University of Canterbury

1980

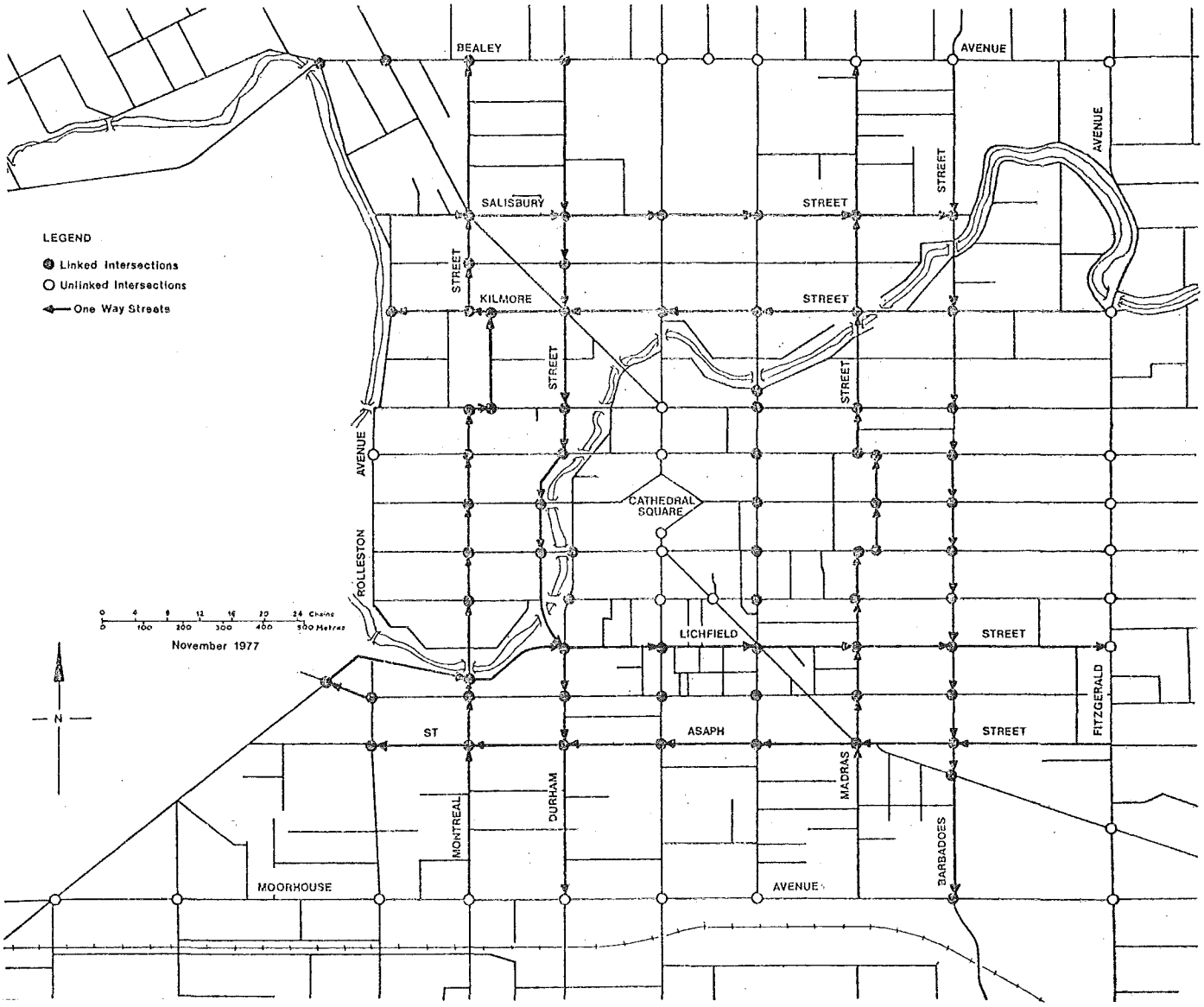
PROTOCOL FOR A COMMUNICATIONS NETWORK

Final year project for

Bachelor of Science with Honours

in Computer Science

by W. J. Wright.



Map of central Christchurch

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 THE PROBLEM

The Christchurch City Council is upgrading their co-ordinated traffic signal control system to allow a more flexible method of controlling urban traffic flows. Essentially the upgrade consists of replacing the existing master controller with a minicomputer data concentrator, and the introduction of microprocessor based devices to interface between the data concentrator and the existing signal installation hardware. Figure 1.1 shows the existing traffic control system.

This project deals with one specific aspect of the upgrade, namely the design of the protocol for the communication links between the data concentrator and the network of microprocessor based devices. In assisting with this upgrade I was responsible for the design of the protocol, while Council Engineers managed the hardware design and selection for the new system.

1.2 WHAT IS A PROTOCOL ?

A protocol is a set of rules which specify the format and relative timing of messages between two communicating processes. A protocol is also known as a Control Procedure or a Line Discipline. ([8])

McNamara ([7] pp 191-192) defines the following as the main communication system problem areas which the rules of a protocol should solve.

1. Error Control - The protocol should have a policy for dealing both with messages that have errors detected in them and with messages which are correct.

2. Sequence Control - Messages should be marked in such a way that they can be properly identified. This helps to avoid duplicate messages and can be used to avoid sequencing

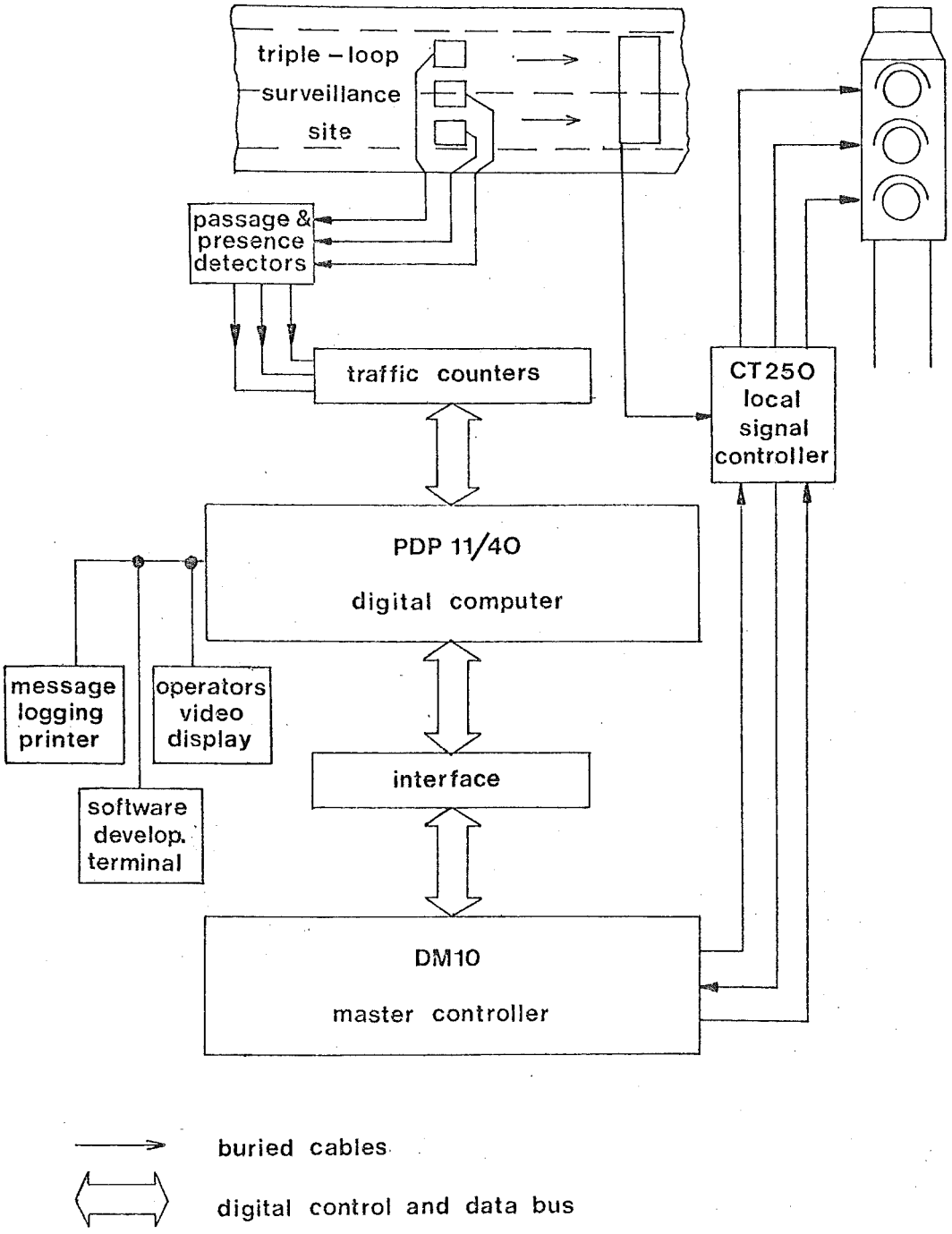


Fig. 1.1 Present Traffic Control System using DM10 Master Controller

problems between stations.

3. Transparency - The transmission of information, containing bit patterns resembling control characters, in such a way that the receiver of the information does not identify the bit pattern as a control character.

4. Timeout Control - When a receiving station does not respond to a message from a transmitter station within a certain time period the receiver is said to have 'timed-out'. Timeout control is the action to be taken by the transmitter in this situation.

5. Startup Control - Starting the transmission of messages in a communication system that has been idle.

The design of the protocol for this network involved considering each of the above problems and the methods available for solving them.

1.3 OVERVIEW OF THE PROPOSED SYSTEM

To aid with the understanding of the project, this section gives an overview of the proposed system. Figure 1.2 shows a block diagram of the new traffic control system while figure 1.3 shows the main hardware components of the new system. A description of each major component follows -

1. HOST COMPUTER - A PDP11/40 minicomputer which is the master computer (HOST) for the new system.

   Features -

   - 124K words of memory
   - 27.5 megabytes of on-line disk storage
   - magnetic tape unit
   - 7 VDU'S and 6 printers
   - IAS Operating system V3.0

   The HOST is responsible for running the entire traffic signal control system and is also used for system development work and data processing for the City Engineer's department.

2. DATA CONCENTRATOR (DC) - A PDP11/04 minicomputer which is used to concentrate data from the various signal

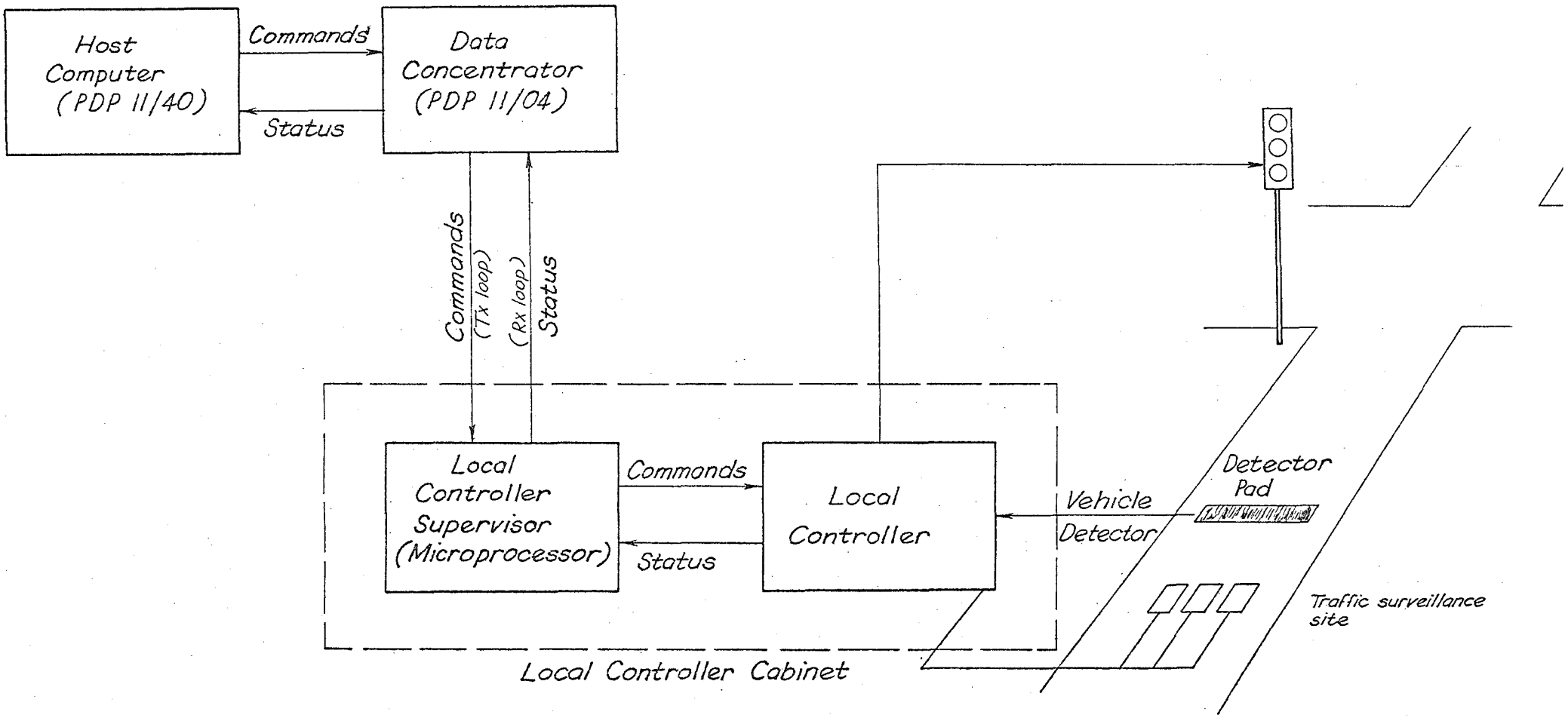


fig. 1.2 Block diagram of the New Traffic Control System

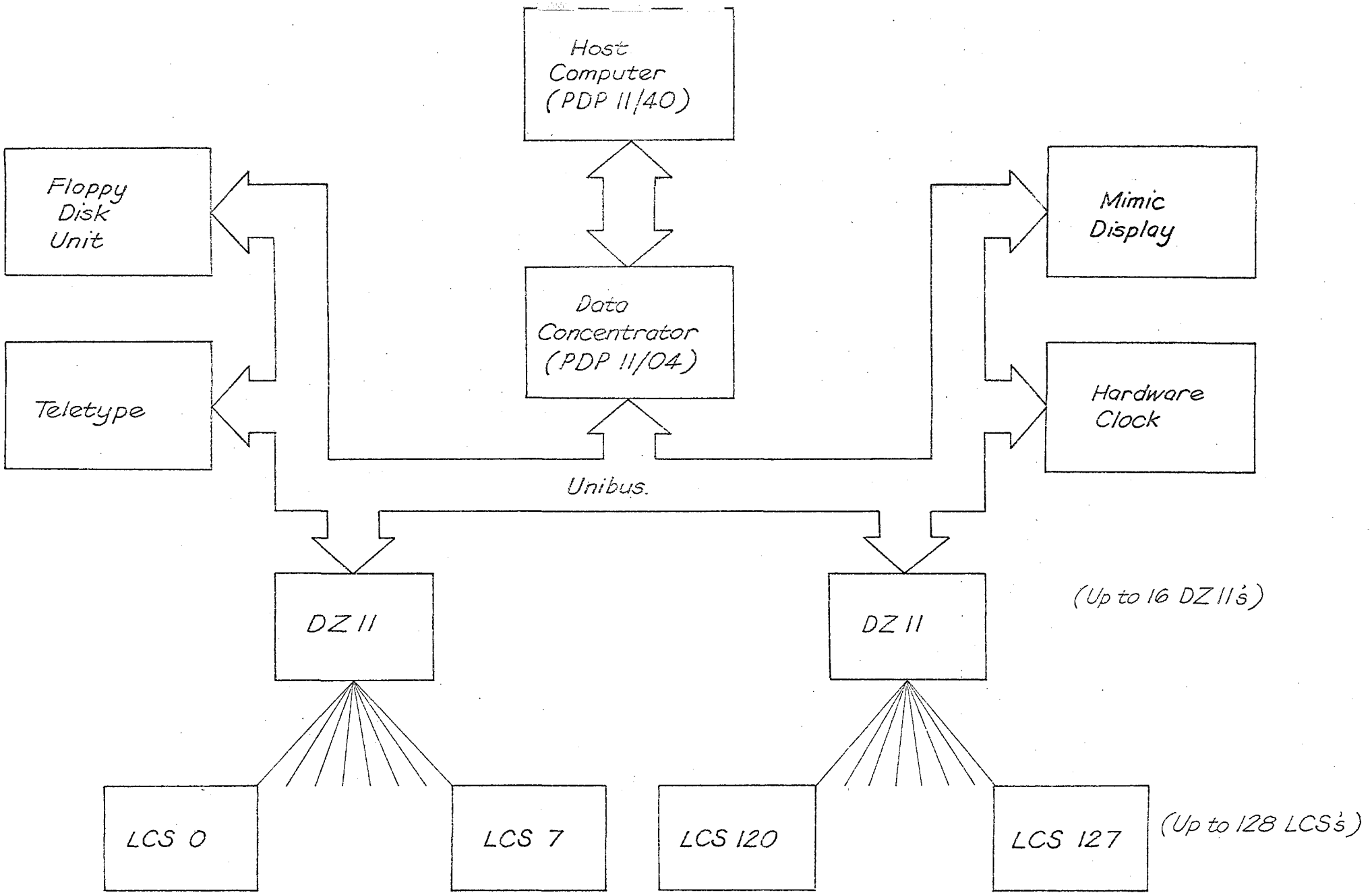


fig. 1.3 Block diagram of the Control Centre Hardware

installations around the city. In addition, should the HOST be non-operational, the data concentrator will be capable of running the traffic signal control system temporarily, according to simplified versions of algorithms used in the HOST.

