COSC 460
Honours Project
1991.

Written by: Lindsay Weir.

Supervisor: Ray Hunt.

Tellabs

Network

Monitor

- project report

Mount Cook Group Ltd.

CONTENTS

1. Introduction

· Company background

The Mount Cook Group Ltd operate a nation-wide commercial and tour group computer network. The network consists of two central computers, the NCR 9400 and 9500 series mini computers together supporting about 150 terminals. The 9400 is used for development work and by the freightlines department while the 9500 is used for tours, accounts and a messaging system. The network also supports two international links to Sydney and Los Angeles. The layout of these links can be seen in Appendix 1. The network is controlled by the Tellabs Network Management System. Although management software is available for configuring the network there exists no data collection and management report system.

The network consists of two layers of multiplexers. The main layer is the backbone of the network which connects the three main centres situated at Riccarton Rd, Christchurch, Newton Data Centre and Queen Street, Auckland as can be seen in diagram 1.

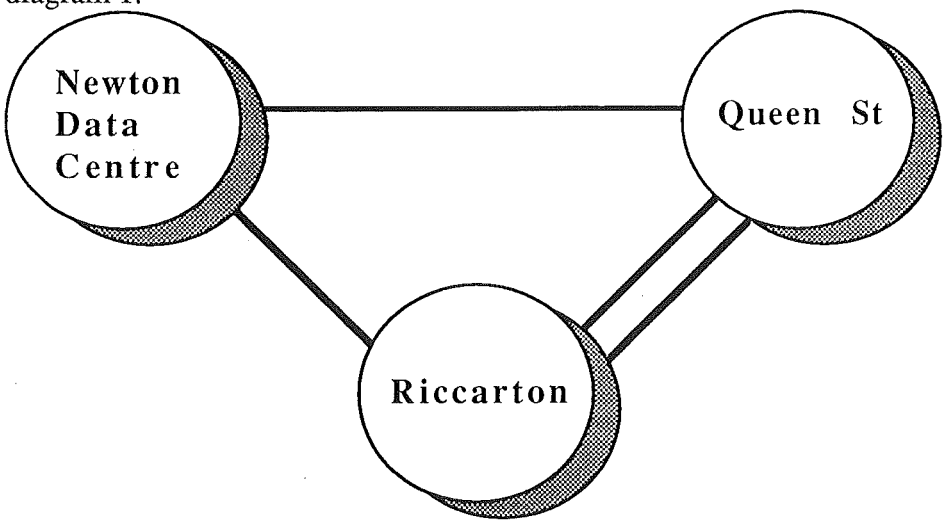


Diagram 1. Network Backbone.

These backbone multiplexers supplied by Tellabs are termed Xplexer's. To these Xplexers are attached local multiplexers called dataplexers to which peripheral equipment such as terminals and printers are attached. The dataplexers are modular and can support up to 32 channels can support up to 32 channels, depending on their size. This can be seen in a hierarchy diagram of the networks multiplexers below in diagram 2.

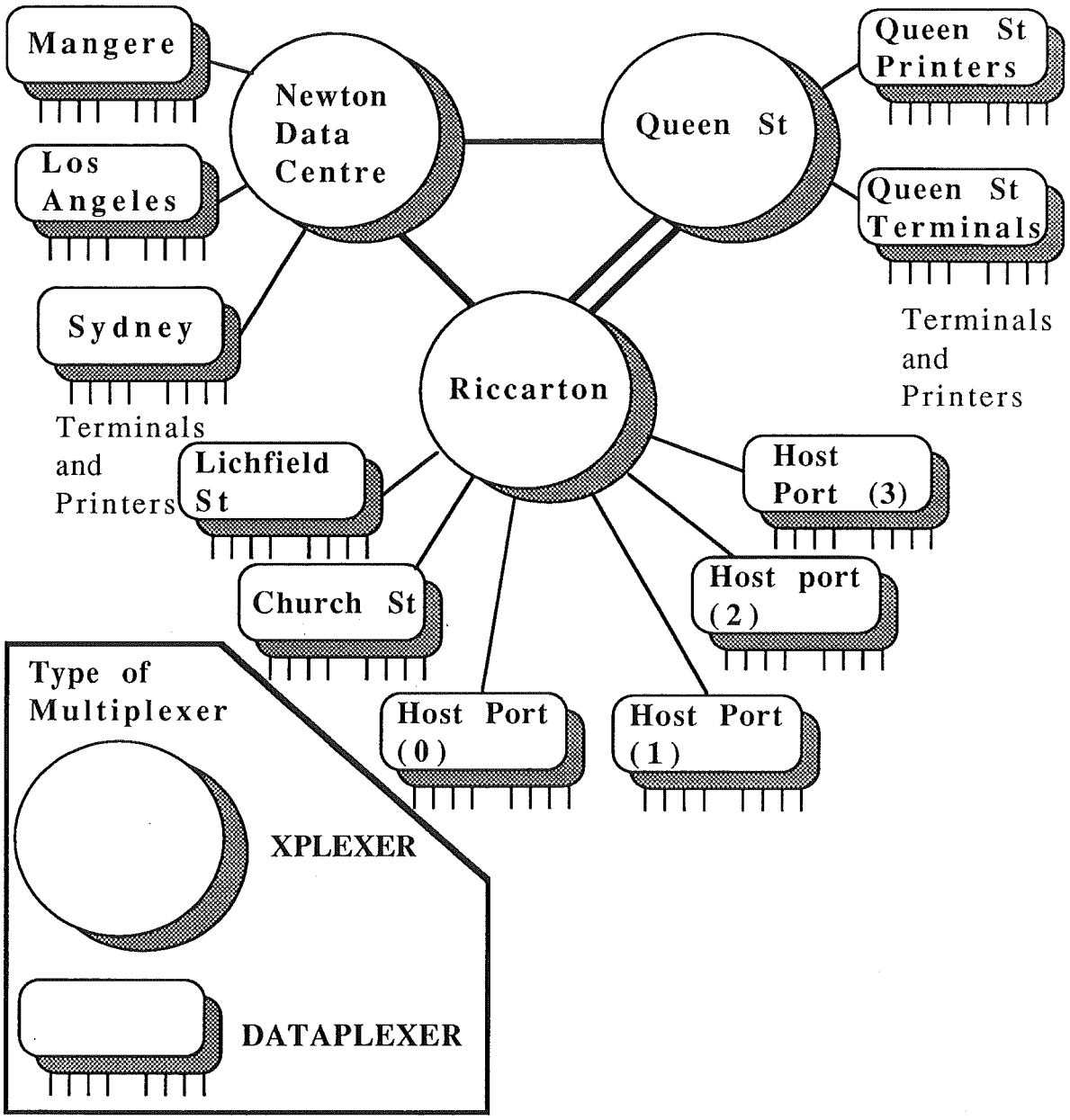


Diagram 2. Multiplexer Hierarchy Showing Xplexers and Dataplexers.

In Appendix B, a full technical diagram of the network is supplied showing the entire configuration of the network.

· Requirements

The requirement of the project was to produce an online tool to help with monitoring system performance of their network on an IBM compatible PC. The system should be a user friendly environment, so that the user should need very little training to operate it efficiently.

Assistance from this network monitoring tool is needed to help in areas such as:

- determining how much traffic is on each line,

- determining if poor user performance is contributed to by the line utilization,
- determining if line speeds could be upgraded or down graded due to the line utilization,
- finding the total traffic of the network,
- helping with future planning based on the present results.
- configuration planning so the layout of the network is available at an instance,
- finding out traffic patterns.

Almost all of these things are unknown at present, so the end product will give an insight into the nature of the network's current performance and be a valuable tool for analysing problem areas of the network. This may include finding dead ports, lines with unacceptable errors rates on and other background problems which were undetectable until the present time.

Configuration of the network's topology at present is based upon observed system performance without much statistical data available on actual performance history.

The requirement of this project was to implement all aspects of a network monitoring system from the hardware interfacing to the database reporting for the Mount Cook Group Ltd. The design consisted of researching all possible approaches available to achieve the most efficient results.

