in this field is set to indicate how many octets the receiver is willing to accept. The window is established by adding the value of the window field to the value in the acknowledgement field.

2.3.1.5 Data Offset

This field specifies the number of 32-bit aligned words that comprise the TCP header and is used to determine where the data begins.

2.3.1.6 Flags

URG: Flag indicates that the urgent pointer is significant.

ACK: Flag indicates that the acknowledgement field is significant.

PSH: Flag specifies that the module is to exercise the push function.

RST: Flag indicates that the connection is to be reset.

SYN: Flag indicates that the sequence numbers are to be synchronised; it is used with the connection-establishment segment as a flag to indicate that handshaking operations are to take place.

FIN: Flag indicates that the sender has no more data to send and is comparable with the end-of-transmission (EOT) of other protocols.

2.3.1.7 Checksum

This field contains the result of a checksum operation done on the whole segment. The purpose of the checksum is to enable the receiver to verify that the segment was received without errors.

2.3.1.8 Urgent Pointer

If the URG flag is set this field is used to signify the octet in which urgent data (also known as out-of-band data) follows. This allows the receiver to skip ahead in its processing to a particular octet and can be handy in the handling of certain control characters.

2.3.1.9 Options

This field was conceived to provide for future enhancements.

2.3.1.10 Padding

The padding field is used to ensure that the header is filled to an even multiple of 32 bits.

2.4 The Internet Protocol (IP) Header

The layout of the fields found in the IP header is shown in Figure 2.2 with the number of bits in each field in brackets.

Version (4)	Head Length(4)
Type of Service (8)	
Total Length (16)	
Identifier (16)	
Flag(3)	Fragment Offset (13)
Time to Live (8)	
Protocol (8)	
Header Checksum (16)	
Source Address (32)	
Destination Address (32)	
Options/Padding (Variable)	
Data (Variable)	

Figure 2.2 IP Header
[Black, 1992b, p 100]

2.4.1 Fields in the header of an IP datagram

2.4.1.1 Version

The version field identifies the version of IP in use. This is required as some network nodes may not have the latest version available. The current version of IP is 4.

2.4.1.2 Header Length

This field contains a value that indicates the length of the IP header in 32-bit words. The value of this field is typically set to 5 because the header without Quality Of Service (QOS) options is 20 octets long.

2.4.1.3 Type of Service (TOS)

The first 3 bits of this field are used to indicate the relative importance of this datagram while the next 3 are used to select delay, throughput and reliability parameters.

2.4.1.4 Total Length

This field specifies the total length of the datagram in octets. This value includes the header as well as data length. The length of the data thus is calculated by subtracting the header length from total length.

2.4.1.5 The Identifier, Flags and Fragmentation offset fields

These three fields are discussed together as they are used to control datagram fragmentation and assembly. The need for IP to do fragmentation while TCP has already segmented large messages into manageable segments, needs to be explained first. When TCP establishes a connection, the sizes of the segments are decided upon by enquiring the maximum acceptable size from both end-point systems and choosing the smallest. In the Internet, however, it might happen that the datagram is routed through a network that cannot handle datagrams of that size. IP must then fragment the datagram even further into smaller fragments. As datagram fragments might not follow the same route through this network it might leave the network through a different gateway. The only place where all the fragments are guaranteed to show up is at the destination system where IP must reassemble them to the original datagram. TCP will then reassemble datagrams to the original message.

The identifier field is used to uniquely identify all fragments of a datagram and it is used together with the source address to identify fragments at the destination. In the flag field, bit position 0 is reserved, while bit position 1 is used to signify whether the datagram can be fragmented (0) or not (1). If fragmented, bit position 2 is used to mark the last fragment (set to 0).

The fragment offset field is used to indicate the relative position of the fragment in the original datagram. The value is initialised to 0 and measured in units of eight octets.

2.4.1.6 Time-to-live

To prevent datagrams from staying in the network too long, this field can be given a maximum value. It acts as a hop counter and as datagrams pass through the network each gateway will decrement this value. When the value reaches zero the datagram is discarded.

2.4.1.7 Protocol

The protocol field is used to indicate the protocol above IP. It is quite similar to the type field found in the Ethernet header (discussed later). The numbers assigned by the Internet standards group are shown in Table 2.1.

2.4.1.8 Header Checksum

This checksum is used to detect damage to the IP header although no checks are made of the user data.

2.4.1.9 Source and Destination Addresses

The source and destination addresses are in the Internet address format. This is a 32 bit field in the format: IP Address = Address + Host Address.

The four octets of the address:

10000000 00000011 00001001 00000001

are written as 128.3.9.1 and translates to network 128.3 and host 9.1 (B-Class address).

IP addresses are classified into five classes which will not be discussed in detail here. It is

Decimal	Key word	Protocol
0		Reserved
1	ICMP	Internet Control Message Protocol
2	IGMP	Internet Group Management Protocol
3	GGP	Gateway-to-Gateway Protocol
4		Unassigned
5	ST	Stream
6	TCP	Transmission Control Protocol
7	UCL	UCL
8	EGP	Exterior Gateway Protocol
9	IGP	Interior Gateway Protocol
10	BBN-MON	BBN-RCC Monitoring
11	NVP-II	Network Voice Protocol
12	PUP	PUP
13	ARGUS	ARGUS
14	EMCON	EMCON
15	XNET	Cross Network Debugger
16	CHAOS	CHAOS
17	UDP	User Datagram Protocol
18	MUX	Multiplexing
19	DCN-MEAS	DCN Measurement Subsystem
20	HMP	Host Monitoring Protocol
21	PRM	Packet Radio Monitoring
22	XNS-IDP	XEROX NS IDP
23	TRUNK-1	Trunk-1
24	TRUNK-2	Trunk-2
25	LEAF-1	Leaf-1
26	LEAF-2	Leaf-2
27	RDP	Reliable Data Protocol
28	IRTP	Internet Reliable TP
29	ISO-TP4	ISO Transport Class 4
30	NETBLT	Bulk Data Transfer
32	MERIT-INP	MERIT International Protocol
33	SEP	Sequens Exchange
34-60		Unassigned
61		Any host internal Protocol
62	CFTP	CFTP
63		Any local network
64	SAT-EXPAK	SATNET and Backroom EXPAK
65	MIT-SUBN	MIT Subnet Support
66	RVD	MIT Remote Virtual Disk
67	IPPC	Internet Plur. Packet Core
68		Any distributed file system
69	SATMON	SATNET Monitoring
70		Unassigned
71	IPCV	Packet Core Utility
72-75		Unassigned
76	BRSAT-MON	Backroom SATNET Monitoring
77		Unassigned
78	WB-MON	Wideband Monitoring
79	WB-EXPAK	Wideband EXPAK
80-254		Unassigned
255		Reserved

Table 2.1 Internet Protocol Numbers [Black, 1992b]

sufficient to say that classes A, B and C are used for networks of different sizes. The D class (starts with 1110) is multicast addresses while the address class starting with 11111 is reserved for future use.

2.4.1.10 Options

This field is used for options selected by Upper Layer Protocols (ULP) and includes security, loose or strict source routing, record routing, stream identification and timestamping.

2.4.1.11 Padding and Data

The padding field is used to ensure that the header aligns exactly on a 32-bit word boundary while the data field contains user data. IP stipulates that no datagram (data and header) may be longer than 65535 octets.

2.5 The Ethernet header

As mentioned earlier, IP uses the services of a datalink protocol to transmit datagrams on the physical medium. One such protocol is Ethernet and as the monitor was developed for Ethernet, it is appropriate also to discuss Ethernet here. As with other protocols, Ethernet adds a header in front of the PDU of IP to form an Ethernet packet. It is the structure of this header that will be our main interest in the discussion.

It is worth noting here that Ethernet, as originally specified, and the IEEE 802.3 standard, as implemented for TCP/IP (used in the test environment), differ as will be shown later.

Ethernet is a CSMA/CD (Carrier Sense Multiple Access network with Collision Detection) and like other IEEE 802 standards the datalink layer is split into two sublayers, MAC (Medium Access Control) and LLC (Logical Link Control). LLC is independent of a specific access method while MAC is protocol specific. This approach gives a 802 network a flexible interface with upper layer protocols (ULP's) such as IP or the OSI's connectionless network protocol.

Destination Address (Octets 0-1)
Destination Address (Octets 2-3)
Destination Address (Octets 4-5)
Source Address (Octets 0-1)
Source Address (Octets 2-3)
Source Address (Octets 4-5)
Ether Type
IP Header, TCP Header, Data
Checksum (Octets 0-1)
Checksum (Octets 2-3)

Figure 2.3 Ethernet Header
[Van Niekerk, 1991, p25.]

The fields of the Ethernet header (Figure 2.3) and the SNAP header of 802.3 networks (Figure 2.4) will not be discussed in detail, it is however necessary to note that the source and destination addresses in these headers are not Internet addresses but Ethernet addresses.

Ethernet addresses are six octet addresses permanently configured into each Ethernet interface. To keep these addresses unique, each Ethernet interface is assigned a unique number by its manufacturer from a range that was assigned to him.

Destination Address (Octets 0-1)		
Destination Address (Octets 2-3)		
Destination Address (Octets 4-5)		
Source Address (Octets 0-1)		
Source Address (Octets 2-3)		
Source Address (Octets 4-5)		
Data Length		
DSAP = 170	SSAP = 170	CONTROL = 3
Protocol ID or Org Code		
Ether Type		
IP Header TCP Header Data		
Checksum (Octets 0-1)		
Checksum (Octets 2-3)		

Figure 2.4 The 802.3 Sub Network Access Protocol (SNAP) Header
[Van Niekerk, 1991. p 25]

Another field worth looking at is **Ethertype** which identifies the ULP. The IEEE assigned codes are given in Table 2.2.

Due to the separate evolution of Ethernet and TCP/IP, it became necessary to define an additional RFC (Request For Comment) to provide guidance on the use of IP datagrams over Ethernet. The approach recommended in RFC 1042 (Standard for the transmission of IP datagrams over IEEE 802 network) [Postel, 1988] is of interest to us.

In this approach the LLC destination and source service access points (DSAP and SSAP) must be set to 170. The SNAP control field can be used to identify the protocol but is usually set to an organisation code of 0.

Ethernet decimal	**Hex**	**Description**
512	0200	XEROX PUP
513	0201	PUP Address Translation
11536	0600	XEROX NS IPD
2048	0800	DOD Internet Protocol (IP)
2049	0801	X.75 Internet
2050	0802	NBS Internet
2051	0803	ECMA Internet
2052	0804	Chaosnet
2053	0805	X.25 level 3
2054	0806	Address Resolution Protocol (ARP)
2055	0807	XNS Compatibility
4049	1000	Berkeley Trailer
21000	5208	BBN Simnet
24577	6001	DEC MOP Dump/Load
24578	6002	DEC MOP Remote Console
24579	6003	DEC DECnet Phase IV
24580	6004	DEC LAT
24582	6005	DEC
24583	6006	DEC
32773	8005	HP Probe
32784	8010	Exelan
32821	8035	Reverse ARP
32824	8038	DEC LANBridge
32823	8098	Appletalk

Table 2.2 EtherType Assignment
[Black, 1992b. p46]

With this setting the Ethertype field is used to identify the protocol according to the coding convention shown in Table 2.2. (Table 2.3 is used for coding the SAP (e.g. 170) for the SNAP convention).

IEEE binary	Internet Decimal	Description
00000000	0	Null LSAP
01000000	2	Individual LLC Sublayer Management
11000000	3	Group LLC sublayer Management
00100000	4	SNA Path Control
01100000	6	DOD Internet Protocol
01110000	14	Proway-LAN
01110010	78	EIA-RS511
01110001	142	Proway-LAN
01010101	170	Subnetwork Access Protocol (SNAP)
01111111	245	ISO DIS 8473
11111111	255	Global DSAP

Table 2.3 The Link Service Access Point (LSAP) [Black, 1992b. p45]

2.6 Other Protocols

In monitoring the network and analysing the traffic, it is possible to come across packets sent by protocols other than TCP or even IP. Although all these protocols and their headers will not be discussed, it is imperative that the headers of a few that might be encountered, are discussed.

2.6.1 User Datagram Protocol (UDP)

Although both UDP and TCP are protocols used to transfer messages across the network, TCP is a connection orientated protocol that will split large messages into smaller segments and ensure that these segments are received and rebuilt into the original message at the destination. UDP on the other hand is a connectionless protocol with no facility to split messages and providing no guarantee of delivery.

Many applications however just require a short request to be sent to another machine, expecting only a short (one datagram) reply. In this case UDP is used as the complexity of TCP is not required. UDP fits in the network system much like TCP. A UDP header (Figure 2.5) is placed in front of the data (e.g. the request) and the PDU is handed to IP which adds an IP header. UDP's protocol identifying code is placed in the IP header instead of TCP's code. The fact that no guarantee of delivery is given presents no problem, the application can time out and retransmit its request if no answer is received within a reasonable time.

Source Port (16)	Dest Port (16)
Length (16)	Checksum (16)
Data >>>>	

Figure 2.5 UDP Header.

As can be seen from the header, UDP uses port numbers in conjunction with IP addresses to identify a conversation, the same as TCP. As UDP does not need a sequence number, the only other fields are the length field giving the length of the PDU (including the header) and the checksum field.

2.6.2 Internet Control Message Protocol (ICMP)

Another alternative protocol is ICMP which handles error messages and other messages intended for TCP/IP software rather than some other program. Typically messages such as, 'Host unreachable', if you are trying to connect to a system, is handled by ICMP. This protocol is also used to get information about the network and applications. For example, PING uses ICMP's ***Echo*** and ***Echo reply*** messages to verify the presence of hosts on the network.

The header of ICMP is even simpler than that of UDP, as the messages are interpreted by the network software itself, there is no need for source and destination port numbers in the header.

2.6.3 Address Resolution Protocol (ARP)

Although ARP is not a TCP/IP protocol, it plays a significant role in an Ethernet network running TCP/IP. To understand its importance, take another look at the type of addressing that IP uses. It can be seen from the IP header that internet addresses are used as source and destination addresses. The Ethernet header on the other hand contains Ethernet addresses. As there is no way to send a packet to a system without knowing its Ethernet address, there must be some way of translating IP's internet addresses to physical hardware Ethernet addresses. The Address Resolution Protocol (ARP) is used to take care of this translation.

Generally ARP works with mapping tables (ARP Cache). In a local area network this table can be used to resolve an IP address to a physical address. If the required address is not in the ARP cache, the protocol sends a broadcast to all systems. This broadcast is called an ARP request and contains the IP address of the target. The system that recognises the IP address as its own will send an ARP reply containing its physical address to the inquiring system. Upon receipt of this reply the ARP cache is updated and future datagrams for that IP address can be sent to the physical address on Ethernet.

2.6.3.1 Fields in the Header

The fields of the ARP header are shown in Figure 2.6.

Hardware (16)
Protocol (16)
Hardware Address Length (8)
Protocol Address Length (8)
Operation Code (16)
Sender H/W Address (48)
Sender IP Address (32)
Target H/W Address (48)
Target IP Address (32)

Figure 2.6 The ARP Packet for Ethernet
[Van Niekerk, 1991]

Protocol Address Length Specifies the length in octets for the protocol address (e.g. IP) in the packet.

Operation Code Indicates an ARP request (1) or an ARP reply (2).

The various address fields in the header are self explanatory. In a request packet all fields except for the hardware address of the target are used while all fields are used in the reply.

2.7 Port Addresses in Perspective

As previously described, TCP port numbers are used in conjunction with IP addresses to identify a conversation between two systems. Although TCP assigns these numbers randomly, certain port numbers (those between 1 and 255) have special meaning in the network. The reason for this can be found in the fact that a host A connecting to a host B must have some way of informing TCP on host B to what application program it wants to connect to. This is done by having certain programs waiting at specified ports (well-known ports) as specified in *'Assigned Numbers'* (RFC 1010) [Reynolds, 1987] of which a subset is listed here in Table 2.4.

Echo	7
Discard	9
FTP (Data)	20
FTP (Command)	21
Telnet	23
SMTP	25

Table 2.4 Some of the Well Known TCP Ports

For example, if a file must be transmitted from host A to host B, an FTP session must be started on host A specifying host B as target. FTP (File Transfer

Protocol) will now establish a connection with host B using B's IP address and a port number of 21 as destination. As port 21 is the official port number for the FTP server, a connection will now be established between the FTP program running on host A and the FTP server on host B. It is important to note that the port number used as source port by host A, is a random number. As nobody is trying to find it, it is not necessary to use a well-known port number, the server on the other hand must use a well-known port number.

If another user on host A also wants to transmit a file to host B, another connection is opened using a different port number as source but the same destination port (21).

The parameters of the connections might be:

	Source		Destination	
	IP Address	TCP	IP Address	TCP
Con 1	198.54.58.6	1348	198.54.58.15	21
Con 2	198.54.58.6	1369	198.54.58.15	21

As illustrated above the destination parameters are identical. Fortunately TCP can distinguish between conversations if only one of the four parameters differ and thus no confusion will arise.

Although it's not the intention to describe the protocols in detail, it is worth noting that when actually transmitting the file, FTP would open a second connection with port 20, leaving the first open as a command connection.

Chapter 3

Packet Drivers

3.1 Introduction

Ethernet is one of the most common network standards. For this reason many vendors have developed interface cards for Ethernet. Unfortunately each vendor developed a different interface for the application programs to access their interface cards. Most developers of software therefore had to make provision in their software to be able to access a variety of interface cards. Apart from the inconvenience of this, application programs had to be updated when a new interface card appeared on the market.

Another drawback of this approach was that while one application was using an interface card, the interface was dedicated to that one application, with the result that each application had to have its own network interface. A workstation on an Ethernet backbone could not use IPX to access a Novell server as well as TCP/IP. A further problem was that when one application program terminated, the computer had to be reconfigured and rebooted in order to load the correct software before another application, using a different protocol stack, could access the network interface.

The solution to the above problems was to develop a way to offer a standard interface to all application programs. Each network interface card can therefore have a software driver offering a standard interface to the application program on the one side, while addressing the interface card in its own unique way. Such a driver could also allow different application programs to share the same network interface at the datalink level. Using the network media's standard type or service access point fields in packets, packets can be delivered to different applications (IPX packets to Novell and TCP/IP packets to another application).

Although Doepnik [1990] lists a host of other advantages, the biggest advantage of such a driver is probably the fact that applications could thus become interface independent. Software developers could develop their applications to address a standard driver, leaving it to the vendors of interface cards to supply drivers for new cards developed.

A packet driver can be described as a TSR program in memory, ready to react on software interrupts from the application program or hardware interrupts from the interface card. Application programs communicate with packet drivers by causing a software interrupt in the range of 0x60 - 0x80 (hexadecimal notation) after placing function numbers to be called and pointers to data in specific registers. The interface on the other hand communicates with the driver by causing a hardware interrupt. As the driver is specific to the interface card, further communication between interface and driver is unique for each interface/driver pair.

It is clear from the above that a packet driver must be notified via parameters, during load time, as to which hardware and software interrupts it should react to. During initialisation of the interface for a specific application, one of the functions provided by the packet driver must be used to inform the packet driver where to deliver packets for that application.

3.2 Packet Driver Specifications

From the above it is clear that a set of specifications had to be developed to ensure a standard interface to all applications. Development of such a specification was done at FTP Software by John Romkey [Romkey, 1989]. The specifications document describes three levels of packet drivers, the first being the **basic packet** driver, which provides minimal functionality but uses very little of the host resources and is fairly simple to implement. The second level described is the **extended packet driver.** These drivers are a superset of the basic driver, supporting some of the less commonly used functions of the network interface, such as multicast and promiscuous mode. It also allows statistics to be gathered on the use of the interface card. The third level provides drivers supporting the **high-performance** functions used for performance improvement and tuning.

The specifications deal with the identification of network interfaces, the initial detecting and initialisation of packet drivers as well as the programming interface that drivers present to application programs.

3.3 Identifying network interfaces

Network interfaces are identified by a triplet of integers, (class, type, number). The first integer (class) describes the media that the interface supports: DEC/Intel/Xerox, Ethernet, IEEE 802.3 Ethernet, Tokenring, etc.

The second describes the type of the interface. The type describes a specific instance of an interface supporting a class of network medium [Romkey, 1989]. For Ethernet the type might be 3Com 3c503, 3Com 3c505, etc. The last integer is used to distinguish between two interfaces where a computer has more than one interface of the same class. The first interface in a class will have a value of 0 here, while the type 0xFFFF is a wildcard that matches any interface in the class. Class and type constants are managed by FTP Software and the type constants for class 1 (DEC/Intel/Xerox, ''Bluebook'' and Ethernet), are shown in Table 3.1.

3.3 Initiating Driver Operations

As described earlier, applications invoke a packet driver with a software interrupt in the range 0x60 to 0x80. The packet driver is loaded specifying the software interrupt to respond to as a parameter, but the application, however, must determine for itself which software interrupt to use in order to invoke the driver. This is done by scanning through the handlers for interrupt vectors 0x60 to 0x80.

The interrupt in use by the interface card will have the string ''PKT DVDR'' in the first 12 bytes immediately following the entry point. Once the software interrupt has been determined, the functions described later must be used to initialise the interface for a certain type only.

INTERFACE TYPE	CODE
3Com 3C500/3C501	1
3Com 3C505	2
Interlan Ni5010	3
BICC Data Networks 4110	4
BICC Data Networks 4117	5
MICOM-Interlan NP600	6
Ungermann-Bass PC-Nic	8
Univation NC-516	9
TRW PC-2000	10
Interlan Ni5210	11
3Com 3C503	12
3Com 3C523	13
Western Digital WD8003	14
Spider Systems S4	15
Torus Frame Level	16
10NET communications	17
Gateway PC-Bus	18
Gateway AT-Bus	19
Gateway MCA-Bus	20
IMC PCNic	21
IMC PCNic II	22
IMC PCNic 8-bit	23
Tigan Communications	24
Micromatic Research	25
Clarkson Multiplexor	26
D-Link 8-Bit	27
D-Link 16-Bit	28
D-Link PS/2	29
Research Machines 8	30
Research Machines 16	31
Research Machines MCA	32
Radix Microsys. 16-Bit	33
Interlan Ni9210	34
Interlan Ni6510	35
Vestra LANMASTER 16-Bit	36
Vestra LANMASTER 8-Bit	38

Table 3.1 Interface Types of Media Class 1
[Romkey, 1989, pA.1]

3.4 Programming Interface

All functions of the programming interface are accessed via the software interrupt determined as discussed above. On entry, the register AH contains the code (Table 3.2) for the desired function.

FUNCTION (+ only extended) (* only high-performance)		**CONSTANT**
driver_info		1
access_type		2
release_type		3
send_pkt		4
terminate		5
get_address		6
reset_interface		7
get_parameters	*	10
as_send_pkt	*	11
set_rcv_mode	+	20
get_rcv_mode	+	21
set_multicast_list	+	22
get_multicast_list	+	23
get_statistics	+	24
set_address	+	25

Table 3.2 Function call numbers
[Romkey, 1989 pB.1]

Handles referred to in the rest of this chapter, is an arbitrary integer value associated with each MAC-level type. It is assigned by the driver during the call to the function **access_type**. No guarantee is given that this is a unique value and applications using two interfaces might find that the values returned from the two drivers might be identical.

The functions described below show the entry conditions (FTP's specifications of register contents before the software interrupt is called), the normal return and the return after an error has been encountered. As ***TCPlot*** requires the basic and some extended functions, these functions will be described, using the C-style notation as found in the original specification [Romkey, 1989].

3.4.1 Driver_Info

This function is used to get information about the interface. The version is assumed to be an internal hardware driver identifier, while the handle (optional in this case) is obtained with the function **access_type**

Entry:

	AH == 1, AL==255	
int handle	BX	(optional)

Error return:

carry flag set
error code DH
Possible errors:
BAD_HANDLE

Non-error return:

carry flag clear	
version	BX
class	CH
type	DX
number	CL
name	DS:SI
functionality	AL

Where the following type values are possible:

```
1 > Basic functions
2 > Basic and extended functions
5 > Basic and high-performance
6 > Basic, high-performance and extended
```

3.4.2 Access_type

```
Entry:
                                    AH == 2
    int         if_class            AL
    int         if_type             BX
    int         if_number           DL
    char far    *type               DS:SI
    unsigned    typelen             CX
    int   far   (*receiver)         ES:DI

Error return
    carry flag set
    error code                      DH
    possible errors:
                NO_CLASS
                NO_TYPE
                NO_NUMBER
                BAD_TYPE
                NO_SPACE
                TYPE_INUSE

Non-error return:
    carry flag clear
    handle                          AX
```

This function initiates access to packets of a specific type. The variable ***type*** is a pointer to a packet type specification while ***typelen*** is the length in bytes of the type field. ***Receiver*** is a pointer to a procedure or subroutine that must be called if packets of the type as specified arrives. If ***typelen*** is set to 0, it indicates that the caller wants to match all packets.

The call to the receiver has the following structure:

```
(*receiver) (handle, flag, len [,buffer])


        int         handle     BX
        int         flag       AX
        unsigned    len        CX
    if AX == 1,
        char far    *buffer    DS:SI
```

When a packet of the specified type is received, the receiver calls the application's receive procedure twice. In the first call AX is set to 0, thereby requesting a pointer to buffer space from the application to copy the packet to. The application should return a pointer to buffer space in ES:DI (no interrupt must be called). If the application places 0:0 (no buffer space available) in ES:DI, the second call will not be made and the packet is discarded. The value in CX is the length of the packet including the MAC header without the Frame Check Sequence and is used by the application to allocate enough buffer space.

On the second call to the receive procedure of the application, AX is set to 1. This call indicates that the packet has been copied into the specified buffer and the application can now process it.

3.4.3 Release_type

This function releases a handle and thus ends access of packets as requested during the **access_type** call.

```
Entry:
        int   release_type          AH==3
        int   handle                BX

Error return:
        carry flag set
        error code                  DH
        possible errors:
              BAD_HANDLE
Non-error return
        carry flag clear
```

3.4.4 Send_pkt

The function is used to send *length* bytes of data starting at *buffer*. The application must place the packet, complete with all headers, in the buffer.

```
Entry:
        int         send_pkt            AH==4
        char far    *buffer             DS:SI
        unsigned    length              CX

Error return
        carry flag set
        error code                      DH
        Possible errors:
              CANT_SEND

Non-error return
        carry flag clear
```

3.4.5 Terminate

Terminate can be used to terminate the driver associated with a handle and set memory free to DOS.

```
    Entry:
                terminate              AH==5
           int  handle                 BX

    Error return:
           carry flag set
           error code                  DH
    Possible errors:
                BAD_HANDLE
                CANT_TERMINATE

    Non-error return:
           carry flag clear
```

3.4.6 Get_address

```
    Entry:
                     get_address            AH==6
           int       handle                 BX
           char far  *buf                   ES:DI
           int       len                    CX

    Error return:
           carry flag set
           error code                       DH
           possible errors:
                BAD_HANDLE
                NO_SPACE
    Non-error return:
           carry flag clear
           length                           CX
```

Get_address is used to place the current local network address of the interface into ***buf***. The buffer, ***buf***, is ***len*** bytes long and the actual length of the address is returned in CX

3.4.7 Reset _interface

The interface associated with a given handle can be reset to a known state, aborting transmits in progress and reinitialising the receiver side, by using **reset_interface**. The address of the interface is set to the ROM address and the receive mode is returned to the default mode. If more than one application are using the driver, a reset should not be allowed by the driver and a CANT_RESET will be returned.

```
Entry:
                    Reset           AH==7
        int         handle          BX

Error_return:
        carry flag set
        error code                  DH
        Possible errors:
                BAD_HANDLE
                CANT_RESET

Non-error return:
        carry flag clear.
```

3.4.8 Set_rcv_mode

This function is used to set the receive mode of the interface associated with a handle. The following are the possible modes:

Mode 1	>	Turn receiver off
Mode 2	>	Receive only packets sent to this interface
Mode 3	>	Mode 2 plus broadcast packets
Mode 4	>	Mode 3 plus limited multicast packets
Mode 5	>	Mode 3 plus all multicast packets
Mode 6	>	All packets (promiscuous mode)

```
Entry:
                    set_rcv_mode    AH==20
        int         handle          BX
        int         mode            CX

Error_return:
        carry flag set
        error code                  DH
        Possible errors:
             BAD_HANDLE
             BAD_MODE

Non-error return:
        carry flag clear.
```

As all interface cards and drivers do not support all the modes, the application must check for a BAD_MODE error return after calling the function set_rcv_mode to ensure correct operation.

3.4.9 Get_rcv_mode

Get_rcv_mode returns the current receive mode of the interface associated with a handle.

```
Entry:
                  get_rcv_mode    AH==21
     int          handle          BX

Error_return:
     carry flag set
     error code                   DH
     Possible errors:
          BAD_HANDLE

Non-error return:
     carry flag clear.
```

3.4.10 Get_statistics

A call to this function returns a pointer to the statistics structure of the interface associated with the handle.

```
Entry:
          Get_statistics          AH==24
     int  handle                  BX

Error_return:
     carry flag set
     error code                   DH
     Possible errors:
          BAD_HANDLE

Non-error return:
     carry flag clear.
     Char far  *stats             DS:SI
```

The statistics structure of the interface consists of seven 32-bit integers stored in the normal 80xx format with the following layout.

```
     struct statistics {
Unsigned long  Packets_in;     /*Totals all handles*/
Unsigned long  Packets_out;
Unsigned long  Bytes_in;       /*Including MAC header*/
Unsigned long  Bytes_out;
Unsigned long  Errors_in;      /* All handles*/
Unsigned long  Errors_out;
Unsigned long  Packets_lost;/*No buffer,card,etc.*/.};
```

3.5 Using the programming interface

When using the programming interface the programmer must be aware of the fact that many networks and protocol families use a byte ordering that differs from that of the PC [Romkey, 1989]. This means that with ***ethertype*** values passed to the function ***access_type,*** the byte order must be swapped to be passed in the order as required by the network. This is true for all numerical values in the packet and care should be taken to swap bytes in such fields to ensure correct interpretation.

While in the ***receiver*** procedure the programmer must realise that this procedure is executed as an interrupt and no interrupts may therefore be called from this procedure. It must further be noted that most packet drivers will save and restore *some* registers, but as the values in registers may change, the ***receiver*** procedure must save and restore the necessary registers to be on the safe side. Rehmann [Rehmann, 1993], also found that he had to set the correct data segment for a variable before addressing that variable in the ***receiver*** procedure.

Chapter 4

The TCPlot Monitor

4.1 Introduction

Any monitoring or analysis of network traffic by a network monitor, requires a monitoring device capable of:

- Inspecting or capturing all packets on the network.
- Timestamping each packet with the exact time of arrival.

Under normal operational conditions a network interface card will only respond to packets addressed to that interface while other traffic (except for broadcasts) is ignored. Most network interface cards, however, can be programmed to accept all packets on the network, regardless of destination address. When programmed like this, the interface card is said to be operating in promiscuous mode and all packets are then available to application programs to do monitoring and analysis. Timestamping packets on the other hand requires that the monitoring device keeps a clock with a high enough resolution to differentiate between the arrival of packets within microseconds of each other.