Features -

- 28K words of memory

- 0.5 M bytes of on-line floppy disk storage

- RSX11-S Operating system V2.2

3. LOCAL CONTROLLER SUPERVISOR (LCS) - A Motorola M6802 microprocessor based device which supervises the operation of a local controller at a particular intersection. The LCS resides in the local controller cabinet.

LCSs receive messages from the data concentrator. A message can instruct the LCS to perform one of several tasks. These include reading the traffic counters and sending the data back to the data concentrator, issuing a signal to the local controller to change a phase at an intersection, or reporting the hardware status of the local controller and/or LCS.

Each LCS contains 4K bytes of RAM and 1K bytes of UV-EPROM. The RAM must be loaded from the data concentrator with traffic control programs which can be run by the control centre at the operator's request. The EPROM will contain a basic communication package and a collection of commonly used subroutines.

4. LOCAL CONTROLLER - The local controller is the electronic device which changes the phases at an intersection, when requested by the LCS.

If the LCS fails, either through a hardware or software fault or a power failure, the local controller automatically takes control of phase changing at the intersection, basing it's decisions on a simple system of time delays and vehicle detection. The local controller is said to have gone into IVA (Independent Vehicle Actuation) mode. In this mode, the operation of the signal installation is independent of any other installation.

5. DZ11 MULTIPLEXORS - The DZ11 is an asynchronous multiplexor that provides an interface between a PDP-11 processor and eight asynchronous communication links. In this network each DZ11 will interface eight LCS's to the data concentrator.

The DZ11, because of it's placement between the parallel data path of the PDP-11 Unibus and the serial data paths of the links to the LCSs, must be able to convert between the two data formats. Detailed technical information on the DZ11 can be found in reference [3].

6. Communication links - The communication lines between a DZ11 and an LCS will be 20 milliamp twisted pair cables. The Tx loop carries the signal from the DC to an LCS while the Rx loop carries the signal from the LCS to the DC. All links will operate at 1200 baud.

7. MIMIC DISPLAY - A wall mounted map of Christchurch City spanning all those intersections which are computer controlled.

The mimic will display visually, via LEDs, the status of each of the intersections currently linked to computer. Status information will be available for intersection hardware as well as the estimated speeds and volumes of traffic passing through an intersection.

CHAPTER 2

INVESTIGATIONS INTO THE PRESENT SYSTEM

This chapter outlines the study made of the existing system, to estimate the data message flows in the planned network. Flow information defined the message handling requirements which the protocol had to meet.

It became obvious, early in the study, that the most commonly issued messages would be those concerned with setting phases and reading traffic counters. These messages were therefore more closely investigated. A description of how frequency data was obtained for these message types follows.

2.1 'SET PHASE' MESSAGES

'Set phase' messages are presently issued by the HOST to the DM10. Under normal operating conditions the flow of 'set phase' messages will be from the HOST to the DC to LCSs.

A diagnostic task, SIGDMP, was available which recorded the issuing of 'set phase' messages by the HOST. The number of messages in any given time interval varies with the time of day and prevailing city traffic conditions. Generally speaking, as traffic intensity increases, the time between phase changes becomes longer and during slack periods the phases are changed more frequently.

A worst case figure for the number of phase set messsages on the DC-LCS links was therefore obtained by running SIGDMP, during the traffic plan OFFPEAK1. During OFFPEAK1 the full cycle of phases at all computer controlled intersections is completed in the shortest possible time.

The present system contains inefficiencies which were revealed by this study; namely that local controllers in IVA mode receive a phase set command every second during a cycle. If this inefficency were removed then the number of 'set phase' messages on the DC-LCS links would be approximately halved.

Figure 2.1 shows the distribution of 'set phase' messages for traffic plans of different cycle length. In the upper diagram the cycle length is 45 seconds and 5 sites are in IVA

mode, while in the lower diagram the cycle length is 60 seconds and no sites are in IVA mode.

'Status return' messages from the LCS to the DC would be expected to have a similar distribution to phase set messages since there will be approximately a one-to-one correspondence between 'set phase' and 'status return' messages.

2.2 COUNTER SITE SERVICING

In the present system the HOST reads the triple-loop surveillance site counters directly via hardware. In the new system an additional set of counters will have to be read. These are connected to the 30-metre detectors, which lie on the approaches to all controlled intersections. It is intended that in the new system the HOST will issue 'read counter' messages to the DC which will pass the message to an LCS. The LCS will read the counters and send a message back to the DC, which will pass the message to the HOST.

Counter service messages are issued at certain fixed positions within a traffic plan cycle. Once again, the plan which cycles in the shortest time was considered, to give a worst case figure.

It was necessary to look at all counting sites and, for each, to work out where in the cycle sampling would occur. The distribution of counter service messages over the cycle could then be obtained.

In the new system a limit will be imposed on the number of 'read counter' messages issued, since otherwise there will be too much data to be processed by the HOST. A realistic estimate of the maximum number of counter service messages is thought by Council Traffic Engineers to be 11 messages per second.

The response to 'read counter' messages in the new system is expected to have a similar distribution since the two messages are directly related.

2.3 COMBINED MESSAGE FLOWS

Figure 2.2 shows the estimated average message rates in the network. It must be remembered that the numbers shown in this figure are only estimates of the average numbers of messages issued per second, during the traffic plan of shortest cycle length. The actual message flows which will occur on the links will only be known with certainty when the system is in operation. This is because of the unknown but presumably complex interactions which exist between the DC, the HOST and the LCS's and also the inability to predict the degree to which

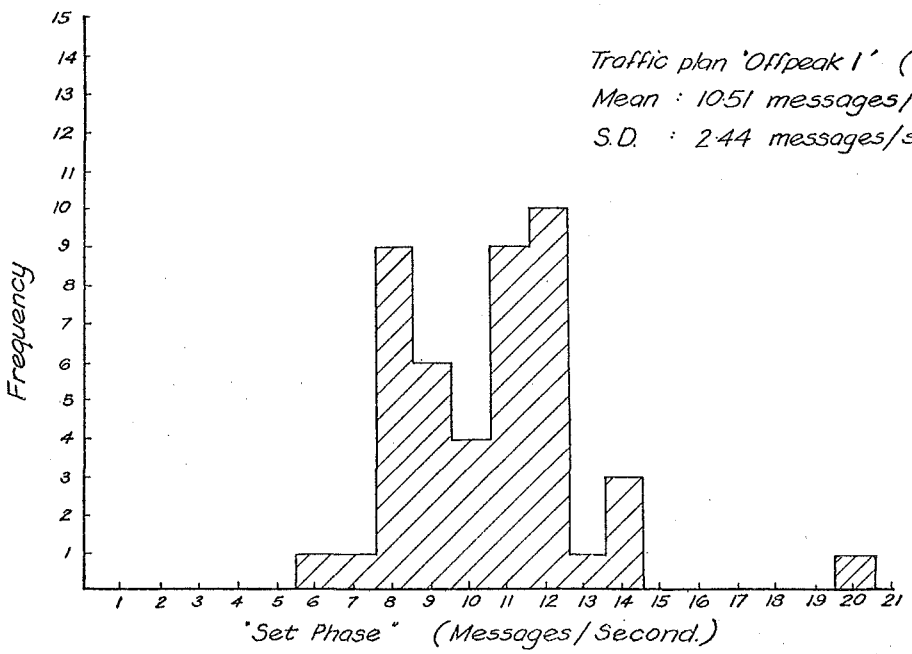


Fig. 2.1 (a)

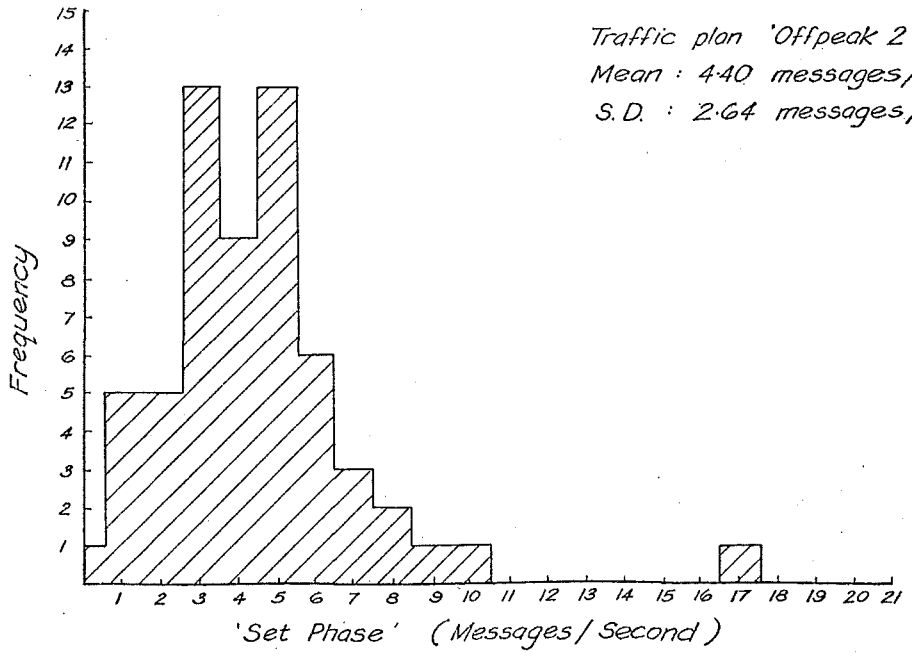
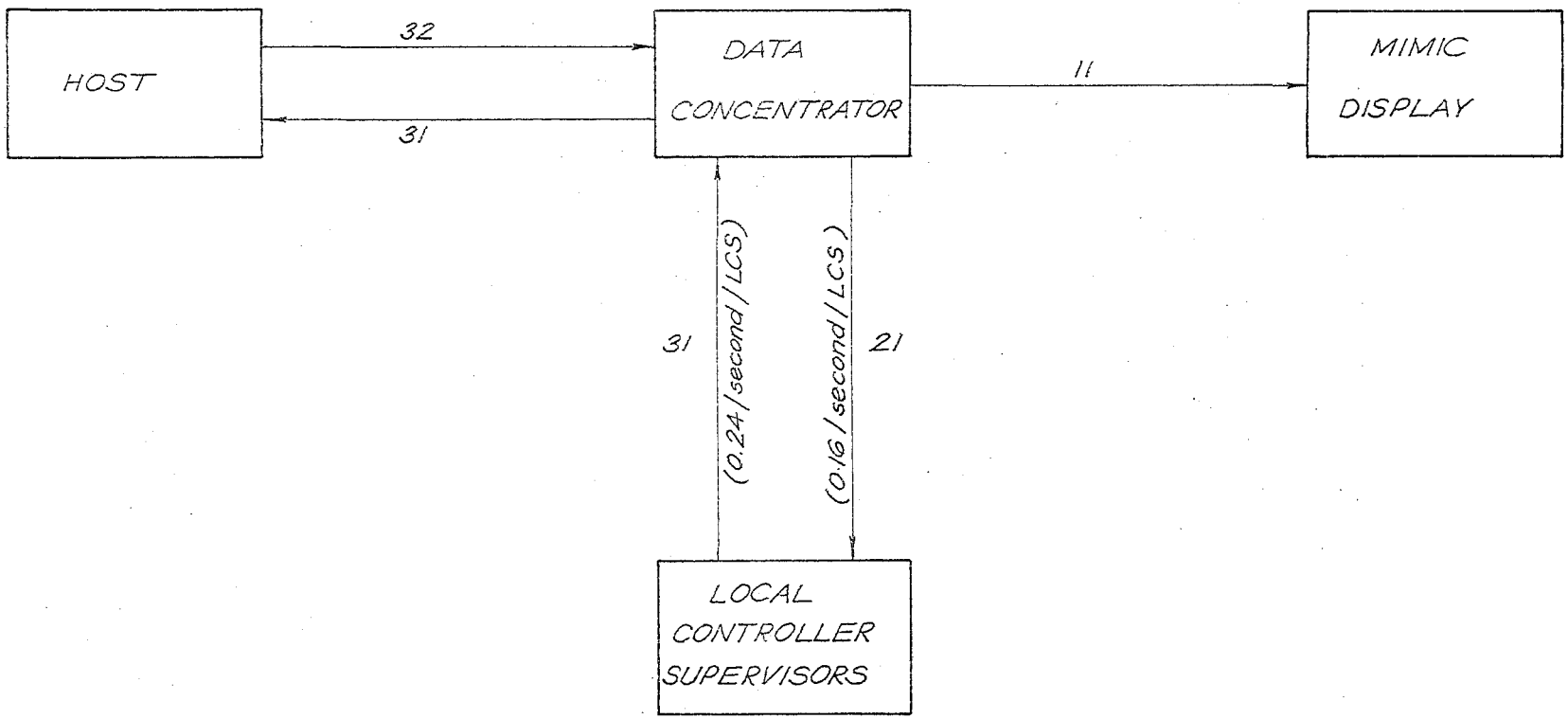


Fig. 2.1 (b)

Measured frequency distributions for the 'Set Phase' messages from the 'Host' to the DM10



Estimated average message rates

(All numbers are 'messages/sec.)

Fig. 2.2

asynchronous operations will overlap in time.

CHAPTER 3

ERROR CONTROL

This chapter discusses the different techniques of error control available and describes an experiment made to test the quality of the links to be used in the network.

3.1 ERROR CONTROL METHODS

The hardware facilities in this network have, built into them, several features to help guard against data transmission errors. These include optical isolators and various line filters as well as parity, framing and overrun checks on all characters received.