· Existing Options

The software package commercially available is the Tellabs 320 Network Monitor which generates various text based reports. It captures the data from two sources. The first of which is an event log process which captures system events. The second uses a second communication channel as if is a terminal into the network administrators system. The platform it runs on is an IBM compatible AT with at least 1Mbyte of RAM and a 30Mbyte hard disk. It runs the XENIX System V Operating System, a PC version of Unix which allows multitasking capabilities of processes and the database management system used is Informix-SQL. The current cost of the Tellabs 320 Network Monitor is around $20,000 which includes the operating system and database management system above. The actual cost would largely consist of the price of XENIX and Informix.

There exists several reasons why this product would not be a feasible economic purchase for the Mount Cook Group Ltd. The first of course is the cost which is, in itself a major drawback for what the actual Network Monitor consists of. A second reason is the suppliers of the Tellabs equipment did not recommend the product. The last reason why the Tellabs 320 Network Monitor would not be appropriate is that the product is unable to collect any statistics from the lower level multiplexers, (dataplexers), which have terminals and printers attached to. This information is actually very valuable and would be of great importance in system performance analysis.

At present statistics produced can show how nodes and links are performing but nothing is available on the actual channels. In the case of a link in the network having a high line utilization there would be no way to see how the channels attached to this link had been performing over a period of time up to the point of it being found. The network management system can be used to see the channel throughput but the accuracy of this result depends on when the actual statistics were last reset. Almost certainly the statistics would not been reset for some long period of time thus making the data invalid for the instance of time required. Even if the statistics had been reset

recently, the results being displayed may not be accurate over a period of time but represent an exceptionally abnormal period of time. This result, if taken, could lead to dramatic system changes which could be unnecessary and expensive. This situation could have easily been avoided if there was a way to keep a record of periodic channel throughput statistics.

The inability to obtain this information from the dataplexers is not entirely the fault of the Tellabs 320 Network Monitor product since the statistics cannot be configured automatically to be sent to a destination like the system events produced from the Xplexers. The Tellabs 320 Network Monitor is capable of obtaining the information however from the second communication channel which acts as a command terminal into the network management system.

This is where an automated process of collecting the statistical data from the dataplexers would be very valuable to have valid data on system channel throughput and buffer utilization.

• Approach

To capture all the required information involves:

- capturing all the existing data being sent from the main multiplexers in the network, i.e the three Xplexers acting as the back-bone on the Mount Cook network.
- somehow retrieving statistics from the lower level multiplexers, i.e. the dataplexers attached to the Xplexers.
- analysing the captured data to make it into a usable form for a database.
- the creation of an application to let the network manager(s) interrogate the captured data for specific system performance statistics on certain areas of the network.

From these requirements we can see that there are two main areas to the project, the capture of statistical data and the database design and construction. We can see that the data capturing consists of two separate parts in itself. So in total we have two separate processes capturing the data and the database as a third process. The two processes which capture data run continuously in the background without any need for interaction with the user. One of these processes is for capturing the system event datagrams produced by the Xplexers while the second process interrogates the dataplexers at regular intervals to obtain statistical information on the channels. The two processes which capture data can be seen below more clearly in diagram 4. The database and any other number of programs can be run concurrently with the two data capturing processes.

Since the project is PC based, the main problem of course is how to run more than one process concurrently on an IBM compatible personal computer.

There are one of two viable approaches which could be taken in order to achieve multi-tasking.

- Use a multi-tasking UNIX like operating system such as XENIX (which the Tellabs Network Monitor 320 uses), or
- Use Microsoft Windows 3.0 in the 386 enhanced mode.

The first option requires the purchase of a relatively expensive operating system, XENIX, which would probably not be used by staff for other jobs.

The second option, under the circumstances is the most viable since the software already exists at the site and is more readily available and cheaper. The interrelation of the two processes and Windows can be seen below in diagram 3.

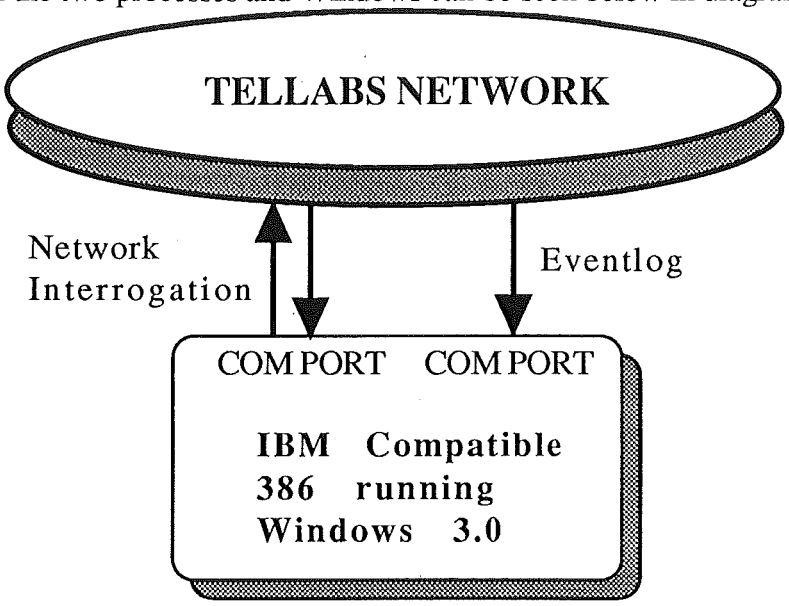


Diagram 3. Interrelationship of Windows and Data Capture Processes.

Here we can see that both processes will run under Microsoft Windows 3.0 which is on a machine running MS-DOS 4.1. One process captures data from the backbone Xplexers in the network while the second process polls each of the dataplexers in turn for statistical information.

This option however requires a 386 PC in order to achieve a relatively true sense of concurrent multi-tasking. The need for the 386 is because a PC, XT, AT or 286 actually stops the previous process when the user switches between applications. This would be acceptable for the situation of only one process capturing data whereby the new interrupt handler would still capture data into its 32K buffer but would still be unable to process it until the capturing process was reactivated. This would allow the computer to occasionally be used for something else like a word processor, but when they were finished the data capturing process would continue to process the captured data. Our situation involves two such processes so it would require each process to be activated for a certain amount of processor time in order to process the captured data.

Since PC's were originally designed as a DOS style operating system, their architecture is not the best to run a windows based graphical user interface environment on. Due to this Windows 3.0 has a very large overhead in disk swap space and is relatively slow when a large number of processes are run simultaneously. When running application programs like dBase IV under Windows the speed of execution is very noticeably slower than running it straight under MS-DOS. This of course may change with future releases of Windows based products as they become more and more efficient.

2. Data Capture

The capturing of the data from the two levels of multiplexers is the most important part of the project. It requires that there is no loss of incoming data or else the results produced by the Network Monitor will be largely invalid.

· Communication ports

The capturing of data from the network is the most essential area needing attention in the Network Monitor. Without reliable data the results of network performance would be inconsistent and meaningless. In a simple approach code could be written which would continuously loop waiting until it read a character from the communication port and then process each individual character arriving. This may be sufficient for a very slow communications channel with very little traffic but there are several problems which will arise with this simple approach. The first of which is if more characters arrive at the communication port while the initial character is being processed, the result of which is the loss of the incoming data. This of course is an unacceptable result. The only viable option is to use interrupt handlers.

• Interrupt handlers

All events on a PC are handled by interrupts. Thus if a key is pressed or a character is detected at the serial port an interrupt is invoked to handle the event. The machine essentially stops what it is doing and saves the current status about what it was doing and then proceeds to handle the interrupt. These interrupt handlers have default actions which can be changed by reassigning the interrupt vector to point to the new location of code written. This can be seen below in diagram 4.