This chapter will describe the development of a network monitor with the above capabilities to gather packet traces for later analysis on a routine basis. Although this was not the primary goal, some on-line capabilities and graphic displays were also built in. This was done because visualisation of LAN traffic with the aid of graphics is the most effective way for the brain to capture the data [Likavec in Dallas, 1991]. The same principle applies when the analyser described in the next chapter uses graphics to analyse a TCP conversation.

4.2 Development Issues

4.2.1 Gathering Packets

IEEE 802.3 Ethernet networks operate at 10 MBit/s (1.25 Mbytes/s) and packets range in size between 64 and 1518 bytes, excluding the preamble. If allowance is made for the minimum inter packet gap of 9.6 microseconds, this will account for 14200 minimum sized or 812 maximum sized packets per second at 100% network capacity. This can be halved as Ethernet will saturate at traffic levels of 55% of the capacity [Sudama, 1990].

Because of the high speed, monitors running in a DOS environment are restricted in that DOS is not a multi-user environment and incoming packets can therefore not be written to disk without running the risk of losing packets. The reason for this is simply that during the I/O time to write one packet to disk, several more packets might arrive at the interface card. Although the packets can be buffered at arrival time, the interface card will ignore packets on the network if there are no buffers available and will only accept packets again after a packet has been processed and a buffer was set free. To avoid this, packets of a trace must be kept in memory and only written to disk when available memory has been filled.

Keeping packets in memory poses yet another problem. If the complete packets were to be kept in memory, the memory will be full within a few seconds on a network with a high load. The alternative is to store only the first 64 bytes of each packet as all the relevant information regarding the packet can be found in this 64-byte header. A monitor using this method can unfortunately not replay a complete conversation, as that would require it to have the data in the rest of the packet. On the positive side it poses no security problem, as no data or passwords can be collected from the network with such a monitor.

4.2.2 Timestamps

The time-of-day clock of an IBM-compatible Personal Computer is updated approximately every 55 ms. In a worst case scenario (small packets and maximum load on Ethernet), more than 780 packets can arrive in this period. Even while such a load is not very practical, the timer resolution might cause problems even at moderate loads. A normal load of mixed packets on the network in the test environment provided for 250 packets per second or approximately one packet every four milliseconds. If the time-of-day clock is used to timestamp the packets on arrival, at least ten packets will thus be shown as having arrived at the same time. To utilise a PC as network monitor, alternative timing methods had to be found as will be shown.

4.2.3 On-line Processing

On-line processing must be kept to a minimum to prevent packet loss similar to that described in 4.2.1. This includes the filtering of packets, moving of packet data from card to memory, timestamping packets, refreshing graphs as well as calculations regarding traffic.

4.3 Addressing the Issues

The issues as mentioned above had to be addressed during development of the **TCPlot** monitor. The fact that disk I/O will cause packet loss had to be accepted, but by being selective about the data stored in memory, the memory usage could be optimised and the overheads minimised.

The approaches taken during the development of the **TCPlot** monitor will be discussed here and a description will be given of the functioning of the monitor.

4.3.1 Memory utilization

As TCPlot was developed mainly with the purpose of analysing TCP conversations on the Ethernet, it follows that **TCPlot** only needs to collect traces of TCP packets while all other packets can be ignored. This approach not only reduces the amount of memory used, but saves time, as not all packet arrivals need to be attended to and no unnecessary packets are copied from the interface to memory.

When addressing the interface card directly, as was done with the original development of **TCPlot**, the full benefit of this approach is not realised. The application program must still inspect each packet's type field to determine the protocol used and must therefore accept all packets, even if they are not copied out to memory.

Later development of **TCPlot** improved on this by implementing the use of a packet driver, which made it possible to be selective about the packet type it wants to attend to. When an application initialises a packet driver, it specifies the type of packets that should be handed to it. In **TCPlot** the packet driver is therefore initialised to operate in promiscuous mode, receiving all packets from Ethernet, but to hand only TCP packets to **TCPlot** and discard all other packets.

Overheads and memory usage can also be reduced by reducing the amount of data moved to memory. As all the information required to analyse TCP conversations can be found in the packet headers, **TCPlot** only moves the first 64 bytes of each TCP packet to memory, discarding the rest of the data. This not only reduces the time to move packets from the buffer to memory, but greatly reduces the memory space required for each packet.

4.3.1.1 Implementation of the TCPlot Monitor

During the initialisation of the packet driver by **TCPlot**, the address of the procedure that must be called when a packet arrives, is handed to the packet driver. When a packet of the correct type (in this case TCP) therefore arrives at the interface, the packet driver calls the receiver procedure requesting a buffer to copy the packet to. As there is no way to specify at this point that only the headers are required, this buffer is large enough to accept a maximum size Ethernet packet.

Once the packet is in the buffer, the packet driver notifies **TCPlot** by calling the receiver a second time with a value of 1 in the AX register. **TCPlot** now moves the first 64 bytes of the packet to memory and set the buffer free for the next packet to arrive. Packets are stored in memory as a linked list with the data in an array of 64 bytes.

To ensure that enough memory is available, the amount of available memory is regularly checked and the gathering of packets is suspended if the available memory drops below a preset threshold. When the threshold is reached, or the monitor suspended by the user, the trace will be written to a file on disk. If **TCPlot** is operating in single trace mode, a file name for the packet trace will be requested from the user, while in auto-trace mode **TCPlot** will assign a unique sequentially numbered name to the file.

The auto mode of **TCPlot** was developed to enable the network manager to get a better picture of the network. The program must therefore set the memory free immediately after the trace file has been written out and start with the gathering of packets for a new trace.

The filenames assigned to the traces in auto mode, is such that the network manager, when using the analyser part of **TCPlot**, can determine the sequence of the traces.

4.3.1.2 Filters

During testing it was found that traces taken on a busy network contained very short portions of conversations due to memory filling too quickly.

Although these conversation portions contained enough packets to point out suspect conversations, it was thought necessary to develop a way to filter excess packets out and only collect packets of a specific conversation, once it has been identified as suspect. As all other packets can be ignored, traces of longer duration are made possible by the use of filters.

Apart from the different modes in which the monitor part of **TCPlot** can be started, provision has therefore been made to allow filters to be set from the options menu. These filters allow the user to enter an IP number and optionally the port numbers of a single conversation, that can be used as filter during packet gathering. When these filters are set, the addresses of packets in the buffer will be checked and only those containing the filter IP number (and if set, the filter port numbers) are moved to memory. Packets not containing the selected IP number or IP number pair, will be discarded and not moved to memory. It is worth noting that while the IP numbers of stations are known, the port numbers are (apart from a few well known ports) assigned randomly. The user must therefore first analyse a trace of traffic on the network to determine the port numbers of the conversation of interest, before port number filters can be set. This can be done by selecting **Select Conv** from the **Options** menu. Such a selection will initiate the capture of a short trace to determine the current active conversations on the net. After analyses of this trace a selection list containing the active conversations is presented to the user. Selecting one of the conversations in the list will set the filters to collect only packets belonging to that conversation.

Another, not so specific, way to collect packets belonging to a certain conversation, is to select the two participants in the conversation of interest and use the IP number of the one least likely to be participating in another conversation, as filter. A conversation between a workstation and a fileserver can therefore effectively be filtered by using the workstation's IP number as filter

4.3.2 Timers

As mentioned above, the normal time-of-day clock provided in a PC does not have the resolution to timestamp arriving packets and an alternative solution had to be found. The different options that were investigated will be described here together with comments on their viability.

4.3.2.1 Interrupts of the Interface Card

The Ethernet Interface Card from ISOLAN that was originally used in developing this monitor, can be programmed to cause interrupts not only on receipt and dispatch of packets, but also on certain given time periods [BICC, 1986]. The interrupt enable register of the card's bit setting corresponds to interrupts on values of 5,10,20, 40 and 80 millisecond intervals respectively. Even though a timer with a 5 millisecond timer was considered marginal in accuracy as far as timestamps were concerned, this was the first solution implemented and tested.

The card was programmed to cause interrupts by the card every 5 milliseconds and the cause of interrupts were identified by reading the interrupt status register to determine actions to be taken. If the interrupt was a timer interrupt, a counter that rolled over on 1000 was incremented and a second counter (for seconds) was updated before the interrupt was cleared.

Although this appeared to work fine under low load, packets were stamped with the same time at traffic peaks. It was also felt that the processing of the extra 200 interrupts per second placed an unnecessary load on the resources of the computer and thereby increasing the risk of losing packets. This approach was therefore abandoned for a more accurate method.

4.3.2.2 Timers of the IBM-compatible PC

The IBM PC family of microcomputers incorporates a high-resolution timer (the Intel 8254 Programmable Interval Timer) which provides three independent 16-bit counters (timers) counting down in binary coded decimal (BCD) or binary and each one can be operated in six modes. These counters are normally used to govern short-duration functions such as RAM refresh and speaker-tone generation, and longer-duration functions such as time-of-day determination [Intel, 1989].

The timer input frequency is 1.19318 MHz and any frequencies that are integer quotients (up to 1/65536) of the base frequency can be generated. The minimum frequency is 18.2065 Hz (corresponding to 54.9 milliseconds) and this time period is known as a tick. Each tick consists of 65536 timing periods and Roden refers to these periods as ticklets [Roden, 1992]. To allow longer periods to be timed, the divisor of Timer-0 is set to 0 (which is essentially the same as 65536) and the output is connected to an interrupt line (IRQ0 / INT 8). The PC responds to this interrupt by simulating a 21-bit timer cascaded from the 16-bit hardware timer. This 21-bit timer (BIOS clock counter) is large enough to hold values up to 24 hours. Interrupt 1Ch is also caused by this timer and can be used by applications to execute a routine every 54.9 milliseconds.

Timer-0 is normally set to the minimum frequency as above to allow the PC to spend as little time as possible simulating the additional timer bits of the BIOS timer. Timer-1 is normally used for DMA memory refresh operations and should not be interfered with. It uses a devisor of 18 which results in a frequency of approximately 66267.8 refreshes per second. This is equivalent to a DMA interrupt being generated every 15.086 microseconds. Timer-2 is most often used to output frequencies to the speaker port but is also available for general use [Whyatt, 1987].

The time-of-day functions normally only need resolution up to the second. Some applications however, require a higher resolution to which there are a few approaches. When timing periods smaller than one tick are required, Timer-0 cannot be used but Timer-2 can be set to the maximum frequency and programmed to start and stop like a stopwatch. Theoretically this provides resolution up to one ticklet although in reality the time it takes to access the timer gates and latencies in the control software, degrade the resolution to about three ticklets [Roden, 1992].

For timing of periods longer than one ticklet, Timer-0 can be set to a higher frequency causing the tick count to increase more quickly. Steps must be taken to keep a separate tick count and the original IRQ0 must be simulated at the correct intervals to preserve the system's tick count. This method, however, has the drawback that it increases system overhead and an IRQ0 frequency to support a millisecond resolution will seriously impact the systems performance. As performance degradation could not be allowed in the monitor this method could not be considered as a possibility in solving the timestamping problem.

4.3.2.3 High-Resolution Timers

A technique to combine the normal system tick count with the current state of Timer-0 to supply a timer with a resolution up to 100 microseconds without increasing system overhead was described by Jerry Jongerius [Jongerius, 1991]. An improved approach to high resolution timing as described by Thomas Roden [Roden, 1992] was, however, implemented in the monitor.

Roden's approach which combines Timer-0 readings with the 8259 Programmable Interrupt Controller (PIC) values, allows for a resolution of 16 microseconds. The essence of high-resolution, non-destructive timing is to combine the ticklet portion of the current time with a tick count. This is done by reading Timer-0 to determine the number of ticklets since the last tick interrupt. This value is then used as the low word of the result while the low word of the system tick count is used as the high word of the result.

Normally Timer-0 is loaded with a devisor of 0 (or 65536) and set to mode 3 which generates a square wave. In this mode the timer is initially loaded with

0 and the output starts high. The counter counts down in two's to the value of two, it is then reloaded with 0 and the output set low before it starts the countdown for the second part of the tick. To convert the current timer state to a ticklet count to use as above, the countdown must be translated to an up count while the fact that the count of 0 represents the maximum value must be taken into consideration. Since counting is in two's, the maximum value is halved, thus producing values from 0 to 32767 which is repeated twice in one tick.

To resolve the ambiguity of the repeated count, the value of the timer output is polled to determine whether it is the first or second countdown. If the value is high, it is the first countdown and the 15th bit is set producing a ticklet count of 0 to 65535.

Although the above produces the desired effect, some precautions must be taken to ensure accuracy. The first of these is to disable interrupts while polling the timer to avoid inconsistencies between tick and ticklet count. If the ticklet count was close to maximum before interrupts were disabled, it could overflow before reading Timer-0 and would then be combined with a tick count of one too low (causing an error of 65536 ticklets). To prevent this error from slipping through, the Programmable Interrupt Controller is interrogated just after reading Timer-0 to see if there is an IRQ0 pending. If so, the tick count may be too low, but as it is also possible that the timer was read just before overflow (on 65535), it is not safe to add to the tick count. The best solution is to assume that the timer was about to overflow and combine it with the tick count [Roden, 1992]. The counter can also be read at a null count

4.4 The TCPlot Monitor modes

The different operating modes of the **TCPlot** monitor can be selected from the **Capture Traffic Menu** after the monitor (Capture Traffic) has been selected from the Main Menu (Shown in Chapter 6 as Figure 6.4 and Figure 6.5).

4.4.1 Hash mode

In this mode a hash is displayed with the arrival of each packet. No on-line processing is required and no packets are moved to memory in this mode. This mode was implemented to give an indication of traffic on the network while collecting statistics, it was also found to be the easiest way to do a quick check to determine whether the packet driver and card were configured correctly and that the monitor is functioning. Figure 4.1 shows the monitor in action in this mode.

4.4.2 Packet Display mode

In this mode the on-line processing done is also negligible. Addresses and packet sizes are read from the buffer and displayed on screen with an indication of the direction a packet was sent. This mode is handy when the network is not very busy and the network manager wants to ensure that the conversation he is interested in, is taking place. For this reason the filters set for packet trace collection, will also be effective in this mode. Figure 4.2 shows the display in this mode.

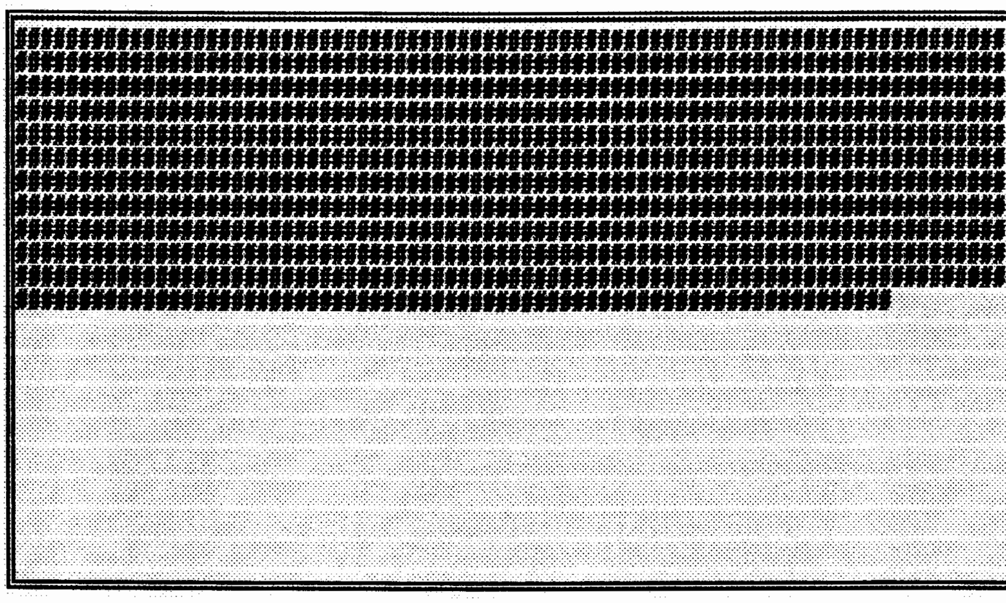


Figure 4.1 The Hash display of the TCPlot monitor.

In this display, the first address shown is the Ethernet address of the sender followed by the Ethernet address of the receiver. This is followed by the IP address of the sender and the IP address of the receiver. The time column, shows the timestamp of the packet as minutes, seconds and microseconds while the last figure is the data length of the IP packet.

4.4.3 Graphics Display mode

Although no packets are moved to memory in this mode, a fair amount of on-line processing is done. This mode shows two bar graphs depicting the number of packets as well as the number of bytes received for the previous ten second period. A new bar is added to each graph approximately every second using the timer interrupt (INT $1C) of the PC. To save processing time in updating the screen display, both graphs are displayed with a vertical baseline. This way, by scrolling the screen one line, only the newest bar needs to be drawn, where if the normal horizontal baseline were used, **TCPlot** would have had to refresh the whole graph.

Due to the quick fluctuation of traffic levels on the Ethernet, provision had to be made to scale the graphs according to the traffic level. This requires some on-line processing and forces TCPlot to redraw the whole graph every time the scale is changed. The maximum value of the largest bar currently on screen is shown at the top of the graph to indicate scale. To give a clearer picture of traffic levels, the LAN activity during the time period of the largest bar, is displayed directly below the maximum bytes received.

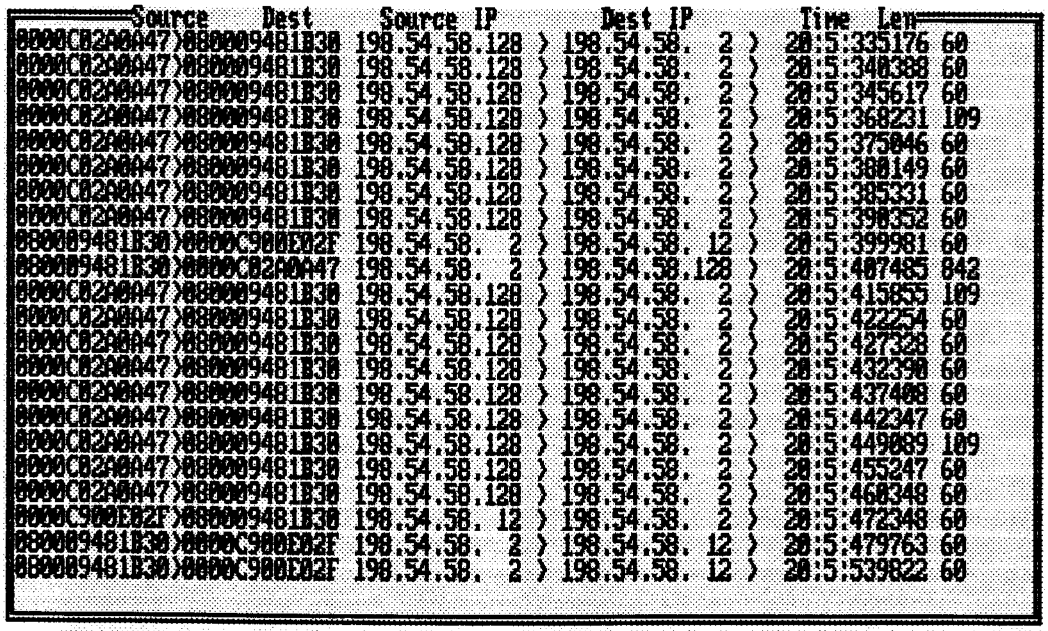


Figure 4.2 The Packet display mode of the TCPlot monitor

The LAN activity, as reported here, is determined by the bytes received per second together with packet overheads as a percentage of the Ethernet maximum (1250000 bytes per second). Overheads per packet can be calculated from the values in Table 4.2. The bytes received as reported by TCPlot, include the MAC header. It is therefore only neccesary to add 160 bytes as overheads to each packet received (preamble of 64 bytes and seperation delay of 96 bytes).

The number of bytes and packets received in the second immediately preceding the drawing of the newest bar, is displayed at the bottom of each graph. The bottom part of the screen in this mode is used to display packets in a similar fashion as packet display mode as shown in Figure 4.3. If the number of packets received during a specific one second period is too small to show on the current scale, an arrow point is displayed instead of the bar. A dot instead of a bar on the other hand, will indicate that no packets were received during that period.

	Minimum packet size	Maximum packet size
Preamble	64 Bytes	64 Bytes
Address fiels and overheads	144 Bytes	144 Bytes
Data frame	46 Bytes	1500 Bytes
Separation delay	96 Bytes	96 Bytes

Table 4.2 Bytes of Ethernet bandwidth used per packet. Adapted from Nemzow [Nemzow, 1988, p228].

With the three preceding modes, no packets were kept in memory and the operation time is therefore not limited. The fact that the timer will roll over in approximately one hour is of little or no relevance here.

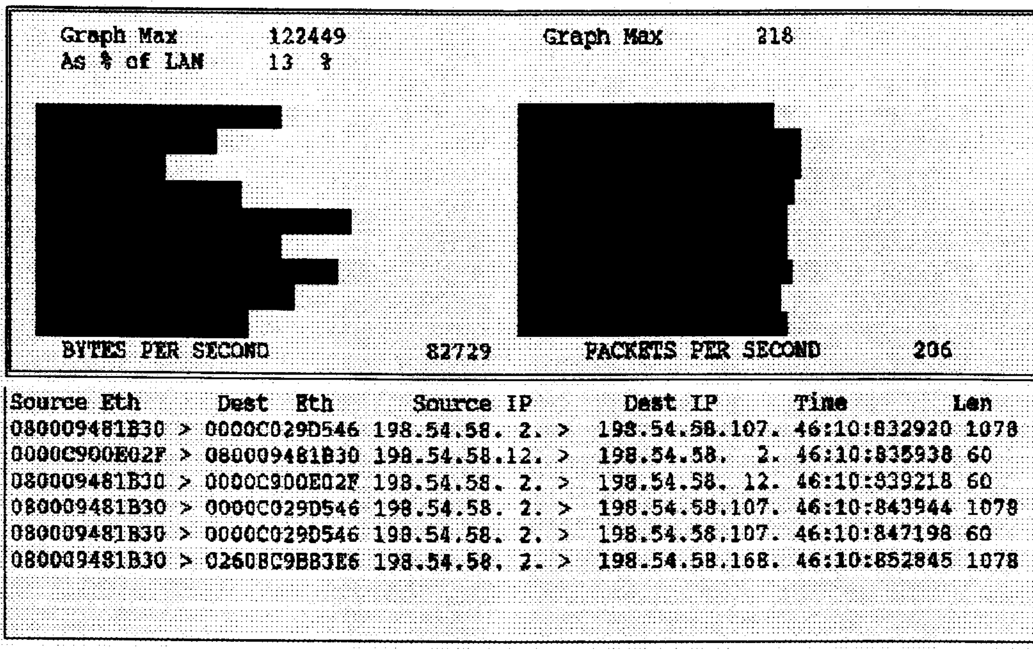


Figure 4.3 The Graphics display mode of the TCPlot monitor.

4.4.4 Store mode

Store mode is the mode for collecting packet traces. Packets must therefore be filtered and moved to memory. Although on-line processing is kept to a minimum in this mode, ensuring maximum time for the receiving and processing of packets, a hash is displayed for each packet accepted as with hash mode.

As memory may fill up in this mode, available memory is checked with each packet saved and in contrast with previous modes, the limit on the duration of this trace is a function of the traffic on the Ethernet and filters set.

4.4.5 Statistics

TCPlot makes use of the packet driver to report statistics. Most of the more sophisticated packet drivers gather statistics on packets arriving at the interface from the time the packet driver is initialised. These statistics include:

- Number of packets received and sent.
- Bytes received and sent.
- Faulty packets received or faulty transmission.
- Packets lost due to buffers not available.

In **TCPlot** statistics can be displayed at any given time by selecting the function from the menu. It should be noted though that statistics reported here are for all packets, not only for the type selected, and that filters as selected have no influence here. Because the statistics are gathered by the packet driver, collection starts as soon as the packet driver has been loaded, irrespective of whether **TCPlot** is running or not.

The monitor, when in store mode, will cause packets to be lost due to the time the disk I/O takes. Care must therefore be taken when using the figures reported. In testing the effectiveness of the monitor, **TCPlot** must therefore be loaded directly after the packet driver, and operated in hash or packet display mode. If, however, effectiveness must be tested while in store mode, the network connection must be removed before ending the trace collection. This will prevent packet loss and make the reported figures usable.

Chapter 5

The TCPlot Analyzer

5.1 Aims of Trace Analysis

As discussed in previous chapters, many factors can cause performance deterioration in the network. Although some of the reasons for such deterioration could be detected using a network monitor such as described in Chapter 4, only a meticulous study of all the packets passed between two hosts during a conversation, can reveal reasons for poor performance that is transparent to the user. Keeping in mind the amount of packets that a network can carry every second, this is no mean task. Such a meticulous study can involve many hours of work, working through packet traces containing many thousands of packets. Trying to interpret their meaning and relation to each other, is even worse. To alleviate the burden on network managers there is a need for a tool to assist them in this type of trace analysis.

Any such analysing tool should therefore be able to:

- Scan the packet trace for symptoms of problems.
- Separate the packets belonging to a certain conversation from the rest of the trace.
- Present a conversation in such a way that the network manager can identify suspicious situations and determine the cause thereof.

5.2 Detecting problematic conversations

To detect a certain condition, that condition must be defined. As almost all reasons for transparent performance deterioration cause duplicate packets to be sent, any excess amount of duplicate packets could be the symptom of a problematic conversation. If the percentage duplicate packets in each conversation are therefore reported, the network manager can identify conversations that need closer scrutiny.

A condition that might cause performance deterioration without initially causing duplicate packets, is the ***Silly Window Syndrome (SWS)*** described by David Clark [Clark, 1982]. The SWS is a degeneration in the throughput which develops over time, during long data transfers. The acknowledgement of a small TCP segment causes another segment of the same size to be sent until some abnormality, or the end of data, breaks the pattern. During large file transfers the SWS can clog the network with many small segments and an equal amount of acknowledgements. Fortunately from the viewpoint of detection, the clogging of the network eventually causes lost segments and therefore massive retransmissions (duplicate packets) [Clark, 1982].

Certain problematic conditions, however, do not cause duplicate packets. One such case would be where the receiver offers a zero window after all packets have been acknowledged. Although this is a perfectly valid action on the part of the receiver when used as flow control, this condition can sometimes arise from a faulty TCP implementation [Shepard, 1991]. Conversations where such conditions exist, will not be detected merely by inspecting conversations for duplicates. If a TCP implementation is faulty, however, the subsequent plotting of any of its conversations with **TCPlot**, will show this fault clearly.

From the above it would appear that detecting problematic conversations in most cases boils down to detecting duplicate packets. We must therefore define a duplicate packet to enable an analyser to detect it.

To expect the analyser to keep a copy of all packets in memory and compare each packet with all previous packets, seems like a waste of resources. An easier way might be to keep track of the sequence number and acknowledgement number of the last packet from each side of the conversation. Any packet in the same direction with the same or lower sequence number or the same sequence number and acknowledging the same packet, may be considered a duplicate for our purposes. It is true that some packets, like control packets to update a window, may satisfy the above criteria without being duplicates. The purpose, however, is not to get an exact duplicate count but rather to pin-point problematic conversations (which incidentally may include conversations where there are excessive window updates without data being sent).

5.2.1 Detecting duplicates

Having established the above criteria to detect duplicate packets and therefore problematic conversations, the implementation in **TCPlot** can now be described.

In **TCPlot** the trace file is processed sequentially and a linked list is constructed containing a record for each conversation found in the trace. Records of the trace (consisting of the full packet header) are read and the fields relevant to TCP are extracted. These values are placed in a conversation record in the linked list of conversations.

A linked list as mentioned above must not only contain information of individual packets, but of total conversations. In **TCPlot** the following were therefore included in this conversation record:

- The source and destination IP addresses.
- The source and destination port numbers.
- A packet counter to determine the total amount of packets in the conversation enabling us to calculate the percentage duplicate packets.
- A counter to keep track of duplicates.
- Previous sequence and acknowledgement numbers from the side of the conversation with the lowest port number.
- Previous sequence and acknowledgement numbers from the side of the conversation with the highest port number.

For each packet record read from file, the linked list of conversations is scanned to determine if the packet belongs to a conversation already in the list. If such a record is found, the packet counter field of that conversation's record is incremented and the packet checked to see whether it is a duplicate packet. This is done by dividing the conversation into two one directional conversations based on their IP numbers. Each packet is tested against the criteria for duplicates as discussed above and if it satisfies the criteria, the duplicate counter field of the conversation's record is incremented. As a result of the fact that only one counter is kept for duplicates, duplicate packets in any direction will cause the counter to be incremented.

If the packet does not belong to a conversation already in the list, a new conversation record is added to the list. Once the end of the trace file is reached, all the conversations can be reported by reporting each record in the conversation list together with number of packets and the percentage duplicates found for each conversation. The number of packets is reported here to show the significance of the duplicate percentage (50 percent of two packets will not carry the same weight as 50 percent of a hundred packets).

5.3 Displaying conversations graphically

When faulty TCP conversations have been detected, the problem must be analysed. In addition to the normal packet trace, **TCPlot** endeavours to assist the network manager with this tedious task by plotting the suspect conversation on a graph. The theory behind such graphical presentation will be discussed before its implementation by **TCPlot**.

5.3.1 Time-sequence plot

In his research on the behaviour of the TCP protocol, Shepard [Shepard, 1991] proposed a way to plot TCP conversations on a time-sequence plot. This type of plot makes any malfunctioning of the TCP protocol easy to spot. Using the same type of time-sequence plot, a conversation between two hosts can be analysed down to packet level to determine the cause of any problems. Considering the number of packets that can be transmitted on Ethernet every second, it is clear that even a trace of a relative short period, can contain several thousands of packets. To analyse a conversation of approximately one hundred packets exchanged during that period, requires the network manager to work through the whole trace. The fact that sequence numbers of TCP packets are based on the octets sent so far, makes it even more confusing. Graphic representation simplifies this by showing only the relevant packet and acknowledgement in such a way that a holistic picture of the conversation can be formed.

The following will illustrate how this method is used to construct a graphical representation from the packet trace in Table 5.1. In this table the **TO** column identifies the receiving host while the **DIR** column can be interpreted as direction of traffic with Host **A** to the left and Host **B** to the right. In a real packet trace, this information must be derived from IP-addresses and port numbers.

Shepard's method [Shepard, 1991] depicts only one directional traffic, a full conversation like that shown in Table 5.1 , must therefore be separated into two conversations, that of Host **A** to Host **B** and that of Host **B** to Host **A**. Only the first of these two will be demonstrated. Table 5.2 shows the conversation with all data not relavant to the conversation half from Host **A** to Host **B** removed.

In Figure 5.1 the packets sent by the sender are plotted by placing a vertical line segment in the time-sequence space starting at the sequence number contained in the sequence number field of the packet and extending upwards for the length of the packet. Shepard ended these vertical line segments with inwards facing arrows, thus making it possible to identify zero length packets as two arrows facing each other (on the graph this looks like a character X).