Although hardware techniques can afford protection to a communication link, there should also be provisions in the the protocol for detecting and correcting errors.

There are two broad categories of error control mentioned in the literature ([2] pp 260-267) -

(i) Forward Error Control - Sufficient redundant data is included with the information transmitted to allow the receiver to detect an error and to infer the correct information from the pattern received. Hence even if the message is affected by noise it should, in theory, only have to be transmitted once.

This method sounds easy, but in practice there are high overheads, as much redundant information needs to be transmitted with a message. Another drawback of the method is that the decoding of a received message is complex.

One application where this method has been successfully applied is the transmission of information from deep space probes to Earth.

(ii) Feedback Error Control - Some redundant data is included with the information transmitted to aid with error detection, but the correction of errors is made by retransmission of the message. This method needs less redundant information than the forward error control method and is regarded as being more

reliable in the usual communication environments. For these reasons, it is the type of error control system adopted in this protocol.

In recent times there has been much study of efficient schemes for detecting errors in transmitted information.

Some of the better known methods described in the literature ([1] pp 3-8 and 3-9) are as follows -

1. Valid and invalid states - Since n bits can represent 2**n combinations then if less than 2**n states are needed to represent information, the remaining states can be used to detect an invalid state.

2. Character parity (Vertical Redundancy Check) - By convention n-1 bits are used to convey information and another bit called the parity bit is appended to these n-1 bits. The number of 1 bits in every group of n bits must be either odd or even and the parity bit is used to maintain this property. If a character is received with wrong parity, it is in error.

   The parity check system is not 100% reliable since double errors can occur in a character which will cancel each other. Parity can only detect errors which affect an odd number of bits but it is a simple method and is often implemented by hardware.

3. Column parity (Longitudinal Redundancy Check) - Similar to character parity except that this time the parity check is performed on bits in the same column of a message to be transmitted. All comments made about character parity apply again except that this form of parity is not commonly hardware implemented.

4. Checksums - The data to be transmitted is treated as unsigned integers and is summed to produce what is known as the checksum. The checksum is transmitted, along with the message, to the receiving station. The receiver receives the message and computes its own checksum which it compares with the transmitted checksum. If both are identical then the message is assumed to have been received correctly.

   The method is not foolproof as double-bit errors in a column can go undetected, but it is still a very practical method.

5. Polynomial Cyclic Codes - A cyclic code message consists of a specific number of data bits and a block check character (BCC). The BCC is generated by taking the remainder, after dividing all the serialised bits in the block of data by a predetermined binary number (the generator polynomial).

   The receiving station divides the cyclic code message by the same generator polynomial and if there is no error the division will produce the same number as the BCC.

The BCC is usually computed and accumulated in a special shift register i.e. special hardware is needed, although slower software algorithms exist.

3.2 CHOICE OF ERROR CONTROL METHOD

The decision as to which method to use in this network involved considering the following two design constraints:

(i) the links between the DC and the LCSs must be essentially error free.

(ii) an efficient rate of data transmission must be maintained between the DC and the LCSs.

To determine which scheme would be best suited to this network, it was necessary to know the error rate of the communication links.

A program was therefore written which could monitor the flow of messages over a link, and record any error conditions which arose [see appendix B]. This program made use of a series of subroutines developed by Mr Tim Slack. These subroutines were used by Mr Slack in the field testing of a prototype LCS and implemented a simple protocol by which the HOST computer and the LCS could communicate.

The hardware configuration in which the error monitor program ran consisted of the prototype LCS connected by 7.4 km of cable to the HOST. LCSs will normally be connected to the DC but at the time this work was done the DC had not been delivered.

The heart of the monitor program was a repetitive 'write-read-compare' cycle. On each cycle, a portion of the LCSs memory was loaded via the communication link from the HOST. Next a message to read back that data was issued and the data read back was compared, against the data originally sent, for errors. The simple protocol checked for parity and framing errors on the link as well as unexpected message responses from the LCS or a lack of response. All errors detected were logged and an interim report on the status of the link was printed, at predetermined message intervals. The interim reports summarised the number of errors recorded, and the total number of messages sent so far, in each link direction.

The data used in each message was 40 characters from the ASCII character set. The complete set of ASCII characters was stored in a 'circular' array and each time a message was to be sent, the next 40 characters were obtained. This technique provided variation in the message data transmitted yet ensured that all ASCII characters were used equally often. The full ASCII character set was used because many messages in the network

will carry binary data which may represent infrequently used ASCII characters.

The error monitor program was run on several occasions, over periods of up to five days and no errors were ever detected on the link. The error rate on the link was found to be less than 1 bit in 100 million which is very low. The high quality of the link is supported by the experiences of remote terminal users who have had similar links connecting them to the HOST for several years now with no noticeable problems.

From this quantitative result it does not seem neccessary to use a polynomial cyclic code on these links. The expected frequency of errors does not justify the extra hardware or software needed to implement it.

A simple character parity check would be acceptable in most cases and it is available automatically from the DZ11 multiplexors, but it would not be very secure if a burst of errors occurred on a link.

I therefore decided to use a combination of character parity and checksum methods. Little extra code is necessary to implement the checksum method yet the added protection that it would give to messages would be extremely valuable.

CHAPTER 4

SELECTION OF PROTOCOL CLASS

This chapter looks at the different classes of protocols available and presents the reasoning behind the selection of the protocol class for this network. The following chapters describe the protocol in detail.

There are three main categories of protocol in use throughout the world at present, the differences being in how the messages are framed.

4.1 CHARACTER ORIENTED PROTOCOLS

This category of protocol uses special characters to delineate various fields of the message. For example STX is used to indicate the start of a message and ETB is used to indicate the end of a block of text.

When a station is receiving a message it must continually test incoming characters for the end of transmission character (EOT) or the end of transmission block character (ETB) so it knows when a message is finished. There is, of course, a possibility that data in the message transmitted will have the same bit pattern as these control characters and so 'character stuffing' techniques are necessary on these data to make them transparent.

A good example of this class of protocol is the BISYNC protocol (Binary Synchronous communication - [7] pp195-205) of IBM.

4.2 BYTE COUNT ORIENTED PROTOCOLS

A typical byte count oriented protocol uses a special block of characters called a header which precedes the remainder of the message. The first byte of the header contains a special character which is used to indicate the start of a message. Successive bytes in the header contain such information as the number of bytes in the data portion of the message, the type of message, and control information.

The data portion of the message follows the header and is of the length specified in the header. A block check character completes the message.

DEC's DDCMP (Digital Data Communications Message Protocol - [4], [7] pp207-213) is a good example of a byte count oriented protocol.

4.3 BIT ORIENTED PROTOCOLS

In a bit oriented protocol the start and finish of messages are marked by a special sequence of bits called the flag character. The flag character is the binary bit pattern 01111110. When a station receives this sequence of bits it knows that the previous 16 bits were part of the block check character and that the bits between those 16 and the previous flag were the message. Of course the receiving station must continually test each incoming bit, along with the preceding 7 bits, to see if it has received the flag character yet.

Special techniques, called 'bit stuffing', must be used if we wish to send a character which has the same bit pattern as the flag.

A typical bit oriented protocol is IBM's SDLC (Synchronous Data Link Control - [7] pp215-220) or CCITT Recommendation X.25.

Bit oriented protocols are designed for efficient use of high speed serial, synchronous, full duplex facilities. As such it is not a suitable protocol class for this network since the hardware works in asynchronous fashion and is byte rather than bit oriented.

4.4 THE SELECTION

For this network there were only two possible protocol types which could be implemented. Initially, neither of these types seemed to offer an advantage over the other but it soon became

clear that there were advantages in using a byte count oriented protocol.

Firstly, a large proportion of the messages on the link will contain binary data. If a character oriented protocol was used then 'character stuffing' would be needed. This would make some messages longer than necessary and hence incur higher link overheads. A byte count protocol, on the other hand, has no transparency problems.

Secondly, a byte count oriented protocol yields information in the header about the size of the message. This information could be very useful, for example, when allocating buffer space for a message. A character oriented protocol has no message size information in the message and hence a maximum sized buffer must be allocated for each message.

Thirdly, the communications link between the data concentrator and the HOST computer uses a byte count oriented protocol. Any message which has to be sent to the HOST from an LCS could therefore be sent with relatively few changes i.e. the message formats in the two systems would be consistent.

So the conclusion of this chapter is that a byte count oriented protocol would be the most suitable for this network.

CHAPTER 5

MESSAGE FORMATS

The first two sections of this chapter describe the types of messages needed in the network and the final section presents a description of the message format adopted.

5.1 MESSAGE TYPES

There are two logically different types of messages used in this protocol.

1. Control messages

2. Data messages

The distinction between the two message types is that data messages contain information which is used to operate the traffic system, while control messages are used to support the operation of the protocol.

5.1.1 CONTROL MESSAGES

Control messages are used to transmit information concerning the control and transmission status of the link between the DC and an LCS. There are two such messages -

1. ACKNOWLEDGE (ACK) - Used by an LCS when a message from the DC is received correctly and does not require a data response. The DC, upon receiving an ACK knows that the last message it transmitted was received correctly by the LCS.

2. NEGATIVE ACKNOWLEDGE (NAK) - The NAK message is issued by an LCS to tell the data concentrator that the last message it sent was received with an error. No information on the nature of the error is returned.

5.1.2 DATA MESSAGES

Data messages contain information which is relevant to the operation of the Traffic Control System. The most frequently issued data messages are -

1. SET PHASE - A message specifying the phase change which should be performed at an intersection.

   The usual flow of this message will be

   HOST ---> Data Concentrator ---> LCS

   or, if the HOST is out of order

   Data Concentrator ---> LCS

   directly.

2. STATUS RETURN - A message issued whenever the internal state of an LCS or local controller changes. This change may take place for several reasons including a phase being changed at an intersection, an LCS fault or a vehicle detector fault.

   The usual flow for this message will be

   LCS ---> Data Concentrator ---> HOST

   or, if the HOST is out of order

   LCS ---> Data Concentrator

   In addition, the DC will make a copy of the status return message and send it to the mimic display.

3. SEND SYSTEM STATUS - A message issued by the data concentrator to find out more about the status of the hardware at a particular intersection.

   The flow of this message is

   Data Concentrator ---> LCS

4. SURVEILLANCE SITE DISPLAY - A message issued to the mimic display, by the data concentrator, which contains the volume and occupancy figures for the intersection over the last time period.

This message will flow as -

HOST ---> Data Concentrator ---> Mimic

If the HOST is out of order then this message will be absent from the system.

5. COUNTER SERVICING - A message sent from

HOST ---> Data Concentrator ---> LCS

or, if the HOST is out of order

Data Concentrator ---> LCS

to read the volume and occupancy counters of the local controller supervised by that LCS.

6. RESPONSE TO COUNTER SERVICING - the message sent from

LCS ---> Data Concentrator ---> HOST

or, if the HOST is out of order

LCS ---> Data Concentrator

which contains the readings of the local controller's volume and occupancy counters.

Several other types of message will also be issued in the network but only infrequently and they will make up only a small, as yet unknown, percentage of the total number of data messages issued. These data messages are -

7. TIME CHECK - Data messages for synchronising one or many LCSs to the time of the Control Centre. Two data messages are necessary to ensure this synchronisation and they are issued when an off-line LCS must be synchronised with the time of the on-line LCSs in the network.

8. SET MEMORY - A message used to write into an LCS's RAM, mainly when a backup traffic plan must be downline loaded to an LCS. The backup plans are used by the LCSs to run the traffic system, should the data concentrator be out of action, and each is ten 8-bit bytes long.

SET MEMORY messages are also essential to load the operational code into LCS RAM, so that the LCS can do useful

work after a power recovery. The memory data is sent to an LCS in a series of messages, each containing up to 42 bytes of data.

9. READ MEMORY - A message sent from the data concentrator to an LCS, requesting a copy of the contents of a specified portion of memory.

10. RESPONSE TO READ MEMORY - The LCS packages a copy of the memory contents into messages up to 42 bytes in length and sends these to the data concentrator.

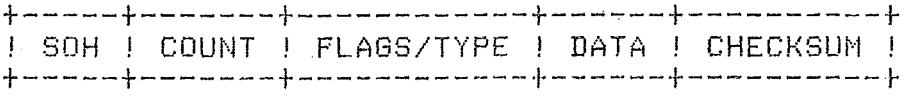
5.2 MESSAGE FORMAT

The message format decided upon is presented below. Several points should be noted about the design.

Firstly it was decided to use a standard message format for both control and data messages so that the action of the receiving station will be independent of the message type. Not all protocols have this standard format; DDCMP in particular has differing formats for it's control and data messages and even has different sychronising characters, SOH for data messages and ENQ for control messages.

Secondly the message format is relatively simple so assembly and disassembly overheads by the DC and LCS will be low.

Figure 5.1 Message format.



where -

SOH = Start Of Header, a special character used to denote the beginning of a new message. The SOH character comes first in a message since it is a sychronising character. If the first character of a new message is not a SOH then it and subsequent characters are ignored until a SOH is received. This scheme is not foolproof but it does offer some protection if the data concentrator and an LCS ever get out of synchronisation.

COUNT = A count of the number of 8-bit bytes in the message, from SOH to the last data byte inclusive. The COUNT byte follows logically in the message since, once the valid start of new message occurs the receiving station must know how many bytes will follow.

The incoming bytes of a message are counted by the receiver so it knows when the message is finished and a new one is expected.

The information in the COUNT field is also used to determine the size of the buffer which must be allocated from the buffer pool to contain the incoming message.

FLAGS/TYPE = the following table is a bit map of the FLAGS/TYPE byte. Bits are numbered with 0 being the least significant.

Bit	Function

7	If message direction is LCS -> DC then
	0 = Tx loop to LCS is working
	1 = Tx loop to LCS not working
	If message direction is DC -> LCS then
	0 = Message is a response to a RELOAD message.
	(RACK - Reload Acknowledge)
	1 = Message is not a response to a RELOAD message.

5,6	Reserved for future use.
0-4	Specify a unique message type, for example ACK, SET MEMORY.

DATA = Up to 42 bytes of information, specific to the message type. The data field may not be needed for some message types, e.g. control messages or READ MEMORY, and so can be omitted.

There are no restrictions on the bit patterns allowed in these 42 bytes since, unlike some protocols, the data is totally transparent. The limit of 42 bytes is imposed by -

1. Memory buffer restrictions in the LCS.
2. The fact that data for loading into LCS memory is formatted on disk in blocks of this size.

CHECKSUM = the byte containing the unsigned orithmetic sum of all the preceding bytes in this message i.e. all bytes from SOH to the last byte of data, inclusive. In some byte count oriented protocols (e.g DDCMP [4]) the checksum is computed as a 16-bit total but in this protocol only an 8-bit checksum is used since it is easier for the LCS's to perform single, rather than double byte additions.

Another reason for limiting the checksum to 8-bits is that the length of the control message becomes four bytes, a convenient size when it is realised that the core allocation routines of the RSX-11S operating system allocate memory in blocks of this size.

CHAPTER 6

PROTOCOL DEFINITION

This chapter presents the definition of the protocol for the Traffic Control network.

6.1 QUEUEING OF MESSAGES

A message transmitted from the DC to an LCS may have originated in the HOST, which will be the case under normal circumstances, or in the DC if the HOST is out of action.

A queue of messages waiting to be transmitted to each LCS is maintained in the DC. The queues are organised according to message priority, this priority being determined by the message source. See Chapter 7 for more details of the implementation.

When a message is to be added to a transmission queue in the DC, several checks are made of the link.