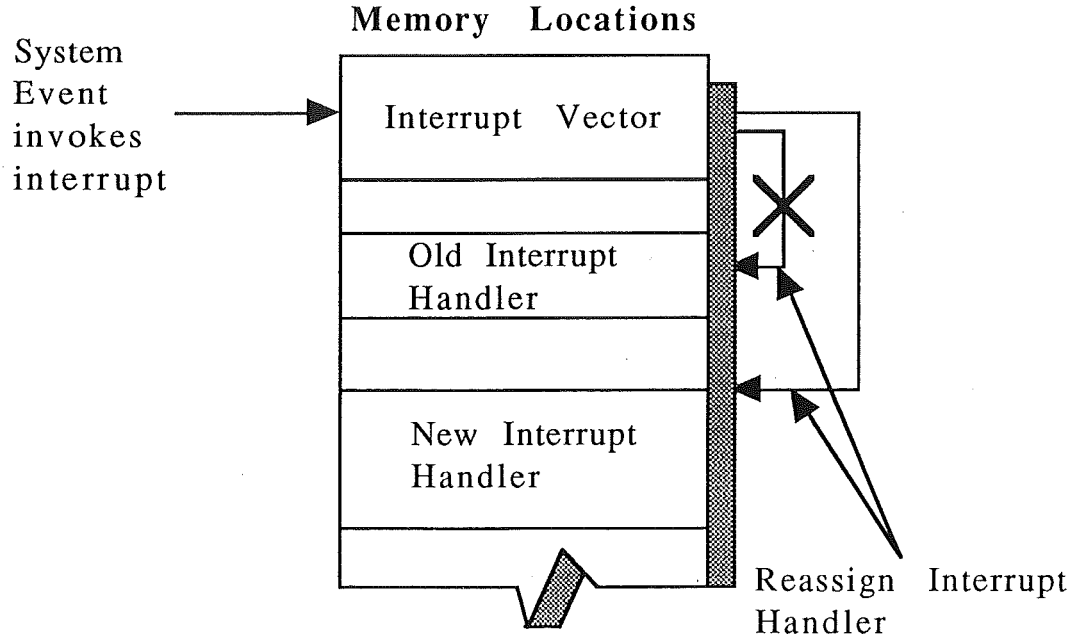


Diagram 4. Reassigning Interrupt Handlers.

The existing location the interrupt vector points to for the communication ports are changed from pointing to the default handlers to modified interrupt handlers for the

particular port. The new interrupt can thus be tailored to the specific needs of the data capture. The main requirement for the new interrupt handler is for the data to be accessible when needed from a buffer.

The new interrupt handler routines for the communication ports were obtained from Hans Nasten, Everyware Mikrodata AB, in Sweden by posting to News. Permission has been granted to use the routines in any way required. This can be confirmed in the reply from Hans Nasten in Appendix C.

This new interrupt handler was only capable of capturing up to 256 characters into a circular buffer. This would be inadequate in the case of the PC being heavily loaded with other processes. For example it would be difficult for the captured data to be processed if a large series of CPU intensive programs were run on the PC. The CPU would be shared among them thus there would be a greater delay in being able to process the captured data. By having a buffer large enough, this lets the PC schedule other jobs without them monopolising the CPU, and if there is any extensive delay in getting back to analyse the data it will still be stored in the buffer.

• Modifications to the Interrupt Handler.

The routines obtained were written in 8086 assembly language and C. Both of the assembly code and the C source code had to be changed to increase the buffer size from 256 characters. A more realistic size for the buffer is 32 Kbytes whereby the buffer can hold a vast amount of data without the need for immediate processing. This required change led to the source code of the interrupt handlers to be changed. The reason behind the need for major change to his source code was the restriction imposed on the number of characters in the buffer. The restriction was in the form of storing the buffer size in an unsigned character. This limits him to $2^8 = 256$ characters stored in the buffer. By changing this to a word size in the assembly code allowed up to $2^{16} = 65536$ characters but a restriction of 32768 characters is enough. By changing this the indexes to the buffer also needed changing both in the assembly code and the C functions. A test for when the limit of 32768 characters was reached was included so that the pointers into the buffer could loop back to 0 so we have a circular buffer. With the increase in the size of the buffer the indexes in the assembly code had to be modified. This consisted of changing how the index number was stored in the register. It previously masked itself to clear the upper half of the register, and only used the lower half of the register, (bl), to store the index. This was modified so the index used the entire register (bx) to store the index.

These newly modified routines were the basis for capturing the data reliably and could be used for both of the communication ports simultaneously. The data being appended to one end of the buffer does not affect data being accessed freely from the other end. The analysis of the data can then be carried out in the next phase to see exactly what data was being transmitted down the lines while new information is still being appended. The actual physical connections of the two processes to the Tellabs Network can be seen in the next section.

• Port Configuration

The actual connections to the Tellabs Network needed a great deal of experimentation with cabling and the configuration of the port at both ends. The connection for both processes were between two RS-232C ports, one on the PC, the other attached to a port of a local dataplexer. With a datascope, (built into a laptop computer), attached between the PC and the host port, analysis of the line could be carried out. By experimenting with the cabling and the configuration of the ports we tried to establish flow control between the dataplexer port and the PC. This was unsuccessful after some considerable experimentation. The conclusion that was found

was that after a line of data was transmitted by definition of the carriage return character, the Tellabs network management system processed the line thus flow control between the two was not held to prevent more data from coming in. This could be further concluded by experimentation with command terminals connected directly to the network.

With the help of the datascope another interesting fact emerged. The network only required the carriage return character to terminate a line of input while it returned both carriage return and line feed characters. It was crucial to know the format of the data returned by the network for the lexical analysis phase of the data analysis, discussed next.

To resolve the problem of flow control a way to suspend the flow of data to the network is required. This is in the way of a short delay at the end of every line of data being sent into the network. The delay set up is long enough in case of very heavy network loading whereby it would take even longer to process the interrogation query.

So with the data being captured accurately from the network the next stage is to analyse the data to make more efficient use of it.

3. Data Analysis

The Network Monitor as described before consists of two totally independent data capturing processes. The first of these processes captures datagrams sent from the network management system. There can be two such Event-Logs configured by the Tellabs network management system. At present only one is configured to a line printer. The second such Event-Log could have been used for the Tellabs 320 Network Monitor package but instead this project will use it. The second process interrogates the lower level multiplexers at regular time intervals which have terminals and printers attached to them. From this data information pertaining to the individual channels can be further analysed.

The analysis of the data is similar for both processes except for the structure of the data being analysed. The raw data is obtained from the buffer which the interrupt handler routines write to. This raw data is stripped to its bare essential and written to specific files depending on the format of the datagram recognised. An example of these datagrams from the Xplexer is shown below in diagram 5.

03/20/91 17:25

331:14.07 002/000	Link S Frames(x100)	180 m	567 T	451 R
331:14.08 002/000	Link E Frames	180 m	7 T	0 R
331:14.09 002/000	Link Tx Utilization %	180 m	3 A	7 P
331:14.10 002/000	Link Rx Utilization %	180 m	6 A	27 P
331:14.11 002/000	Link Virtual Circuits	180 m	8 A	10 P
331:14.05 002/001	Link Characters(x1000)	181 m	777 T	1465 R
331:14.06 002/001	Link I Frames(x100)	181 m	409 T	522 R
331:14.07 002/001	Link S Frames(x100)	181 m	544 T	439 R
331:14.08 002/001	Link E Frames	181 m	8 T	0 R
331:14.09 002/001	Link Tx Utilization %	181 m	5 A	12 P
331:14.10 002/001	Link Rx Utilization %	181 m	10 A	31 P
331:14.11 002/001	Link Virtual Circuits	181 m	9 A	11 P
331:14.01 002	Sys. Packets/Sec.	180 m	39 A	67 P
331:14.02 002	Sys. Virtual Calls	180 m	25 A	29 P
331:14.03 002	Sys. Buffer Utilization	180 m	7 A	9 P
331:14.04 002	Sys. 0 LC Buffer Util	180 m	20 A	20 P
331:14.05 002/002	Link Characters(x1000)	181 m	709 T	526 R
331:14.06 002/002	Link I Frames(x100)	181 m	228 T	213 R

03/20/91 17:28

```
 331:13.00 002/002.008                                  Channel Down
 331:18.00 002/002.008     0 Tx        0 Rx             Call Disconnect
                                                        (Normal Disconnect)
 331:15.00 002/003.028 "gotops"                         Call Dial
 331:17.00 002/003.028 001/002.024                      Call Connect
03/20/91 17:30
 331:18.00 002/000.001     0 Tx        0 Rx             Call Disconnect
                                                        (Normal Disconnect)
 331:18.00 002/003.019   506 Tx 22967 Rx                Call Disconnect
                                                        (Normal Disconnect)
03/20/91 17:33
 331:20.00 002/003.022  1165 Tx 50442 Rx                Count
 331:13.00 002/002.008                                  Channel Down
 331:18.00 002/002.008     0 Tx        0 Rx             Call Disconnect
                                                        (Normal Disconnect)
 331:20.00 000/005.006  3321 Tx 49716 Rx      Count
 331:19.17 002/003.019                                  Call Timed Out
```

Diagram 5. Example of datagrams produced by the Xplexers.