Timestamp	TO	Dir	Seq	Ack	Win	Length
23:000	HOST A	<	260	98	20	6
23:178	HOST B	>	98	266	50	10
23:282	HOST A	<	266	108	20	20
23:325	HOST B	>	108	286	50	10
23:350	HOST B	>	118	286	50	10
23:400	HOST A	<	286	118	20	20
23:425	HOST B	>	128	316	50	10
23:508	HOST A	<	316	118	20	20
23:611	HOST B	>	118	336	50	10
23:682	HOST A	<	336	138	20	20
23:755	HOST B	>	138	356	50	10
23:828	HOST A	<	356	148	20	50

Table 5.1 A simplified example showing part of a conversation between two TCP hosts.

The acknowledgements are plotted using the values as contained in the acknowledgement field of the packets returned from the receiver (Host B) to the sender (Host A) and the timestamp of the packet. By connecting these points an acknowledgement line is formed (Figure 5.2). The window line (topmost line on the plot) is plotted by computing an end-of-window value (as offered by the receiver) by adding the value in the window field to the value in the acknowledgement field. To make inbound packets, containing the same acknowledgement or window values as previously received packets, visible, a down tick is placed on the acknowledgement line and an up tick on the window line corresponding to the timestamp of the packet.

Timestamp	TO	Dir	Seq	Ack	Win	Length
23:000	HOST A	<	-	98	20	-
23:178	HOST B	>	98	-	-	10
23:282	HOST A	<	-	108	20	-
23:325	HOST B	>	108	-	-	10
23:350	HOST B	>	118	-	-	10
23:400	HOST A	<	-	118	20	-
23:425	HOST B	>	128	-	-	10
23:508	HOST A	<	-	118	20	-
23:611	HOST B	>	118	-	-	10
23:682	HOST A	<	-	138	20	-
23:755	HOST B	>	138	-	-	10
23:828	HOST A	<	-	148	20	-

Table 5.2 TCP-trace as shown in table 5.1 with all data not relevant to the conversation of Host A (sender of data) to Host B (receiver of data), removed.

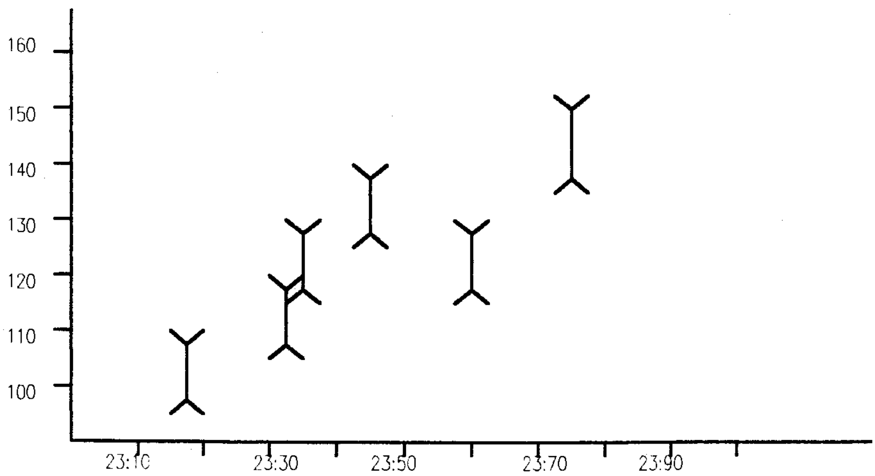


Figure 5.1 Time sequence plot showing six packets from HOST A to HOST B

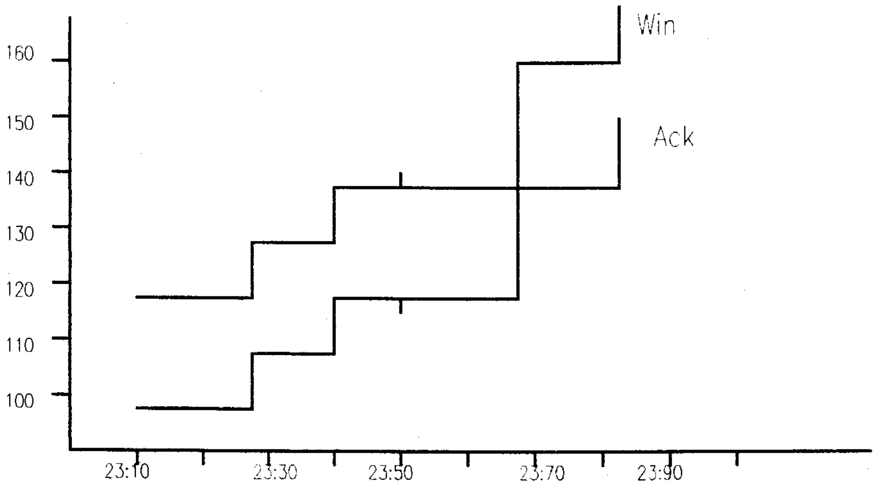


Figure 5.2 Time sequence plot showing the window and acknowledgement lines for the packets in Table 5.2

The space between these two lines can be thought of as the window and whenever these lines touch, the window is effectively closed and the sender must refrain from sending new data before a window is offered again. Under normal circumstances this will happen when receipt of a previously sent packet (and therefore octets contained in it) is acknowledged.

If both the packets as well as the window are plotted together, a complete time-sequence plot (Figure 5.3) is the result. Note that all the important information regarding the packets (Sequence number, acknowledgement number, length, window and arrival time) can readily be extracted from the plot. The plot shows the duplicate packet sent at 23:508 at first glance, while even in this simple example, this is not so readily seen in the trace itself. Of even more importance is that the pattern of re-transmitted or duplicate packets can now be seen clearly. Even more difficult to pick up in the trace will be packets outside the window or a prolonged closed window, yet in a plot as described above these situations will stand out immediately.

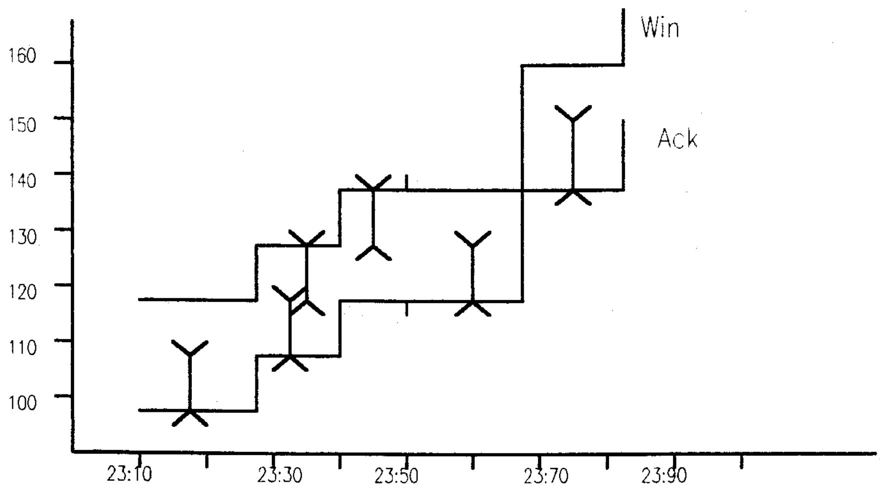


Figure 5.3 Complete time sequence plot showing six packets from HOST A to HOST B within the window

5.3.2 Implementation by TCPlot

With the development of **TCPlot**, the focus was on alleviating the burden on network managers when analysing trace files. For this reason the process of analysing traces and plotting conversations, was simplified to a great extent.

The first step in the analysis of traces by **TCPlot** is to show the traces collected by the monitor part of **TCPlot** to the user in the form of a selection list, enabling him to select the file of interest from the list with the press of a key. The next step is to analyse the selected trace file, isolating the different conversations that are fully or partially represented in the trace file and for each conversation detecting the number of duplicate packets.

This information is then displayed in a selection list, showing the number and percentage of duplicate packets next to each conversation. The user can now, on the strength of these duplicate reports, decide a conversation is suspect and can with the press of a key plot the selected conversation on a time-sequence plot. This time-sequence plot, while using the principles of the method described under 5.3.1, differs from that method in that packet-lines are not terminated with arrow heads, nor is any indication given for zero length packets. The reason for this deviation from Shepard's method [Shepard, 1991] can be found in the scale used. Where packets that arrived close together were plotted on small scale, the arrow heads tended to produce a confusing plot.

5.3.2.1 Analysing the trace file

When a trace file is selected for analysis, the file is scanned for conversations and a linked list containing one record for each conversation is built in memory as discussed under 5.2.1. The logic of this process is given in Figure 6.15 where the procedure handling this process is discussed.

Once this linked list has been created, it is processed and a selection list is created containing an entry for each conversation in the file.

To allow the user to make printouts for record purposes, **TCPlot** makes provision for printing a list of conversations or full traces from a specified trace file.

5.3.2.2 Plotting the conversation

When a conversation is selected from the list, **TCPlot** breaks the display string for the conversation down into the trace file name and the two port numbers of the conversation. These are then used when calling the procedure **PrintGraph** (see 6.3.2.13) to plot the conversation.

Before any plotting can be done, the conversation of interest must be isolated. This is done by reading through the trace file sequentially, using the port numbers received from the selection as filter to process all the packets belonging to the conversation. Although the time-sequence plot will be for traffic in one direction only, all packets, from **A** to **B** as well as **B** to A, are processed at this point. This is because a time-sequence plot, as described above, does not only need information about packets sent by the source host. To be able to plot the acknowledgement and window lines, it also needs information contained in the packets returning from the destination host.

The first implementation of **TCPlot** attempted to plot the time-sequence plot directly from the trace file. It was soon realised, however, that rescaling and scrolling through the plot requires the conversation to be plotted numerous times with different scales. Sequential processing proved too slow for this. The answer lay in the construction of yet another linked list. This one containing information regarding all the packets of the conversation to be plotted. The records of this second linked list contains apart from the information in the packet header, also the timestamp given by **TCPlot** when the packet arrived.

Each record in the list contains the following:

- IP numbers
- Port numbers
- Sequence number
- Acknowledgement number
- Offered window size
- Data length of packet
- Arrival timestamp

As certain calculations must be done with the 32-bit sequence and acknowledgement numbers, the most significant bit of these numbers must be stripped to allow the usage of the LongInt data type of Pascal (failing to do so will provide false negative numbers). The probability of a conversation's sequence number going from 31 to 32 significant bits during the trace is low and it was considered safe to strip bit 32 in this case.

The time-sequence plot is plotted by processing the list in memory. The list is transversed and each record plotted as it is processed. Depending whether the record holds information about a packet from the source or the destination, either both the window and acknowledgement lines will be updated or a packet will be plotted.

The first record in the list serves as a baseline for the plot. If this record contains packet data from maximum sized packets or a mixture of the two, it is clear that one scale cannot be used in all plots. To plot packets with a size difference of more than 1400 bytes on a screen with a resolution of 480, while ensuring that all packets stay visible, is not possible. An even bigger problem is to show a window of 4096 bytes on the same plot as telnet packets in such a way that the plot shows the packets and acknowledgement lines in enough detail to be of value.

While window size can be problematic in some cases, it is clear that a window so large as described above, would hardly pose a problem. For this reason no effort was made during the development of **TCPlot** to show the window line out of position just to make it visible. Plots of telnet conversations will therefore show no window line at normal scale, it will, however show up if the scale is made small enough.

As far as the scaling of plots in general is concerned, the approach taken by **TCPlot** is to try and determine the type of conversation and by using set parameters for conversation types, do some initial scaling. The user can then adjust the scale to display a proper plot or step through the trace by using the arrows and function keys.

Detecting the most likely scale for a conversation is done in two ways. The first test is to determine whether the source port is 23. This would mean a telnet session with known small packets and the scale can be adjusted to show an initial plot of some value. If the source port is not 23, then an average packet size (determined while creating the linked list) is used to estimate the best scale. Unfortunately this approach is not so effective as a conversation may very well contain 200 small packets and only one larger than a 1000 bytes. This may give an average of 500 bytes, which will not be representative of any of the packets in the conversation. The user will therefore be responsible for the final ajustments to the scale as described below.

Provision has been made to allow the user to adjust the X-scale by using the F5 and Shift-F5 keys. Likewise the F6 and Shift-F6 keys are used to change the Y-scale. Whenever one of these keys is pressed with the plot on screen, the screen is cleared. The linked list is processed again and a new plot is drawn on screen using the new scale.

To allow the user to move between the beginning and the end of the trace, the left and right arrows will scroll the plot left or right. This works fine for relative short traces, but as the sequence numbers increase with time (scrolling the plot), the plot tends to disappear off the top of the screen and the scale must be adjusted to see it. Because adjustment of the scale will strip detail from the plot, this is not always the answer. **TCPlot** therefore also provides a function key (F4) that will cause the plot to be drawn of packets later in the trace ignoring those at the beginning. The trace of later packets will now start with a base of the first packet plotted, eliminating the problem as described above. This technique is especially handy in cases where one or two packets were received at the beginning of the trace and the rest towards the end, leaving a long inactive period between the initial and later packets.

The time-sequence plot is discussed in more detail under 7.3.3 (p 147).

5.3.2.4 Selecting conversation direction

As mentioned before, a time-sequence plot shows only the one half of a conversation. In the way the plotting of a conversation was implemented in **TCPlot**, the initial plot will be that of traffic going from the host whose port number is first in the entry of the selection list. The duplicate packets reported in the selection list are, however, from both halves of the conversation.

It might therefore be that when selecting a conversation showing excessive duplicates, the initial plot shows no problems. In such a case the problem is with the other half of the conversation and a time-sequence of that must be plotted. To plot the second half of the conversation, a **Reverse Direction** function has been implemented in **TCPlot.** The F2 function key toggles **TCPlot** to plot the normal or reverse direction conversation.

5.3.2.5 Filters

The filters set from the options menu of **TCPlot** is meant to filter packets when collecting packets for a trace file. In the analyser part of **TCPlot**, filter settings have no relevance when plotting a conversation. The filter setting does however influence the printing of a complete trace file, thus making it possible for the user to print a trace containing only the packets in a certain conversation. This is handy if a comparison is to be made between a trace and the time-sequence plot on screen.

5.3.2.6 Getting help

The standard F1 function key will provide a help screen in TCPlot. This help function is context sensitive to a certain extent. It will display general help when in any part of the program, but when in the graphic screen of a time-sequence plot, a screen explaining the use of the function keys in the plot, will be displayed.

Chapter 6

The TCPlot Program

6.1 Introduction

The purpose of this chapter is twofold. The structure of **TCPlot** will firstly be discussed to give the reader insight into the program. This discussion will be aided by a discussion of the different menus and selections presented by the program.

In the second part of the chapter, the technical detail of **TCPlot** will be discussed in the form of a technical reference. This discussion will show the different components of **TCPlot** and endeavour to explain the functions and interface of each procedure. Where the logic of a procedure requires it, algorithms will be provided

6.2 Structure

The structure of **TCPlot** can be described best with a structure chart. The chart in Figure 6.1 shows how the main **TCPlot** functions interact.

6.2.1 User Interface

When **TCPlot** is started, it tests for the presence of a packet driver. If no packet driver is found in memory the user is notified (Figure 6.2) and the program is aborted.

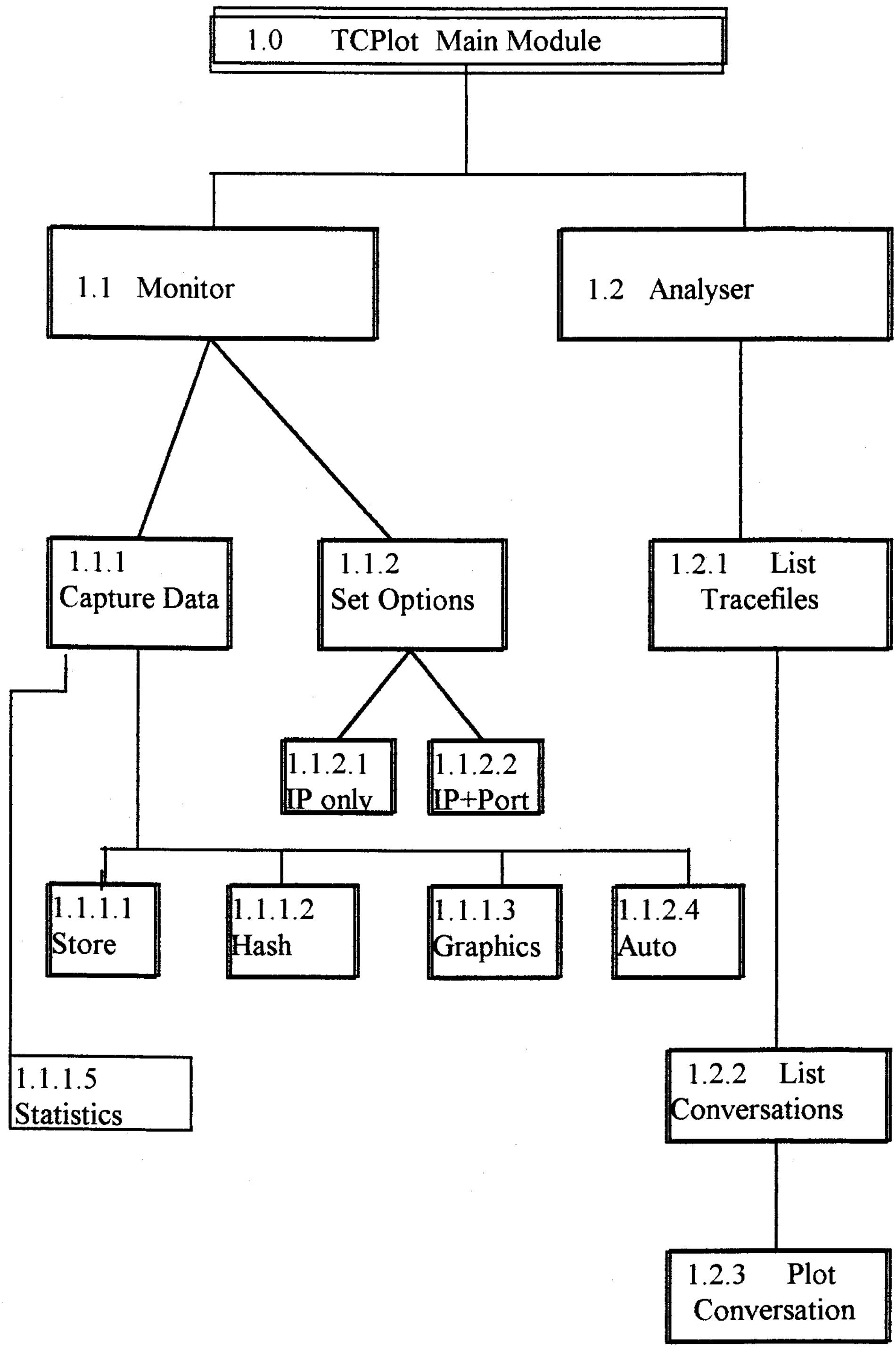


Figure 6.1 The TCPlot program

If, however, a packet driver is detected, **TCPlot** checks the driver and interface card and reports the capabilities thereof to the user (Figure 6.3)

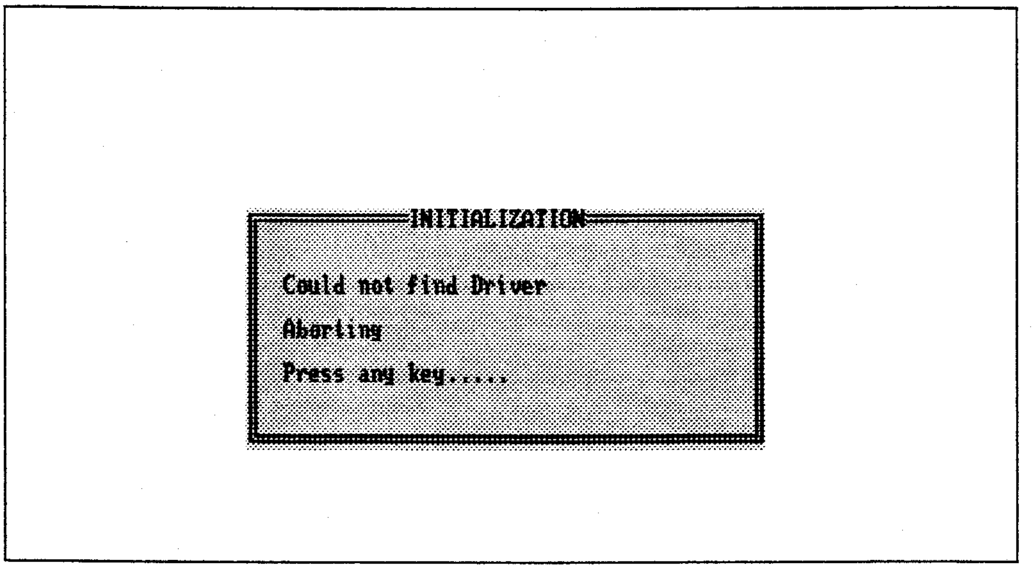


Figure 6.2 Warning to user if no packet driver present

6.2.1.1 The Main menu

Once the user accepts the initialization message, the main menu of **TCPlot** (Figure 6.4) is displayed. From here the user can select the capture traffic (**monitor**), **analyser** or **options** menus, or **TCPlot** can be started in continuous capturing mode by selecting the **auto** mode.

6.2.1.2 The Capture traffic menu

The capture traffic menu (Figure 6.5) contains a selection of all the functions of the monitor part of **TCPlot** as described in Chapter 4. To return to the main menu from any sub-menu, the user must press the escape key.

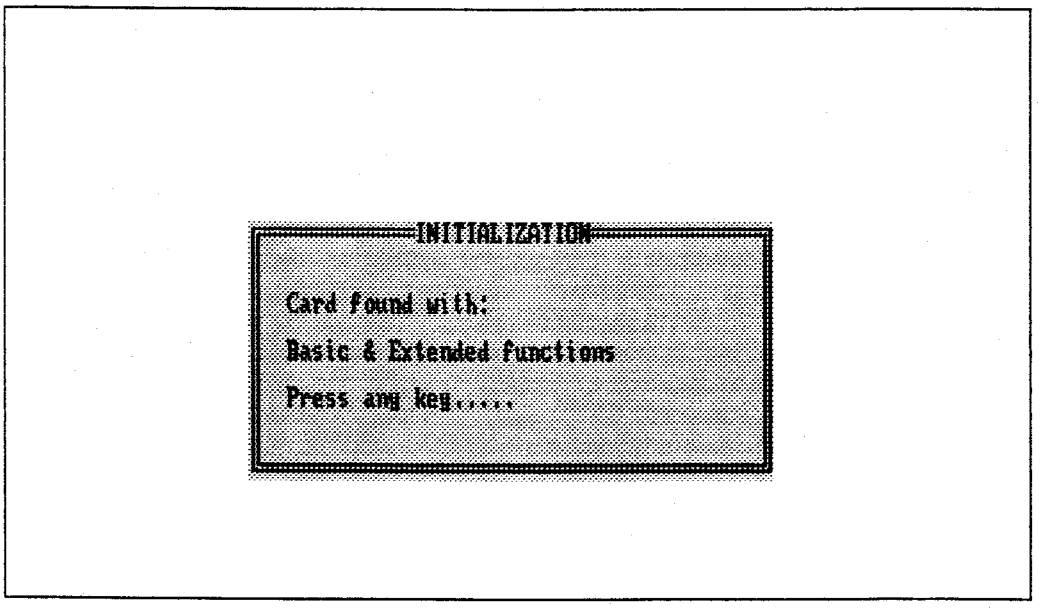


Figure 6.3 The capabilities of the interface card as reported to the user on start-up.

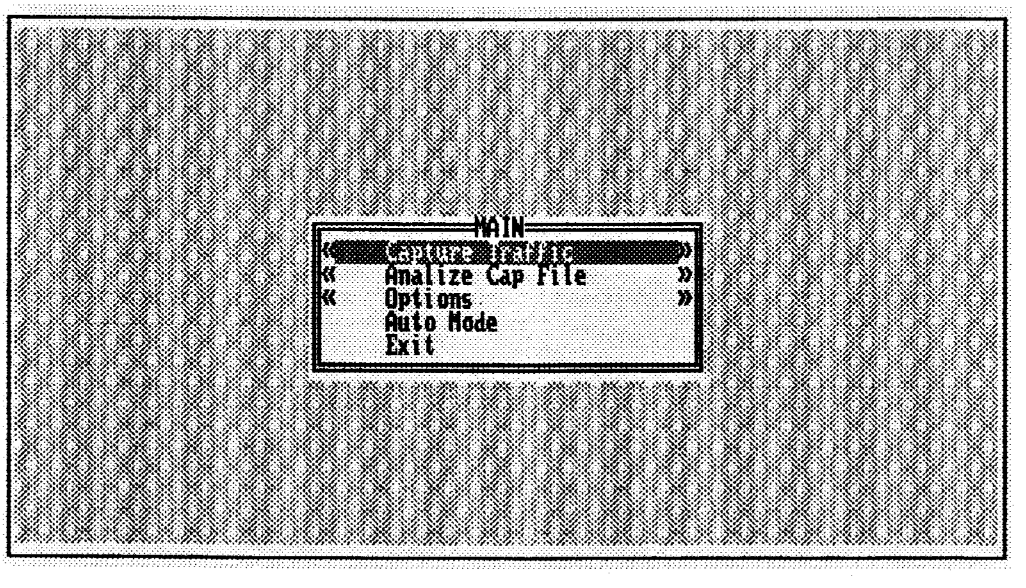


Figure 6.4 The **TCPlot** Main menu

6.2.1.3 The Analyser menu

When the **Analyser menu** (Figure 6.6) is selected from the main menu, the user is presented with three choices. The first, **Print Trace** will display a list of trace files and on selection of a file it will print a list (trace) of all the packets in the selected trace file. This trace contains the source and destination addresses and port numbers as well as sequence and acknowledgement numbers of each packet. As this report is wider than 80 columns, a 132-column printer (or condensed print) must be used when printing a trace file.

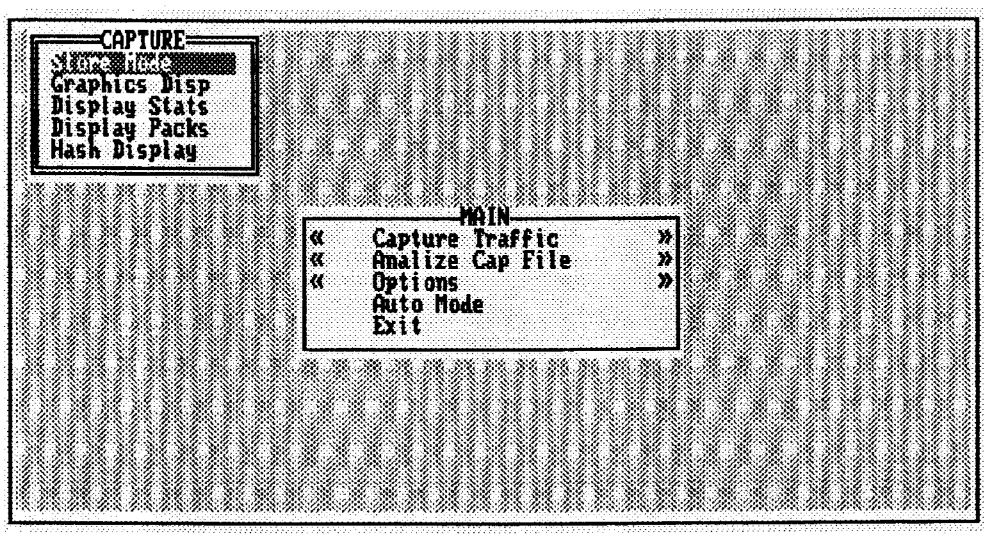


Figure 6.5 The Capture traffic menu

The second choice, **Print C-List,** will again display a list of trace files and on selection of a file it will print a list of all the conversations contained in the selected trace file. This printout will also show the number of packets as well as the percentage of duplicates in each conversation. Examples of both the trace and the conversation list are shown in Chapter 7.

If **Select & Display** is selected, a selection list of all the trace files in the current directory will be displayed (Figure 6.7). Because the trace files in the directory can be more than the lines available on the screen, this selection list will display the first 20 entries and allow the user to scroll to the others. When **enter** is pressed on any of these entries, a selection list of conversations (Figure 6.8) is displayed.

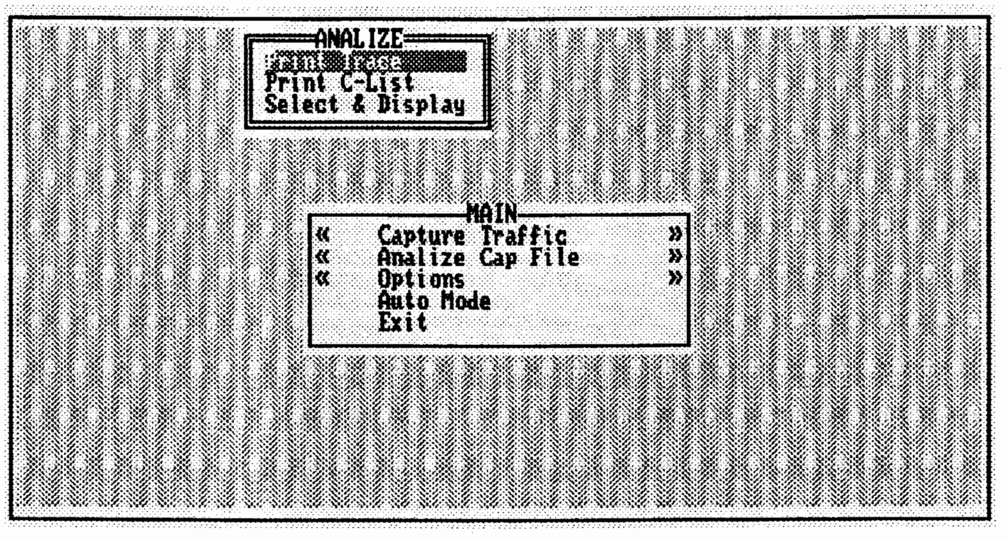


Figure 6.6 The Analyser menu

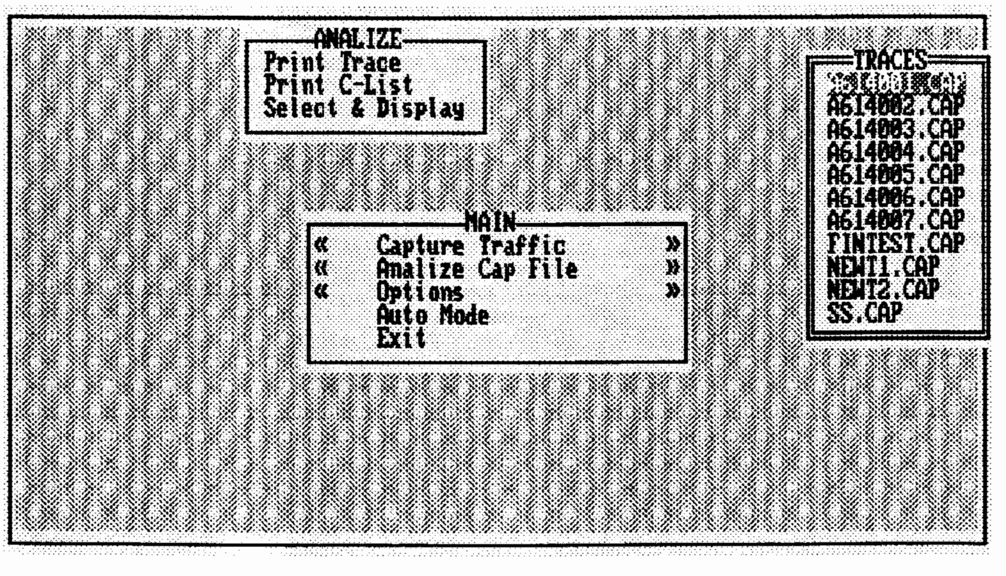


Figure 6.7 Selection list when Select & Display is selected.