(i) Rx loop - The Rx current loop carries the data signal from the LCS to the DC. If this loop is broken then the DC cannot receive messages from the LCS.

(ii) Tx loop - The Tx current loop carries the data signal from the DC to the LCS. If this loop is broken then the LCS will not receive any message sent by the DC. The DC cannot check the status directly, but instead it relies on the information contained in bit 7 of the FLAGS/TYPE byte, of messages from the LCS.

(iii) Link closed - If the link closed flag is set (see Section 6.3) then no messages can be sent from the DC to the LCS.

If the link fails any of these checks then the message at the head of the appropriate transmission queue is discarded and the message source notified. The next message to be added is then inserted in the queue according to its priority. This dequeueing-queueing mechanism prevents the queues from growing unboundedly and consuming precious memory under link fault

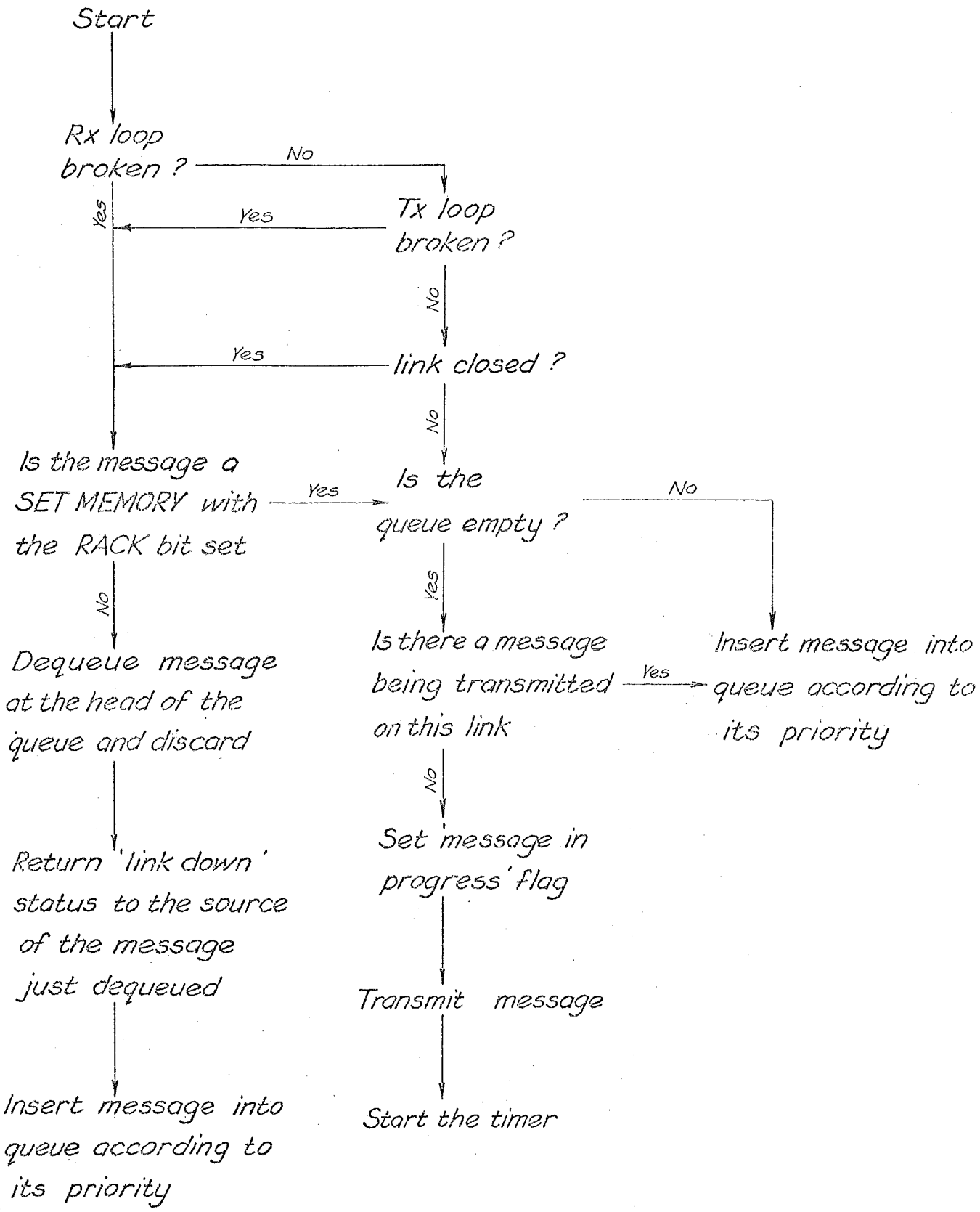


fig. 6.1 Handling by LCS driver of a message from the I/O Packet queue

conditions.

If the link passes the checks then the message is inserted in the queue according to its priority.

Figure 6.1 shows the decisions made before a message is to be added to a transmission queue.

6.2 PRE-TRANSMISSION LINK CHECKS

Once the preceding message has been processed, the message at the head of the queue is ready for transmission. Before the DC begins the transmission of a message, it once again checks the link. If the link fails any of these checks then the message is not transmitted and the message source is alerted with a 'link down' status. The checks are -

(i) Rx loop - If this loop is broken then the DC would have little chance of receiving any response which the LCS might make, so it would not be advisable to send the message. Instead the Tx loop is broken by the DC, forcing the LCS into backup mode (if it still has power).

(ii) Tx loop - A breakage on this loop means the LCS cannot receive any message sent from the DC, so the DC does not send it. Since the Tx loop is broken, the LCS is already operating in backup mode.

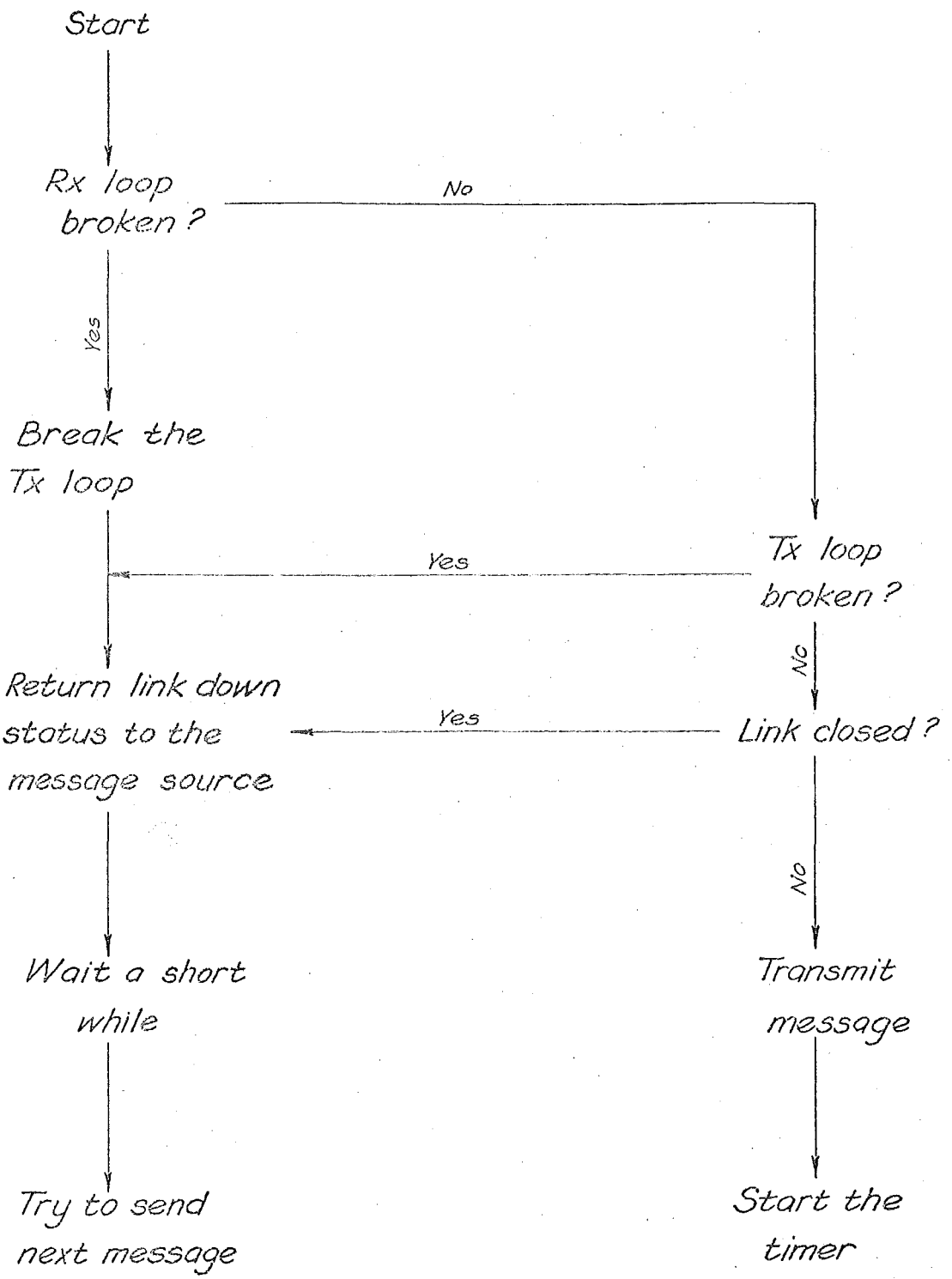
(iii) Link closed - If the link is closed then a major link problem exists so the message is not sent.

Figure 6.2 shows the decisions made before a message is dequeued and transmission of it's contents started. If the link fails, then a short time delay will be made before transmission of the next message is attempted. This allows time for the link status to change, otherwise an entire series of messages could be discarded in quick succession.

6.3 MESSAGES INITIATED BY THE DATA CONCENTRATOR

Once the DC has transmitted a message to an LCS, it waits for a reply.

Messages whose function is to 'write' data to an LCS, e.g. SET MEMORY or SET PHASE, require either an ACK or a NAK reply. An ACK message from the LCS signifies that the message sent by the DC was correctly received. A NAK message indicates that the message was received with errors. An LCS may issue a NAK response to a message from the DC for any of the following reasons -



Communication link testing before message transmission

Fig. 6.2

1. Checksum Error - The checksum computed by the LCS, which received the message, did not match the checksum which the DC computed and transmitted.

2. Buffer Unavailable - A buffer was not available in the LCS to store the incoming message.

3. Overrun - The receiving hardware in the line driver and/or LCS, or the LCS software, was not able to respond fast enough to incoming bytes and so one or more bytes of the message were lost.

4. Header Error - The header section of a message contained illegal values. For example, an invalid function code or a message length which exceeded the allowable limit. This provides protection against errors such as the DC or an LCS writing the wrong value in a header field.

Messages whose function is to 'read' data from an LCS, e.g. COUNTER SERVICING or READ MEMORY, require slightly different responses. If the message is received correctly by the LCS then the response message contains the data requested. If the message is received incorrectly then a NAK message is issued by the LCS.

To detect a lack of response from the LCS, a timer is started in the DC immediately after the last byte of the message has been transmitted. One timer will exist for each message currently in progress. When a response is received from the LCS the timer is reset. If, however, the timer is not reset before it exceeds a critical value then the message is said to have 'timed out'. I have defined the critical value to be 3 seconds. This will give ample time for a response to be received.

If the DC receives a NAK response, an incorrect data message, or times out, then it will retransmit the message. There is a defined limit of 3 on the number of times a message can be transmitted and when this limit is exceeded the message is aborted and the message source is alerted. It is up to the source to decide what action to take if, say, a SET PHASE command has to be aborted. An attempt is then made to send the next message queued to this LCS.

If 5 consecutive messages to an LCS are aborted then the link to that LCS will be closed, by setting the 'link closed' flag, since a serious problem must exist. The closure means that no messages will be sent from the DC to the LCS though the LCS is still free to transmit to the DC. A diagnostic message is sent to the system log and the link can only be reopened by operator intervention or a RELOAD message from the LCS (see startup notes).

If a message receives, as reply, an ACK or a correct data message then the source of the original message is notified of the success and an attempt is made to send the next message.

6.3.1 EXAMPLES OF MESSAGES INITIATED BY THE DC

The message exchange examples on the following pages, between the DC and an LCS, use the notation below.

By arbitrary convention the DC is the left hand side of the diagram while an LCS is on the right.

Symbol	Meaning
---------->	Message sent, received correctly.
----------/--->	Message sent, received with errors.
----------//--->	Message sent, not received.