This process is in two phases, the first of which is the lexical analysis phase which reads each individual character and tries to form recognizable combinations of characters. The second phase is to parse the recognized combinations of characters until recognizable datagrams are found. Upon finding one, the selected contents are written to that file. In the situation of an unrecognizable string of characters, the process will wait until it can resynch itself with the next time stamp as a form of error recovery in the case of capturing the data from the Xplexers. In the case of capturing the data from the dataplexers the processing of the data is terminated, the buffer is cleared and then interrogation of the next dataplexer is started. The actual structure of this input from the dataplexer is somewhat less structured and varies a great deal more than that of the Xplexer's so error recovery is far more important. A more detailed explanation of these two data analysis operations follows.

· Lexical Analysis Phase

The lexical analysis phase takes individual characters out of the buffer supplied by the communication port interrupt handler, and proceeds to try to make matches against well structured combinations of characters.

The matches are based upon checking the characters in the input against regular expressions until the longest possible match has been found. Such regular expressions may be of the following form:

```
{D}                    [0-9]          =    any digit,
[1-9][0-9]* | [0]                     =    which translates to either
                                           a '0' or a number
                                           beginning with a 1-9
                                           followed by an optional
                                           number of 0-9's,
{D}{D}"/"{D}{D}"/"{D}{D}              =    a date field.
{D}{D}":"{D}{D}                       =    a time field.
```

The lexical analysis stage takes advantage of a UNIX software tool called LEX. This stage was done on the SUN Workstations running SunOS at Canterbury University. What LEX does is to take a series of regular expressions, like the ones above, as its input. When LEX is run over it the end result is the equivalent portable C

source code. This is portable in the sense that the produced code can be taken to the PC and compiled there without any modifications.

The input to LEX has been changed from reading its input directly from keyboard to reading each character from the buffers supplied by the interrupt handler for the communication ports. This was necessary because of the nature of the input being from the buffer. LEX as a requirement needs to be able to push this data back onto the source from where it came. Since the source was changed, a stack was implemented to take this data back temporarily until it was needed again. So when the next character is needed the stack is checked first. If there is a character then it is taken otherwise the next character is taken from the buffer.

More information about regular expressions can be found in many computer science textbooks, for example in the SUN Workstation manuals[1].

Essentially what the lexical analysis phase does is to take each character at a time until it can find the longest match to one of the regular expressions. Upon finding a match, the action to the right-hand side of it is carried out. This may be nothing at all or to return a *token* to the parser, which will be discussed next. The token itself is just a symbol to represent what regular expression has been matched. For example:

```
{D}{D}"/"{D}{D}"/"{D}{D}        return(DATE);
```

When a sequence of characters matches what looks like a date, the lexical analyser returns 'DATE', a token, to the parser. If however the lexical analyser doesn't match anything then it just keeps looking for the next match.

With the tokens being passed back from the lexical analyser stage, the next stage is to look at the structure of the returned tokens until a datagram is matched or an error occurs whereby recovery will take place. This applies to both the event log and the command port in analysing the structure of their data.

· Syntax Analysis Phase

The second stage of data analysis consists of examining the tokens returned from the lexical analysis phase. These form the basis for the syntax analysis phase. As previously mentioned these tokens returned represent a match of a sequence of characters like a 'date' or a 'time'. A BNF (Backus Normal Form) syntax description of the input was worked out for each of the Event-Log and the command port processes. This is used to match sequences of these character strings until complete datagrams are matched. A BNF description is a formal way of showing how all the possible combinations that these matched strings of characters can be sequenced.

An example of how these tokens can be matched to form valid datagrams for the Xplexer node statistics is shown below in the BNF description of the datagrams as the input.

```
Event_Log   :   Time_Based_Datagram
            |   Event_Log Time_Based_Datagram
            |   error Event_Log
            ;
```

This shows that the Event-Log consists of a single or a series of datagrams in a group beginning with a time stamp. The other case is if an invalid datagram is matched. Here if it is invalid, the recovery process is to wait until the next time a time stamp appears.

---

[1] See references.

```
Time_Based_Datagram     :   Time_Stamp
                        |   Datagram_List
                        ;
```

The next stage is in breaking up the definition of a Time_Based_Datagram into a time stamp followed by a datagram list.

```
Datagram_List     :   Xplexer_datagram Datagram_List
                  |   Datagram_List Xplexer_Datagram
                  ;
```

The datagram list is defined as one or more datagrams.

```
Xplexer_Datagram  :   one of many rules to
                      describe the datagrams
```

The last definition in the BNF description is simplified to show the definition of specific datagrams.

Due to the nature of the input from the Xplexers being an infinite source, the grammatical rules were required to be right recursive to avoid the internal stack of the parser from overflowing.

In the case of the command port process interrogating the dataplexers, the information is not always so well structured as the Xplexer datagrams. For example when interrogating a dataplexer, there may be someone else using the port so no connection would be available. In this situation the syntax analysis phase would have to recognize that this would have happened and terminate the analysis of the dataplexer statistics.

The command port process was thus of a more inconsistent nature. The actual format of the data to be collected was known but the extra responses generated by the network in response to being interrogated was inconsistent and sometimes unknown. This led to problems which could only be resolved by waiting for these unknown responses from the network. The need to know these responses comes from the way in which the command process carries out its interrogation. The command port process waits to begin its phase of interrogation by constanty checking the contents of the file DATETIME.TXT. The contents of the file contains the current time from the network and can be used as a basis for when to start the next interrogation pass. When this is reached, the process reads the contents of a batch file to interrogate one dataplexer and sends it down the communication port into the Tellabs Network. The actual data sent is just user keystrokes used to access the required menus on a command terminal in the Network Manager's system. The responses which are thus captured in response to the interrogation are then analysed. The actual method of analysis is just the same as for the Event-Log process mentioned above except that the data format is different.

In response to matching one of the rules in the input, the action associated with the rule is carried out. This action, or series of actions usually consists of writing the required captured data to specific files depending on what the information was that was captured. The responses are similar in many respects between the Event-Log and the command port processes. In the case of invalid input both methods have their own way of recovering. The Event-Log simply ignores every subsequent datagram until it finds another time stamp. The interrogation process on the other hand stops analysis of the data whenever an error is encountered.

Below in diagram 6 is the Event-Log process in all of its stages.