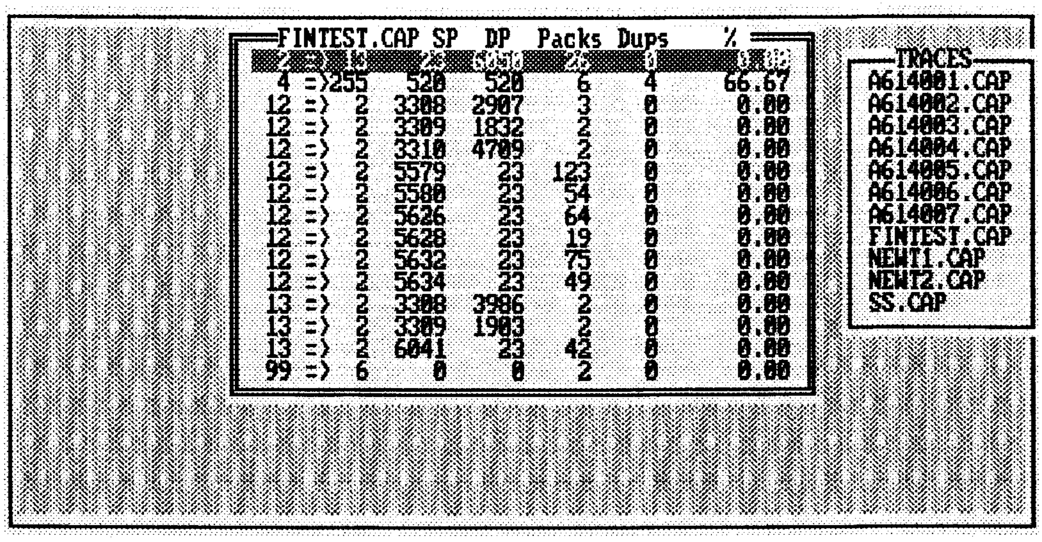


Figure 6.8 Selection list of conversations.

This selection list has as header the name of the trace file containing the conversations as well as a legend to the columns of figures. The entry for each conversation contains the last byte of the two IP numbers in the addresses, the port numbers of the conversation, the number of packets in the conversation as well as the number and percentage of duplicate packets in the conversation.

If any of the conversations in the list appears suspect (has a high percentage of duplicate packets in context with the number of packets), the user can move the highlighted bar to that conversation and press **enter** to see a time-sequence plot of that specific conversation.

6.2.1.3.1 Time-Sequence plot

Every conversation consists of two halves, traffic from A to B with acknowledgements from B to A and traffic from B to A with acknowledgements from A to B.

Two time-sequence plots can therefore be derived from each conversation. The time-sequence plot that will be displayed first (Figure 6.9), is that of the traffic moving from the left to the right port number as displayed in the conversation entry.

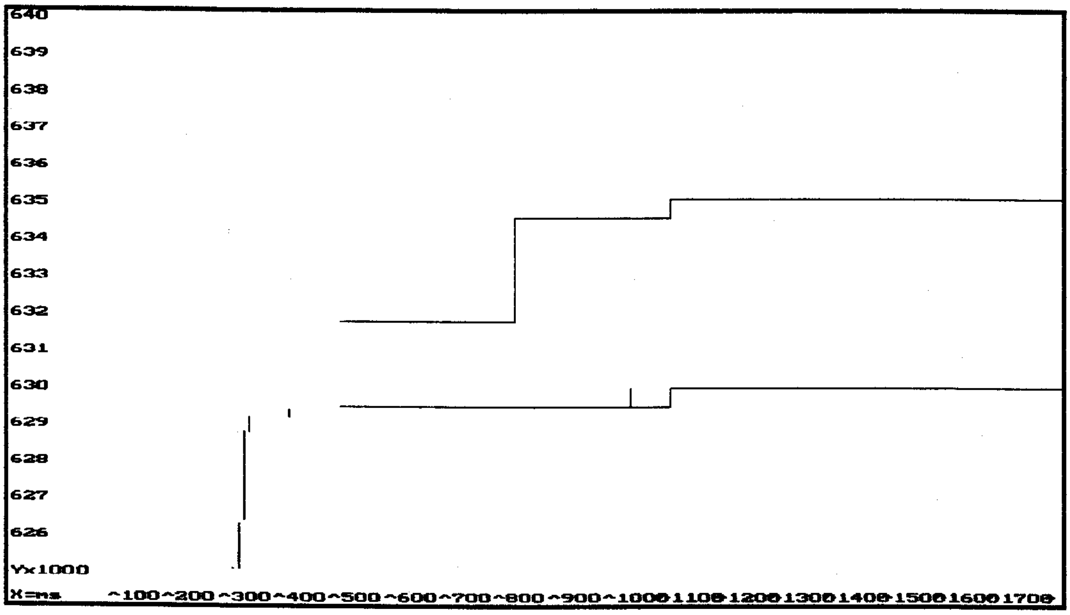


Figure 6.9 Time-sequence plot (Normal direction)

The second plot (Figure 6.10), called reverse direction by **TCPlot**, can be displayed by pressing F2 with the first plot on screen. This will take the user back to the conversation selection list and any selection now made, will be of the reverse direction plot. Pressing F2 again with a plot on screen, will return the user to normal direction display.

The scale bar on the X-axis depicts microseconds in increments of 100 microseconds. This scale is used to plot the packet according to relative time of arrival and do not refer to the time of a packet's timestamp. The scale on the Y-axis refers to a sequence number relative to the sequence number of the first packet of the conversation contained in the trace file.

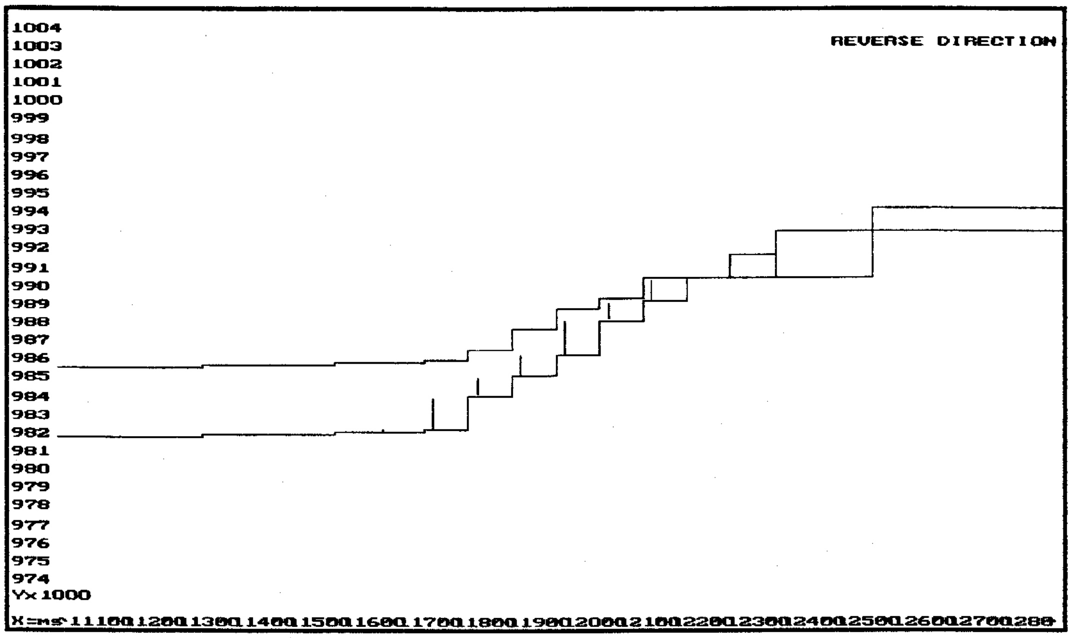


Figure 6.10 Time-sequence plot (Reverse direction).

As discussed in Chapter 5, scaling of a time-sequence plot may present problems. Although adjustments to the scale are made by **TCPlot** based on the data of the conversation, the user must adjust the scale manually to get the best results. This can be done by using the special function keys while the plot is displayed on screen to get the best plot. Provision has been made to adjust the Y-axis scale (F6 and Shift-F6) and the X-axis scale (F5 and Shift-F5). In addition to the scale adjustments, provision has been made for an enlargement factor (Up/Down arrows) as well. This enlargement factor is required to be able to show packets of 1500 bytes in some cases and packets of 64 bytes in others.

To follow the trace through from beginning to end, the left and right arrows can be used. The legend for these keys is available in the form of a help screen (Figure 6.11) by pressing F1 with the plot on screen.

6.2.1.4 The Options menu

The **Options menu** (Figure 6.12), when selected from the main menu, allows the user to set, inspect or clear the filters used to filter packets during the capturing of traffic or printing of trace files. Of the four options, only the **Set Filter** and **Select Conv** selections require some discussion.

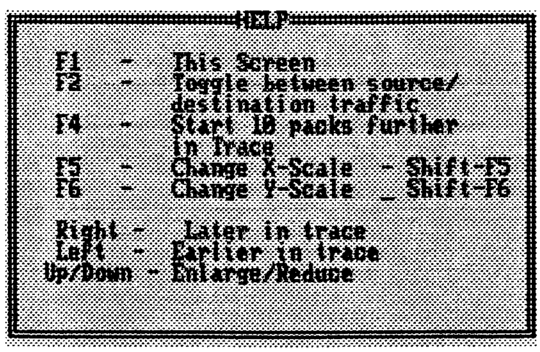


Figure 6.11 The Time-sequence plot help screen.

When **Set Filter** is selected, the set filter window (Figure 6.13) is displayed. The current filter settings are displayed and the user is then required to enter new settings. An IP number setting of zeros will clear all the filters, while any invalid IP number (containing numbers larger than 255) will be set to zero.

If a valid IP number has been entered, the user is requested to enter the port number filters.

Again, port numbers of zero, will cause the port filter to be disabled, thus leaving only the IP number filter enabled.

The **Select Conv** selection is used to determine active conversations currently on the network before allowing the user to select a conversation to capture. The IP and port filters will be set to the correct values automatically on selection of a conversation. On selection of this option, a red capture screen will be displayed for approximately five seconds while a short trace is collected from the network. The network activity can be estimated from this screen.

After the trace has been captured, the temporary trace file is analysed and active conversations are displayed in a selection list similar to that in Figure 6.8. The conversation of interest can now be selected.

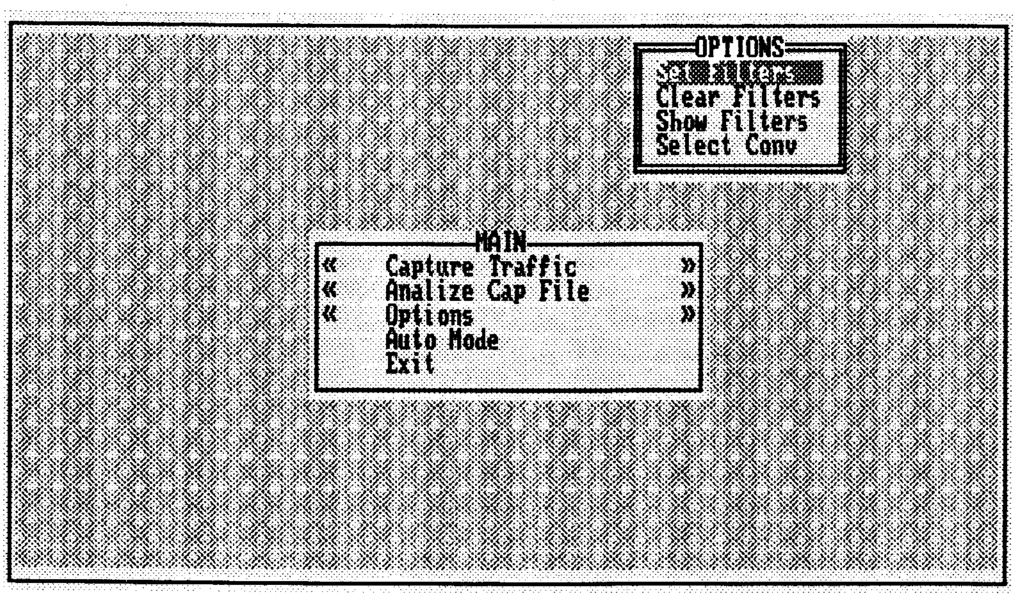


Figure 6.12 The option menu

6.2.1.5 Auto Mode

When **Auto Mode** is selected, the monitor will start in continuous store mode, displaying asterisks on screen as packets arrive. In this mode all packets will be collected and written out to disk as soon as the available memory drops below 10K. The file name used to store the trace under is a function of the date and time. As soon as disk IO is completed, the collection of packets for the next trace commences. This process will continue until the user aborts it by pressing a key.

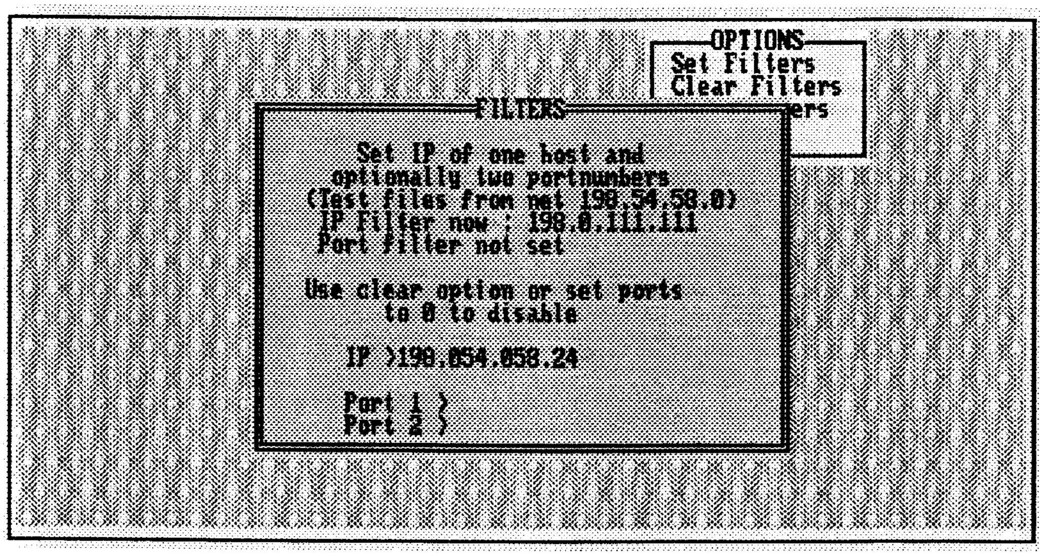


Figure 6.13 The Set filter window

6.3 Technical Reference

The **TCPlot** program was developed in such a way that it consists of a main program, containing only the menu structure, and a number of units. Some of these units are Turbo Pascal standard units and will be mentioned but not discussed here. The **Pick Unit**, which is a public domain menu system, was used instead of

Pascal's **Turbo Vision**. **Turbo Vision** was found to be too expensive with memory, thus leaving little memory to collect and store packets. Because this unit was not developed by the author and is only used as a menu system, it will only be described where relevant but its procedures and functions will not be discussed. As the focus of **TCPlot** is on the collection of packet traces and the analyses thereof, the units dealing with these functions will firstly be discussed.

6.3.1 The NetUW Unit

This unit handles all the functions related to monitoring traffic on the network and capturing packets for a trace file.

Although this unit only makes five procedures available to other programs, it contains several supporting procedures. The public procedures are:

DispStats
GetName
InitDriver
NetStart
SetFilter

Other procedures in the unit that fill a supporting role are:

AutoName
CalcBarDisp
DisplayShort
GetDispStr
GetPort
ProcessBuff
RecvPkt

ReadIP
SekBar
StoreBuff
StrAdr
StrIp
StrClickTime
SekBar
WriteFile
WriteIP

6.3.1.1 The AutoName procedure

AutoName uses values received from the procedures **Gettime** and **Getdate** to construct a unique name string. This string is used to name trace files when **TCPlot** is operating in **Auto Mode**.

6.3.1.2 The CalcBarDisp procedure

The purpose of this procedure is to display a summary string containing Ethernet address, IP address and time of arrival for each packet. It uses the function **GetDispStr** to build this string and displays it on the bottom half of the screen while bar graphs are displayed in the top half.

Because the graphs are displayed in the active window on the top half of the screen, this procedure uses the **Scroll** and **StrXY** procedures (discussed under the **Library** unit) to scroll only the bottom part of the screen. An absolute screen write displays the summary string.

6.3.1.3 The DispStats procedure

The **DispStats** procedure calls the procedure **GetStats** with a parameter of type **DrvStatsRec.** The structure of **DrvStatsRec** is:

```
DrvStatsRec = Record
     PackIn     : LongInt;
     PackOut    : LongInt;
     BytesIn    : LongInt;
     BytesOut   : LongInt;
     ErrIn      : LongInt;
     ErrOut     : LongInt;
     PackLost   : LongInt;
end;
```

Of the above, the number of packets, the number of bytes received as well as the number of packets lost (due to the lack of buffer space) are reported in a window.

6.3.1.4 The GetDispStr function

This function is used by **CalcBarDisp.** It accepts the packet length as parameter and returns a string containing Ethernet address, IP address and time of arrival. **GetDispStr** uses the functions **StrAdr, StrIp** and **StrClickTime** to build the return string.

6.3.1.5 The GetName procedure

The **GetName** procedure is called by **NetStart** when **StoreMode** for a single trace is selected. It requests a file name for the trace and hands it back in the parameter **FName** with the string '.cap' appended.

6.3.1.6 The GetPort procedure

This procedure is used to read port numbers in the address of packets in a specified buffer. **GetPort** accepts three parameters. The first, **Start**, is of type word and is used to indicate the position of the required port number in the buffer. The second parameter, **Pac**, is of type Ibuff and specifies the buffer while the third, **WO**, of type word is used to return the port number.

6.3.1.7 The InitDriver Procedure

The purpose of this procedure is to determine the presence of a packet driver and interface card. It will abort the program if no installed packet driver is found. If a packet driver is found, it will determine the type of interface card and if installed, will report the capabilities of that card to the user.

If a valid packet driver and interface card was found, this procedure will attempt to initialise the interface card to operate in promiscuous mode and hand all TCP packets to **TCPlot**. Failure to do so will be reported to the user.

This procedure makes use of three procedures found in the **DrvU unit** that will be discussed later. The first of these is the procedure **DriverInfo** which will return all information concerning a loaded packet driver in a data structure **Dinfo** of the type **DrvInfoRec.**

The procedure **AccessType** is called to initialise the packet driver to hand TCP packets to **TCPlot** and to inform the packet driver of the address of the receiving procedure of **TCPlot.** All parameters are placed in a data structure **AccRec** of the type **AccessTypeRec** before calling this procedure.

The procedure **SetRxMode** is called to set the operation mode of the interface card. In **TCPlot** it is called with '6' as a parameter, thus setting the operation mode of the interface to promiscuous.

6.3.1.8 The NetStart procedure

This is the main procedure of the monitor part of **TCPlot** and is called when a selection is made from the **Capture Traffic** menu. A single parameter, **FMode**, is used to determine the mode of operation of this procedure. Although four modes are possible, only **BarGraphMode** and **StoreMode** cause direct action in this procedure. The other modes determine actions to be taken by the procedure **ProcessBuff** in the main loop of the procedure.

The **NetStart** procedure forms the heart of the monitor and therefore the logic of this procedure is shown in Figure 6.14.

If **BarGraphMode** was selected this procedure sets Interrupt $1C to the address of the procedure **SekBar** on entry and resets it on exit. **SekBar** will then update the graphs on screen every second. In this mode the screen is also prepared for the graphic display and cleared on exit.

When **StoreMode** is received as parameter, this procedure will call the procedures **GetName** and **WriteFile** to get a file name from the user and write the trace to disk before terminating.

```
if selected mode = BarGraphMode
        prepare screen
        point user interrupt to bar procedure
repeat  { Wait for packets / Start main loop}
        if a packet in buffer
                process the packet
        if packet arrived    {Packet driver called}
                             {receiver procedure}
                copy packet to buffer
until memory full or a ESC pressed
Case selected mode of
        BarGraphMode
                reset user interrupt
        StoreMode
                Get name for file
                Write packet headers to tracefile
end.
```

Figure 6.14 Algorithm of the NetStart procedure.

After the initial preparations, **NetStart** enters a loop to wait for packets arriving at the interface card. The loop will terminate only when the Escape key is pressed or if the available memory drops too low. In this main loop of the procedure, a constant check is done to determine the presence of a packet in the incoming buffer. If a packet is in the buffer, shown by **BuffAvail** with a value of 1, the procedure **ProcessBuff** is called to process this packet and make the buffer available to incoming packets again.

Any packet arriving at the interface card during this time will be handed to the packet driver. The packet driver will determine if the packet should be handed to **TCPlot** and cause a software interrupt if so. This interrupt will cause the receiver procedure (**RecvPKT**) of **TCPlot** to copy the packet into the incoming buffer (if it's available).

6.3.1.9 The ProcessBuff procedure

This procedure is called from within the main loop of the **NetStart** procedure whenever a packet is placed in the incoming buffer. The action of this procedure is determined by the mode in which the **NetStart** procedure was started.

ProcessBuff first checks whether any filters are set. If no filters are set, processing is done as described in the next paragraph. If the IP-filter is set, however, the IP numbers of the packet in the incoming buffer will be retrieved and compared with the IP-filter. Should the packet's IP number satisfy the IP-filter and the port number filter is set, the port numbers are retrieved and compared with the filter. Only packets satisfying the set filter are passed on for processing, while all other packets are ignored and the buffers set free without being processed. The logic of this procedure is such that no IP number or port number is retrieved from the buffer if it is not required by the filter setting, thus ensuring the smallest possible overheads.

In **HashMode** the only action is to write a hash character to the screen. In the modes **StoreMode**, **PackDisplayMode** and **BarGraphMode**, the procedures **StoreBuff**, **DisplayShort** and **CalcBarDisp** respectively will be called to process the packet in the incoming buffer.

Whether the buffer was processed or not (as in the case of **HashMode**), it is released by setting **BuffAvail** to 0 and the packet count is incremented before the procedure is exited.

6.3.1.10 The RecvPkt procedure

This is an assembler procedure that is executed during an interrupt from the packet driver. Because values in registers can be changed by this procedure, the registers AX, BX, CX and DX are saved on entry to this procedure and restored on exit. On arrival of a packet for **TCPlot**, this procedure is called once or twice by the packet driver. On the first call the packet driver indicates a new arrival with a value of 0 in the AX register. In this case the **RecvPkt** procedure will return a pointer to a buffer for the packet (or 0:0 if no buffer is available).

If a buffer was not available, the packet driver will discard the packet, otherwise it will call **RecvPkt** again with the value 1 in the AX register. The **RecvPkt** procedure will in this case copy the packet to the buffer and set the value of the variable **BuffAvail** to 1.

6.3.1.11 The ReadIP procedure

This procedure is used to enter an IP address from the keyboard. The input is checked for validity and once the four bytes of the address have been entered, the values are moved to a LongInt and handed back to the caller in the parameter **RIP** of type LongInt.

6.3.1.12 The SekBar procedure (Interrupt)

The purpose of this procedure is to update the bar graphs for both bytes per second and packets per second.

When in **BarGraphMode** the interrupt vector of Interrupt $1C points to this procedure and it will therefore be executed approximately eighteen times per second. Because the updating is required only every second, a counter, **Sek18**, is kept to ensure that the update is done during every eighteenth interrupt. When updating the graphs, the procedure **DrawHorzSet** that will be discussed later, is used.

6.3.1.13 The SetFilter procedure

This procedure accepts one parameter of type char. If a 'F' is received as parameter, this procedure displays a window showing the current filter setting and requests new settings from the user. Depending on the values of the input, global filters are set for IP number or IP number as well as port numbers. The input of zeros will clear all filters. A parameter value of 'S' will only display current filters in a window without the option to change them while a value of 'C' will clear all filters.

6.3.1.14 The StoreBuff procedure

StoreBuff is called by **ProcessBuff** if the **NetStart** procedure was started in **StoreMode**. This procedure stores information about packets in memory with the purpose of creating a trace file. This is done by firstly moving the first 64 bytes (the header) of the packet into a data structure of type **PackHeadType. PackHeadType** has the following structure:

```
PackHeadType = Array [ 1..PSize] of byte
```

PSize is a constant set to 64, this value can be changed if the need arises to develop **TCPlot** to also store part of the data of packets.

This structure, together with a time record (**TimeRec**), is then placed in a data structure of type **PackRec.** A linked list is then formed consisting of structures of type **PackRec. DataP** is a pointer to a structure of **PackRec.**

The structures of **PackRec** and **TimeRec** are shown below:

```
    TimeRec = Record
        Hours     : Word;
        Min       : Word;
        Sek       : Word;
        Hun       : LongInt;
    end;

    PackRec = Record
        PackHead       :  PackHeadType;
        Time           :  TimeRec;
        Next           :  DataP;
    end;
```

The time values placed in **TimeRec** are arrived at by calling the procedure **GetOwnTime** that will be discussed later. Once the required data has been placed in the structure, the procedure **Add**, which is internal to the **StoreBuff** procedure, is called with a pointer to the new structure as parameter, to place the new structure in the linked list in memory.

6.3.1.15 The StrAdr function

This function accepts the position in the buffer where an address starts and returns the Ethernet address as a string.

6.3.1.16 The StrIP function

This function takes as parameter a character 'S' or 'D'. Depending on the parameter, the source or destination IP address of the packet in the buffer, will be returned as a string.

6.3.1.17 The StrClickTime Function

The function **StrClickTime** uses the procedure **GetOwnTime** to obtain the time and returns a string showing the time in minutes, seconds and microseconds.

6.3.1.18 The WriteFile procedure

This procedure is called to write the linked list of packet records (**PackRec**) in memory to a file on disk. It accepts a single parameter, **CapFile**, that is used as file name.

6.3.1.19 The WriteIP Procedure

WriteIP accepts a parameter **WIP** of type LongInt and displays it on screen as a four byte IP address.

6.3.2 The PlotU unit

This unit contains most of the procedures for the analyser part of TCPlot. The public procedures of this unit are:

ConvToMenu
FileList
GetConversation
PrintConv
PrintGraph
ReadFile
SelectDisp

Other procedures in the unit that fill a supporting role are:

AutoFilter
AutoFilterSet
DisposeConvList
DisposeConvInfo
FiveSec
MakeList
MenuGraph
PackToWord
PackToLong
ShowConvs
StoreConv
WriteLstIP
XScaleBar
YScaleBar

6.3.2.1 The AutoFilter procedure

The purpose of the procedure is to provide the procedure **AutoFilterSet** with the information regarding a TCP conversation, thus allowing **AutoFilterSet** to set filters to collect only packets of that conversation. On entry, **AutoFilter** starts the monitor in store mode to collect a five second trace of packets on the network. It then calls the procedures **GetConversation** and **ConvToMenu** to analyse the trace and place conversations in a selection list respectively.

6.3.2.2 The AutoFilterSet procedure

When called by **AutoFilter**, this procedure will extract an IP number and port numbers from the selected menu entry. These values are then used to set filters to collect only packets of the selected conversation.

6.3.2.3 The ConvToMenu procedure

The procedure **ConvToMenu** receives a file name in the parameter **TFile**. It then calls the procedure **GetConversation** with **TFile** as parameter. The linked list produced by the procedure **GetConversation** is then used to create a selection list of conversations and their statistics. Depending on the value received in the parameter **ConvMode**, the procedure will either call the procedure **MenuGraph** (if ConvMode = 'N') or the procedure **AutoFilterSet** (If ConvMode = 'A'), with a pointer to the entry in the selection list.

6.3.2.4 The DispConvList and DispConvInfo procedures

Both these procedure are used to dispose of linked lists of conversations or conversation information respectively, thereby setting the memory used by these lists free.

values are then placed in a data structure of type **ConVRec**.

As indicated in the data structure of **ConvRec**, values are placed in the fields marked with an asterisk before the procedure **StoreConv** is called with this data structure as parameter.

The structure of **ConVRec** is:

```
    ConVRec = Record
             PCounter          :  Word;
             PrevSeqNoL        :  LongInt;
             PrevAckNoL        :  LongInt;
             PrevSeqNoH        :  LongInt;
             PrevAckNoH        :  LongInt;
             PackDup           :  Word;
    *        PackSeqNo         :  LongInt;
    *        PackAckNo         :  LongInt;
    *        SourceIP          :  LongInt;
    *        DestIP            :  LongInt;
    *        SourcePort        :  Word;
    *        DestPort          :  Word;
    *        PWindow           :  Word;
             Next              :  Word;
end;
```

6.3.2.8 The MakeList procedure

This procedure is local to the procedure **PrintGraph** and is used to extract only packet records belonging to a certain conversation. Information of each packet is placed in a data structure of the type **CVRec** which in turn is placed in a linked list in memory. The structure of **CVRec** is shown below with **CVP** as pointer to a structure of type **CVRec** while **TimeRec** has the same structure as discussed under **StoreBuff**.

The structure of **CVRec** is:

```
    CVRec      = Record
         SourceIP        : LongInt;
         DestIP          : LongInt;
         SourcePort      : Word;
         DestPort        : Word;
         PackSeqNo       : LongInt;
         PackAckNo       : LongInt;
         PWindow         : Word;
         DLen            : Word;
         T               : TimeRec;
         Next            : CVP
end;
```

6.3.2.9 The MenuGraph procedure

This procedure is called by **ConvToMenu** when an entry is selected. **MenuGraph** calls the procedure **PrintGraph** with the file name and two query port numbers as parameters. The port numbers used in the call are extracted from the item that the menu entry points to. The order of the two query ports in the parameter list depends on the value of the boolean variable **Reverse**. This variable is set by pressing F2 on the graphics screen and determines which half of the conversation is plotted.

6.3.2.10 The PackToLong procedure

As with the procedure **PackToWord** (6.3.2.11), this procedure is used to extract 32 bit values from packet structures and returns it as a value of the type LongInt.

6.3.2.11 The PackToWord procedure

The need for this procedure arises from the fact that values are stored with the low byte first in the packets. Three parameters are accepted, **Pac**, which is the packet data structure containing the value, **Wo**, in which the value is returned as a word and **Start**, which is used to indicate the position of the value in **Pac**.