6.3.2 NORMAL 'READ'

DC LCS

'read' message

---------->

data received

<----------

6.3.3 NORMAL 'WRITE'

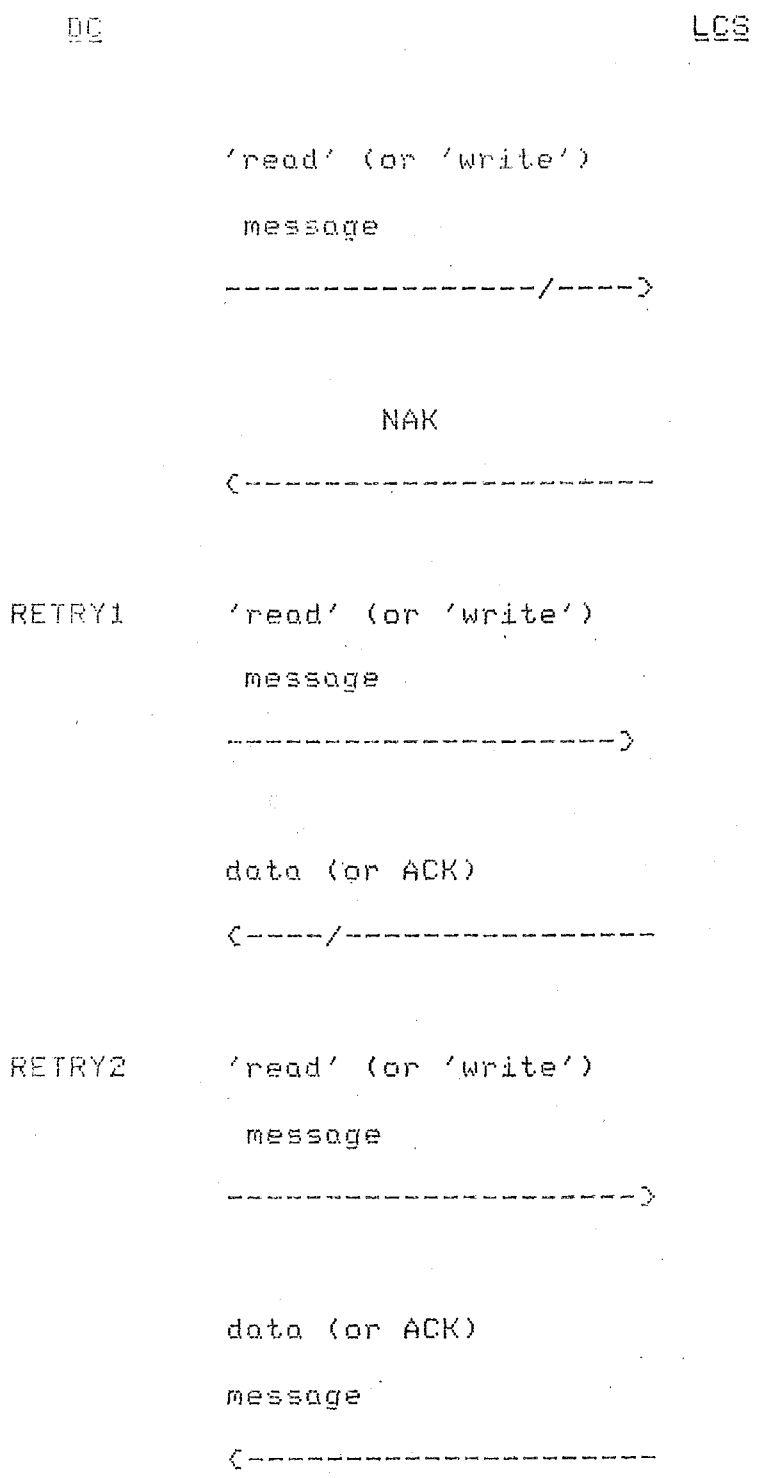
'write' message

---------->

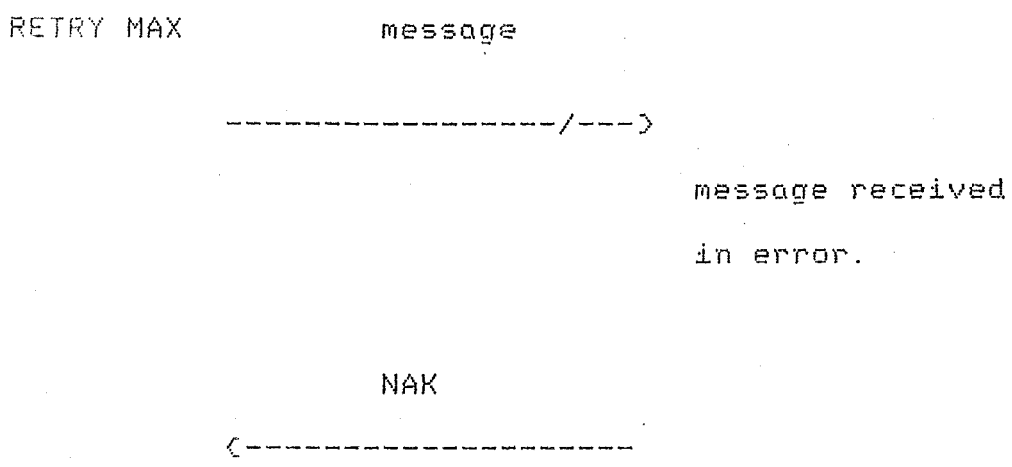
ACK

<----------

6.3.4 'READ' OR 'WRITE' WITH ERRORS



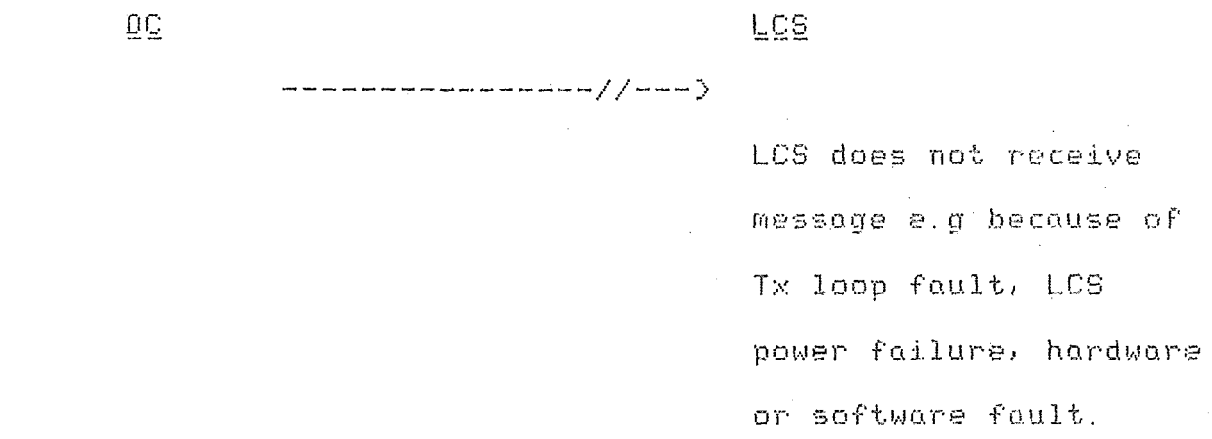
6.3.5 MESSAGE ABORTED


RETRY MAX
message
message received
in error.
NAK
Message has failed
MAX times so the
message is aborted.
(MAX is defined to be 3)

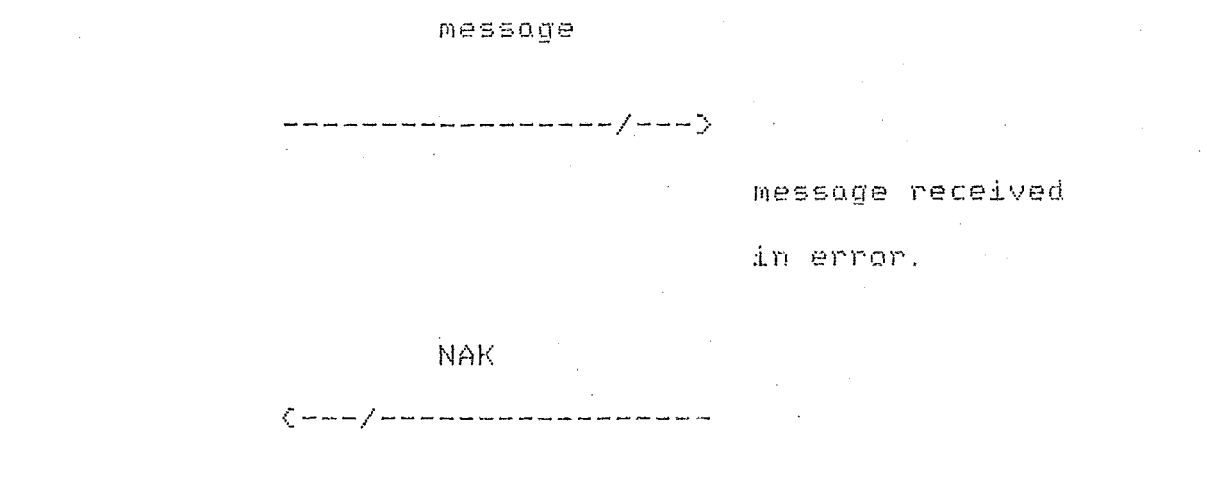
6.3.6 TIMEOUT OF 'READ' OR 'WRITE' MESSAGE

A timeout may be caused in the DC by any of the following situations and will be treated the same as a NAK received from an LCS.

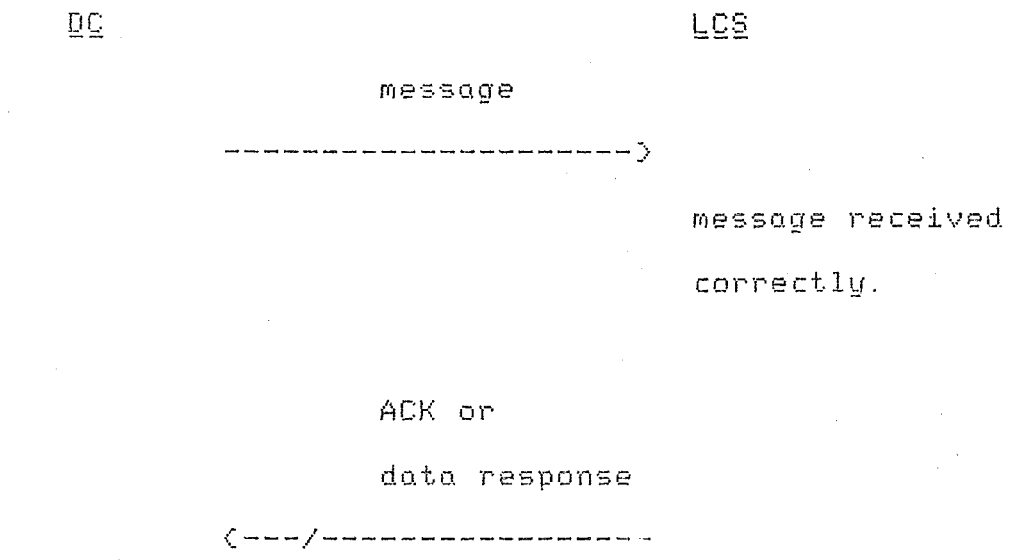
6.3.6.1 LCS does not receive message -



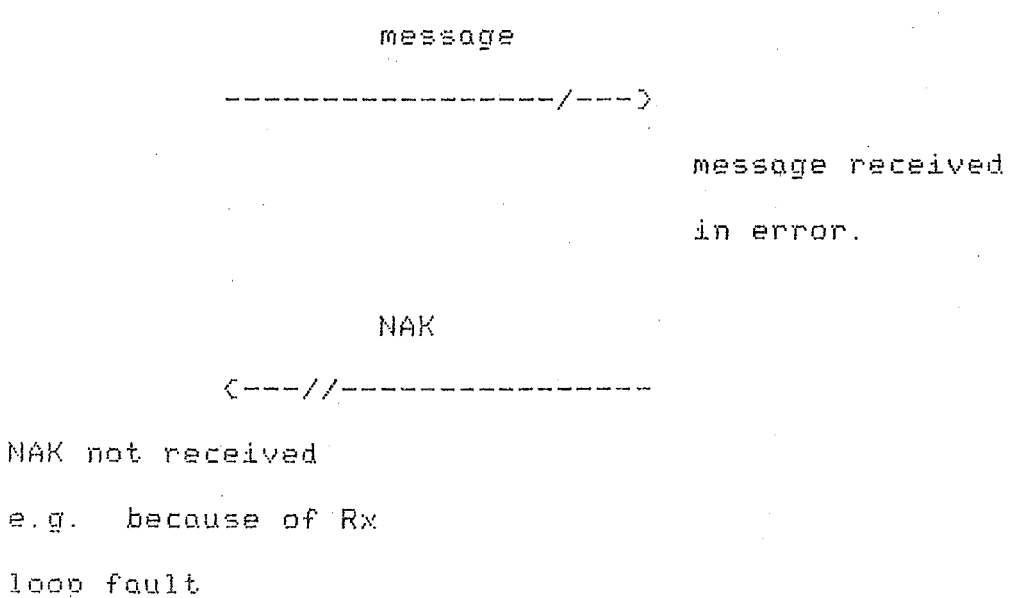
6.3.6.2 Message received in error, NAK received in error -



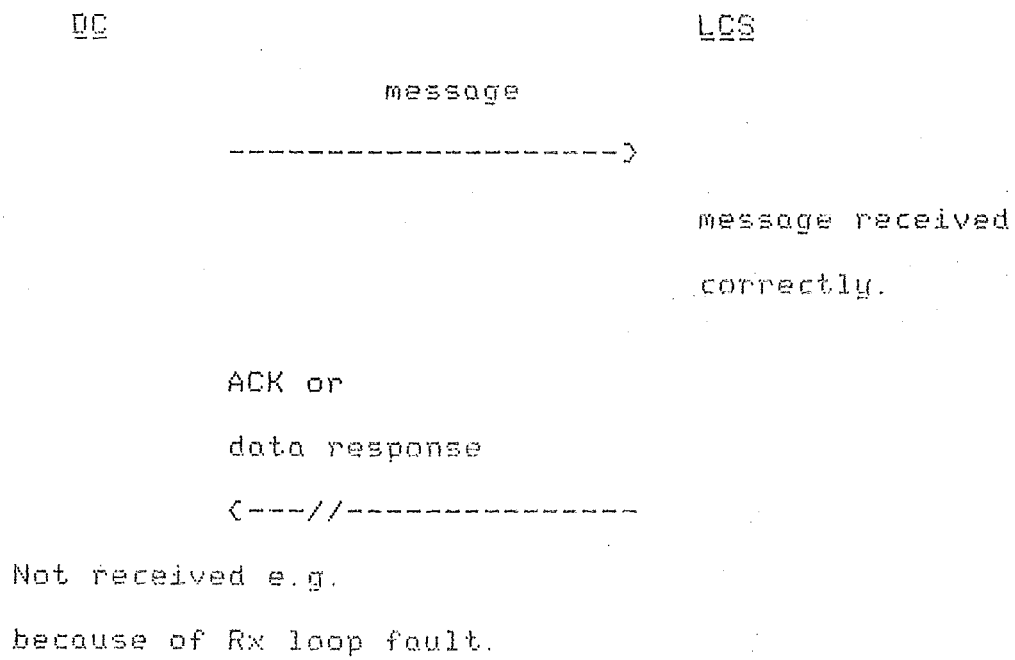
6.3.6.3 ACK or data received in error -



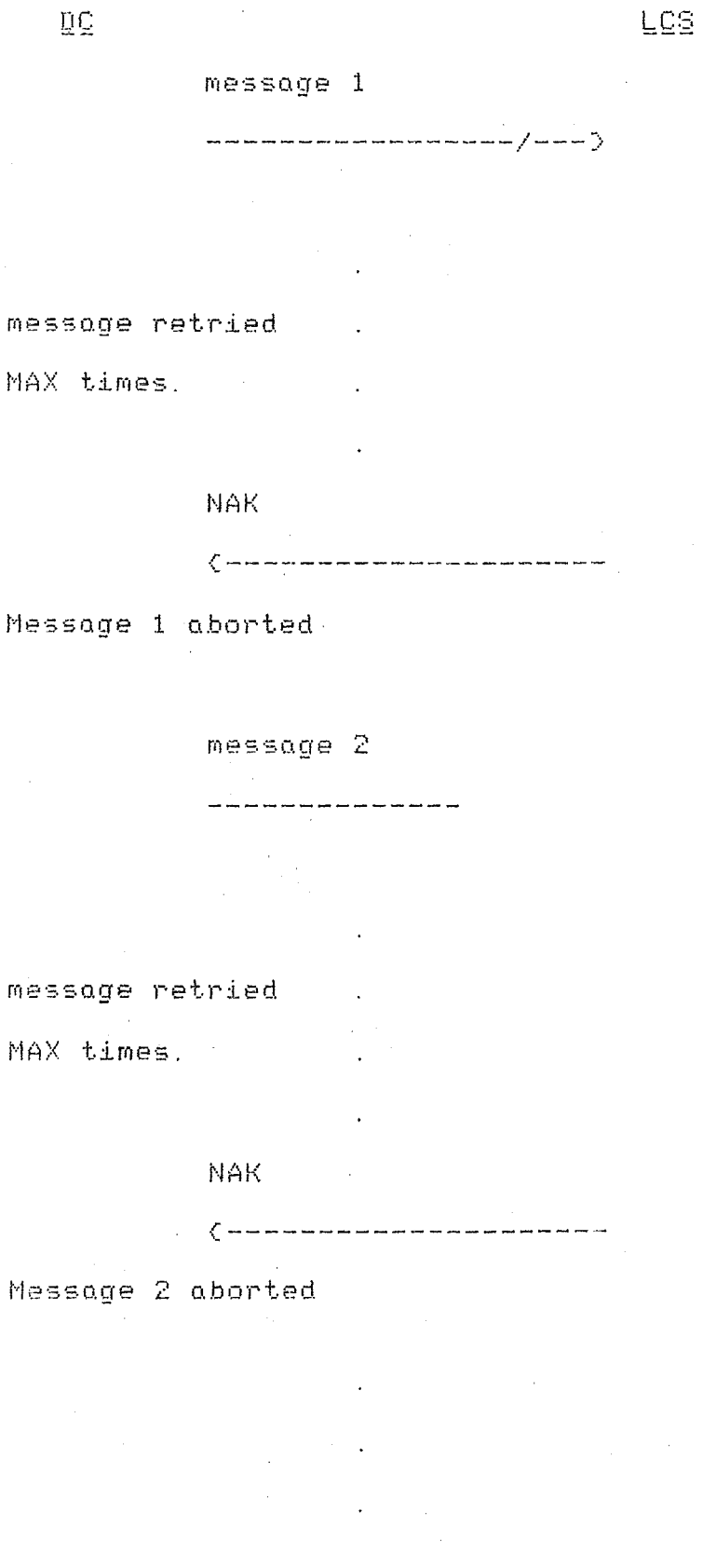
6.3.6.4 NAK not received -

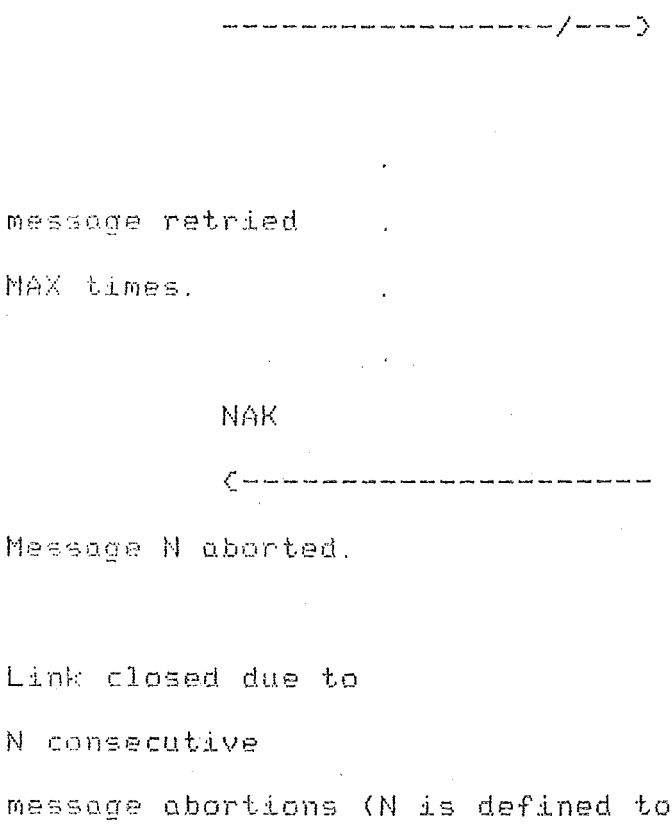


6.3.6.5 ACK or data not received -



6.3.7 LINK CLOSED




----------/--->
message retried
MAX times.
NAK
<----------
Message N aborted.
Link closed due to
N consecutive
message abortions (N is defined to be 5).