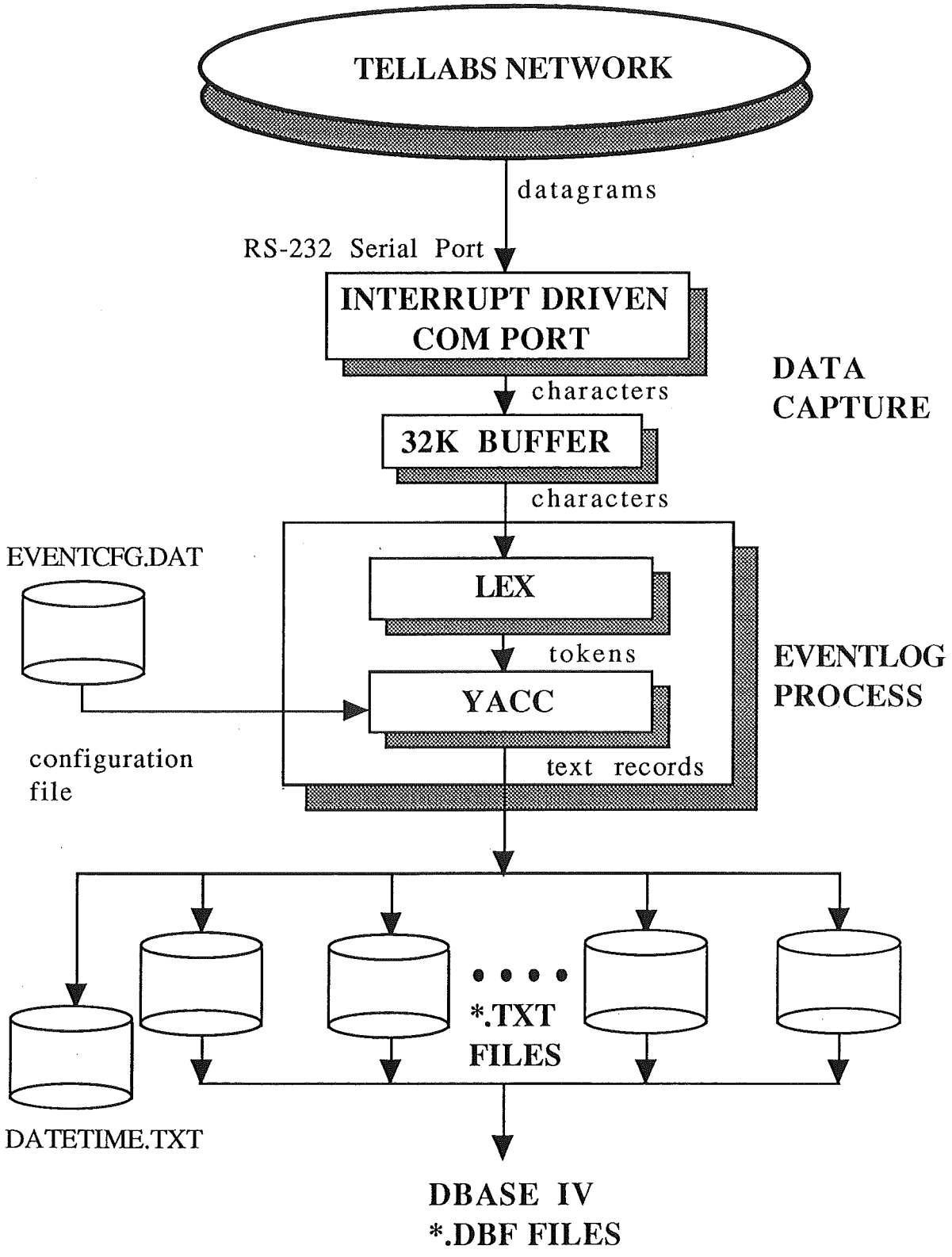


Diagram 6. Stages Of The Event Log Process.

We can look at the stages of the Event-Log process to see how it all fits together. The datagrams sent from the network are captured by the interrupt handler at

the communication port and placed into a buffer. This was the data capture stage as described in section 2. The next stage is the data analysis stage where the characters stored in the buffer are read through the lexical analysis stage to form recognisable strings. The last stage is the syntax analysis stage just described. The actual software used is BYACC which is a PC version of the SUN tool YACC. These tools take the BNF description of the input and return the equivalent code in C. When valid datagrams are recognised the relevant information is written to text files whereby they can be appended into database files at a later time. Both LEX and BYACC produce C source code for a table driven parser which is included with the rest of the code for each of the two processes.

The interrogation process below in diagram 7 uses the same method of data capture and analysis as the Event-Log process. The only difference is the length of time that the parsing of the input required. The Event-Log is an infinite source of datagrams while the interrogation process just processes the results of one interrogation of a dataplexer at a time then it exits until the next interrogation.

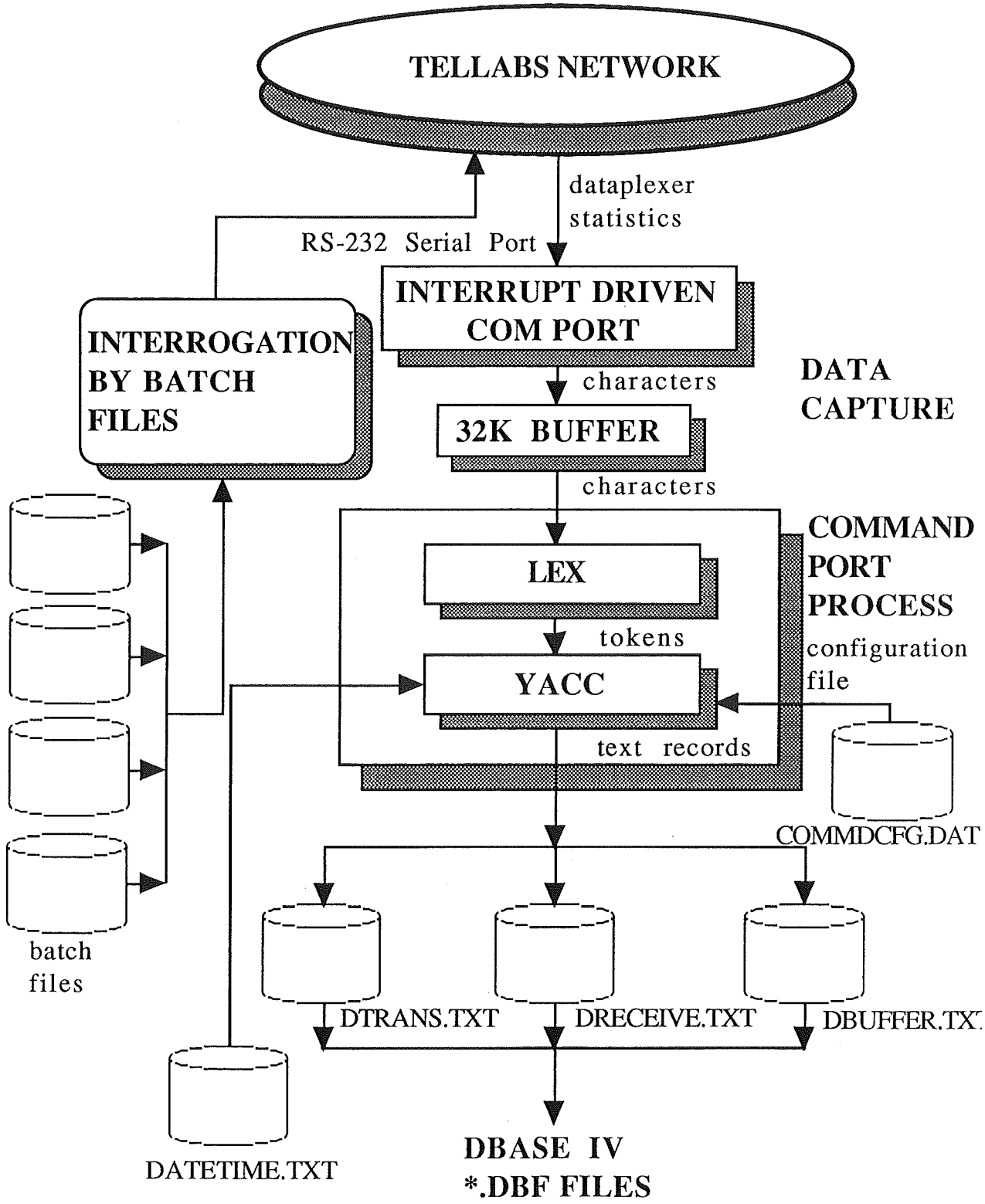


Diagram 7. Stages Of The Command Port Process.

The actual data sent to the network needed to get the responses necessary from the network consist of menu commands contained in batch files. These are just transmitted down the communication port when the interrogation of a port is needed.