6.3.2.12 The PrintConv procedure

This procedure accepts a single parameter, **FName** of type string. The procedure firstly calls the procedure **DisposeConvList** to dispose any existing conversation lists in memory. It then calls the procedure **GetConversation** to analyse the trace file named in **FName** and build a new linked list, containing all the conversations and statistics of **FName**.

Once this list has been constructed, it is transversed and a list containing all conversations, as well as the percentage duplicate packets in each conversation, is printed.

6.3.2.13 The PrintGraph procedure

The procedure **PrintGraph** is responsible for plotting a specific TCP conversation on a time-sequence plot. It accepts as parameters the two TCP ports numbers of the conversation to be plotted as well as the name of the trace file.

It first calls the local procedure **MakeList** to extract information relevant to the conversation from the trace file. Information thus extracted is placed in memory to enhance drawing and rescaling of the time-sequence plot. The second action is to call the procedure **DetectScale** to determine the initial scaling and enlarging factors for the plot.

The local procedures **XScaleBar** and **YScaleBar** are used to draw the scale on the X and Y axis respectively. If rescaling is done, these procedures are called again to draw the new scales.

The linked list in memory is processed sequentially and the packets, acknowledgement line and window lines are plotted. The variables **PrevWin, PrevP, PrevAck** and **PrevAckP** are used to store values of the previous packet, thus making it possible to draw a line, like with the acknowledgement line, from the previous value to the new value. Values used for the window line are arrived at by adding the offered window of the acknowledging packet to the acknowledgement number.

The procedure also provides for rescaling by scanning the keyboard in a loop until either escape, or one of the rescaling function keys, is pressed. If a function key is pressed, the scale is adjusted and the time-sequence plot redrawn.

6.3.2.14 The ReadFile procedure

This procedure's main function is to print a hard-copy of the packet trace file whose name is received in the parameter **FName**. If the variables **IPFilterSet** and **PortFilterSet** are TRUE, the values in the global variables **OptIP, OptPort1** and **OptPort2** are used as a filter. In such a case, a trace containing only packets belonging to the selected TCP conversation, will be printed.

6.3.2.15 The SelectDisp procedure

This procedure accepts the parameter **SMode** of type char. It produces a menu with all the trace files (ending with .cap), in the current directory, as entries. When one of these entries is selected, one of the **FileList** procedures (see 6.3.2.5) is called with pointers to the selected menu entry. Corresponding to the values 'P', 'T' and 'C' in **SMode,** one of the procedures **Filelist, FileList2 or FileList3** will be called by this procedure.

6.3.2.16. The ShowConvs procedure

When called **ShowConvs** accepts a file name as parameter and calls three procedures with this file name as parameter. To get the conversations in the trace file, the procedure **GetConversation** is firstly called. **ConvToMenu** is then called to process the link list produced by **GetConversation,** creating a menu with these conversations and their statistics as entries.

Finally the procedure **DisposeConvList** is called to dispose the linked list and set the memory free.

6.3.2.17 The StoreConv procedure

This procedure is called by the procedure **GetConversation** and accepts a data structure of the type **ConVP** (a pointer to type **ConVRec**) as parameter.

The purpose of this procedure is to keep record of each conversation found in the trace file. It must also detect duplicate packets in conversations and keep track of the number of packets as well as the

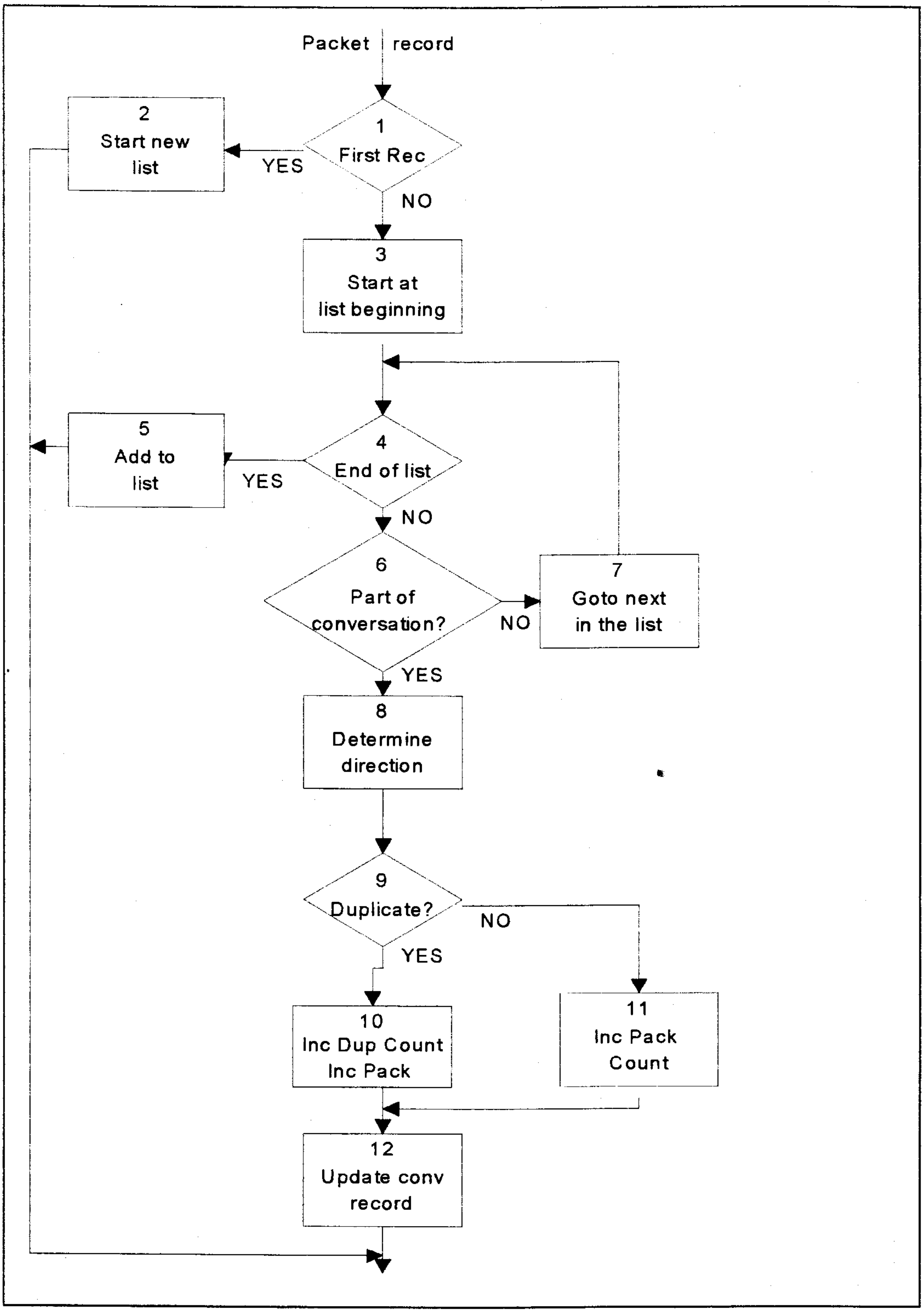


Figure 6.15 Logic of the StoreConv procedure

number of duplicate packets in each conversation. All this information is kept in a linked list consisting of a data structure, of the type **ConVRec**, for each conversation. The logic of this procedure is depicted in Figure 6.15.

6.3.2.18 The WriteLstIP procedure

This procedure accept an IP address as a LongInt in the parameter **WIP**. The IP number is then printed in the format xxx.xxx.xxx.xxx, as used in the printing of traces.

6.3.2.19 The XScaleBar and YScaleBar procedures

These procedures are responsible for displaying the scale bars of the X and Y axis of the time-sequence plot respectively.

6.3.3 The AsmTim unit

This unit is the implementation of the high resolution timer discussed in Chapter 4. Its only two public procedures are **GetOwnTime** and **StopTimer.** Other procedures and functions of this unit are declared as external and are located in the object file **Astim.obj** which is linked into this unit.

6.3.3.1 The GetOwnTime procedure

When called this procedure gets the system time and then calls the **_HrTime** function of the timer to get the microsecond count. The time is returned as **Hour, Min,** and **Sek** of type word, while the ticklet count from **_HrTime** is converted to microseconds and returned in **Mil** as type LongInt.

6.3.3.2 The _Hrt_Close procedure

_Hrt_Close is an external procedure that unhooks all interrupts installed by **_Hrt_Open**.

6.3.3.3 The _Hrt_Open procedure

This is an external initialisation procedure for the timer that will install the interrupt and clear the tick count.

6.3.3.4 The _HrTime function

This external function determines the high resolution time. It returns the ticklet count (accumulated since the first call to **_Hrt_Open**), in a 32 bit variable.

6.3.3.5 The StopTimer procedure

StopTimer's only function is to stop the high resolution clock by calling the external procedure **_hrt_close**. This procedure must be called when **TCPlot** terminates, failing to do so will cause the computer to hang because the timer will update memory locations now used for other purposes.

6.3.3.6 The TimeExit procedure

This is an exit procedure that will call **StopTimer** when the **TCPlot** terminates for whatever reason.

6.3.4 The Library unit

This unit contains an assortment of supporting procedures and functions that are used throughout the program. The procedures and functions used by **TCPlot** will be discussed in alphabetical order with the public procedures and functions first, followed by the local ones.

6.3.4.1. The Beep procedure

This procedure accepts two parameters, **Hz** and **Ms**. When called the procedure will cause a sound with **Hz** as frequency for **Ms** long.

6.3.4.2 DrawHorzSet

The **DrawHorzSet** procedure is used to display a bar graph of the number of bytes and number of packets that arrived each second, for the previous ten seconds.

As parameters it accepts and returns a data array, **BarN**, in which the values of the past ten seconds are stored. The other parameters are **NewH**, in which the value of the new bar to be added is placed and two values (**BaseLine** and **YPos**) that are used to locate the graph on screen.

On entry the new value is shifted into the data array, shifting the oldest value out, and then scanned to determine the maximum value in the array. This maximum value is then passed to the function **GetMax** to determine the scaling factor now required.

If there was no change from the previous scale, the graph is scrolled one line up and the new bar added at the bottom. If, however, the new value caused a change in scale, the whole graph is redrawn on the new scale. The maximum height is shown in the right hand corner.

6.3.4.3 The Hex function

This function takes a byte as parameter and returns the hexadecimal value as a string;

6.3.4.4 The HexWord function

This procedure takes a word as parameter and returns its hexadecimal value as a string.

6.3.4.5 The HexLong function

This procedure takes a LongInt (32 bit) value as parameter and returns its hexadecimal value as a string.

6.3.4.6 The KeyProc procedure

The **KeyProc** procedure reports the keyboard scan code and the ASCII code of a character in the parameters **_KeyPos** and **_Ascii** respectively, when a key has been pressed.

6.3.4.7 The NoCursor and NormCursor procedures

These two procedures are used to hide and redisplay the cursor. They are called when a message is displayed in a window to prevent the displaying of a flashing cursor as well.

6.3.4.8 The StrXY procedure

The procedure **StrXY** accepts as parameters absolute screen co-ordinates in **X** and **Y**. The string passed to the procedure in the parameter **St**, is then displayed at the absolute co-ordinates, regardless of any active window.

6.3.4.9 The Scroll procedure

This procedure is used to scroll a selected part of the screen, a variable number of lines in a given direction. The area to be scrolled is determined by values received in the parameters **X**, **Y**, **X1** and **Y1**, while the number of lines scrolled are determined by the value of the parameter **Lines**.

The parameter, **Direction**, can have the values **UpDir** or **DownDir** and determine the direction of scrolling while the parameter, **Attribute**, determines the characters in new lines.

6.3.4.10 The procedure XYWrite

XYWrite is procedure used to display the value of the variable at the screen co-ordinates received in parameters **X** and **Y**. By using an un-typed parameter, strings as well as numbers in variables of different types, can be displayed. The type of the variable passed to the procedure is indicated in the parameter **T**.

6.3.4.11 The GetMax function

This function returns the highest value in an array of **Num** values. The array (**Data**) as well as **Num** are accepted as parameters.

6.3.4.12 The CBar procedure

This procedure is used to display a bar in the two sets of bar graphs that are used by the monitor part of **TCPlot**. It accepts as parameters the **Height** of the bar to be drawn as well as the position where the bar must be placed. The position is indicated as X and Y co-ordinates in the parameters **BaseLine** and **YPos** respectively.

Because this procedure is internal to **DrawHorzSet** and is used to draw the newest bar of a set of bars, the block of the screen reserved for the bar graph is scrolled one line before the new bar is drawn on the new line.

6.3.5 The DRVU unit

This is a unit containing all the procedures required to address the packet driver discussed in chapter 3. All the procedures in this unit are publicly known with the exception of the procedure **TestVec**. The unit contain declared constants for all the possible return codes by the packet driver as well as constants for the implemented initiation values.

Where the logic of the procedures were discussed in chapter 3, the discussion here will only show its implementation in Pascal

6.3.5.1 The AccessType procedure

This procedure initiates access to the packet driver. It has one parameter, **AccRec** of the type **AccessRecType** which is used to specify the type of packet **TCPlot** requires from the packet driver. The same parameter is used to hand the address of the receiving procedure to the packet driver. The handle used for this agreement is handed back to **TCPlot** in the same structure.

In **TCPlot** this procedure is called by **InitDriver** with the **Type_** field set to point to the variable **TypeField** and **Receiver** to point to procedure **RecvPkt.** As **TCPlot** must operate in promiscuous mode, **If_Type** is set to **AnyType** and **TypeLen** to 0. The **If_Class** field is set according to the **DInfo.Class** field received from calling the procedure **DriverInfo.**

The structure of **AccessRecType** is:

```
AccessRecType = Record
     If_Class : Byte;
     If_Type  : Word;
     If_No    : Byte;
     Type_    : Pointer;
     TypeLen  : Word;
     Receiver : Pointer;
end;
```

6.3.5.2 The DriverInfo procedure

The **DriverInfo** procedure is called by **InitDriver** when **TCPlot** starts. It provides information about the interface card in the parameter **DInfo** of type **DInfoRec.** **TCPlot** uses this information to determine whether the functions needed, are offered by the interface or whether to abort.

The structure of **DInfoRec** is:

```
DInfoRec = Record
     Vers        : Word;
     Class       : Byte;
     Type_       : Word
     Number      : Byte;
     NameP       : Pointer;
     Funct       : Byte;
end.
```

In the structure above, **TCPlot** uses the **Class** field for the **AccessType** procedure and the **Funct** field to determine the functions available on the interface card.

6.3.5.3 The FindPktInt procedure

This procedure is used to determine whether a packet driver is loaded and if so to determine the software interrupt to address it. The procedure is called when **TCPlot** is started before any other procedures and returns values in two global variables. The first, **DrvIntFound**, is a boolean variable that is set to true if a driver is present. **TCPlot** will abort if this procedure returns with **DrvIntFound** set to false. The second variable, **PckInt**, is set to the value of the interrupt vector of the packet driver and is used by other procedures when addressing the packet driver.

FindPktInt starts at the lowest possible interrupt vector (60H) and calls the internal procedure **TestVect** to determine whether this is the interrupt for the packet driver. If not, the next interrupt vector will be tested until the correct one is found or 80H is reached in which case it sets **DrvIntFound** to false. When called, the **TestVec** procedure checks for the string 'PKT DRVR' in the first 12 bytes following the entry point of the interrupt. If this sting is found, it returns with **DrvIntFound** set to true, if not **DrvIntFound** is set to false.

6.3.5.4 The SetRxMode procedure

This procedure is used to set the receive mode of the interface card. The required mode setting is handed to **SetRxMode** in the parameter **Mode**, of type word, when called. In the case of **TCPlot**, the procedure is called with the value 6 as parameter. Thus setting the interface card to promiscuous mode.

6.3.5.5 The GetStats procedure

When called, the procedure **GetStats** returns the statistics of packets received by the interface card in the variable **DStats** of type **DrvStatsRec**. The structure of **DrvStatsRec** is discussed under 6.3.1.3.

6.3.5.6 The DrvRelease procedure

This procedure is called before **TCPlot** terminates to notify the packet driver that **TCPlot** no longer requires packets of the type specified in **AccessType**, to be handed to it. If this is not done, other applications trying to use the packet driver, will not operate correctly.

Chapter 7

TCPlot in Action

7.1 Introduction

In this chapter, a trace file is analysed to show the effectiveness of the **TCPlot** analyser. Several tests, done to evaluate **TCPlot**'s ability to collect all packets from the **Ethernet**, are also discussed. These tests were done under a variety of conditions and using different interface cards, to determine the influence on **TCPlot**'s operation.

7.2 The Monitor at work

The primary function of the monitor part of **TCPlot** is the collection of packets. It is therefore necessary to test **TCPlot**'s ability to keep up with the rate of traffic on the network without losing packets. The various factors that can have an influence on this ability were therefore examined and the tests done are discussed in this section.

As part of the evaluation, tests were done with different packet drivers. It was found that the older packet drivers for the Western Digital and SMC Ethernet interface card (**WD800** series), do recognise the card and its capabilities but do not operate correctly in promiscuous mode. The driver, typically used with Western Digital and SMC interface cards in Novell networks, (**SMC8000.com**), does not report an error when **TCPlot** requests promiscuous mode, but does not initialise the card to the correct mode. When using **TCPlot** with one of the above drivers, only packets directed to the management station as well as broadcasts will be seen. The packet drivers available in the **CRYNWR** packet driver collection **PKTD11C.zip**, seem to be able to switch the interface card to promiscuous mode. For Western Digital and SMC interface cards, the **SMC_WD** driver has been used in all the tests. **PKTD11C.zip** is available with anonymous ftp, but will also be included with the electronic version of **TCPlot**.

7.2.1 Using an 8-Bit Interface

This test was done to determine how a 386DX 40 MHz computer with an 8-Bit SMC Ethernet adapter will handle various levels of traffic on the network whilst running **TCPlot**.

The first test was done when traffic consisted of mostly small sized packets (Telnet) at a rate of approximately 96 packets per second. During the six minutes the test lasted, an average of 106016 bits/s were transmitted on the network with peaks not much higher. If this is expressed as a percentage of the Ethernet bandwidth, the traffic was a little more than 2% of the bandwidth (determined as discussed in Chapter 4). The test was done with **TCPlot** in the **BarGraphMode** to allow for the maximum on-line processing by **TCPlot**. As was expected, no packets were reported lost by the packet driver in the six minute test.

It was found, however, that if the packet driver is loaded and not initialised, as would happen if the computer is booted and left idle or loaded with another application, the packet driver registers lost packets. This is probably due to the way that the packet driver initialises itself during loading. The test results as reported by **TCPlot** after reading the packet drivers' statistics, are shown in Table 7.1.

The test was repeated during a time that the activity on the network was higher. The traffic on the network during this test consisted of small Telnet as well as maximum sized Ethernet packets in a ratio of approximately 1:1. The average packet count was 196 packets per second while 1619183 bits per second were received.

Values of various counters six minutes after packet driver was loaded (machine idle)		After running TCPlot in BarGraphMode for six minutes	
Packets	4010	Packets	37489
Bytes In	1475239	Bytes In	6245980
Lost	131	Lost	131
LAN activity 2%			

Table 7.1 Results of an 8-bit interface with low traffic load.

This accounted for an average use of about 18% of the Ethernet bandwidth. As can be expected, there were peaks of traffic as well as fairly idle times during the sample period. The results are shown in Table 7.2.

The final test reported on here, was a prolonged test lasting 30 minutes, with the **TCPlot** monitor operating in **BarGraphMode**. During this test period, the classes in the computer laboratories of Technikon OFS started and large numbers of student loaded applications from the file server simultaneously. This caused LAN activity peaks of 35% of the Ethernet bandwidth.

Values of various counters six minutes after packet driver was loaded (machine idle)		After running TCPlot in BarGraphMode for six minutes	
Packets	4781	Packets	75446
Bytes In	1902097	Bytes In	74765349
Lost	55	Lost	55
LAN activity 18%			

Table 7.2 Results of an 8-bit interface with moderate traffic load.

The average LAN activity during the test period was 19%, with an average of 405 packets per second arriving at the interface. The results of this test are shown in Table 7.3. As can be seen, no packets were lost even with these high traffic levels and prolonged test.

Values of various counters before start of test		After running TCPlot in BarGraphMode for thirty minutes	
Packets	1271	Packets	730031
Bytes In	523854	Bytes In	298839660
Lost	2	Lost	2
Average LAN activity 19% - Peak LAN activity 35%			

Table 7.3 Results of an 8-bit interface with high traffic load after 30 minutes.

Although the LAN activity during these tests was far from the theoretical maximum of Ethernet, an Ethernet network will become saturated at approximately 55% LAN activity [Nemzow, 1988]. This, together with Sudama's statement [Sudama, 1990] that even the most heavily loaded LANs at DECNet was found to carry less than a thousand of the theoretical eight thousand packets per second, led the author to believe that **TCPlot** will be able to handle most network traffic without packet loss.

7.2.2 Using a 16-Bit interface

Given that any 16-Bit interface should out perform an 8-bit interface, together with the fact that no packets were reported lost with the 8-bit interface, made testing with this interface redundant.

7.2.3 The effect of filters

The use of filters cause additional overheads that can have an influence on the ability of TCPlot to capture packets. The additional overheads are due to the fact that addresses and port numbers of all arriving packets must be retrieved from the buffer and compared with the filter values. Only then can a decision be made to ignore the packet or to move it to memory.

Values of various counters before start of test		After running TCPlot in BarGraphMode for six minutes	
Packets	37489	Packets	147649
Bytes In	6245980	Bytes In	32794540
Lost	61	Lost	61
Average LAN activity 10%			

Table 7.4 Results of an 8-bit interface with moderate traffic load after TCPlot has been in action for six minute with IP and Port filters set.

To reduce the overheads, the implementation of filters in **TCPlot** will only retrieve an address if the corresponding filter is set. To determine the effect of filters in the worst case therefore, the test described here was done with both the IP as well as the port filters set. As can be seen from the results in Table 7.4, no packets were reported lost during the six minutes test on a network with moderate traffic, even with the use of filters.

7.3 The Analyzer at work

To illustrate the time saving capabilities of the **TCPlot** analyser, a trace file taken on a relative quiet network for approximately 20 seconds, is shown in Appendix A. This trace was intentionally taken on a quiet network (24 packets per second or approximately 0.2% of the bandwidth), to show the need for a tool such as **TCPlot** even when traffic is minimal.

7.3.1 The collected trace

The format of this trace is typical of traces produced by most network management packages. It shows the time of arrival of each packet in seconds and microseconds, the source and destination IP addresses and the source and destination port addresses. In addition to this, some will report the sequence and acknowledgement numbers and the offered window and data length of the packet.

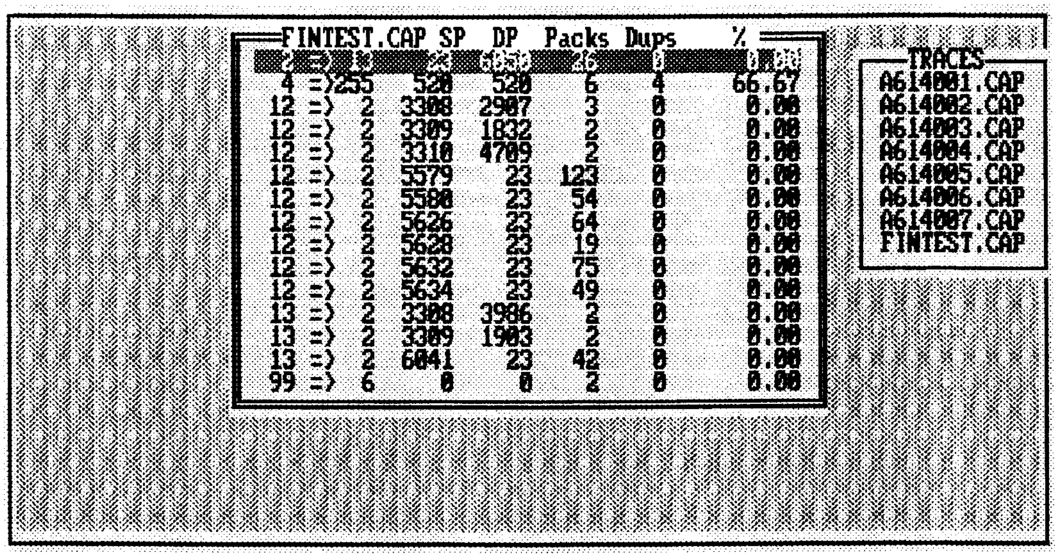
TCPlot's analysis of the trace file, showing the conversations that took place during the time packets were collected, is shown in Figure 7.1. This analysis of conversations shows only one conversation with duplicates, but as that conversation consists of only six packets, it is not deemed of interest. For the sake of this illustration we will focus our attention on the conversation with the most packets. This is the telnet (port 23) conversation between a file server (198.54.58.2) and port 5579 of a terminal controller (198.54.54.12).

An attempt to follow this conversation in the above trace, is difficult and time consuming. (If the trace was taken on a busy network, it would have been even worse.) Even when irrelevant data is filtered from the trace, leaving only packets of the conversation as in Appendix B, the task of interpreting the trace is not simple.

7.3.2 Filtered trace

As the IP addresses of the hosts are known, the trace file is normally reduced to relevant data by removing all packets except those between two specified hosts. In the trace as shown in Appendix A, this strategy would have been of little value, as terminals connected to a terminal controller (multiplexer), do not have their own IP addresses. It will correctly reduce the trace to a trace containing only packets sent between the controller and the fileserver, but the trace will still contain numerous conversations.

The ability of **TCPlot** to analyse a trace file and report on the different conversations contained therein, makes it possible for **TCPlot** to use the port numbers of conversations as filter. The trace can now be reduced to a trace containing only packets belonging to the conversation of interest.



FINTEST.CAP			SP	DP	Packs	Dups	%
4	=>	255	520	520	6	4	66.67
12	=>	2	3308	2907	3	0	0.00
12	=>	2	3309	1832	2	0	0.00
12	=>	2	3310	4709	2	0	0.00
12	=>	2	5579	23	123	0	0.00
12	=>	2	5580	23	54	0	0.00
12	=>	2	5626	23	64	0	0.00
12	=>	2	5628	23	19	0	0.00
12	=>	2	5632	23	75	0	0.00
12	=>	2	5634	23	49	0	0.00
13	=>	2	3308	3986	2	0	0.00
13	=>	2	3309	1903	2	0	0.00
13	=>	2	6041	23	42	0	0.00
99	=>	6	0	0	2	0	0.00

Figure 7.1 Analyses of the conversations in Fintest.cap

The effect of using a filter for both IP number and port numbers is shown in Appendix B. Of the 471 packets contained in the trace file, the 123 packets belonging to the conversation of interest remains in the filtered trace. It is clear that even by using this reduced trace, it still remains a formidable task to work through this trace to detect any problems.

7.3.3. Time-sequence plot

With the use of **TCPlot's** plotting feature, however, the above can be done in a matter of seconds. Once the conversation to be plotted has been selected, the plot will be displayed and can be scaled.

For the purpose of this illustration, the plot of the reverse direction conversation is discussed here. After the reverse direction has been selected, the plot was scaled down (F6) and scanned from beginning to end (using the right arrow). The time-sequence plot in Figure 7.2 shows the time slice 2s 297892µs to 3s 416222µs. Due to the small scale the first packet visible on the plot, is the packet that arrived at 2s 697972µs with a data length of 19 bytes. As can be clearly seen on the plot, the packets sent to the terminal were acknowledged promptly, and no duplicates were sent. The terminal, however, closed its window briefly (window and acknowledgement lines meet) during this time (between the packet that arrived at 3s 188372µs and the packet at 3s 416222µs), an indication that the terminal was busy and could not accept any data. The zero window offered can also be seen in the filtered trace (Appendix B).

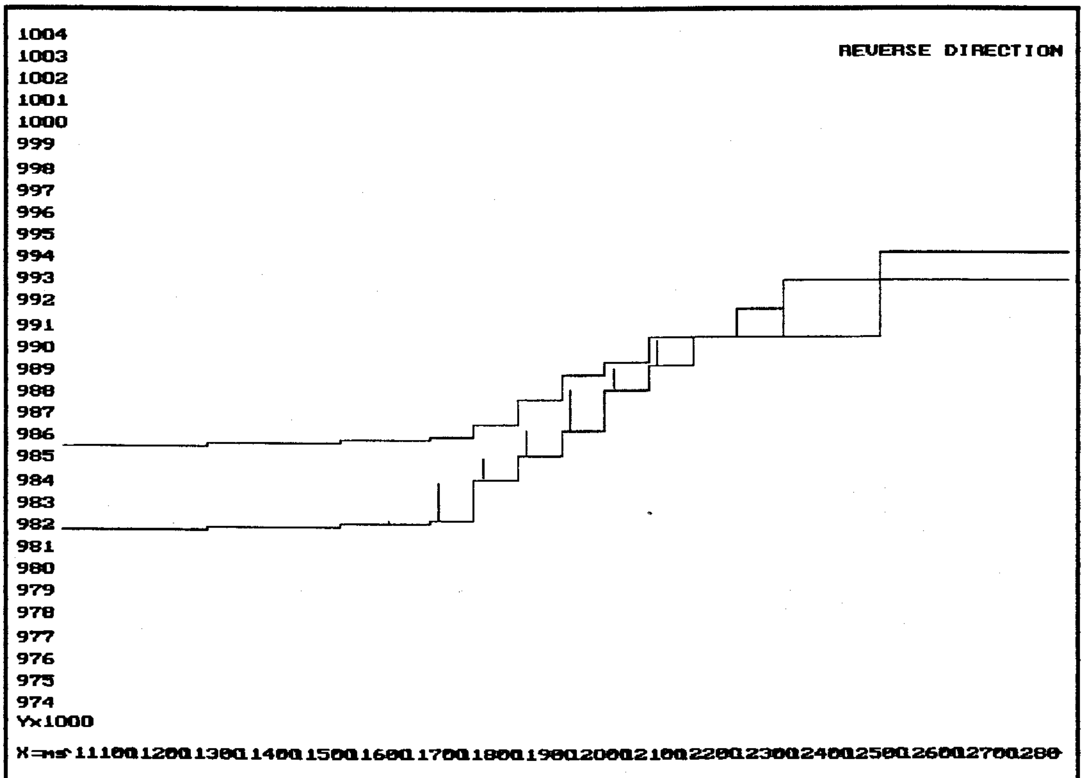


Figure 7.2 Time-sequence plot of trace in Appendix B.

Although this is normal and of no consequence in this trace, prolonged or frequent zero windows might point to a faulty station. This illustrates the ease of detecting and inspecting such occurrences, in context with the other packets in the conversation, using the plotting facility of **TCPlot**.

7.4 The effect of approximation

When scaling down packet sizes to enable **TCPlot** to display them on the screen, sequence numbers are divided by the scaling factor and rounded off to produce a whole number screen co-ordinate. This rounding off has the effect that when the screen co-ordinate is multiplied with the original scaling factor, the result will differ from the original sequence number.