6.4 MESSAGES INITIATED BY THE LCSs

Most of the messages initiated by the LCSs will be sent spontaneously, to report a change in the status of either the LCS, or the local controller it is monitoring. These messages contain information which is useful but not critical to the operation of the traffic system.

If the message is received correctly by the DC then the information in the message will be passed to the HOST, which is monitoring the status of all the LCS's and the local controllers. The message will also be copied and sent to the mimic display.

If the DC receives the message with errors or has no available buffer space then the message is ignored entirely.

The LCSs never expect a reply of any description from the DC in response to messages which they initiate. One exception to this rule is the RELOAD message, which is illustrated below.

6.5 RESTARTING A LINK

Loss of power to an LCS causes the random access memory to be corrupted and the local controller to go into IVA mode. It is important to be able to reload the RAM, and get the LCS reconnected to the network as soon as possible once the power is restored. This is the startup process.

It makes sense to let the LCS rather than the DC initiate and control the startup process on a link since -

1. Once the DC knows the LCS is 'down' it does not have to waste time repeatedly trying to restart the link. It waits instead until the LCS is functioning and sends a RELOAD message.

2. Only those LCS's which require restarting at any one time will get attention. If the DC initiated the startup it would have to poll all the LCS's to find those needing a reload. This process could become very time consuming and complicated.

The following is a description of the startup process.

1. The LCS has its power restored and code in the ROM is activated causing, among other things, a RELOAD message to be sent to the DC. At this stage the DC may or may not be powered up but the LCS sends its message regardless.

2. The RELOAD message tells the DC that the LCS has recovered from a power failure and wants its RAM reloaded. If this unsolicited message is received incorrectly by the DC then the DC ignores it. If the messsage is received correctly

then the DC will send the first of a series of messages to reload the LCS's memory - the only time an unsolicited message is acknowledged by the DC. The LCS will repeat the RELOAD message at 15 second intervals until a successful SET MEMORY message is received.

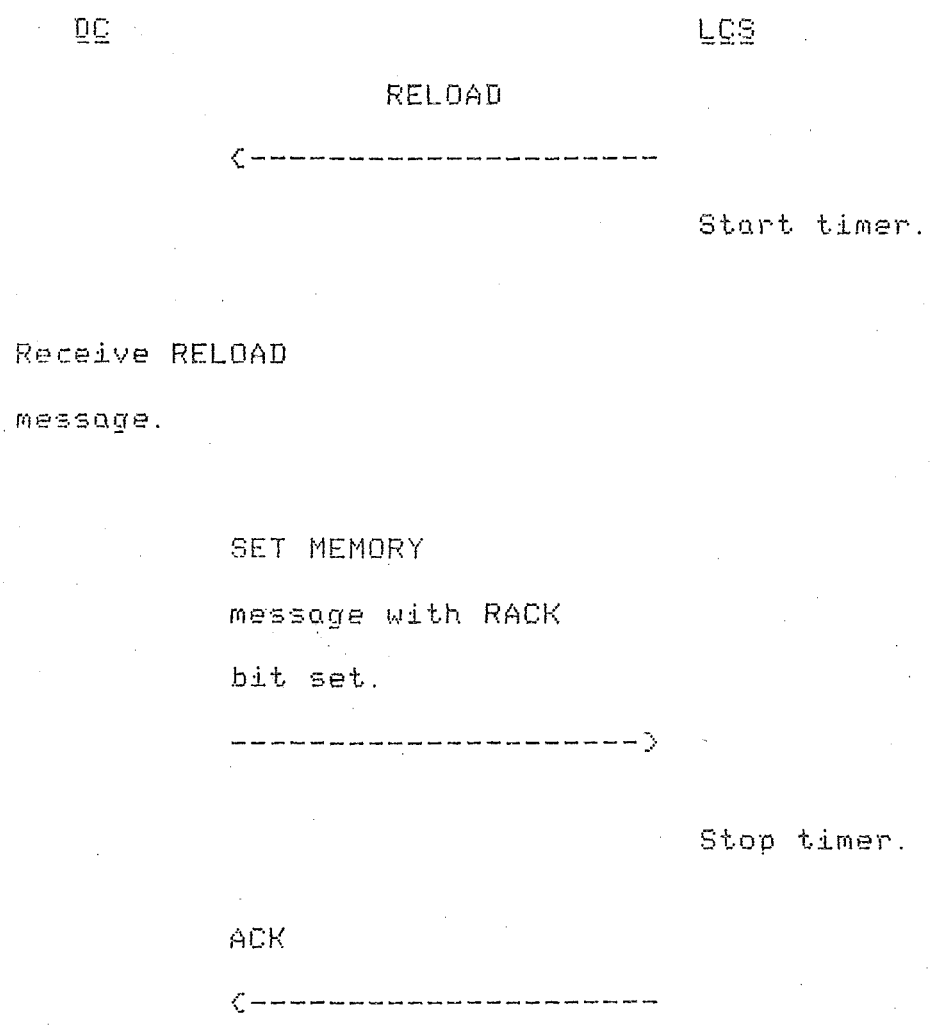
3. The high priority SET MEMORY message sent to the LCS will have the RACK (Reload Acknowledge) bit in the flags byte set to indicate to the LCS that the RELOAD message was correctly received by the DC. Any message received by the LCS, while it is trying to restart the link with the DC, will be ignored if the RACK bit is not set.

4. The LCS acknowledges the SET MEMORY message with an ACK, if it was received correctly, otherwise a NAK. If the SET MEMORY message fails or 'times out' 3 times then the DC discards the SET MEMORY message and waits for the LCS to issue the next RELOAD message. When the ACK is received the DC declares the link open, if it was previously closed, by clearing the 'link closed' flag. The DC can then proceed to complete the loading of RAM with additional SET MEMORY messages.

6.5.1 RESTART EXAMPLES

The following are examples of different situations which can arise during the normal startup of a link to an LCS.

6.5.1.1 Normal startup -



The DC now proceeds to RELOAD the remainder of the LCS's memory, using SET MEMORY messages.

6.5.1.2 RELOAD message fails -

DC LCS

RELOAD

<---/----------

Start timer.

DC ignores unsolicited

message with errors.

!

! 15 seconds

!

!

v

Timer expires.

Send another RELOAD.

RELOAD

<----------

Start timer.

receive RELOAD

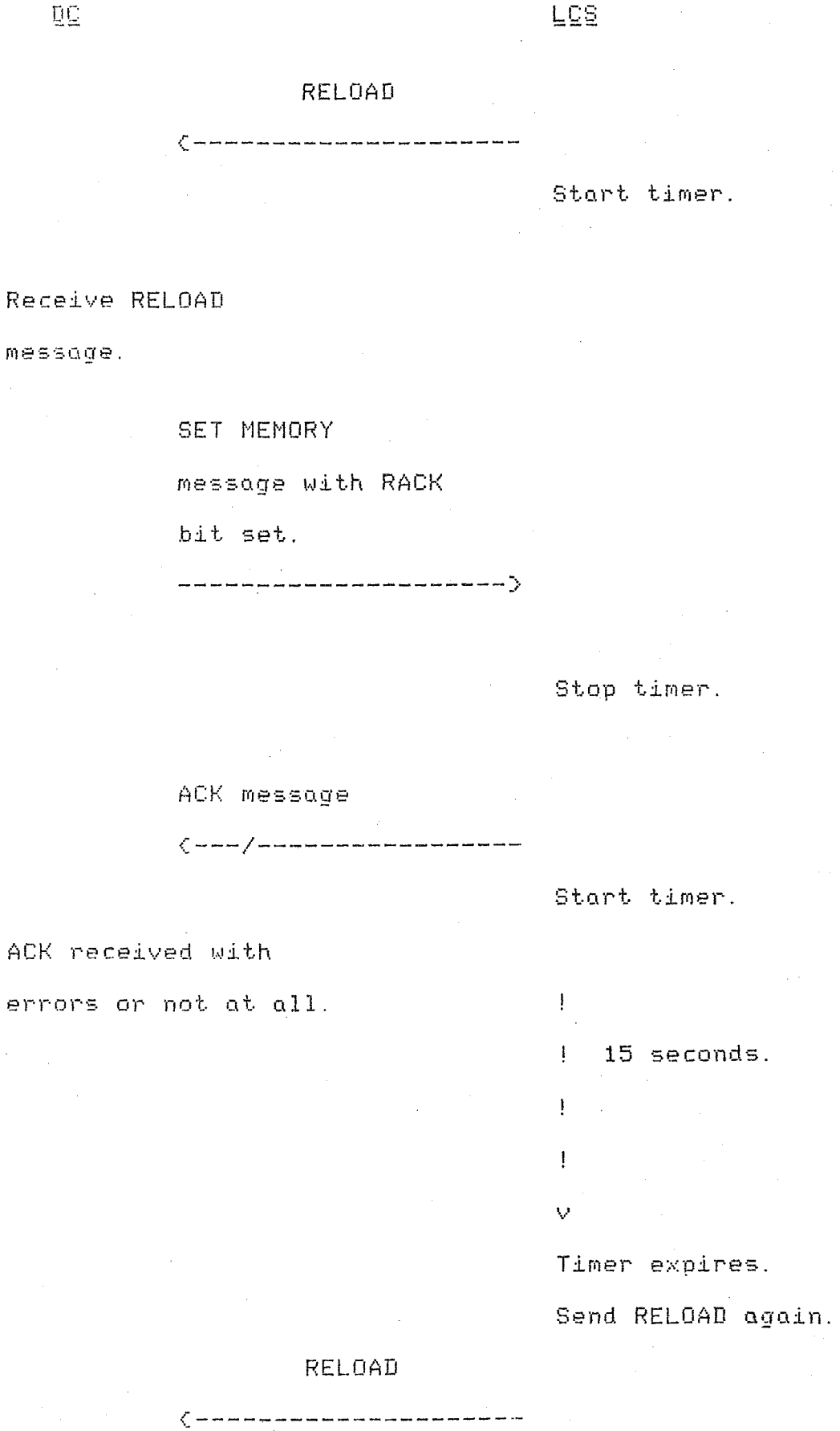
message.

.

.

Same as for 6.5.1.1 above.

6.5.1.3 DC does not receive the ACK message -



.

.

.

Same as for 6.5.1.1 above.

CHAPTER 7

PROTOCOL IMPLEMENTATION

This chapter describes the implementation of the message handling protocol in the data concentrator, within the constraints of the RSX-11S Executive.

The code to implement the protocol is contained in a standard RSX-11S I/O driver called the LCS driver, which is linked into the Executive at system generation time.

7.1 AN OVERVIEW OF THE RSX-11S I/O PROCESS

An I/O driver calls and is called by the Executive to service an external I/O device or devices.

It is usual, in RSX-11S, for all I/O devices to be declared to the Executive via a series of tables. The tables contain the static attributes of the devices, such as allowable functions and interrupt addresses, as well as dynamic device attributes, such as controller status and lists of pending I/Os.

When a task wishes to make an I/O request it issues a QIO$ (Queue I/O) system directive to the Executive. The Executive performs validity checks on the QIO$ parameters and if these checks are successful then the Executive generates a data structure called an I/O packet, which is inserted into a device specific queue of I/O packets.

The Executive also notifies the LCS driver that an I/O packet has been queued to the device. From information contained in the I/O packet the driver can initiate the I/O operation and then continue with other activities. At a later stage when the previously issued I/O operation interrupts, the driver is called to process the interrupt.

Upon completion of processing for an I/O request, the driver calls the $IODON (I/O Done) procedure which returns status information, on the completed I/O, to the task which issued the I/O request.

For more details of RSX-11S see [5] and [6]. Although these manuals are written for RSX-11M, RSX-11S is similar in many ways.

7.2 HANDLING MESSAGES TO LCSs

This section describes how messages to be sent to LCSs are handled. The handling process is represented diagrammatically in figure 7.1.

It was desirable, in this implementation, to have a queue of I/O requests for each LCS. This allows full use to be made of the communication links and the DZ11 multiplexors, since transmission of messages from the data concentrator to different LCSs can then be overlapped.

The initial approach was to consider each of the 128 LCSs as a separate I/O device which is declared to the Executive. It soon became apparent that this approach was infeasible since the memory requirements for the necessary declaration tables were excessive.

After much thought, a method was devised for bypassing the memory requirements problem. Instead of having 128 LCS devices declared to the system, a pseudo-device called LCO: was declared. Any task wishing to send a message to an LCS queues the LCS number, message size and message address to device LCO:, via the QIO$ directive. This causes an I/O packet to be inserted in the queue of I/O packets for device LCO: and the LCS driver notified of this insertion. Based on information in the I/O packet, the LCS driver will then relink the packet into an internal, priority based queue, one of which exists for each LCS.

By bypassing the normal RSX-11S device approach with a pseudo-device, a separate queue of requests for each LCS can therefore be maintained with acceptable memory requirements.

Once it is time for a message to be transmitted, the message is accessed via the size and address stored in the I/O packet, and is transmitted a byte at a time to the LCS concerned. A reply is then awaited from the LCS.