4. File Locking

The communication between the Event-Log and the Command Port processes is via a text file containing the systems current date and time. The date and time fields are written to the text file DATETIME.TXT by the Event-Log process. The date and time fields are taken from the time stamp placed on packets at arrival time. The reason for using this time is that the network's time is more reliable than the PC's internal clock. This is used as the basis for when the second process may start its interrogation of the dataplexers. This process continuously checks the time from the file and if it exceeds the period required it then starts the next interrogation cycle of the dataplexers.

From diagram 8, a representation of the inter-communication between the processes, we can see that the two processes are making accesses to the same file, namely DATETIME.TXT.

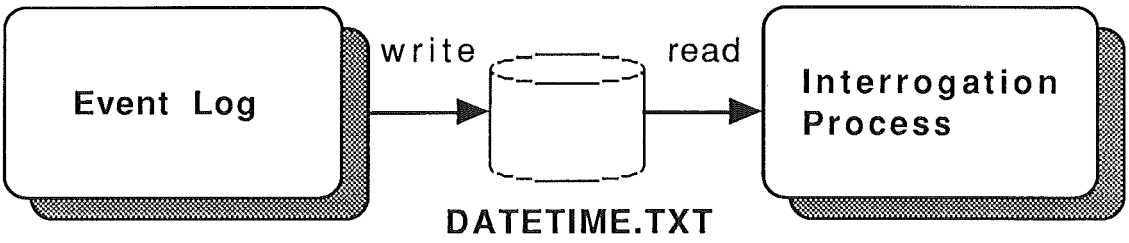


Diagram 8. Communication between the two Data Capture processes.

This leads to the need to have some data integrity. With both processes accessing the same file we can have the situation of one file trying to read from the file while the other is writing to it. This causes havoc in determining what was read from the file in the case of the file being written to at the same time.

Below in diagram 9 we can see the two processes accessing the one file and the effects of the file locking mechanisms that have to be imposed.

The Event-Log process extracts the current date and time from the datagrams it processes from the network and writes it to the DATETIME.TXT file which the interrogation process uses as the basis for when to start its interrogation of the dataplexers. In the diagram below we can see that the Event-Log process would write to the text file while the interrogation would be reading from it.

The Event-Log opens the file DATETIME.TXT, and waits until it can place a lock on it. This is done using some compiler dependent locking functions. When it is successful it writes the information to the file, unlocks it using the same functions and then closes it. If the interrogation process tries to place a lock on the file during this period in order to read from it, it must wait until the lock is released. We can see these critical sections by the shaded boxes below.

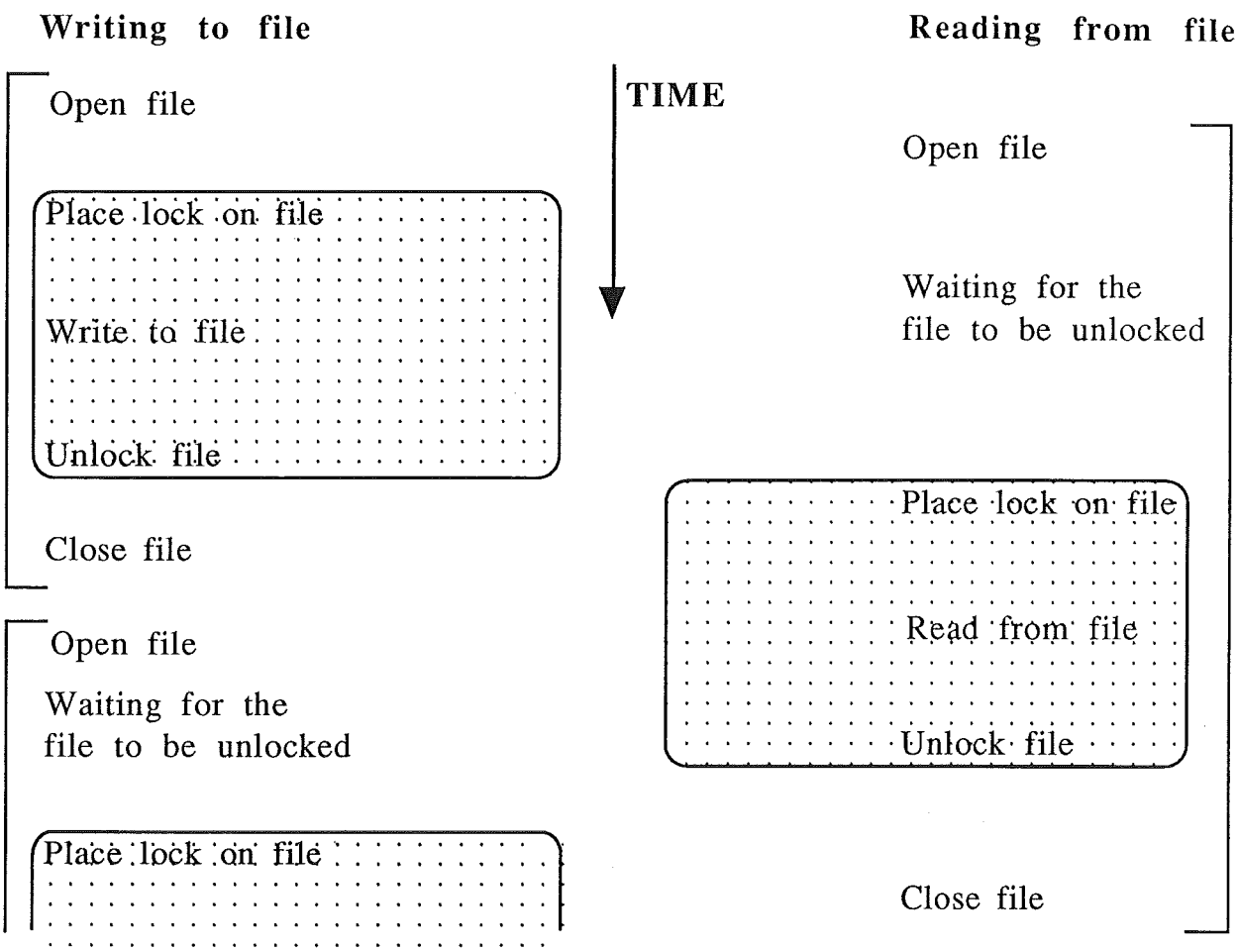


Diagram 9. Accessing Files via Locking Mechanisms.

5. Database Design and Construction

· Requirements

The objectives of the Network Monitor is to produce statistics on the performance of the Tellabs network and to help in finding possible problems.

To produce accurate results on the network performance requires the data to be captured as reliably as possible. This was achieved by using interrupt handlers as was shown in the data capturing stage of the project in section 2 above. The data is captured by the two processes as described earlier, namely the *event log* and *command port* processes.

There is a great deal of data redundancy in the input from the Tellabs network. This redundancy is removed at the lexical analysis stage. The actual redundant information mainly consists of descriptive information about the datagrams from the Event-Log process and headings of the statistical reports returned by the network management system from the interrogation process. This redundant data can be obtained from having a pre-existing database file containing the relative information about each of the datagrams. The specific textual description can be obtained by a field in the datagrams which describes the type of datagram, i.e a foreign key relating the

datagram to the primary key in the textual description file. As stated this is redundant information and would waste valuable disk space when it is easily retrievable from the database.

There were several options available for the database management system to use. dBase IV seemed the most accessible and was an industry standard. The actual version used was the Developers Edition v1.1. This allows the final product to be used without having to buy the dBase IV product for the company. This is done by creating an object code module which is run with the runtime libraries.

The database application can be broken up into two separate stages, the design of the separate modules to the database and the application which holds these individual modules together.

The actual design phase of the database consists of several separate sections. These are the design of:

- the database files,
- the queries,
- forms,
- report layout,
- application program.

The fist two of these can be seen in more detail in the accompanying TELLABS NETWORK MONITOR - SOURCE CODE LISTING manual. This gives the structure of the data for the files and the format of queries.
The last three items can be seen in more detail in the accompanying TELLABS NETWORK MONITOR - USER MANUAL. This shows the format of the forms, reports and the Network Monitor Application program.