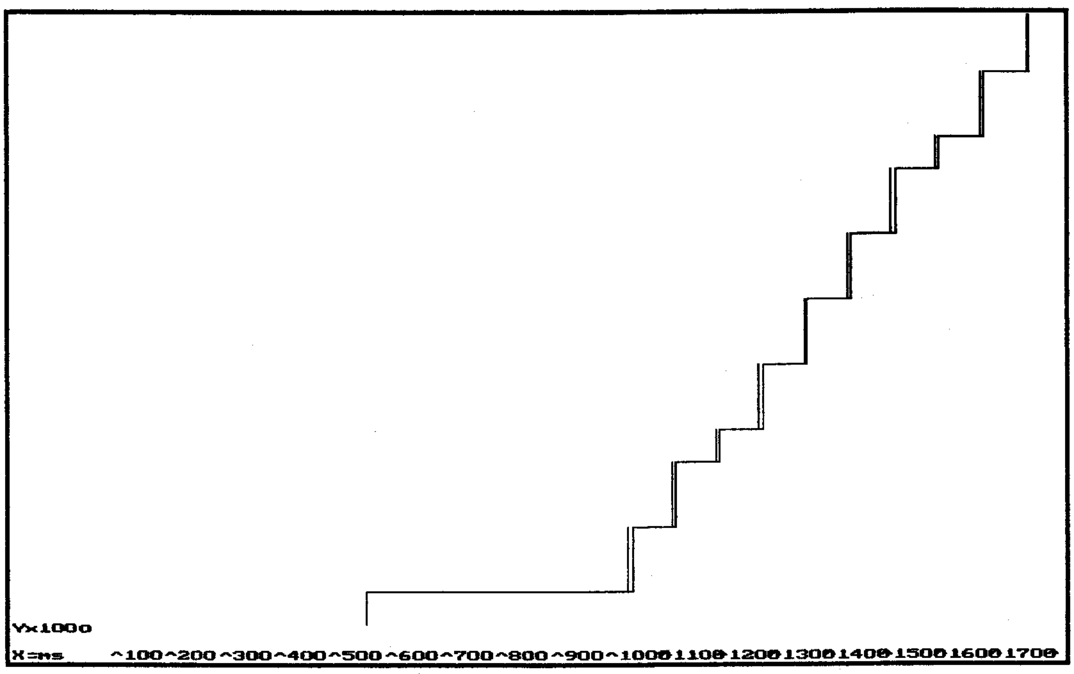


Figure 7.3 A Time-sequence plot enlarged in the correct way.

In **TCPlot** this has an effect when a plot is scaled down and then enlarged using the enlargement factor. When a time-sequence plot is therefore displayed as automatically scaled by **TCPlot** and the user wants to enlarge it, care must be taken to do this in the correct order. The scaling factor must be reduced first (Shift-F6) until a beep indicates that no further reduction is possible. Only then should the enlargement factor be used as shown in Figure 7.3. Figure 7.4 shows the effect of enlarging the time-sequence plot first. Here the rounded off co-ordinates has been multiplied by an enlargement factor resulting in a value that differs from that of the original sequence number to such an extent that packets are shown at half their size. As the scale is now reduced (Shift-F6) keeping the enlargement the same, packets and acknowledgements will vary between half and full size (due to rounding off).

Although the effect described above can cause confusion, it can be avoided by using the scale reduction first.

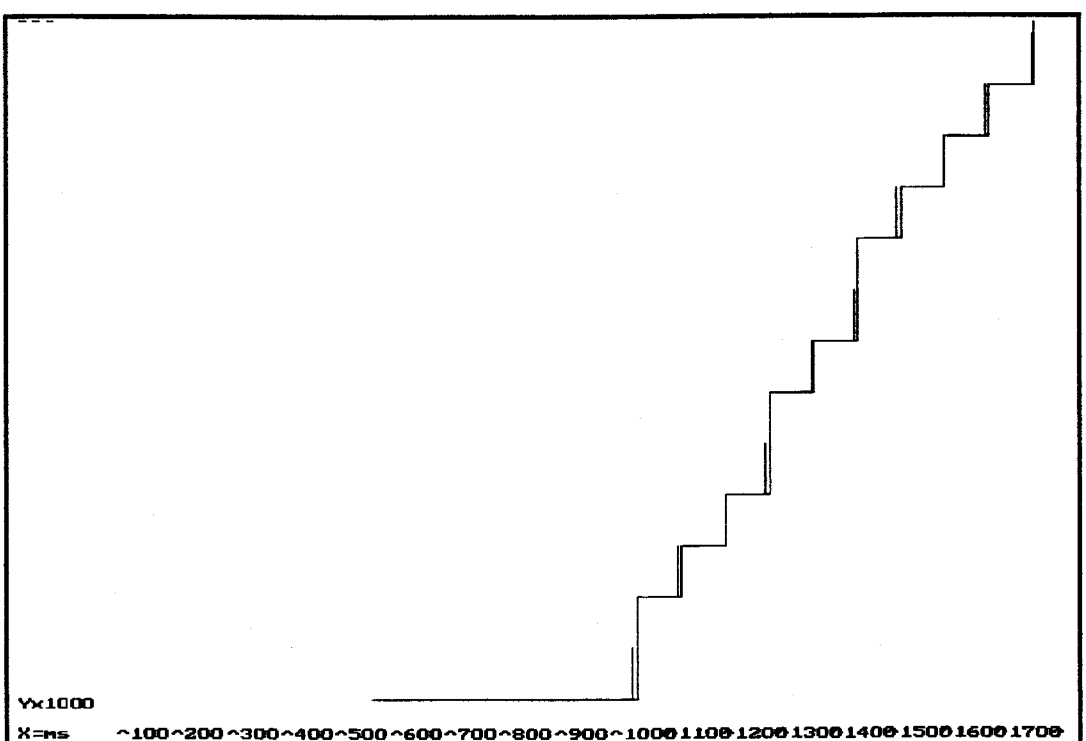


Figure 7.4 The effect of using the enlargement factor before reducing the scale

7.5 Interesting plots

During the testing period many time-sequence plots were inspected and checked against the traces. In this section a few interesting plots are discussed.

7.5.1 Packets outside the window

In Figure 7.5 the trace of a short conversation, taken from a fairly busy network, is shown. On the resulting time-sequence plot (Figure 7.6), the second packet shown is clearly outside the window. In the trace that is the packet that arrived at 56s 67603µs.

Sek	MicroS	Source IP	Dest IP	SPort	DPort	Seq	Ack	Win	DLen
49	226685	198.54.58.2	198.54.58.107	10283	4377	510856235	42571	8192	1024
49	229229	198.54.58.107	198.54.58.2	34377	1028	42571	510857259	1024	0
49	235082	198.54.58.107	198.54.58.2	34377	1028	42571	510858283	1024	0
49	238075	198.54.58.2	198.54.58.107	10283	4377	510858283	42571	8192	824
49	239921	198.54.58.107	198.54.58.2	34377	1028	42571	510859107	1024	0
56	47192	198.54.58.107	198.54.58.2	34377	1028	42571	510859107	1024	55
56	49118	198.54.58.2	198.54.58.107	10283	4377	510859107	42626	8192	49
56	57792	198.54.58.107	198.54.58.2	34377	1028	42626	510859156	1024	55
56	62696	198.54.58.2	198.54.58.107	10283	4377	510859156	42681	8192	1024
56	67603	198.54.58.2	198.54.58.107	10283	4377	510860180	42681	8192	474
56	85459	198.54.58.107	198.54.58.2	34377	1028	42681	510860654	1024	55
56	90504	198.54.58.2	198.54.58.107	10283	4377	510860654	42736	8192	1024
56	93080	198.54.58.107	198.54.58.2	34377	1028	42736	510861678	1024	0
56	94944	198.54.58.2	198.54.58.107	10283	4377	510861678	42736	8192	310
56	359311	198.54.58.107	198.54.58.2	34377	1028	42736	510861988	1024	0
59	166150	198.54.58.107	198.54.58.2	34377	1028	42736	510861988	1024	55
59	169861	198.54.58.2	198.54.58.107	10283	4377	510861988	42791	8192	697
59	172160	198.54.58.107	198.54.58.2	34377	1028	42791	510862685	1024	0

Figure 7.5 Filtered trace showing a packet arriving outside the window.

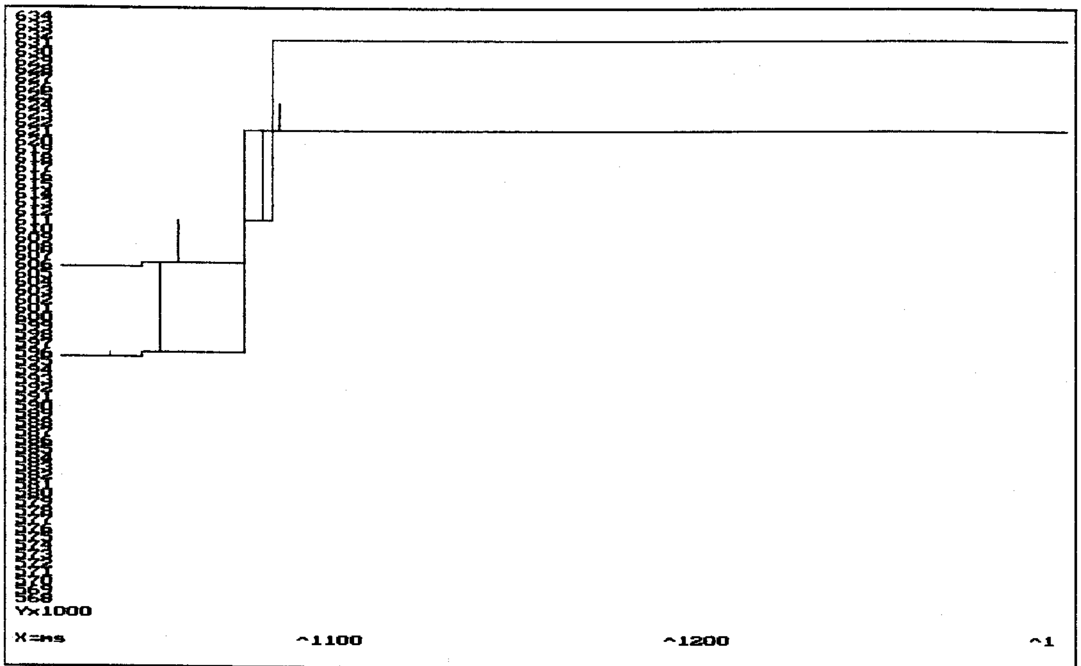


Figure 7.6 Time-sequence plot showing a packet outside the window.

On closer inspection it is clear that although this packet was sent outside the window, the receiver accepted the packet. This can be deducted from the acknowledgement number in the next packet. If we presume that the TCP implementations of both the receiver and sender are not at fault, the sender would not have sent and the receiver would not have accepted the packet outside of the offered window.

The only explanation therefore is that there was an acknowledgement packet from the receiver acknowledging the packet at 56s 62696μs and that this acknowledgement was received by the sender, thus allowing him to send the next packet. The fact that this packet apparently arrived at the sender but not at the **TCPlot** station, could be due to the fact that the sender and receiver were close to each other while the **TCPlot** station was monitoring the network on a distant segment. The packet could therefore have been received by the sender, but lost before it reached the

TCPlot station. The only other explanation would be that the packet arrived at the monitoring station, but this station failed to collect the packet from the network interface.

7.5.2 Normal time-sequence plots

A trace taken of a conversation whilst a large file (1 Mb) was copied to a file server from a slow PC-workstation produced the time-sequence plot shown in Figure 7.7.

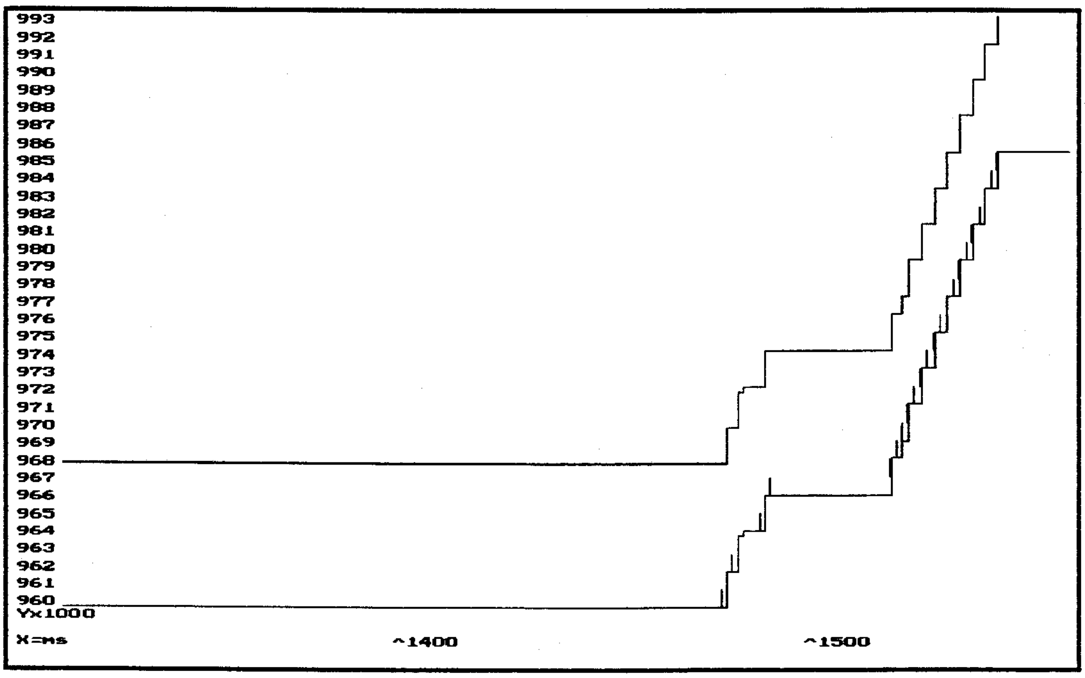


Figure 7.7 Time-sequence plot of a file copied from a slow PC to a file server.

The plot clearly shows that whilst the file server offers a large window (8K), the PC manages to send only two packets (1K each) before an acknowledgement is received.

Figure 7.8 shows the time-sequence plot of a trace taken whilst a large file was copied from a file server to a PC. Here it is clear that the PC offers a fairly small window and the file server transmits a full window of data at a time. The two horizontal steps in the figure shows that the PC delayed acknowledgement of packets on those occasions. This could have been caused by disk IO when data was written away to a file. The small packets in the beginning of the trace are the keystrokes to initiate the file transfer.

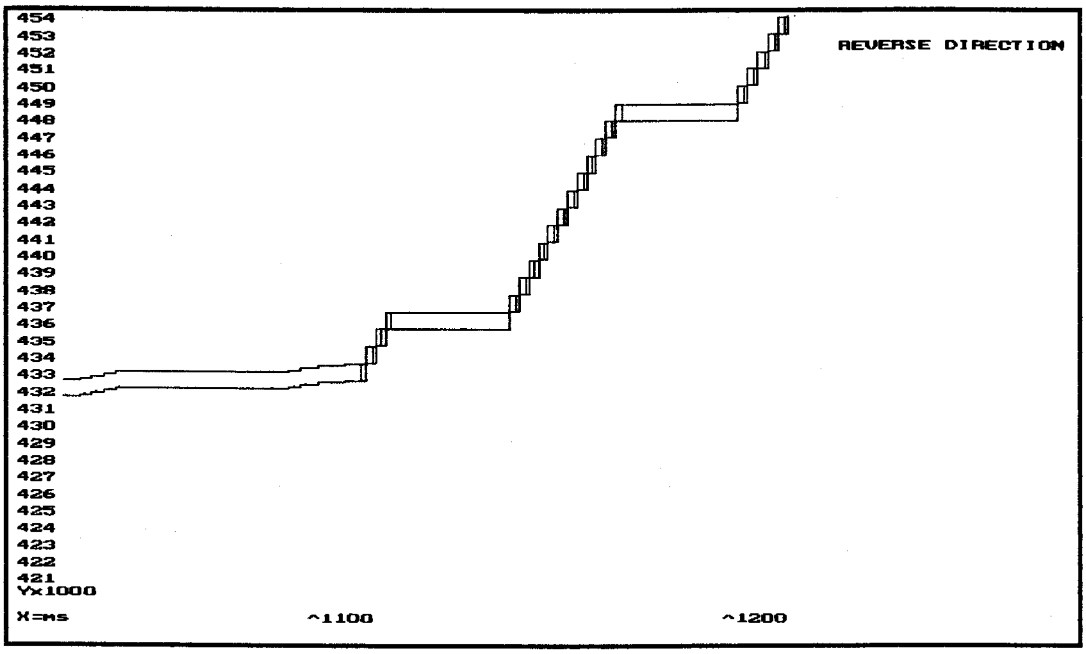


Figure 7.8 Time-sequence plot of a file copied from a file server to a PC.

7.5.3 Suspect time-sequence plots

While the plots as described under 7.5.2 are the normal plots expected, tests with a fast PC as workstation produced completely different results. The conversation when a large file was copied from a file server to a 486DX PC produced a plot similar to that in Figure 7.8. The conversation between a 486DX PC and a file server whilst a large file was copied from the PC to the file server, however, produced results as shown in Figure 7.9.

At first glance this pointed to a TCP implementation that was not functioning correctly. Similar results were, however, obtained with different TCP implementations (**NCSA**'s ftp and **LANMAN**). Inspection of the trace used to produce the plot in Figure 7.9 showed that the plot was a true reflection of the packets in the trace file. In the trace it can be seen that after a window of 8K was offered by the fileserver, up to sixteen 1K packets were transmitted by the PC before waiting for an acknowledgement.

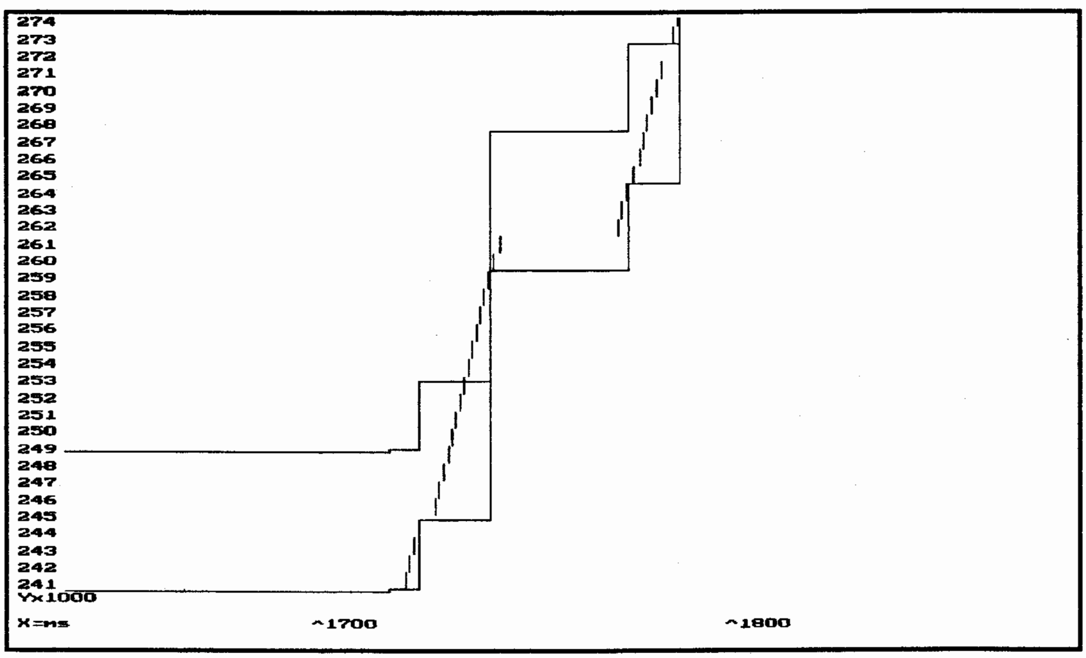


Figure 7.9 Time-sequence plot of a file copied from a 486DX PC to a file server

In Figure 7.9 it can be seen that an acknowledgement packet, offering a new window, should have been received before the Y-axis value (Sequence number) of 253. As there were no such acknowledgement in the trace file, together with the fact that several traces showed similar discrepancies, led the author to believe that packets were lost despite the reports by the packet driver as discussed under 7.2.1.

To determine the integrity of the trace file, the trace file was printed showing the values in the ID-field of each packet. As the ID-field values of packets from the PC to the file server can be expected to be sequential in this case, any missing numbers indicated packets not represented in the trace. Inspection of the trace showed that approximately 2% of the packets were not recorded in the trace and could therefore be seen as lost by the monitoring station.

The fact that packets were lost but not reported as lost by the packet driver, indicated that the packet driver might be at fault. Van Niekerk [Van Niekerk, 1994] confirmed that the loss of packets is eminent when using packet drivers in promiscuous mode on a busy network. In his experience in developing commercial network software, this loss can be reduced or eliminated by addressing the interface card directly. (**TCPlot** was originally developed to address the interface card directly but was modified to use packet drivers in order to enable testing with different interface cards.)

According to Van Niekerk, memory management software such as EMM386, also contributes to the loss of packets due to their utilisation of interrupts. To determine the effect of the EMM386 program, tests were done on the same monitoring station after removing EMM386. The resulting plot (Figure 7.10) showed a marked improvement

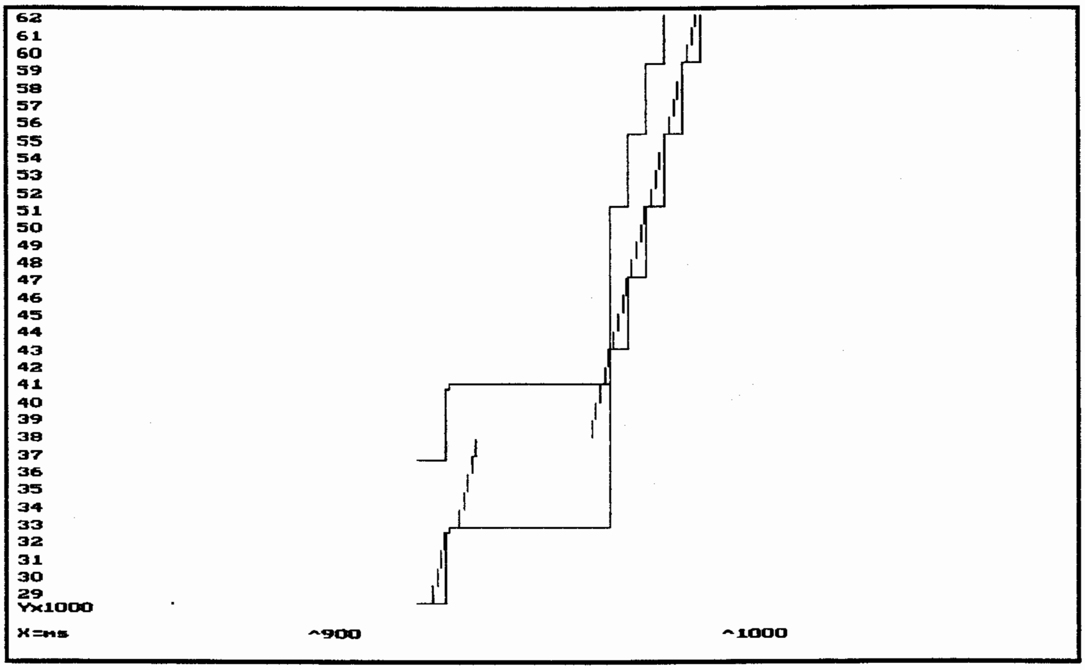


Figure 7.10 Time-sequence plot of a file copied from a 486DX PC to a file server after EMM386 has been removed from the monitor station.

Another possible explanation for the lost packets can be found in studies done at Xerox PARC by Westley Irish [Irish, 1994]. In his studies of the Ethernet network at Xerox PARC, Irish found that most network interface cards adhere to the specification of a 9.6 microsecond gap between frames (IFG) when transmitting normal packets. The same could, however, not be said when a card transmitted a packet directly after a collision was detected. He found that a number of interface card types allowed for a much too short IFG after a collision was detected, while others will inject a short signal burst into the network, thus damaging the IFG.

The net result of situations described above is that the receiving interface cards will not be able to retrieve packets, following a too short IFG, from the network. As the controller does not see the packet, it will not be reported as a lost packet, thereby causing \`\`undetected'' packet loss. According to Irish, this is particularly true for an interface card operating in promiscuous mode. Using a digital oscilloscope, Irish determined that with an Ethernet load of 25%, a packet loss of between 1% and 5% is possible.

Comparison between Figure 7.8 and Figure 7.9 shows that when the server was transmitting, all packets arrived safely, while when the workstation were transmitting, packets were lost. This leaves the possibility that a situation as described by Irish could be a role player here.

7.6 Getting the best results

When using **TCPlot** for the first time to display a conversation, the user may be confronted with a single line that does not seem to represent anything. In this section a few of these cases will be discussed.

An important fact to note is that when capturing traffic on a very quiet network, a single conversation consisting of one packet every second or so, can show up as containing 30 packets in the trace after 30 seconds. On a time sequence-plot, however, where the X-axis scale is in microseconds, two packets will have to be shown as arriving ten screens apart, thus showing a horizontal line. Extensive scaling would be required to show these packets on one screen. The normal conversations of TCP, where there will be several packets within microseconds of each other, however, will produce a workable initial plot.

Another cause for concern is that the number of packets shown on the plot seem less than that reported when analyses of the trace was done. Remember that the analyses also reports acknowledgements as well as zero length packets as belonging to the conversation. Although these packets have an effect on the acknowledgement and window lines, they will, for obvious reasons, not show up as packets on the plot.

A few initial plots and the actions required to scale them properly will be discussed below.

7.6.1 Flat horizontal plot

This might be the plot of acknowledgements only. If the acknowledging host has no data to send, only zero length packets will be sent, giving a horizontal line. With this plot on screen, press F2 (selecting reverse direction). Press enter again to display the reverse plot.

On the other hand it might be the plot of a quiet time in the conversation. Press the right arrow a few times and scroll through the conversation to see if there was any activity within the next period.

A nearly straight horizontal line (with small steps) might indicate a wrong scale. In this case the scale can be adjusted by pressing Shift-F6 a few times. If this results in a beep, enlarge the plot by pressing the up arrow.

7.6.2 Plot with only a vertical line

As explained earlier in the thesis, plots might start in the middle of a conversation, thus causing a faulty line at the beginning of the plot. A vertical line across the screen can be a manifestation of this. Scale the plot by pressing F6 until other lines appear and use the right arrow to scroll into the conversation

7.6.3 Scrolling through a conversation

To scroll through a conversation, press the right arrow. As sequence numbers increase, the plot will tend to disappear from the top of the screen. By alternatively pressing F6 and right arrow, the plot can be kept on screen, although smaller. Alternatively the F4 key can be used to start the plot later into the trace.

7.7 Areas for future research

TCPlot was developed to demonstrate the possibility to alleviate the task of the network manager with the use of graphics to display conversations. Determination of faulty conversations by **TCPlot**, however, rests mainly on the detection of duplicate packets.

An expert system developed to analyse conversations, using a set rule base, would complement the technique as described in this thesis and should be considered for further research.

Chapter 8

Conclusion

8.1 Conclusion

Development of **TCPlot** was started with the goal of using graphics to enable network managers to diagnose faulty TCP conversations. The basic steps required to reach this goal were listed in section 1.6.

Reaching the first of these milestones provided two options, namely using packet drivers or addressing the interface card directly. Technical information regarding the interface cards from different manufacturers, are hard to come by, yet this option was first explored and a network monitor was successfully developed for an 8-bit **ISOLAN** interface card.

Although this monitor performed well, this approach was abandoned and another monitor, using packet drivers, was developed. The need to evaluate and use the monitor on different networks with a variety of interface cards, prompted this change. For the same reason the final version of **TCPlot** uses the monitor with packet drivers. While the use of packet drivers had distinct advantages, later evaluation has shown possible disadvantages especially where lost packets are concerned. It can be concluded that with any future development of similar network tools, serious consideration should be given to addressing the interface card directly.

The high resolution timer described in section 4.3.2.3 proved effective although the running of **TCPlot** in automatic mode for prolonged time, produced strange timestamps. This is due to the resolution range selected when implementing the timer (see Table 4.1). Although this places a restraint on the use of **TCPlot**, normal use should not require monitoring sessions of more than an hour. If the monitor part is used on its own to display packets on screen (**Packet Display Mode**), however, **TCPlot** must be terminated and restarted hourly.

The strategy developed to identify problematic conversations depends on the detection of duplicate packets. While the duplicate packet count produced with **TCPlot** is not accurate, it serves the purpose of pointing out suspicious TCP conversations.

The last of the milestones were reached with the development of the analyser part of **TCPlot**. During evaluation, it was proven that by plotting a conversation on a time-sequence plot, **TCPlot** not only saves the network manager time, but makes it possible to detect patterns in the conversation easily. It was shown that to do the same from a trace file, would, if possible at all, require extensive analyses of the trace file.

Apart from the above, the ease with which **TCPlot** can identify faulty conversations, giving the percentage of duplicates for each conversation, also makes it valuable as a first line monitor and fault detection tool.

Although the loss of packets (see section 7.5.3) is cause for concern, refining the interface between **TCPlot** and the interface card should solve this problem. If, on the other hand, lost packets are caused by situations as described by Irish [Irish, 1994], the whole network will be suffering from this undetected lost packet syndrome. This could be the cause of throughput degradation and should be addressed.

8.2 Areas for future research

TCPlot was developed to demonstrate the possibility to alleviate the task of the network manager with the use of graphics to display conversations. Determination of faulty conversations by **TCPlot**, however, rests mainly on the detection of duplicate packets.

An expert system developed to analyse conversations, using a set rule base, would complement the technique as described in this thesis and should be considered for further research.

With the development of switching hubs to minimise traffic on Ethernet segments, a tool such as **TCPlot** may be isolated from problematic conversations. Future research can be directed towards developing remote stations to monitor the network on different segments. These remote stations could be SNMP manageable to make the monitoring from a central station possible.

Future research could also be directed towards the development of tools to represent other traffic of protocols where applicable.