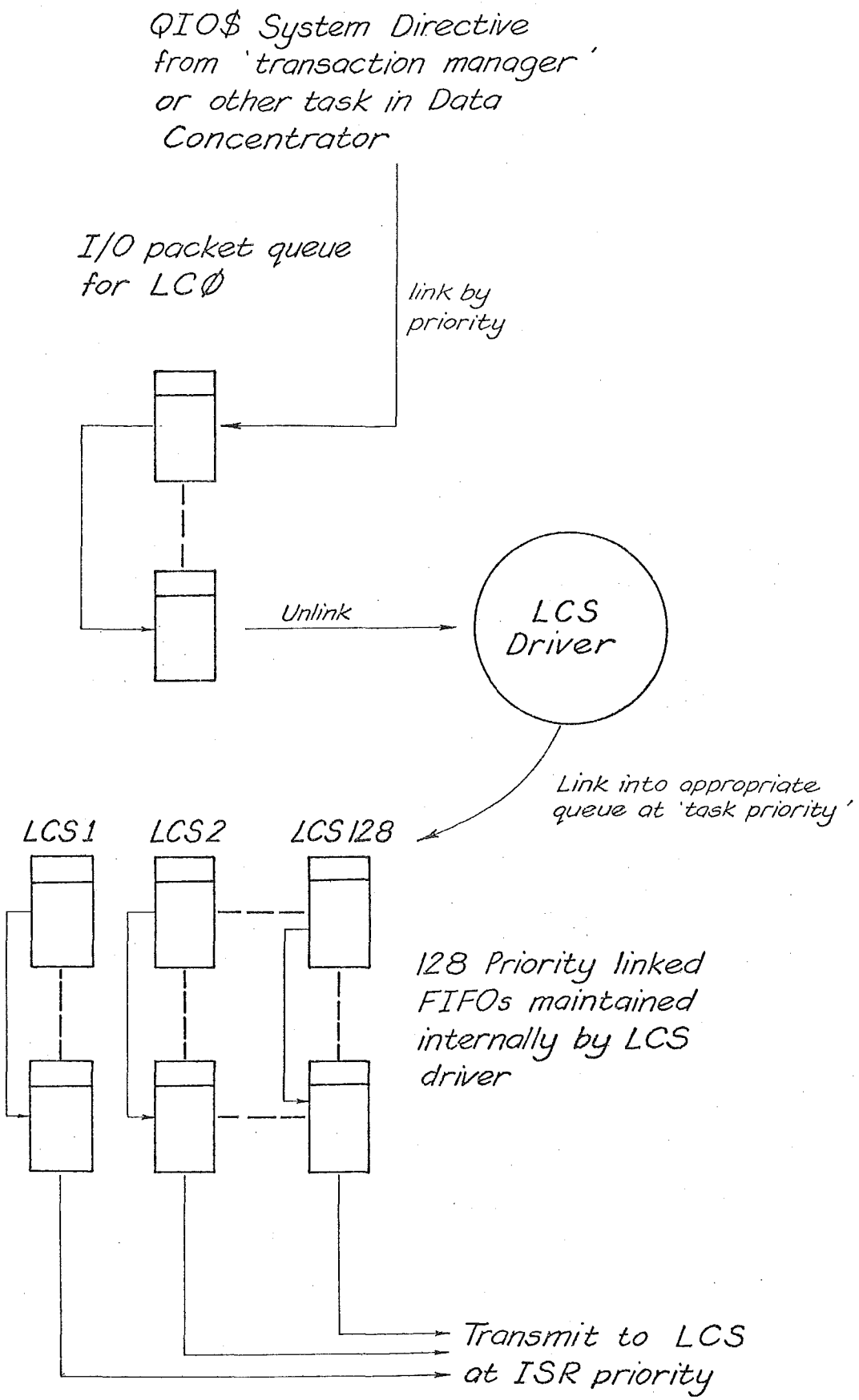
7.3 HANDLING MESSAGES FROM LCSs

This section describes how messages received from LCSs are dealt with by the LCS driver. The process is represented diagrammatically in figure 7.2.

Messages arrive from LCSs a byte at a time, the bytes being stored directly into a DZ11's silo, which can hold up to 64 bytes of data.

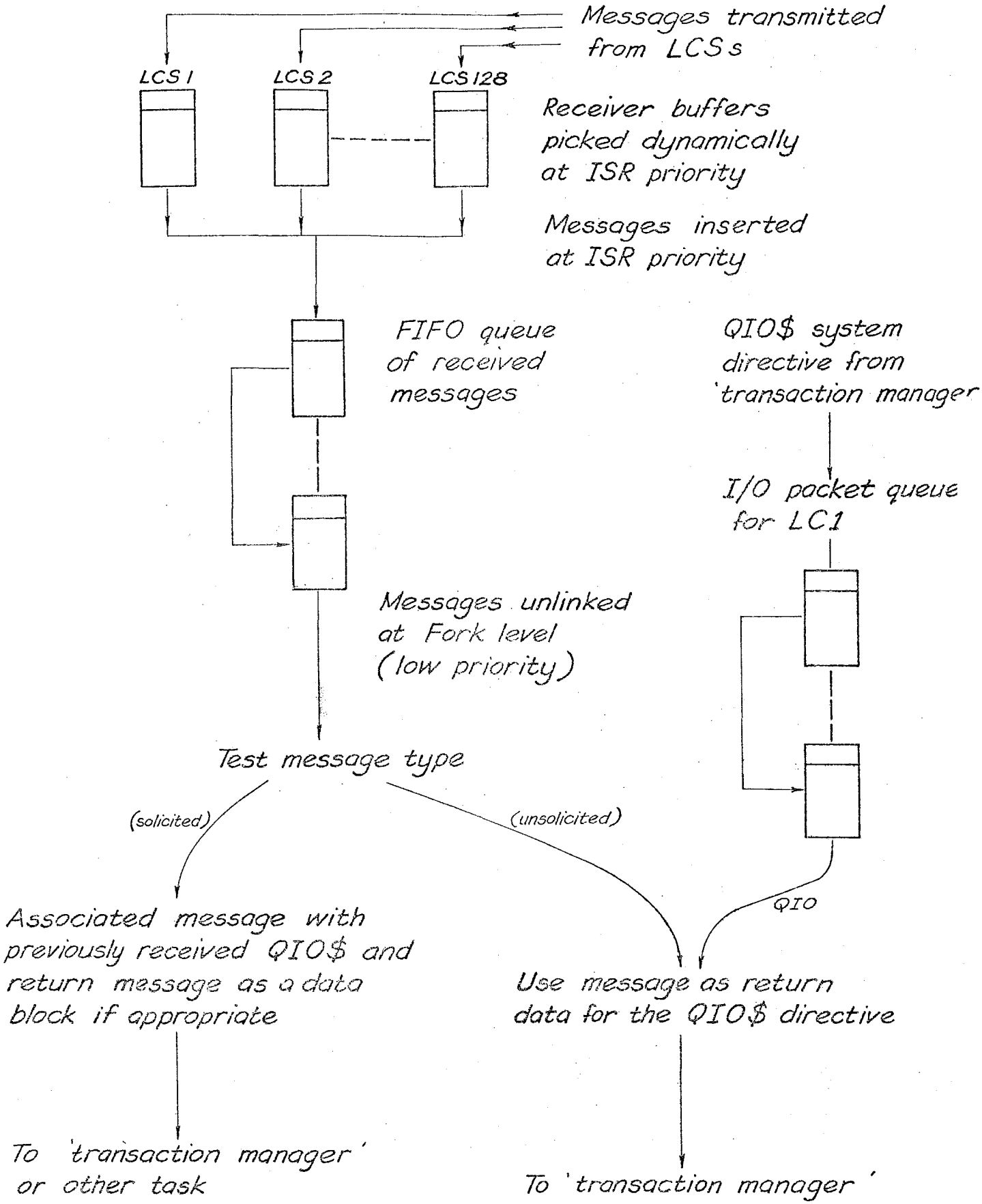
There are three different options available for clearing a silo -

(i) after each byte received



Handling by LCS driver of messages directed to LCSs

fig. 7.1



Handling by LCS driver of LCS messages received by Data Concentrator

fig. 7.2

(ii) after 16 bytes have arrrived since the silo was last emptied,

or

(iii) after a pre-determined time interval.,

It was decided not to use option (i) since the overheads of entering the interrupt service routine so frequently would be excessive. A combination of op ons (ii) and (iii) was therefore chosen with the time interval being one second to prevent messages less than 16 bytes in length from sitting in the silo too long.

Once the COUNT byte of a message has been received, a memory buffer is allocated by the LCS driver using the core allocation routine $ALOCB (Allocate Block), which will hold at least COUNT bytes.

If no buffer can be allocated then a flag will be set and the COUNT and subsequent bytes of the present message being received on this link are discarded. If the message is solicited, then an attempt will be made to retransmit it, otherwise no action will be taken.

If buffer allocation is successful then the COUNT byte will be stored in the first byte of the buffer. Subsequent bytes will be stored in successive positions of the buffer and added to the checksum so far.

When the message has been received, it must undergo several checks. This additional processing cannot be done at interrupt service routine level since interrupt processing would last too long, resulting in possible data overrun in a DZ11 silo.

The additional processing is therefore done at fork level which is between interrupt and task level ([5] pp 2-9,2-15). When a message has been received it is entered into the FIFO queue of received messages for that LCS. Once the interrupt service routine has terminated, the received messages can be retrieved from the queue and processed.

When processing a received message, several cases exist -

1. Message contains errors, or is a NAK

If the message received contains errors, or is a NAK, then the memory buffer containing the message will be deallocated using the system routine $DEACB (Deallocate Block). If the message is solicited then an attempt will be made to send the earlier message from the DC again, otherwise no action will be taken.

2. Message is an ACK

If the message is an ACK, then the driver associates it with the earlier message sent to the LCS, and for which the ACK is the response. The driver then declares the 'I/O done', using $IODON, for the I/O packet that generated the earlier message, and the ACK is discarded.

3. Message is a response to 'Read'

If the message is a response to a 'Read' message, then a buffer will have been previously allocated by the task which issued the 'Read'. The address of the buffer will be contained in the I/O packet and the data returned by the LCS will be copied into this buffer. The $IODON procedure is then called to declare the 'Read' complete.

In 2 and 3 above, the LCS driver then attempts to dequeue and transmit another message to the LCS.

4. Message is unsolicited

If the message received is unsolicited, e.g. a status return, then the message must be dispatched to a task. Unfortunately, the LCS driver, being part of the Executive, must conform to system conventions which state that a driver cannot issue a QIO$ directive to dispatch messages. A driver does, however, have access to all the system tables and it could manipulate these tables in such a way as to effect a QIO$. This approach would be complicated and difficult to debug.

Instead the following approach was chosen since it is simpler and conforms to all the system conventions. A pseudo-device called LC1: was created which is used for communication between a task called TXMNGR and the LCS driver.

TXMNGR issues I/O requests at regular intervals, via the QIO$ mechanism, to device LC1. The LCS driver, on the other hand, builds up a FIFO queue of entries pointing to unsolicited messages which have to be dispatched. When the LCS driver receives an I/O packet from TXMNGR it calls the $IODON procedure passing the size and address of the unsolicited message, referred to in the entry at the head of

the FIFO, in the I/O status block as shown below.

I/O Status Block

Byte	Function
0	driver disposition code
1	size in bytes of the unsolicited message
2,3	address of the unsolicited message

TXMNGR, since it is a task can then issue the message to the appropriate destination using a QIO$ directive. The QIO$s from TXMNGR to device LC1: are therefore 'dummy' QIO$s which allow the LCS driver to issue messages.

CHAPTER 8

PROTOCOL STATISTICS

This section describes the reasons for making the protocol software gather statistics and outlines the statistics which are thought to be desirable. Appendix C presents the format of the statistics block designed.

8.1 USE OF STATISTICS

As part of the normal operation of the protocol software, simple statistics will be kept on each link by the data concentrator and the appropriate LCS. The statistics will be used to -

(i) Monitor any errors which occur on the links. It is expected that this will enable any minor link problems to be detected before they become serious.

(ii) Offer information on the level of link traffic over a certain time period. If performance is below expectation then changes will need to be made to the LCS driver. This information will also enable the monitoring of any effects which may arise, as more LCSs are added to the network.

(iii) Aid the debugging of the software

8.2 DESIRABLE STATISTICS

The following are desirable statistics to be kept by the protocol software. The statistics will be kept in a series of 8 and 16-bit counters. It is planned that a task will execute once a day to produce a report of the day's link operations, based on the contents of these counters. The counters will be zeroed after each reading and if they overflow a bit in a counter status word will be set to indicate this fact to the task reading the counters.

1. Message Statistics

Message statistics will be kept, for each link, on the relative frequency of each of the different types of message which can be transmitted or received by the data concentrator or an LCS.

There are five main message types (see Chapter 5) and each will require a 16-bit counter (2 bytes) since they will be issued frequently. The remaining 4 types of message will each be accounted for in an 8-bit counter, as they are issued less frequently.

Subtotal = 14 bytes/link

2. Error Statistics

Statistics will be kept, for each link, on the frequency of the following error conditions

- data concentrator timeouts,
- LCS timeouts,
- bad parameters received in a message e.g. a data field longer than the maximum allowed,
- checksum errors,
- parity errors,
- framing errors,
- overrun errors,
- Rx loop breakages,
- Tx loop breakages,
- unrecognisable messages

Each of these errors would be expected to occur very infrequently and so an 8-bit counter for each is sufficient.

Subtotal = 10 bytes/link.

3. Operating Statistics

Operating statistics will be kept for the LCS driver. These are -

(i) the mean queue length of messages waiting to be transmitted.

(ii) the average waiting time before a message is transmitted.

(iii) the number of times with that no message buffer is available for an incoming message.

Each of these operating statistics is kept in a separate word.

Subtotal = 6 bytes/link.

==========

Total = 30 bytes/link.

For the proposed maximum of 128 LCS's, 3840 bytes of memory are therefore needed to record the statistics described. At the time of writing it is not clear just how much memory will be available for this purpose. If the statistics must be pruned to save memory, then the error statistics should be retained on all links but the message and operating statistics implemented on a just a few selected links.

The operating statistics may be quite time consuming to calculate. If these statistics cannot be easily obtained, without affecting the performance of the driver, then they should be collected on a reduced set of links.

CHAPTER 9

CONCLUSIONS

This project has involved the investigation of the proposed Christchurch City Council network and the design of a protocol to suit its requirements.

The various design decisions have been documented in this report as well as a definition of the protocol and an outline of the implementation in the RSX-11S operating system.

A protocol design almost identical to that presented has been adopted and implemented by the Council, though at the time of writing it has not yet been tested.

Future work may be directed towards an evaluation of the performance of the network, with refinements being made to the LCS driver as the real-time interactions of the different network components become clearer.

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Mr G.A.Cox and Dr M.A.Maclean, for their help and enthusiasm with this project. I would also like to thank the Christchurch City Council's Traffic Engineering Division for the useful advice given by their staff, and the use of their PDP11/40 to prepare this report.

REFERENCES

[1] ABRAMS, BLANC AND COTTON. 'Computer Networks: A Tutorial', IEEE Press 1975.

[2] BURTON, H.O AND SULLIVAN, D.D. 'Errors and Error Control' in 'Computer Communications' - Edited by Paul E. Green and Robert N. Lucky, IEEE Press 1974.

[3] DIGITAL EQUIPMENT CORPORATION. 'DZ11 Users Guide',

[4] DIGITAL EQUIPMENT CORPORATION. 'DDCMP Specification 4.0' (Order number AA-D599A-TC)

[5] DIGITAL EQUIPMENT CORPORATION. 'RSX-11M Guide to Writing an I/O Driver'. (Order number AA-2600D-TC)

[6] DIGITAL EQUIPMENT CORPORATION. 'RSX-11M Executive Reference Manual'.

[7] MCNAMARA, J.E. 'Technical Aspects of Data Communication', Digital Press.

[8] STELMACH, E.V. 'Introduction To Minicomputer Networks', Digital Press.