• User Queries

Queries on the database require user input as to which aspects of the network will be extracted for the reports. The user is prompted with an input screen to select desired ranges for the search on the relevant database files. The user input screen initially contains default values consisting of the option for all information to be extracted. An example of such a menu driven interface can be seen in the illustrations of the NETWORK MONITOR - USER MANUAL and on the next page in diagram 10. These default values are extracted from a database file called MEMVARS.DBF and loaded into memory variables. The actual field values shown in the user input form are the memory variables so when a field value is changed, the memory variable is automatically updated. These memory variable values are then used in the actual query on the database file(s) to filter out the required tuples in the database. The actual memory fields, prefixed with "M_" for memory variable, are:

M_FROMDATE	M_TODATE
M_FROMHOUR	M_TOHOUR
M_FROMMINS	M_TOMINS
M_FROMNODE	M_TONODE
M_FROMLINK	M_TOLINK
M_FROMCHAN	M_TOCHAN

They can be seen in the input form below in diagram 10 for queue statistics.

TELLABS NETWORK

QUEUE STATISTICS SELECTION MENU

FROM DATE :	01/01/1900	TO DATE :	31/12/2050
FROM TIME :	00:00	TO TIME :	23:59
FROM NODE :	001	TO NODE :	001
FROM LINK :	009	TO LINK :	009
FROM CHANNEL :	003	TO CHANNEL :	003

Diagram 10. Input Screen for Queue Statistics.

Here we can see the need for the complex user queries on the database to effectively extract useful information on the networks performance. The query must be as flexible as possible to cover as large or as small an area as required. The retrieval time from the database can be affected from these complex queries. The memory variables are used to filter out rows in the database (tuples) based on the values in specific rows. For example the DATE field can be filtered by using the M_FROMDATE and M_TODATE memory variables. For example we can specify that the date being searched is >= M_FROMDATE and <= M_TODATE where the two memory variables are the values the user inputs in the above form as the two ranges for the search.

The reason for having to use memory variables is due to the nature of dBase IV's queries. This refers to Query By Example used from the control centre in dBase IV. The reason is that every file needed in a query must be linked to at least one other file by the use of a foreign key. This does not allow for the use of a file such as MEMVARS.DBF described above for the pure reason of using its values as filters in the query. This type of query would however be trivial using another query language such as SQL where the *join* of the database files is not explicitly required to match the foreign key in one database with the other database. A dBase IV application program can be written in one of three ways:

- using the control centre, a menu driven application generator which also allows the embedding of dBase code,
- using explicitly dBase code, or
- using SQL which dBase IV supports.

Due to the nature of dBase IV, SQL is a separate entity and cannot be embedded in dBase code which is what we would have liked. Likewise the dBase code cannot be embedded into the SQL code. This seems to be an unfortunate shame on the side of dBase IV as both systems have their own benefits and pitfalls and it would be a more powerful and useful system if both could be intertwined. So the actual method of construction used was via the control centre. This method is very good for prototyping developments and for producing applications quickly.

· Design of the Database

The design of the database application itself can be broken up into three main areas:

- the configuration of the existing network,
- the production of text based statistical reports on the system performance, and
- updating the database by adding new records.

· Configuration

The configuration of the network is a means of labelling the nodes, links and channels in the network for ease of reference. This labelling gives a more user friendly meaning to the network as a whole, instead of having to view the network as a sequence of unfriendly numbers. Reports on the topology of the network can be produced and also the provision for queries into the network is available. For example this might be "what is the name of link 5 which goes from node 0". This would return to the user that the link connects Newton Data Centre to Sydney. The configuration also allows new features to be added or removed via the update submenu.

· Statistics

This is the main area of the Network Monitor, it consists of the production of text based reports based upon the data captured from the network itself. The basic layout of the reports were specified by the Mount Cook Group Ltd. One requirement was to produce the name of the node, link and channel if appropriate on the report with each entry. By doing this the observed query access time was dramatically slowed down. This was because for each entry in the report, a search was needed for the node name and another one for the link name. Each of these searches, on different database files, lengthened the time needed to process the query. Due to this excess time it was better to do away with the extra information in favour of the faster retrieval speed.

The statistical reports can be broken up into two sections, one for Xplexer results and one for dataplexer results.

The Xplexer statistics consist of reports generated by user queries on node statistics, link statistics, and queue statistics. The dataplexer statistics consist of buffer utilization and channel throughput.

6. Updating Database

Since the Network Monitor application program does not have to be running all the time, the data from the analysis stage is written to text files. These files at some point in time must be updated into the database. This can be either to update all the files into the database or to update selected files.

The problem with updating the text files into the database is that once again we can have the situation of being able to read and write to the text file as it is being updated.

As we saw before a method for keeping data integrity is once again needed. This is in the form of placing a lock on the file before any action is taken on it. Since dBase IV does not support any feature needed to perform this operation, some other method was needed. A small C routine was written which opened up the text file to be updated and placed a lock on it. After this the entire contents was copied to a temporary

file and the original file size was set to 0. The lock basically stops any conflicts over access to the file. By copying the contents of the original file to the temporary file, the lock on the original file can be removed immediately. At this point the temporary copy of the original text file can be updated into the database by using the dBase IV option '*append*''. This will have no conflicts as no-one else should be accessing it. When the file is finished, it is removed by the MS-DOS command DEL. The program created to do this updating of the original file to the temporary file is TXTTOTMP.EXE.

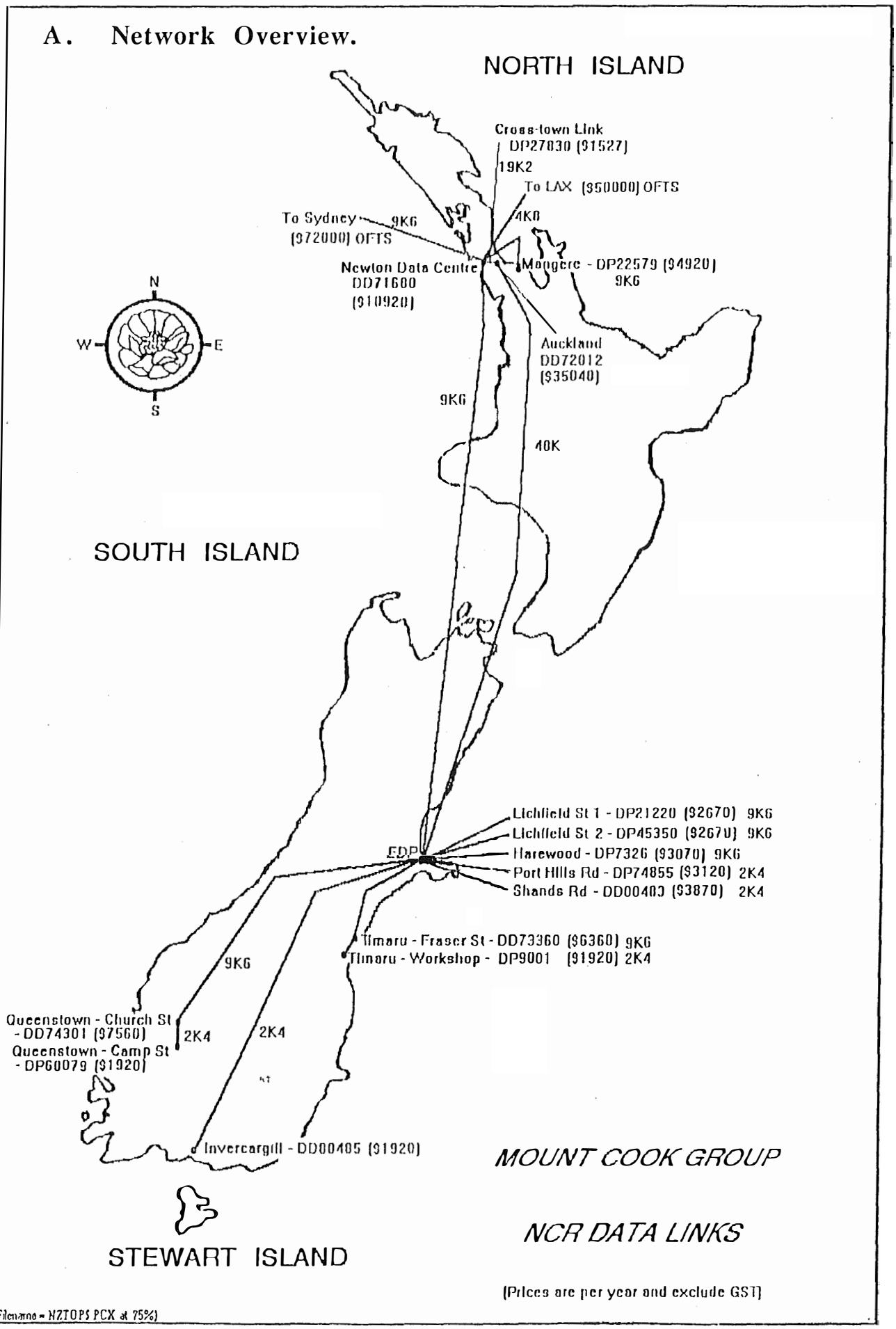
Another option in the updates menu is to clean up unused files. This refers to some of the files not being used but the text files are being appended to. This refers to the files not being used for reports etc in the current application. The program DELFILES.EXE removes the file by opening it and changing its file size to 0.