Appendix A

Trace of all packets in Fintest.cap

Sek	MicroS	Source IP	Dest IP	SPort	DPort	Seq	Ack	Win	DLen
46	520607	198.54.58.13	198.54.58.2	6041	23	97220939	90351867	512	1
46	586391	198.54.58.12	198.54.58.2	5580	23	96017216	682910547	384	0
46	6298	198.54.58.2	198.54.58.12	23	5580	682910547	96017216	8192	65
46	16704	198.54.58.2	198.54.58.12	23	5580	682910612	96017216	8192	162
47	76169	198.54.58.2	198.54.58.12	23	5580	682910774	96017216	8192	24
47	186325	198.54.58.12	198.54.58.2	5580	23	96017216	682910798	384	0
47	330902	198.54.58.12	198.54.58.2	5632	23	97333781	219471615	384	1
47	336086	198.54.58.2	198.54.58.12	23	5632	219471615	97333782	8192	1
47	409863	198.54.58.12	198.54.58.2	5632	23	97333782	219471616	384	1
47	416097	198.54.58.2	198.54.58.12	23	5632	219471616	97333783	8192	1
47	587353	198.54.58.12	198.54.58.2	5632	23	97333783	219471617	384	0
47	596267	198.54.58.2	198.54.58.12	23	5580	682910798	96017216	8192	68
47	676304	198.54.58.2	198.54.58.12	23	5580	682910866	96017216	8192	22
47	730475	198.54.58.12	198.54.58.2	5626	23	97044397	2033906804	384	1
47	732185	198.54.58.12	198.54.58.2	5628	23	97257532	119757789	384	1
47	736158	198.54.58.2	198.54.58.12	23	5628	119757789	97257533	8192	1
47	786503	198.54.58.12	198.54.58.2	5580	23	96017216	682910888	384	0
47	811216	198.54.58.12	198.54.58.2	5626	23	97044398	2033906804	384	2
47	816082	198.54.58.2	198.54.58.12	23	5626	2033906804	97044400	8192	8
47	896742	198.54.58.12	198.54.58.2	5628	23	97257533	119757790	384	1
47	898451	198.54.58.12	198.54.58.2	3308	2907	42131138	910247824	0	1
47	899471	198.54.58.2	198.54.58.12	2907	3308	910247824	42131139	8192	0
47	901870	198.54.58.12	198.54.58.2	3309	1832	44255417	923763643	384	1
47	902878	198.54.58.2	198.54.58.12	1832	3309	923763643	44255418	8192	0
47	906036	198.54.58.2	198.54.58.12	23	5628	119757790	97257534	8192	1
47	971032	198.54.58.12	198.54.58.2	5626	23	97044400	2033906812	384	3
47	976147	198.54.58.2	198.54.58.12	23	5626	2033906812	97044403	8192	8
48	87822	198.54.58.12	198.54.58.2	5628	23	97257534	119757791	384	0
48	370366	198.54.58.12	198.54.58.2	5626	23	97044403	2033906820	384	3
48	376138	198.54.58.2	198.54.58.12	23	5626	2033906820	97044406	8192	3
48	487088	198.54.58.12	198.54.58.2	5626	23	97044406	2033906823	384	0
48	529857	198.54.58.12	198.54.58.2	5632	23	97333783	219471617	384	1
48	536145	198.54.58.2	198.54.58.12	23	5632	219471617	97333784	8192	1
48	690926	198.54.58.12	198.54.58.2	5626	23	97044406	2033906823	384	3
48	692636	198.54.58.12	198.54.58.2	5632	23	97333784	219471618	384	0
48	696200	198.54.58.2	198.54.58.12	23	5626	2033906823	97044409	8192	3
48	891470	198.54.58.12	198.54.58.2	5626	23	97044409	2033906826	384	0
49	90362	198.54.58.12	198.54.58.2	5626	23	97044409	2033906826	384	3
49	96228	198.54.58.2	198.54.58.12	23	5626	2033906826	97044412	8192	3
49	287546	198.54.58.12	198.54.58.2	5626	23	97044412	2033906829	384	0
49	326443	198.54.58.2	198.54.58.13	23	6050	306562434	97432524	8192	7
49	336634	198.54.58.2	198.54.58.13	23	6050	306562441	97432524	8192	129
49	347083	198.54.58.2	198.54.58.13	23	6050	306562570	97432524	8192	240
49	356413	198.54.58.2	198.54.58.13	23	6050	306562810	97432524	8192	42
49	410778	198.54.58.12	198.54.58.2	5579	23	96015081	683097129	384	1
49	416298	198.54.58.2	198.54.58.12	23	5579	683097129	96015082	8192	1
49	426338	198.54.58.2	198.54.58.13	23	6050	306562852	97432524	8192	22
49	491384	198.54.58.12	198.54.58.2	5632	23	97333784	219471618	384	1
49	496264	198.54.58.2	198.54.58.12	23	5632	219471618	97333785	8192	1
49	519430	198.54.58.13	198.54.58.2	6050	23	97432524	306562874	230	0
49	570332	198.54.58.12	198.54.58.2	5579	23	96015082	683097130	384	1
49	576272	198.54.58.2	198.54.58.12	23	5579	683097130	96015083	8192	1
49	687064	198.54.58.12	198.54.58.2	5632	23	97333785	219471619	384	0
49	688777	198.54.58.12	198.54.58.2	5579	23	96015083	683097131	384	0
49	729897	198.54.58.12	198.54.58.2	5579	23	96015083	683097131	384	1
49	736272	198.54.58.2	198.54.58.12	23	5579	683097131	96015084	8192	1

49	839347	198.54.58.13	198.54.58.2	6050	23	97432524	306562874	512	0
49	895883	198.54.58.12	198.54.58.2	5579	23	96015084	683097132	384	0
49	969860	198.54.58.12	198.54.58.2	5579	23	96015084	683097132	384	1
49	973213	198.54.58.12	198.54.58.2	5626	23	97044412	2033906829	384	1
49	976317	198.54.58.2	198.54.58.12	23	5579	683097132	96015085	8192	1
50	16302	198.54.58.2	198.54.58.12	23	5626	2033906829	97044413	8192	0
50	50587	198.54.58.12	198.54.58.2	5628	23	97257534	119757791	384	1
50	56316	198.54.58.2	198.54.58.12	23	5628	119757791	97257535	8192	1
50	86841	198.54.58.12	198.54.58.2	5579	23	96015085	683097133	384	0
50	187089	198.54.58.12	198.54.58.2	5628	23	97257535	119757792	384	0
50	371078	198.54.58.12	198.54.58.2	5626	23	97044413	2033906829	384	1
50	372792	198.54.58.12	198.54.58.2	5632	23	97333785	219471619	384	1
50	376417	198.54.58.2	198.54.58.12	23	5632	219471619	97333786	8192	1
50	376816	198.54.58.2	198.54.58.12	23	5626	2033906829	97044414	8192	1
50	450167	198.54.58.12	198.54.58.2	5626	23	97044414	2033906830	384	2
50	453520	198.54.58.12	198.54.58.2	5632	23	97333786	219471620	384	1
50	456394	198.54.58.2	198.54.58.12	23	5632	219471620	97333787	8192	1
50	456783	198.54.58.2	198.54.58.12	23	5626	2033906830	97044416	8192	2
50	530892	198.54.58.12	198.54.58.2	5626	23	97044416	2033906832	384	2
50	536375	198.54.58.2	198.54.58.12	23	5626	2033906832	97044418	8192	2
50	586860	198.54.58.12	198.54.58.2	5632	23	97333787	219471621	384	0
50	609922	198.54.58.12	198.54.58.2	5579	23	96015085	683097133	384	1
50	613275	198.54.58.12	198.54.58.2	5626	23	97044418	2033906834	384	2
50	616481	198.54.58.2	198.54.58.12	23	5626	2033906834	97044420	8192	2
50	616893	198.54.58.2	198.54.58.12	23	5579	683097133	96015086	8192	3
50	787545	198.54.58.12	198.54.58.2	5626	23	97044420	2033906836	384	0
50	789258	198.54.58.12	198.54.58.2	5579	23	96015086	683097136	384	0
50	850084	198.54.58.12	198.54.58.2	5628	23	97257535	119757792	384	1
50	856481	198.54.58.2	198.54.58.12	23	5628	119757792	97257536	8192	1
50	965531	198.54.58.13	198.54.58.2	3309	1903	52259942	1636621444	512	1
50	966599	198.54.58.2	198.54.58.13	1903	3309	1636621444	52259943	8190	0
50	988157	198.54.58.12	198.54.58.2	5628	23	97257536	119757793	384	0
51	116491	198.54.58.2	198.54.58.12	23	5626	2033906836	97044420	8192	1
51	251044	198.54.58.12	198.54.58.2	5626	23	97044420	2033906837	384	1
51	410401	198.54.58.12	198.54.58.2	5626	23	97044421	2033906837	384	1
51	416481	198.54.58.2	198.54.58.12	23	5626	2033906837	97044422	8192	0
51	571461	198.54.58.12	198.54.58.2	5579	23	96015086	683097136	384	1
51	576545	198.54.58.2	198.54.58.12	23	5579	683097136	96015087	8192	1
51	688184	198.54.58.12	198.54.58.2	5579	23	96015087	683097137	384	0
51	896819	198.54.58.12	198.54.58.2	5628	23	97257536	119757793	384	1
51	906643	198.54.58.2	198.54.58.12	23	5628	119757793	97257537	8192	1
51	964509	198.54.58.13	198.54.58.255	513	513	4456448	16842752	13548	48
52	51402	198.54.58.12	198.54.58.2	5579	23	96015087	683097137	384	1
52	56611	198.54.58.2	198.54.58.12	23	5579	683097137	96015088	8192	1
52	87661	198.54.58.12	198.54.58.2	5628	23	97257537	119757794	384	0
52	130429	198.54.58.12	198.54.58.2	5626	23	97044422	2033906837	384	3
52	133783	198.54.58.12	198.54.58.2	5628	23	97257537	119757794	384	1
52	136593	198.54.58.2	198.54.58.12	23	5628	119757794	97257538	8192	1
52	136993	198.54.58.2	198.54.58.12	23	5626	2033906837	97044425	8192	1
52	188168	198.54.58.12	198.54.58.2	5579	23	96015088	683097138	384	0
52	291697	198.54.58.12	198.54.58.2	5628	23	97257538	119757795	384	1
52	293410	198.54.58.12	198.54.58.2	5626	23	97044425	2033906838	384	0
52	296672	198.54.58.2	198.54.58.12	23	5628	119757795	97257539	8192	1
52	370724	198.54.58.12	198.54.58.2	5626	23	97044425	2033906838	384	3
52	376622	198.54.58.2	198.54.58.12	23	5626	2033906838	97044428	8192	1
52	487464	198.54.58.12	198.54.58.2	5628	23	97257539	119757796	384	0
52	489154	198.54.58.12	198.54.58.2	5626	23	97044428	2033906839	384	0
52	610753	198.54.58.12	198.54.58.2	5626	23	97044428	2033906839	384	3
52	616663	198.54.58.2	198.54.58.12	23	5626	2033906839	97044431	8192	1
52	691347	198.54.58.12	198.54.58.2	5626	23	97044431	2033906840	384	3
52	696681	198.54.58.2	198.54.58.12	23	5626	2033906840	97044434	8192	5
52	720201	198.54.58.13	198.54.58.2	6041	23	97220947	90351875	512	1
52	816654	198.54.58.2	198.54.58.13	23	6041	90351875	97220948	8192	0
52	850767	198.54.58.12	198.54.58.2	5626	23	97044434	2033906845	384	3
52	856677	198.54.58.2	198.54.58.12	23	5626	2033906845	97044437	8192	5
52	931365	198.54.58.12	198.54.58.2	5579	23	96015088	683097138	384	1
52	933088	198.54.58.12	198.54.58.2	5580	23	96017216	682910888	384	1
52	936702	198.54.58.2	198.54.58.12	23	5580	682910888	96017217	8192	1
52	937102	198.54.58.2	198.54.58.12	23	5579	683097138	96015089	8192	1
52	987473	198.54.58.12	198.54.58.2	5626	23	97044437	2033906850	384	0
53	90999	198.54.58.12	198.54.58.2	5579	23	96015089	683097139	384	1
53	92708	198.54.58.12	198.54.58.2	5580	23	96017217	682910889	384	0

53	96696	198.54.58.2	198.54.58.12	23	5579	683097139	96015090	8192	1
53	200266	198.54.58.13	198.54.58.2	6041	23	97220948	90351875	512	1
53	206738	198.54.58.2	198.54.58.13	23	6041	90351875	97220949	8192	1
53	288257	198.54.58.12	198.54.58.2	5579	23	96015090	683097140	384	0
53	364310	198.54.58.13	198.54.58.2	6041	23	97220949	90351876	512	1
53	366826	198.54.58.2	198.54.58.13	23	6041	90351876	97220950	8192	13
53	446872	198.54.58.2	198.54.58.13	23	6041	90351889	97220950	8192	37
53	516850	198.54.58.2	198.54.58.13	23	6041	90351926	97220950	8192	7
53	600014	198.54.58.13	198.54.58.2	6041	23	97220950	90351933	512	0
53	626881	198.54.58.2	198.54.58.13	23	6041	90351933	97220950	8192	7
53	839904	198.54.58.13	198.54.58.2	6041	23	97220950	90351940	512	0
53	895933	198.54.58.12	198.54.58.2	5626	23	97044437	2033906850	384	1
53	917137	198.54.58.2	198.54.58.12	23	5626	2033906850	97044438	8192	109
54	88286	198.54.58.12	198.54.58.2	5626	23	97044438	2033906959	384	0
54	480461	198.54.58.13	198.54.58.2	6041	23	97220950	90351940	512	1
54	486948	198.54.58.2	198.54.58.13	23	6041	90351940	97220951	8192	1
54	691023	198.54.58.12	198.54.58.2	5580	23	96017217	682910889	384	1
54	696967	198.54.58.2	198.54.58.12	23	5580	682910889	96017218	8192	1
54	720573	198.54.58.13	198.54.58.2	6041	23	97220951	90351941	512	1
54	726992	198.54.58.2	198.54.58.13	23	6041	90351941	97220952	8192	1
54	800627	198.54.58.13	198.54.58.2	6041	23	97220952	90351942	512	1
54	806921	198.54.58.2	198.54.58.13	23	6041	90351942	97220953	8192	1
54	852087	198.54.58.12	198.54.58.2	5632	23	97333787	219471621	384	1
54	856934	198.54.58.2	198.54.58.12	23	5632	219471621	97333788	8192	1
54	893281	198.54.58.12	198.54.58.2	5580	23	96017218	682910890	384	0
54	988596	198.54.58.12	198.54.58.2	5632	23	97333788	219471622	384	0
54	40075	198.54.58.13	198.54.58.2	6041	23	97220953	90351943	512	0
55	92120	198.54.58.12	198.54.58.2	5580	23	96017218	682910890	384	1
55	96944	198.54.58.2	198.54.58.12	23	5580	682910890	96017219	8192	1
55	200552	198.54.58.13	198.54.58.2	6041	23	97220953	90351943	512	1
55	207028	198.54.58.2	198.54.58.13	23	6041	90351943	97220954	8192	1
55	287665	198.54.58.12	198.54.58.2	5580	23	96017219	682910891	384	0
55	410908	198.54.58.12	198.54.58.2	5580	23	96017219	682910891	384	1
55	417055	198.54.58.2	198.54.58.12	23	5580	682910891	96017220	8192	1
55	440129	198.54.58.13	198.54.58.2	6041	23	97220954	90351944	512	0
55	491495	198.54.58.12	198.54.58.2	5634	23	97430456	299397593	384	3
55	497125	198.54.58.2	198.54.58.12	23	5634	299397593	97430459	8192	43
55	572121	198.54.58.12	198.54.58.2	5632	23	97333788	219471622	384	1
55	577034	198.54.58.2	198.54.58.12	23	5632	219471622	97333789	8192	1
55	588681	198.54.58.12	198.54.58.2	5580	23	96017220	682910892	384	0
55	651158	198.54.58.12	198.54.58.2	5579	23	96015090	683097140	384	1
55	654509	198.54.58.12	198.54.58.2	5626	23	97044438	2033906959	384	1
55	657082	198.54.58.2	198.54.58.12	23	5579	683097140	96015091	8192	1
55	687482	198.54.58.12	198.54.58.2	5634	23	97430459	299397636	384	0
55	689195	198.54.58.12	198.54.58.2	5632	23	97333789	219471623	384	0
55	787802	198.54.58.12	198.54.58.2	5579	23	96015091	683097141	384	0
55	817021	198.54.58.2	198.54.58.12	23	5626	2033906959	97044439	8192	0
55	920662	198.54.58.13	198.54.58.2	6041	23	97220954	90351944	512	1
55	927160	198.54.58.2	198.54.58.13	23	6041	90351944	97220955	8192	1
55	971851	198.54.58.12	198.54.58.2	5580	23	96017220	682910892	384	1
55	973563	198.54.58.12	198.54.58.2	5626	23	97044439	2033906959	384	1
55	977005	198.54.58.12	198.54.58.2	5634	23	97430459	299397636	384	4
55	987179	198.54.58.2	198.54.58.12	23	5634	299397636	97430463	8192	12
55	987621	198.54.58.2	198.54.58.12	23	5626	2033906959	97044440	8192	36
56	51094	198.54.58.12	198.54.58.2	5579	23	96015091	683097141	384	1
56	54458	198.54.58.12	198.54.58.2	5634	23	97430463	299397648	384	7
56	57160	198.54.58.2	198.54.58.12	23	5634	299397648	97430470	8192	17
56	57578	198.54.58.2	198.54.58.12	23	5579	683097141	96015092	8192	1
56	67184	198.54.58.2	198.54.58.12	23	5634	299397665	97430470	8192	35
56	87578	198.54.58.12	198.54.58.2	5580	23	96017221	682910893	384	0
56	117596	198.54.58.2	198.54.58.12	2907	3308	910247824	42131139	8192	1
56	132048	198.54.58.12	198.54.58.2	5632	23	97333789	219471623	384	1
56	133771	198.54.58.12	198.54.58.2	5634	23	97430470	299397700	384	7
56	137204	198.54.58.2	198.54.58.12	23	5632	219471623	97333790	8192	1
56	147636	198.54.58.2	198.54.58.12	23	5634	299397700	97430477	8192	153
56	164251	198.54.58.13	198.54.58.2	6041	23	97220955	90351945	512	0
56	188270	198.54.58.12	198.54.58.2	5626	23	97044440	2033906995	384	0
56	189982	198.54.58.12	198.54.58.2	5579	23	96015092	683097142	384	0
56	211406	198.54.58.12	198.54.58.2	5626	23	97044440	2033906995	384	1
56	214760	198.54.58.12	198.54.58.2	5634	23	97430477	299397853	384	3
56	217227	198.54.58.2	198.54.58.12	23	5626	2033906995	97044441	8192	37
56	217582	198.54.58.2	198.54.58.12	23	5634	299397853	97430480	8192	0

56	227294	198.54.58.2	198.54.58.12	23	5634	299397853	97430480	8192	49
56	240706	198.54.58.13	198.54.58.2	6041	23	97220955	90351945	512	1
56	247102	198.54.58.2	198.54.58.13	23	6041	90351945	97220956	8192	1
56	292256	198.54.58.12	198.54.58.2	5579	23	96015092	683097142	384	1
56	293969	198.54.58.12	198.54.58.2	5634	23	97430480	299397902	384	6
56	297226	198.54.58.2	198.54.58.12	23	5579	683097142	96015093	8192	1
56	299098	198.54.58.12	198.54.58.2	5632	23	97333790	219471624	384	0
56	307395	198.54.58.2	198.54.58.12	23	5634	299397902	97430486	8192	104
56	391272	198.54.58.12	198.54.58.2	5626	23	97044441	2033907032	384	0
56	480746	198.54.58.13	198.54.58.2	6041	23	97220956	90351946	512	1
56	487109	198.54.58.2	198.54.58.13	23	6041	90351946	97220957	8192	1
56	488234	198.54.58.12	198.54.58.2	5579	23	96015093	683097143	384	0
56	489946	198.54.58.12	198.54.58.2	5634	23	97430486	299398006	384	0
56	531075	198.54.58.12	198.54.58.2	5580	23	96017221	682910893	384	1
56	537132	198.54.58.2	198.54.58.12	23	5580	682910893	96017222	8192	1
56	611672	198.54.58.12	198.54.58.2	5579	23	96015093	683097143	384	1
56	617209	198.54.58.2	198.54.58.12	23	5579	683097143	96015094	8192	1
56	692268	198.54.58.12	198.54.58.2	5632	23	97333790	219471624	384	1
56	693981	198.54.58.12	198.54.58.2	5580	23	96017222	682910894	384	0
56	697154	198.54.58.2	198.54.58.12	23	5632	219471624	97333791	8192	1
56	720314	198.54.58.13	198.54.58.2	6041	23	97220957	90351947	512	0
56	771288	198.54.58.12	198.54.58.2	5579	23	96015094	683097144	384	1
56	777182	198.54.58.2	198.54.58.12	23	5579	683097144	96015095	8192	1
56	800868	198.54.58.13	198.54.58.2	6041	23	97220957	90351947	512	1
56	807151	198.54.58.2	198.54.58.13	23	6041	90351947	97220958	8192	1
56	851888	198.54.58.12	198.54.58.2	5632	23	97333791	219471625	384	1
56	857232	198.54.58.2	198.54.58.12	23	5632	219471625	97333792	8192	1
56	893075	198.54.58.12	198.54.58.2	5579	23	96015095	683097145	384	0
56	988387	198.54.58.12	198.54.58.2	5632	23	97333792	219471626	384	0
57	40840	198.54.58.13	198.54.58.2	6041	23	97220958	90351948	512	2
57	48097	198.54.58.2	198.54.58.13	23	6041	90351948	97220960	8192	240
57	57290	198.54.58.2	198.54.58.13	23	6041	90352188	97220960	8192	43
57	172381	198.54.58.12	198.54.58.2	5626	23	97044441	2033907032	384	3
57	187321	198.54.58.2	198.54.58.12	23	5626	2033907032	97044444	8192	3
57	280994	198.54.58.13	198.54.58.2	6041	23	97220960	90352231	426	1
57	287243	198.54.58.2	198.54.58.13	23	6041	90352231	97220961	8192	1
57	331810	198.54.58.12	198.54.58.2	5580	23	96017222	682910894	384	1
57	337290	198.54.58.2	198.54.58.12	23	5580	682910894	96017223	8192	1
57	391055	198.54.58.12	198.54.58.2	5626	23	97044444	2033907035	384	0
57	441030	198.54.58.13	198.54.58.2	6041	23	97220961	90352232	512	2
57	448097	198.54.58.2	198.54.58.13	23	6041	90352232	97220963	8192	240
57	457757	198.54.58.2	198.54.58.13	23	6041	90352472	97220963	8192	144
57	491293	198.54.58.12	198.54.58.2	5580	23	96017223	682910895	384	0
57	571839	198.54.58.12	198.54.58.2	5634	23	97430486	299398006	384	3
57	577471	198.54.58.2	198.54.58.12	23	5634	299398006	97430489	8192	47
57	650822	198.54.58.12	198.54.58.2	5626	23	97044444	2033907035	384	3
57	657322	198.54.58.2	198.54.58.12	23	5626	2033907035	97044447	8192	3
57	680588	198.54.58.13	198.54.58.2	6041	23	97220963	90352616	323	0
57	688716	198.54.58.12	198.54.58.2	5634	23	97430489	299398053	384	0
57	731485	198.54.58.12	198.54.58.2	5579	23	96015095	683097145	384	1
57	734848	198.54.58.12	198.54.58.2	5626	23	97044447	2033907038	384	7
57	737342	198.54.58.2	198.54.58.12	23	5579	683097145	96015096	8192	3
57	747359	198.54.58.2	198.54.58.12	23	5626	2033907038	97044454	8192	6
57	812224	198.54.58.12	198.54.58.2	5626	23	97044454	2033907044	384	5
57	817416	198.54.58.2	198.54.58.12	23	5626	2033907044	97044459	8192	3
57	827376	198.54.58.2	198.54.58.12	23	5626	2033907047	97044459	8192	3
57	897807	198.54.58.12	198.54.58.2	5579	23	96015096	683097148	384	1
57	899521	198.54.58.12	198.54.58.2	3310	4709	51456317	673009922	384	1
57	900612	198.54.58.2	198.54.58.12	4709	3310	673009922	51456318	8192	0
57	907360	198.54.58.2	198.54.58.12	23	5579	683097148	96015097	8192	3
57	971965	198.54.58.12	198.54.58.2	5634	23	97430489	299398053	384	3
57	977461	198.54.58.2	198.54.58.12	23	5634	299398053	97430492	8192	49
57	988550	198.54.58.12	198.54.58.2	5626	23	97044459	2033907050	384	0
58	51026	198.54.58.12	198.54.58.2	5579	23	96015097	683097151	384	1
58	54379	198.54.58.12	198.54.58.2	5632	23	97333792	219471626	384	1
58	57453	198.54.58.2	198.54.58.12	23	5579	683097151	96015098	8192	3
58	67631	198.54.58.2	198.54.58.12	23	5634	299398102	97430498	8192	109
58	131908	198.54.58.12	198.54.58.2	5580	23	96017223	682910895	384	1
58	133620	198.54.58.12	198.54.58.2	5632	23	97333793	219471626	384	2
58	136974	198.54.58.12	198.54.58.2	5634	23	97430498	299398211	384	6
58	137748	198.54.58.2	198.54.58.12	23	5632	219471626	97333795	8192	3
58	138146	198.54.58.2	198.54.58.12	23	5580	682910895	96017224	8192	1

58	147737	198.54.58.2	198.54.58.12	23	5634	299398211	97430504	8192	117
58	164531	198.54.58.13	198.54.58.2	6041	23	97220963	90352616	512	0
58	188227	198.54.58.12	198.54.58.2	5579	23	96015098	683097154	384	0
58	212925	198.54.58.12	198.54.58.2	5634	23	97430504	299398328	384	3
58	217400	198.54.58.2	198.54.58.12	23	5634	299398328	97430507	8192	12
58	291892	198.54.58.12	198.54.58.2	5632	23	97333795	219471629	384	3
58	295244	198.54.58.12	198.54.58.2	5634	23	97430507	299398340	384	6
58	296958	198.54.58.12	198.54.58.2	5580	23	96017224	682910896	384	0
58	307554	198.54.58.2	198.54.58.12	23	5634	299398340	97430513	8192	61
58	372723	198.54.58.12	198.54.58.2	5632	23	97333798	219471632	384	3
58	374446	198.54.58.12	198.54.58.2	5634	23	97430513	299398401	384	6
58	377455	198.54.58.2	198.54.58.12	23	5634	299398401	97430519	8192	14
58	387593	198.54.58.2	198.54.58.12	23	5634	299398415	97430519	8192	51
58	387936	198.54.58.2	198.54.58.12	23	5632	219471632	97333801	8192	3
58	451919	198.54.58.12	198.54.58.2	5634	23	97430519	299398466	384	3
58	457508	198.54.58.2	198.54.58.12	23	5634	299398466	97430522	8192	46
58	532547	198.54.58.12	198.54.58.2	5632	23	97333801	219471635	384	3
58	537508	198.54.58.2	198.54.58.12	23	5632	219471635	97333804	8192	3
58	588511	198.54.58.12	198.54.58.2	5634	23	97430522	299398512	384	0
58	611575	198.54.58.12	198.54.58.2	5580	23	96017224	682910896	384	1
58	617439	198.54.58.2	198.54.58.12	23	5580	682910896	96017225	8192	1
58	692169	198.54.58.12	198.54.58.2	5632	23	97333804	219471638	384	3
58	697474	198.54.58.2	198.54.58.12	23	5632	219471638	97333807	8192	3
58	789187	198.54.58.12	198.54.58.2	5580	23	96017225	682910897	384	0
58	894355	198.54.58.12	198.54.58.2	5632	23	97333807	219471641	384	0
58	932201	198.54.58.12	198.54.58.2	5580	23	96017225	682910897	384	1
58	937443	198.54.58.2	198.54.58.12	23	5580	682910897	96017226	8192	1
59	91618	198.54.58.12	198.54.58.2	5580	23	96017226	682910898	384	0
59	172154	198.54.58.12	198.54.58.2	5579	23	96015098	683097154	384	1
59	177488	198.54.58.2	198.54.58.12	23	5579	683097154	96015099	8192	1
59	288880	198.54.58.12	198.54.58.2	5579	23	96015099	683097155	384	0
59	412110	198.54.58.12	198.54.58.2	5579	23	96015099	683097155	384	1
59	417523	198.54.58.2	198.54.58.12	23	5579	683097155	96015100	8192	1
59	492706	198.54.58.12	198.54.58.2	5580	23	96017226	682910898	384	1
59	497542	198.54.58.2	198.54.58.12	23	5580	682910898	96017227	8192	1
59	588093	198.54.58.12	198.54.58.2	5579	23	96015100	683097156	384	0
59	652201	198.54.58.12	198.54.58.2	5579	23	96015100	683097156	384	1
59	653913	198.54.58.12	198.54.58.2	5632	23	97333807	219471641	384	1
59	657585	198.54.58.2	198.54.58.12	23	5632	219471641	97333808	8192	11
59	657986	198.54.58.2	198.54.58.12	23	5579	683097156	96015101	8192	1
59	688602	198.54.58.12	198.54.58.2	5580	23	96017227	682910899	384	0
59	731363	198.54.58.12	198.54.58.2	5580	23	96017227	682910899	384	1
59	737652	198.54.58.2	198.54.58.12	23	5580	682910899	96017228	8192	1
59	788977	198.54.58.12	198.54.58.2	5632	23	97333808	219471652	384	0
59	790691	198.54.58.12	198.54.58.2	5579	23	96015101	683097157	384	0
59	897502	198.54.58.12	198.54.58.2	5580	23	96017228	682910900	384	0
0	51940	198.54.58.12	198.54.58.2	5579	23	96015101	683097157	384	2
0	57714	198.54.58.2	198.54.58.12	23	5579	683097157	96015103	8192	4
0	97855	198.54.58.2	198.54.58.12	23	5579	683097161	96015103	8192	7
0	188426	198.54.58.12	198.54.58.2	5579	23	96015103	683097168	384	0
0	197719	198.54.58.2	198.54.58.12	23	5579	683097168	96015103	8192	7
0	267705	198.54.58.2	198.54.58.12	23	5579	683097175	96015103	8192	18
0	292098	198.54.58.12	198.54.58.2	5580	23	96017228	682910900	384	1
0	297692	198.54.58.2	198.54.58.12	23	5580	682910900	96017229	8192	1
0	347845	198.54.58.2	198.54.58.12	23	5579	683097193	96015103	8192	52
0	353076	198.54.58.12	198.54.58.2	5579	23	96015103	683097245	384	0
0	371217	198.54.58.12	198.54.58.2	5632	23	97333808	219471652	384	3
0	377694	198.54.58.2	198.54.58.12	23	5632	219471652	97333811	8192	3
0	398592	198.54.58.2	198.54.58.12	23	5579	683097245	96015103	8192	240
0	488191	198.54.58.12	198.54.58.2	5580	23	96017229	682910901	384	0
0	489904	198.54.58.12	198.54.58.2	5632	23	97333811	219471655	384	0
0	588500	198.54.58.12	198.54.58.2	5579	23	96015103	683097485	256	0
0	590316	198.54.58.2	198.54.58.12	23	5579	683097485	96015103	8192	256
0	772753	198.54.58.12	198.54.58.2	5632	23	97333811	219471655	384	3
0	787778	198.54.58.2	198.54.58.12	23	5632	219471655	97333814	8192	3
0	789311	198.54.58.12	198.54.58.2	5579	23	96015103	683097741	256	0
0	791033	198.54.58.2	198.54.58.12	23	5579	683097741	96015103	8192	234
0	932487	198.54.58.12	198.54.58.2	5632	23	97333814	219471658	384	3
0	947785	198.54.58.2	198.54.58.12	23	5632	219471658	97333817	8192	3
0	966548	198.54.58.13	198.54.58.2	3308	3986	42842917	1725165637	512	1
0	967627	198.54.58.2	198.54.58.13	3986	3308	1725165637	42842918	8189	0
0	988460	198.54.58.12	198.54.58.2	5579	23	96015103	683097975	256	0

1	11518	198.54.58.12	198.54.58.2	5580	23	96017229	682910901	384	2
1	17857	198.54.58.2	198.54.58.12	23	5580	682910901	96017231	8192	4
1	41486	198.54.58.13	198.54.58.2	6050	23	97432524	306562874	512	1
1	47958	198.54.58.2	198.54.58.13	23	6050	306562874	97432525	8192	52
1	93758	198.54.58.12	198.54.58.2	5632	23	97333817	219471661	384	3
1	107871	198.54.58.2	198.54.58.12	23	5632	219471661	97333820	8192	3
1	120962	198.54.58.13	198.54.58.2	6050	23	97432525	306562926	512	0
1	189129	198.54.58.12	198.54.58.2	5580	23	96017231	682910905	384	0
1	251606	198.54.58.12	198.54.58.2	5632	23	97333820	219471664	384	3
1	257914	198.54.58.2	198.54.58.12	23	5632	219471664	97333823	8192	3
1	289503	198.54.58.12	198.54.58.2	5579	23	96015103	683097975	384	0
1	391392	198.54.58.12	198.54.58.2	5632	23	97333823	219471667	384	0
1	528006	198.54.58.2	198.54.58.12	23	5580	682910905	96017231	8192	13
1	558760	198.54.58.2	198.54.58.12	23	5580	682910918	96017231	8192	240
1	568091	198.54.58.2	198.54.58.12	23	5580	682911158	96017231	8192	58
1	689111	198.54.58.12	198.54.58.2	5580	23	96017231	682911216	256	0
1	731879	198.54.58.12	198.54.58.2	5579	23	96015103	683097975	384	3
1	737855	198.54.58.2	198.54.58.12	23	5579	683097975	96015106	8192	3
1	897868	198.54.58.12	198.54.58.2	5579	23	96015106	683097978	384	0
1	988258	198.54.58.12	198.54.58.2	5580	23	96017231	682911216	384	0
2	211670	198.54.58.12	198.54.58.2	5579	23	96015106	683097978	384	6
2	217909	198.54.58.2	198.54.58.12	23	5579	683097978	96015112	8192	3
2	292265	198.54.58.12	198.54.58.2	5579	23	96015112	683097981	384	6
2	297892	198.54.58.2	198.54.58.12	23	5579	683097981	96015118	8192	4
2	372863	198.54.58.12	198.54.58.2	5579	23	96015118	683097985	384	3
2	374581	198.54.58.12	198.54.58.2	5632	23	97333823	219471667	384	1
2	377991	198.54.58.2	198.54.58.12	23	5579	683097985	96015121	8192	3
2	387918	198.54.58.2	198.54.58.12	23	5632	219471667	97333824	8192	1
2	451957	198.54.58.12	198.54.58.2	5579	23	96015121	683097988	384	6
2	457912	198.54.58.2	198.54.58.12	23	5579	683097988	96015127	8192	5
2	532551	198.54.58.12	198.54.58.2	5579	23	96015127	683097993	384	6
2	537928	198.54.58.2	198.54.58.12	23	5579	683097993	96015133	8192	4
2	588513	198.54.58.12	198.54.58.2	5632	23	97333824	219471668	384	0
2	611573	198.54.58.12	198.54.58.2	5579	23	96015133	683097997	384	7
2	618015	198.54.58.2	198.54.58.12	23	5579	683097997	96015140	8192	5
2	692178	198.54.58.12	198.54.58.2	5579	23	96015140	683098002	384	6
2	697972	198.54.58.2	198.54.58.12	23	5579	683098002	96015146	8192	19
2	772785	198.54.58.12	198.54.58.2	5579	23	96015146	683098021	384	5
2	788533	198.54.58.2	198.54.58.12	23	5579	683098021	96015151	8192	181
2	851881	198.54.58.12	198.54.58.2	5579	23	96015151	683098202	256	6
2	858068	198.54.58.2	198.54.58.12	23	5579	683098202	96015157	8192	13
2	868208	198.54.58.2	198.54.58.12	23	5579	683098215	96015157	8192	91
2	932609	198.54.58.12	198.54.58.2	5579	23	96015157	683098306	256	6
2	948332	198.54.58.2	198.54.58.12	23	5579	683098306	96015163	8192	113
3	13299	198.54.58.12	198.54.58.2	5579	23	96015163	683098419	256	6
3	17964	198.54.58.2	198.54.58.12	23	5579	683098419	96015169	8192	0
3	28584	198.54.58.2	198.54.58.12	23	5579	683098419	96015169	8192	191
3	92456	198.54.58.12	198.54.58.2	5579	23	96015169	683098610	128	6
3	98087	198.54.58.2	198.54.58.12	23	5579	683098610	96015175	8192	13
3	108209	198.54.58.2	198.54.58.12	23	5579	683098623	96015175	8192	96
3	173197	198.54.58.12	198.54.58.2	5579	23	96015175	683098719	128	6
3	188372	198.54.58.2	198.54.58.12	23	5579	683098719	96015181	8192	118
3	252240	198.54.58.12	198.54.58.2	5579	23	96015181	683098837	0	6
3	332779	198.54.58.12	198.54.58.2	5579	23	96015187	683098837	128	6
3	405989	198.54.58.4	198.54.58.255	520	520	33554432	33619968	0	492
3	407164	198.54.58.4	198.54.58.255	520	520	33554432	33619968	0	492
3	408330	198.54.58.4	198.54.58.255	520	520	33554432	33619968	0	492
3	409508	198.54.58.4	198.54.58.255	520	520	33554432	33619968	0	492
3	410695	198.54.58.4	198.54.58.255	520	520	33554432	33619968	0	492
3	412015	198.54.58.4	198.54.58.255	520	520	30932992	33619968	0	452
3	414350	198.54.58.12	198.54.58.2	5579	23	96015193	683098837	256	6
3	416222	198.54.58.2	198.54.58.12	23	5579	683098837	96015199	8192	256
3	493503	198.54.58.12	198.54.58.2	5632	23	97333824	219471668	384	1
3	498055	198.54.58.2	198.54.58.12	23	5632	219471668	97333825	8192	1
3	588876	198.54.58.12	198.54.58.2	5579	23	96015199	683099093	128	0
3	689120	198.54.58.12	198.54.58.2	5632	23	97333825	219471669	384	0
3	989711	198.54.58.12	198.54.58.2	5579	23	96015199	683099093	384	0
3	991047	198.54.58.2	198.54.58.12	23	5579	683099093	96015199	8192	136
4	118347	198.54.58.2	128.100.75.10	4968	70	180416027	507257430	8192	0
4	188658	198.54.58.12	198.54.58.2	5579	23	96015199	683099229	384	0
4	453117	198.54.58.12	198.54.58.2	5632	23	97333825	219471669	384	1
4	458167	198.54.58.2	198.54.58.12	23	5632	219471669	97333826	8192	1

4	565997	198.54.58.13	198.54.58.2	6050	23	97432525	306562926	512	1
4	568276	198.54.58.2	198.54.58.13	23	6050	306562926	97432526	8192	4
4	589548	198.54.58.12	198.54.58.2	5632	23	97333826	219471670	384	0
4	612611	198.54.58.12	198.54.58.2	5632	23	97333826	219471670	384	1
4	618216	198.54.58.2	198.54.58.12	23	5632	219471670	97333827	8192	1
4	693205	198.54.58.12	198.54.58.2	5632	23	97333827	219471671	384	1
4	698199	198.54.58.2	198.54.58.12	23	5632	219471671	97333828	8192	1
4	801479	198.54.58.13	198.54.58.2	6050	23	97432526	306562930	512	0
4	893678	198.54.58.12	198.54.58.2	5632	23	97333828	219471672	384	0
4	967201	198.54.58.13	198.54.58.2	6050	23	97432526	306562930	512	1
4	978236	198.54.58.2	198.54.58.13	23	6050	306562930	97432527	8192	1
5	201531	198.54.58.13	198.54.58.2	6050	23	97432527	306562931	512	0
5	282089	198.54.58.13	198.54.58.2	6050	23	97432527	306562931	512	1
5	288309	198.54.58.2	198.54.58.13	23	6050	306562931	97432528	8192	1
5	442141	198.54.58.13	198.54.58.2	6050	23	97432528	306562932	512	1
5	448276	198.54.58.2	198.54.58.13	23	6050	306562932	97432529	8192	1
5	493138	198.54.58.12	198.54.58.2	5579	23	96015199	683099229	384	1
5	494849	198.54.58.12	198.54.58.2	5634	23	97430522	299398512	384	3
5	498384	198.54.58.2	198.54.58.12	23	5634	299398512	97430525	8192	3
5	618322	198.54.58.2	198.54.58.12	23	5579	683099229	96015200	8192	0
5	652698	198.54.58.12	198.54.58.2	5579	23	96015200	683099229	384	1
5	658325	198.54.58.2	198.54.58.12	23	5579	683099229	96015201	8192	1
5	682150	198.54.58.13	198.54.58.2	6050	23	97432529	306562933	512	1
5	688320	198.54.58.2	198.54.58.13	23	6050	306562933	97432530	8192	1
5	688957	198.54.58.12	198.54.58.2	5634	23	97430525	299398515	384	0
5	789193	198.54.58.12	198.54.58.2	5579	23	96015201	683099230	384	0
5	812255	198.54.58.12	198.54.58.2	5579	23	96015201	683099230	384	1
5	818358	198.54.58.2	198.54.58.12	23	5579	683099230	96015202	8192	1
5	921683	198.54.58.13	198.54.58.2	6050	23	97432530	306562934	512	0
5	973320	198.54.58.12	198.54.58.2	5634	23	97430525	299398515	384	6
5	978435	198.54.58.2	198.54.58.12	23	5634	299398515	97430531	8192	6
5	989878	198.54.58.12	198.54.58.2	5579	23	96015202	683099231	384	0
6	2187	198.54.58.13	198.54.58.2	6050	23	97432530	306562934	512	1
6	8412	198.54.58.2	198.54.58.13	23	6050	306562934	97432531	8192	1
6	52350	198.54.58.12	198.54.58.2	5632	23	97333828	219471672	384	1
6	55699	198.54.58.12	198.54.58.2	5634	23	97430531	299398521	384	6
6	58395	198.54.58.2	198.54.58.12	23	5634	299398521	97430537	8192	4
6	58795	198.54.58.2	198.54.58.12	23	5632	219471672	97333829	8192	1
6	68427	198.54.58.2	198.54.58.12	23	5634	299398525	97430537	8192	6
6	133132	198.54.58.12	198.54.58.2	5632	23	97333829	219471673	384	1
6	134845	198.54.58.12	198.54.58.2	5634	23	97430537	299398531	384	3
6	138424	198.54.58.2	198.54.58.12	23	5634	299398531	97430540	8192	5
6	138823	198.54.58.2	198.54.58.12	23	5632	219471673	97333830	8192	1
6	241684	198.54.58.13	198.54.58.2	6050	23	97432531	306562935	512	0
6	292690	198.54.58.12	198.54.58.2	5634	23	97430540	299398536	384	0
6	294403	198.54.58.12	198.54.58.2	5632	23	97333830	219471674	384	0
6	318450	198.54.58.2	198.54.58.12	23	5579	683099231	96015202	8192	1
6	452189	198.54.58.12	198.54.58.2	5579	23	96015202	683099232	384	1
6	455541	198.54.58.12	198.54.58.2	5634	23	97430540	299398536	384	3
6	458498	198.54.58.2	198.54.58.12	23	5634	299398536	97430543	8192	3
6	532855	198.54.58.12	198.54.58.2	5579	23	96015203	683099232	384	1
6	538452	198.54.58.2	198.54.58.12	23	5579	683099232	96015204	8192	12
6	588821	198.54.58.12	198.54.58.2	5634	23	97430543	299398539	384	0
6	608506	198.54.58.2	198.54.58.12	23	5579	683099244	96015204	8192	7
6	613589	198.54.58.12	198.54.58.2	5634	23	97430543	299398539	384	3

Appendix B

Filtered Trace

Sek	MicroS	Source IP	Dest IP	SPort	DPort	Seq	Ack	Win	DLen
49	410778	198.54.58.12	198.54.58.2	5579	23	96015081	683097129	384	1
49	416298	198.54.58.2	198.54.58.12	23	5579	683097129	96015082	8192	1
49	570332	198.54.58.12	198.54.58.2	5579	23	96015082	683097130	384	1
49	576272	198.54.58.2	198.54.58.12	23	5579	683097130	96015083	8192	1
49	688777	198.54.58.12	198.54.58.2	5579	23	96015083	683097131	384	0
49	729897	198.54.58.12	198.54.58.2	5579	23	96015083	683097131	384	1
49	736272	198.54.58.2	198.54.58.12	23	5579	683097131	96015084	8192	1
49	895883	198.54.58.12	198.54.58.2	5579	23	96015084	683097132	384	0
49	969860	198.54.58.12	198.54.58.2	5579	23	96015084	683097132	384	1
49	976317	198.54.58.2	198.54.58.12	23	5579	683097132	96015085	8192	1
50	86841	198.54.58.12	198.54.58.2	5579	23	96015085	683097133	384	0
50	609922	198.54.58.12	198.54.58.2	5579	23	96015085	683097133	384	1
50	616893	198.54.58.2	198.54.58.12	23	5579	683097133	96015086	8192	3
50	789258	198.54.58.12	198.54.58.2	5579	23	96015086	683097136	384	0
51	571461	198.54.58.12	198.54.58.2	5579	23	96015086	683097136	384	1
51	576545	198.54.58.2	198.54.58.12	23	5579	683097136	96015087	8192	1
51	688184	198.54.58.12	198.54.58.2	5579	23	96015087	683097137	384	0
52	51402	198.54.58.12	198.54.58.2	5579	23	96015087	683097137	384	1
52	56611	198.54.58.2	198.54.58.12	23	5579	683097137	96015088	8192	1
52	188168	198.54.58.12	198.54.58.2	5579	23	96015088	683097138	384	0
52	931365	198.54.58.12	198.54.58.2	5579	23	96015088	683097138	384	1
52	937102	198.54.58.2	198.54.58.12	23	5579	683097138	96015089	8192	1
53	90999	198.54.58.12	198.54.58.2	5579	23	96015089	683097139	384	1
53	96696	198.54.58.2	198.54.58.12	23	5579	683097139	96015090	8192	1
53	288257	198.54.58.12	198.54.58.2	5579	23	96015090	683097140	384	0
55	651158	198.54.58.12	198.54.58.2	5579	23	96015090	683097140	384	1
55	657082	198.54.58.2	198.54.58.12	23	5579	683097140	96015091	8192	1
55	787802	198.54.58.12	198.54.58.2	5579	23	96015091	683097141	384	0
55	51094	198.54.58.12	198.54.58.2	5579	23	96015091	683097141	384	1
56	57578	198.54.58.2	198.54.58.12	23	5579	683097141	96015092	8192	1
56	189982	198.54.58.12	198.54.58.2	5579	23	96015092	683097142	384	0
56	292256	198.54.58.12	198.54.58.2	5579	23	96015092	683097142	384	1
56	297226	198.54.58.2	198.54.58.12	23	5579	683097142	96015093	8192	1
56	488234	198.54.58.12	198.54.58.2	5579	23	96015093	683097143	384	0
56	611672	198.54.58.12	198.54.58.2	5579	23	96015093	683097143	384	1
56	617209	198.54.58.2	198.54.58.12	23	5579	683097143	96015094	8192	1
56	771288	198.54.58.12	198.54.58.2	5579	23	96015094	683097144	384	1
56	777182	198.54.58.2	198.54.58.12	23	5579	683097144	96015095	8192	1
56	893075	198.54.58.12	198.54.58.2	5579	23	96015095	683097145	384	0
57	731485	198.54.58.12	198.54.58.2	5579	23	96015095	683097145	384	1
57	737342	198.54.58.2	198.54.58.12	23	5579	683097145	96015096	8192	3
57	897807	198.54.58.12	198.54.58.2	5579	23	96015096	683097148	384	1
57	907360	198.54.58.2	198.54.58.12	23	5579	683097148	96015097	8192	3
58	51026	198.54.58.12	198.54.58.2	5579	23	96015097	683097151	384	1
58	57453	198.54.58.2	198.54.58.12	23	5579	683097151	96015098	8192	3
58	188227	198.54.58.12	198.54.58.2	5579	23	96015098	683097154	384	0
59	172154	198.54.58.12	198.54.58.2	5579	23	96015098	683097154	384	1
59	177488	198.54.58.2	198.54.58.12	23	5579	683097154	96015099	8192	1
59	288880	198.54.58.12	198.54.58.2	5579	23	96015099	683097155	384	0
59	412110	198.54.58.12	198.54.58.2	5579	23	96015099	683097155	384	1
59	417523	198.54.58.2	198.54.58.12	23	5579	683097155	96015100	8192	1
59	588093	198.54.58.12	198.54.58.2	5579	23	96015100	683097156	384	0
59	652201	198.54.58.12	198.54.58.2	5579	23	96015100	683097156	384	1
59	657986	198.54.58.2	198.54.58.12	23	5579	683097156	96015101	8192	1
59	790691	198.54.58.12	198.54.58.2	5579	23	96015101	683097157	384	0
60	51940	198.54.58.12	198.54.58.2	5579	23	96015101	683097157	384	2
60	57714	198.54.58.2	198.54.58.12	23	5579	683097157	96015103	8192	4
0	97855	198.54.58.2	198.54.58.12	23	5579	683097161	96015103	8192	7
0	188426	198.54.58.12	198.54.58.2	5579	23	96015103	683097168	384	0

0	197719	198.54.58.2	198.54.58.12	23	5579	683097168	96015103	8192	7
0	267705	198.54.58.2	198.54.58.12	23	5579	683097175	96015103	8192	18
0	347845	198.54.58.2	198.54.58.12	23	5579	683097193	96015103	8192	52
0	353076	198.54.58.12	198.54.58.2	5579	23	96015103	683097245	384	0
0	398592	198.54.58.2	198.54.58.12	23	5579	683097245	96015103	8192	240
0	588500	198.54.58.12	198.54.58.2	5579	23	96015103	683097485	256	0
0	590316	198.54.58.2	198.54.58.12	23	5579	683097485	96015103	8192	256
0	789311	198.54.58.12	198.54.58.2	5579	23	96015103	683097741	256	0
0	791033	198.54.58.2	198.54.58.12	23	5579	683097741	96015103	8192	234
0	988460	198.54.58.12	198.54.58.2	5579	23	96015103	683097975	256	0
1	289503	198.54.58.12	198.54.58.2	5579	23	96015103	683097975	384	0
1	731879	198.54.58.12	198.54.58.2	5579	23	96015103	683097975	384	3
1	737855	198.54.58.2	198.54.58.12	23	5579	683097975	96015106	8192	3
1	897868	198.54.58.12	198.54.58.2	5579	23	96015106	683097978	384	0
2	211670	198.54.58.12	198.54.58.2	5579	23	96015106	683097978	384	6
2	217909	198.54.58.2	198.54.58.12	23	5579	683097978	96015112	8192	3
2	292265	198.54.58.12	198.54.58.2	5579	23	96015112	683097981	384	6
2	297892	198.54.58.2	198.54.58.12	23	5579	683097981	96015118	8192	4
2	372863	198.54.58.12	198.54.58.2	5579	23	96015118	683097985	384	3
2	377991	198.54.58.2	198.54.58.12	23	5579	683097985	96015121	8192	3
2	451957	198.54.58.12	198.54.58.2	5579	23	96015121	683097988	384	6
2	457912	198.54.58.2	198.54.58.12	23	5579	683097988	96015127	8192	5
2	532551	198.54.58.12	198.54.58.2	5579	23	96015127	683097993	384	6
2	537928	198.54.58.2	198.54.58.12	23	5579	683097993	96015133	8192	4
2	611573	198.54.58.12	198.54.58.2	5579	23	96015133	683097997	384	7
2	618015	198.54.58.2	198.54.58.12	23	5579	683097997	96015140	8192	5
2	692178	198.54.58.12	198.54.58.2	5579	23	96015140	683098002	384	6
2	697972	198.54.58.2	198.54.58.12	23	5579	683098002	96015146	8192	19
2	772785	198.54.58.12	198.54.58.2	5579	23	96015146	683098021	384	5
2	788533	198.54.58.2	198.54.58.12	23	5579	683098021	96015151	8192	181
2	851881	198.54.58.12	198.54.58.2	5579	23	96015151	683098202	256	6
2	858068	198.54.58.2	198.54.58.12	23	5579	683098202	96015157	8192	13
2	868208	198.54.58.2	198.54.58.12	23	5579	683098215	96015157	8192	91
2	932609	198.54.58.12	198.54.58.2	5579	23	96015157	683098306	256	6
2	948332	198.54.58.2	198.54.58.12	23	5579	683098306	96015163	8192	113
3	13299	198.54.58.12	198.54.58.2	5579	23	96015163	683098419	256	6
3	17964	198.54.58.2	198.54.58.12	23	5579	683098419	96015169	8192	0
3	28584	198.54.58.2	198.54.58.12	23	5579	683098419	96015169	8192	191
3	92456	198.54.58.12	198.54.58.2	5579	23	96015169	683098610	128	6
3	98087	198.54.58.2	198.54.58.12	23	5579	683098610	96015175	8192	13
3	108209	198.54.58.2	198.54.58.12	23	5579	683098623	96015175	8192	96
3	173197	198.54.58.12	198.54.58.2	5579	23	96015175	683098719	128	6
3	188372	198.54.58.2	198.54.58.12	23	5579	683098719	96015181	8192	118
3	252240	198.54.58.12	198.54.58.2	5579	23	96015181	683098837	0	6
3	332779	198.54.58.12	198.54.58.2	5579	23	96015187	683098837	128	6
3	414350	198.54.58.12	198.54.58.2	5579	23	96015193	683098837	256	6
3	416222	198.54.58.2	198.54.58.12	23	5579	683098837	96015199	8192	256
3	588876	198.54.58.12	198.54.58.2	5579	23	96015199	683099093	128	0
3	989711	198.54.58.12	198.54.58.2	5579	23	96015199	683099093	384	0
3	991047	198.54.58.2	198.54.58.12	23	5579	683099093	96015199	8192	136
4	188658	198.54.58.12	198.54.58.2	5579	23	96015199	683099229	384	0
5	493138	198.54.58.12	198.54.58.2	5579	23	96015199	683099229	384	1
5	618322	198.54.58.2	198.54.58.12	23	5579	683099229	96015200	8192	0
5	652698	198.54.58.12	198.54.58.2	5579	23	96015200	683099229	384	1
5	658325	198.54.58.2	198.54.58.12	23	5579	683099229	96015201	8192	1
5	789193	198.54.58.12	198.54.58.2	5579	23	96015201	683099230	384	0
5	812255	198.54.58.12	198.54.58.2	5579	23	96015201	683099230	384	1
5	818358	198.54.58.2	198.54.58.12	23	5579	683099230	96015202	8192	1
5	989878	198.54.58.12	198.54.58.2	5579	23	96015202	683099231	384	0
6	318450	198.54.58.2	198.54.58.12	23	5579	683099231	96015202	8192	1
6	452189	198.54.58.12	198.54.58.2	5579	23	96015202	683099232	384	1
6	532855	198.54.58.12	198.54.58.2	5579	23	96015203	683099232	384	1
6	538452	198.54.58.2	198.54.58.12	23	5579	683099232	96015204	8192	12
6	608506	198.54.58.2	198.54.58.12	23	5579	683099244	96015204	8192	7

Bibliography

Ben-Artzi, A., Chandna,A. & Warrier, U. (1990, July) *Network Management of TCP/IP Networks: Present and Future.* IEEE Network Magazine, Vol 4 (4), pp 35-43.

BICC Data Network Limited. (1986, June) *The 4100 series Controller Interface Manual.* Hemel Hempstead, United Kingdom.

Black, U. (1992a). *Network Management Standards.* New York: McGraw-Hill.

Black, U. (1992b). *TCP/IP and Related Protocols.* New York : McGraw-Hill.

Case, J.D.; Fedor,M.; Schoffstall,M.L.; Davin,J (1988, August) *Simple Network Management Protocol.* Request For Comment 1067, DDN Network Information Centre,.SRI International.

Clark, D.C.(1982, July) *Window and Acknowlegment Stategy in TCP.* Request For Comment 813, DDN Network Information Centre, SRI International.

Comer, D.E. & Stevens, D.L. (1991) *Internetworking with TCP/IP Volume II.* New Jersey: Prentice Hall.

Dallas, I.N.; Spratt.E.B & Cabanel, J.P. (1991) *Issues in LAN Management, II.* (Procedings of the IFIP TC6/WG6, 4a International Symposium on the management of Local Communications Systems, Canterbury, U.K., 18-19 September, 1990) North-Holland: Elsevier Science Publishers.

Derfler, F.J. (1990, June) *Lan Analyzers.* PC Magazine, Vol 9 (12), pp 205-241.

Doepnik, JR (1990). *Packet Drivers made simple* (File Packet_d.109 in PktDrv9.zip). Utah State University, Utah.

Falaki, S.O. & Sorensen, S.A. (1992) *Traffic Measurement on a Local Area Computer Network.* Computer Communications , Vol 15 (3) , pp 192-197.

Fisher, S. (1989, December) *The debate between SNMP and CMIP rolls on.* Lantimes. pp 192-197.

Greenfield, D. (1991, September) *Network Management Filters Down to the Desktop.* Datacommunications, Vol 20, pp 39-42.

Halsall, F. & Modiri, M. (1990) *Protocol analyser for the monitor and analysis of OSI networks.* Computer Communications, Vol 13(9), pp 533-541.

Herman, J. (1990, November) *Enterprise Management Vendors Shoot it out.* Datacommunications, Vol 19, pp 92-110.

Held, G (1992) *Network Management: Techniques, Tools and Systems.* Chicester, England: John Wiley and Sons Ltd.

ISO DIS 7498/4, (1987). *Information Processing System - Open System Interconnection - Basic Reference Model Part 4.* OSI Management Framework. Geneva: ISO.

Intel Corporation. (1989) *Intel 386 Board Technical Reference Manual,* Santa Clara, California.

Irish, W. (1994) *Performance problems on high utilization Ethernets,* Electronic ariticle to be published as : Investigations into observerved performance problems on high utilization Ethernet networks, PARC Blue White Report. Avaliable from wirish@parc.xerox.com.

Jander, M. (1991, September) *CMIP Gets a New Chance.* Datacommunications, Vol 20, pp 51-56.

Jander, M. (1993, January). *Diagnosing and Test Equipment.* Datacommunications, Vol 22, pp 104-106.

Jongerius, J. (1991, July). *Accurately Timing Window Events Without Timer Reprogramming.* Microsoft Systems Journal.

Kauffels, F. (1992) *Network Management* (translated from German by S.S. Wilson). Workinham: Addison-Wesley. (Original work published 1992).

Lew, H.K & Robertson, J. (1989, August) *TCP/IP network management with an eye towords OSI.* Datacommunications, Vol 18, pp 123-130.

Michalski, A (1991) *Managing the Array of rings in Local Communication Systems..* Issues in LAN Management, II.

Miller, M.A. (1992) *Troubelshooting TCP/IP : Analizing the Protocols of the Internet.* San Mateo : M&T Books.

Mogul, J.C. (1990) *Efficient Use of Workstations for Passive Monitoring of Local Area Networks.* Palo Alto : Digital Equipment Corporation Western Research Laboratory.

Nemzow, M. (1988) *Keeping the Link: Ethernet installation & Management.* New York: McGraw-Hill.

Perkins, C. (1990) *Managing Stuctured Networks.* Paper presented at the Data Communications Technology Update International Conference in Pretoria.

Postel, J.B.; Reynolds, J.K. (1988, February) *Standard for transmission of IP datagrams over IEEE 802 networks.* Request For Comment 1042. DDN Network Information Centre, SRI International.

Prinsloo, P.W. (1991) *Modern Hardware and Software Techniques for managing Ethernet (IEEE 802.3) Local Area Networks* Paper presented at the joint SAIEE/CSSA Symposium in Pretoria.

Protogeros, A. & Ball, E. (1990) *Traffic Analyser and Generator.* Computer Communications, Vol 13(8), pp 469-477.

Rehmann, O (1993) PKTDRVR Interface for Turbo Pascal 7.0. (Public Domain file available on Internet).

Reynolds, J.K; Postel, J.B. (1987, May) *Assigned Numbers.* Request For Comment 1010. DDN Network Information Centre, SRI International.

Roden, T. (1992, September) *High-Resolution Timing,* Dr.Dobbs Journal, Vol 7 (9), pp 42-48,110.

Romkey, J. (1989). *PC/TCP Packet Driver Specification Version 1.09,* Wakefield: FTP Software, Inc.

Sekkaki, A & Westphall, C.B. (1991) *Heterogeneous LANs Management.* Issues in LAN Management, II.

Shepard, T.J. .(1991) *TCP Packet Trace Analysis.* Unpublished Masters Thesis, Massachusetts Institute of Technology, Cambridge.

Spanier, S. (1988) *Designing and implementing af a LAN monitoring tool.* Computer Communications, Vol 11(2), pp 85-89.

Stevens, W.R. (1993) *TCP/IP Illustrated, Volume I: The Protocol.* New York: Addison-Westley.

Sudama, R. & Dah-Ming, C. (1990) *Experiences of Designing a Sophisticated Network Monitor.* Software-Practice and Experience, Vol. 20(6), pp 555-570.

Van Niekerk, G.P. (1991) *TCP/IP.* (based on Introduction to Internet Protocols. Hendrick, C.L. Rutgers University). Unpublished seminar.

Van Niekerk, G.P. (1994), Personal communication, 13 September.

White, D. (1989, July) *Internet Management SNMP and CMOT: Two ways to do the same thing.* Lan Magazine pp 147-150.

Whyatt, A.L. (1987) *Using assembly Language.* Carmel, Indiana. Que Corporation.

Samevatting

Waar hoëvlak protokolle die gebruiker afskerm van onderliggende probleme in die netwerk, is dit vir die netwerkbestuurder juis nodig om vroegtydig bewus te wees van sluimerende probleme. Alhoewel daar hulpmiddels, in die vorm van netwerk-bestuurspakkette, bestaan om hom behulpsaam te wees met die identifisering en ontleding van netwerkfoute, is hulle dikwels ontoereikend.

Bestaande netwerkbestuurspakkette maak tot 'n mate voorsiening vir die ontleding van verkeer op 'n netwerk. Waar 'n enkele verbinding egter as 'n geheel beskou moet word, word verslae van alle pakkies op die netwerk as voldoende beskou. Die verant-woordelikheid vir die ontleding van hierdie verslae bly dié van die netwerkbestuurder.

Hierdie tesis beskryf die ontwikkeling van 'n program wat die monitering en ontleding van 'n enkele TCP gesprek op 'n Ethernet-netwerk moontlik maak. Deur die identifisering van foutiewe verbindings te vergemaklik en die grafiese voorstelling daarvan moontlik te maak, lewer **TCPlot** 'n bydrae tot netwerkbestuur.