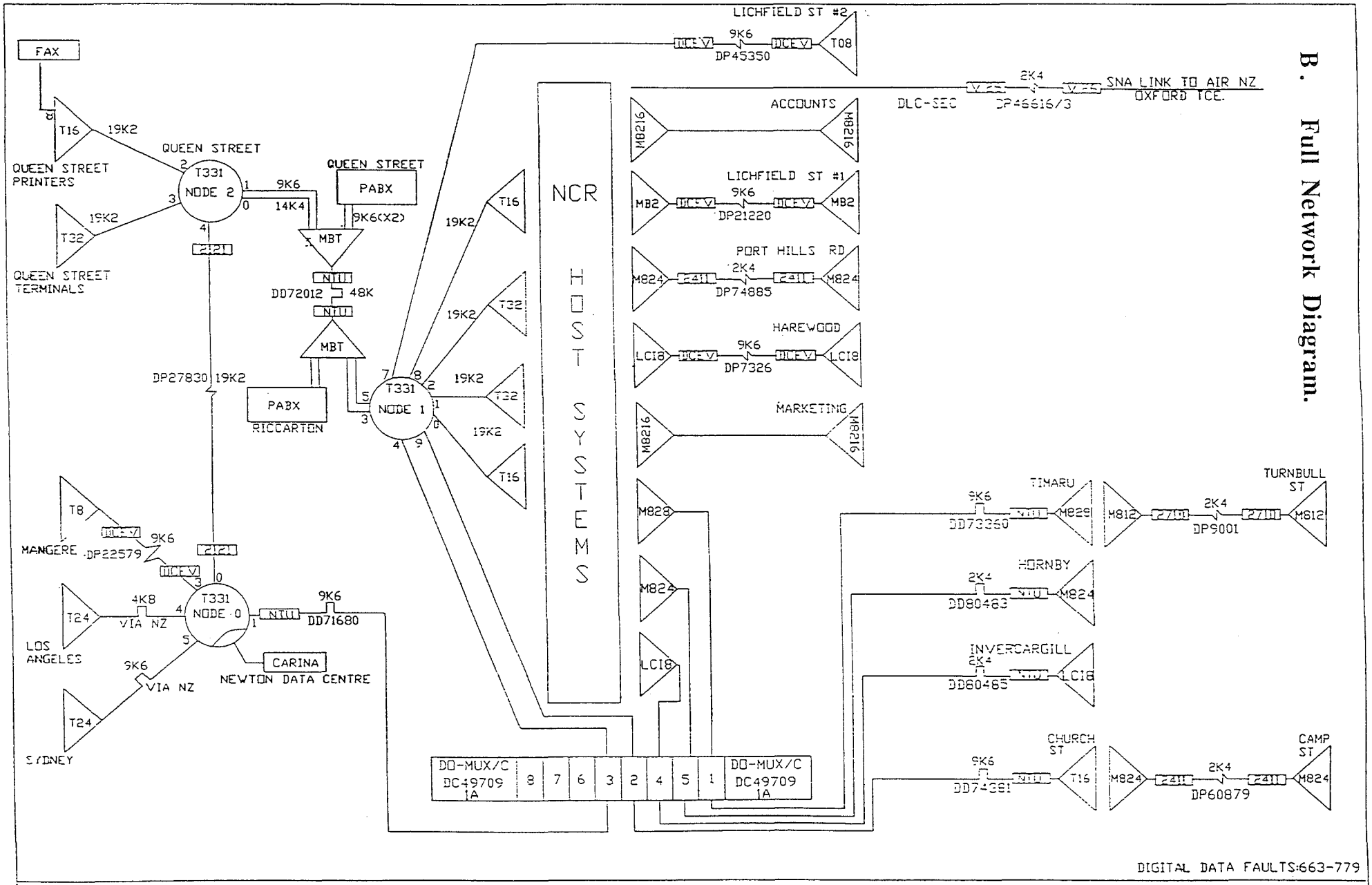
7. Conclusion

Although the Tellabs Network Monitor captures and produces text based reports, there are several possibilities for producing graphical representations. Dbase IV itself cannot produce any graphical output short of lines and boxes but there are ways of producing higher quality graphical output. One such way is by use of a product made by Ashton Tate, the makers of dBase IV, who supply a graph potting 'add on' to dBase IV. Another cheaper option to Mount Cook is to use Microsoft EXCEL to read in the dBase files and plot the required features.

Appendix

A. Network Overview.





THE MOUNT COOK GROUP LTD
COMMUNICATIONS NETWORK

DRAWN 28/09/91 VERSION 7.1
EFFECTIVE 28/09/91
NETWORK.AS.MRH

C. Interrupt Handler Routines - permission of use.

From nasten@gurkan.Stupi.SE Thu Mar 28 03:33:23 1991
Received: from csc.canterbury.ac.nz (cantva) by cosc.canterbury.ac.nz (4.1/SMI-4.0)
id AA01322; Thu, 28 Mar 91 03:33:20 NZS
Return-Path: nasten@gurkan.Stupi.SE
Received: from gurkan.Stupi.SE by csc.canterbury.ac.nz; Thu, 28 Mar 91 03:35
+1200
Received: by gurkan.Stupi.SE (5.61/1.35) id AA00369; Wed, 27 Mar 91 16:30:32
+0100
Date: Wed, 27 Mar 91 16:30:31 MET
From: nasten@gurkan.Stupi.SE
Subject: Re: COM ports
To: lindsay@cosc.canterbury.ac.nz
Message-Id: <9103271530.AA00369@gurkan.Stupi.SE>
In-Reply-To: <9103270838.AA00581@cosc.canterbury.ac.nz>; from
"lindsay@cosc.canterbury.ac.nz" at Mar 27, 91 8:38 pm
X-Mailer: ELM [version 2.3 PL11]
Status: R

Feel free to use the com routines any way you want..

Hans Nasten
Email : nasten@gurkan.stupi.se

Palsundsgatan 3 B
S-117 31 STOCKHOLM
SWEDEN

References.

Ashton Tate	Using The Menu System, 1990.
Ashton Tate	Various dBase IV v.1.1 Reference Manuals.
Dettmann, Terry	DOS Programmers Reference Guide,
Kernighan,B.W. and Ritchie, D.M.	The C Programming Language, Prentice Hall Software Series, 1988.
Pratt, Phil	Microcomputer Database Management Using dBase IV, Boston, 1990.
SUN Microsystems	Programming Utilities for the Sun Workstations.
Tellabs,	Xplexer Manuals.
Tellabs,	Dataplexer Manuals.
Tellabs,	Tellabs 320 Network Monitor - users manual.