APPENDIX A

GLOSSARY

ACK - The message sent from an LCS to the data concentrator, to acknowledge the reception of a correct message.

Channel - See link.

Control Centre - The set of rooms in the Christchurch Civic Offices where the Traffic Control staff and hardware are located.

Counting Site - On-street locations where triple-loop vehicle detectors have been installed. These are the principal data collection points for the present system. Also known as a triple-loop surveillance site.

Data Concentrator - The PDP11/04 minicomputer which concentrates data from the network of LCSs. See figures 1.2 and 1.3.

DC - See data concentrator.

Detector - An under-road device which can detect the passage of vehicles across it. Information from detectors is used to guide the changing of phases at an intersection, when the local controller is in IVA mode.

DM10 - The DM10 is the master controller of the present system. It converts phase control signals from the HOST into cable voltages which it sends to the local controllers to make them change phase. The DM10 also controls the system, from traffic plans stored in it's memory, when the HOST is out of action.

In the new system the DM10 will be replaced by the data concentrator.

DZ11 - A DZ11 is an asynchronous multiplexor used to interface 8 LCSs to the data concentrator.

Executive - Synonym for operating system.

FIFO - A queue operating under a First-In-First-Out discipline.

HOST - The PDP11/40 minicomputer which is connected to the data concentrator by the Interprocessor Link.

Installation - One or more controlled intersections connected to a single local controller.

$IODON - An RSX-11S procedure which returns I/O status information to a task when an I/O request initiated by that task is completed.

IVA - Independent Vehicle Actuation mode for a signal installation. Phase switching is performed under the control of the local controller's detectors, instead of the Control Centre. When a signal installation is in IVA mode it operates independently of all other signal installations.

LCS - Local Controller Supervisor.

LED - Light-Emmitting Diode.

Link - The 20-milliamp, twisted-pair cables between the data concentrator and an LCS.

Local Controller - The electronic device which changes the phases at an installation.

Mimic Display - A wall mounted map of Christchurch, which displays status information on each of the computer-controlled intersections.

NAK - The message sent from an LCS to the data concentrator, to reject a message from the data concentrator.

Occupancy - The total time for which a single vehicle detector registered that a vehicle was present over it's inductive loop. This value is the accumulation of times for all vehicles crossing the loop in a defined counting period.

Phase - A particular combination of red and green lamps which allow a defined set of vehicle movements through an intersection. Only one phase can be operative at any time.

Pseudo Device - A device declared to the RSX-11S Executive which does not exist physically, but which has a logical function.

QIO$ - An RSX-11S system directive which is used by tasks to Queue an I/O request to an I/O device.

RACK - Reload Acknowledge. See FLAGS/TYPE byte description in Chapter 5.

RSX-11S - The operating system used in the data concentrator.

Silo - A buffer in a DZ11, which can hold up to 64 bytes of data and 64 bytes of control information associated with that data.

Solicited Message - A message is solicited if it is a message from an LCS which is a response to a message previously issued by the data concentrator. For example READ MEMORY.

Status Return - A message issued from an LCS to report a change in status of either the LCS or the local controller it is supervising.

Surveillance Site - See counting site.

Traffic Plan - A definition, for a set of co-ordinated traffic installations, of the way in which phases at each installation change with time.

Traffic Counters - Counting hardware which is connected to a triple-loop vehicle detector and records volume and occupancy data for the site.

Triple-loop Vehicle Detectors - Inductive loops, sunk into the road on the main arterial roadways, which are connected to traffic counters. As a vehicle passes over the loops it is detected and volume and occupancy information in the counters is updated.

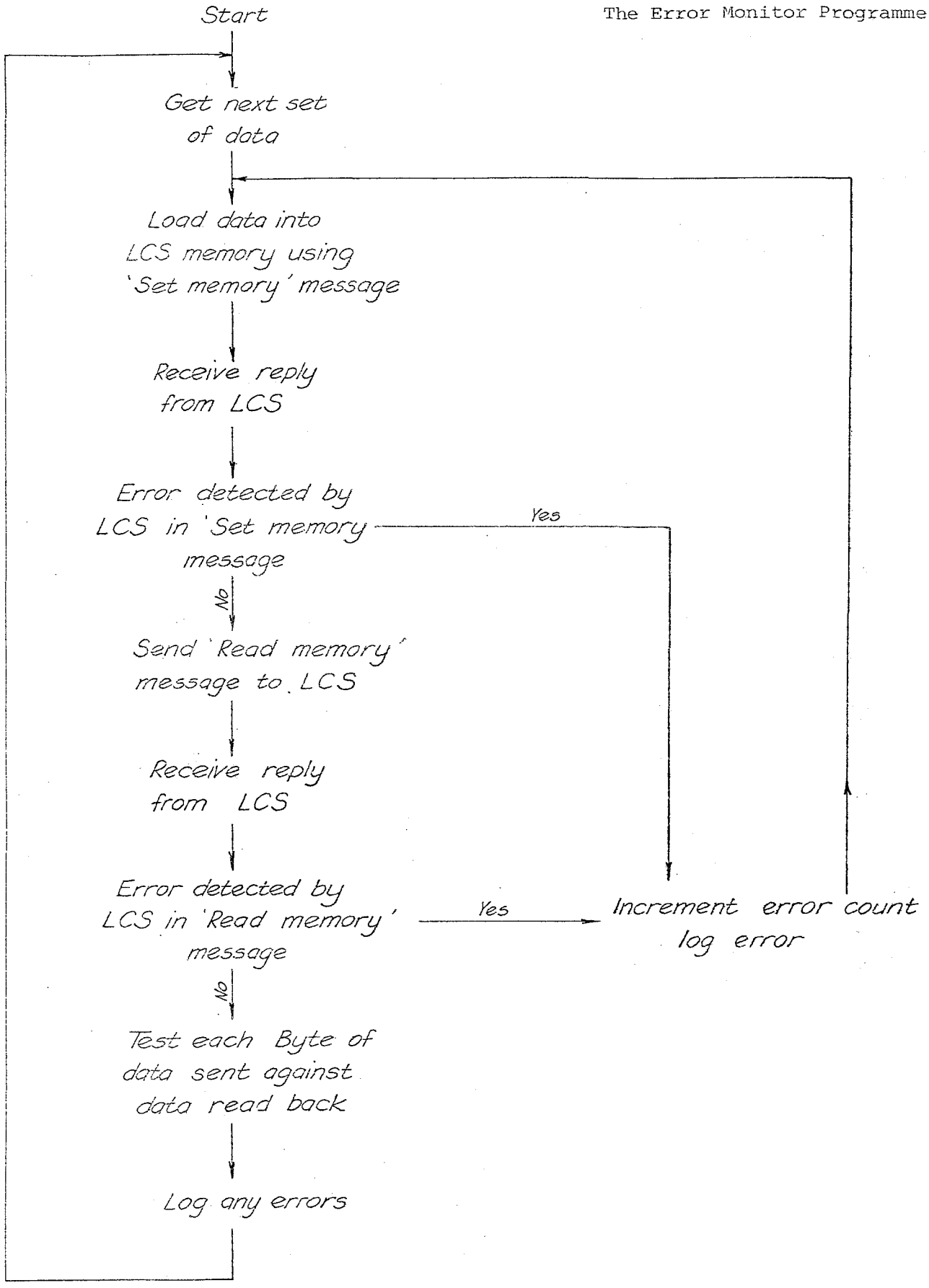
Triple-loop Surveillance Site - See counting site.

TXMNGR - Transaction Manager, a task which runs in the data concentrator.

Unsolicited Message - A message is unsolicited if it is issued spontaneously from an LCS, to the data concentrator, without prompt from the data concentrator. For example a Status Return message.

Volume - The number of vehicles per unit time recorded at a counting site.

APPENDIX B
The Error Monitor Programme



Flow chart of Link Error Monitor program

fig. B.1

```
      C     MONITOR FOR COMMUNICATION ERRORS BETWEEN LCS AND PDP11/40
      C     ==========================================================
      C
0001        INTEGER ERR,TYPE,TRACE,THREE
0002        INTEGER EFLG,SFRDF,TWFAL,TFRAL,TFTMO
0003        REAL LCSERR,PDPERR
0004        BYTE CHARS(128),CODE(42),MEMRED(42),LCSADR(3),RESP(42)
0005        BYTE DMY(9),HMS(8)
0006        COMMON / DIAG / TRACE
0007        COMMON / COMMS / LUN,EFLG,SFRDF,TWFAL,TFRAL,TFTMO
      C
0008        DATA IOSME,IONAK,IORME,IOSST / 6,21,10,5 /
0009        DATA LCSADR / 40,229,192 /
0010        DATA THREE / 3 /
      C      CALL ASSIGN(6,'DP6:LOGFILE.DAT',0)
0011        EFLG=16
0012        INTVAL=10800                        ! LOGGING INTERVAL FOR
      C                                         ! SUMMARY MESSAGES.
0013        KOUNT=0                             ! HOW MANY MESSAGES SENT
      C                                         ! THIS INTERVAL.
0014        LUN=1                               ! LCS LOGICAL UNIT #
0015        L=42                                ! # BYTES TO READ
0016        TRACE=0                             ! TRACE FLAG
0017        TRANUM=0                            ! TRANSACTION NUMBER
0018        NXTCHR=1                            ! POINTER TO NEXT CHAR.
0019        LCSERR=0                            ! NUMBER OF ERRORS PDP --> LCS
0020        PDPERR=0                            ! NUMBER OF ERRORS  LCS --> PDP
0021        SNTCNT=0                            ! COUNT OF TRANSACTIONS PDP --> LCS
0022        RCVCNT=0                            ! COUNT OF TRANSACTIONS LCS --> PDP
      C
0023        CALL SETERM                      !SET TERMINAL (LCS) CHARACTERISTICS
      C
      C     INITIALISE ARRAY OF ASCII CHARACTERS
      C
0024        DO 20 I=1,128
0025        CHARS(I)=I-1
0026     20 CONTINUE
0027        CODE(1)=229      ! ES
0028        CODE(2)=192      ! C0
      C
      C     SET UP THE NEXT 40 BYTE MESSAGE
      C
0029     30 TRANUM=TRANUM+1                     ! INCREMENT TRANSACTION NUMBER
0030        DO 40 I=3,42
0031        CODE(I)=CHARS(NXTCHR)
0032        NXTCHR=NXTCHR+1
0033        IF(NXTCHR.GT.128)NXTCHR=1
0035     40 CONTINUE
      C
0036     50 CALL LSEND(LUN,IOSME,L,CODE,ERR)
0037        SNTCNT=SNTCNT+1                     ! INCREMENT MESSAGES SENT COUNT
0038        KOUNT=KOUNT+1
0039        IF(KOUNT.LT.INTVAL)GO TO 55
0041        KOUNT=0
```

```
0042          CALL DATE(DMY)
0043          CALL TIME(HMS)
0044          WRITE(6,15)DMY,HMS
0045          WRITE(6,14)SNTCNT,LCSERR,RCVCNT,PDPERR,TRANUM
      C
0046       55 IF(ERR.EQ.0)GO TO 70
0048          LCSERR=LCSERR+1                          ! ERROR IF GET TO HERE
0049          IF( (LCSERR+PDPERR).GE.50)GO TO 9999   ! TOO MANY ERRORS SO STOP
0051          CALL DATE(DMY)
0052          CALL TIME(HMS)
0053          IF(ERR.GE.0)GO TO 65
0055       60 IF(ERR.EQ.-4)WRITE(6,1)DMY,HMS,TRANUM    ! PARITY ERROR
0057          IF(ERR.EQ.-66)WRITE(6,2)DMY,HMS,TRANUM    ! FRAMING ERROR
0059          GO TO 50
0060       65 WRITE(6,3)DMY,HMS,TRANUM                  !BLOCK LENGTH ERROR
0061          GO TO 50
      C
0062       70 CALL LRECV(LUN,TYPE,RLEN,RESP,ERR)
0063          RCVCNT=RCVCNT+1
0064          IF(TYPE.EQ.IOSST)GO TO 80
0066          CALL DATE(DMY)
0067          CALL TIME(HMS)
0068          PDPERR=PDPERR+1
0069          IF( (LCSERR+PDPERR).GE.50)GO TO 9999     ! TOO MANY ERRORS SO STOP
      C
0071          IF(TYPE.NE.IONAK)GO TO 75
0073          WRITE(6,4)DMY,HMS,TRANUM                 ! NAK BY LCS
0074          GO TO 50
      C
0075       75 WRITE(6,5)DMY,HMS,TRANUM,TYPE            ! UNEXPECTED RESPONSE
0076       77 CALL LRECV(LUN,TYPE,RLEN,RESP,ERR)       ! FLUSH QUEUE
0077          IF(ERR.NE.2)GO TO 77
0079          GO TO 50
      C
0080       80 CALL LSEND(LUN,IORME,THREE,LCSADR,ERR)   ! REQUEST TO READ LCS ME C
0081          SNTCNT=SNTCNT+1                          ! INCREMENT MESSAGES SENT C
0082          KOUNT=KOUNT+1
0083          IF(KOUNT.LT.INTVAL)GO TO 90
0085          KOUNT=0
0086          CALL DATE(DMY)
0087          CALL TIME(HMS)
0088          WRITE(6,15)DMY,HMS
0089          WRITE(6,14)SNTCNT,LCSERR,RCVCNT,PDPERR,TRANUM
      C
0090       90 CALL LRECV(LUN,TYPE,RLEN,MEMRED,ERR)
0091          RCVCNT=RCVCNT+1                 ! INCREMENT MESSAGES RECEIVED COUNT
0092          IF(TYPE.EQ.IORME)GO TO 110
0094          CALL DATE(DMY)                ! SOME SORT OF ERROR IF YOU REACH HE E
0095          CALL TIME(HMS)
0096          PDPERR=PDPERR+1
0097          IF( (LCSERR+PDPERR).GE.50)GO TO 9999   !TOO MANY ERRORS SO STOP
      C
0099          IF(TYPE.NE.IONAK)GO TO 100
0101          WRITE(6,4)DMY,HMS,TRANUM     ! NAK BY LCS
```

```
0102          GO TO 80
      C
0103       100 WRITE(6,5)DMY,HMS,TRANUM,TYPE     ! UNEXPECTED RESPONSE
0104       105 CALL LRECV(LUN,TYPE,RLEN,RESP,ERR)   ! FLUSH QUEUE
0105          IF(ERR.NE.2)GO TO 105
0107          GO TO 80
      C
0108       110 IF(ERR.EQ.0)GO TO 170
0110          CALL DATE(DMY)                ! ERROR IF GET TO HERE
0111          CALL TIME(HMS)
0112          PDPERR=PDPERR+1
0113          IF( (LCSERR+PDPERR).GE.50)GO TO 9999   ! TOO MANY ERRORS SO STOP
      C
0115          IF(ERR.NE.-4)GO TO 120
0117          WRITE(6,6)DMY,HMS,TRANUM     ! PARITY ERROR
0118          GO TO 30
      C
0119       120 IF(ERR.NE.-66)GO TO 130
0121          WRITE(6,7)DMY,HMS,TRANUM    ! FRAMING ERROR
0122          GO TO 30
      C
0123       130 GO TO (140,150,160,170)ERR
0124          WRITE(6,8)DMY,HMS,TRANUM    ! UNKNOWN ERROR
0125          GO TO 30
      C
0126       140 WRITE(6,9)DMY,HMS,TRANUM    ! BLOCK LENGTH ERROR
0127          GO TO 30
      C
0128       150 WRITE(6,10)DMY,HMS,TRANUM   ! NO LCS RESPONSE
0129          GO TO 30
      C
0130       160 WRITE(6,11)DMY,HMS,TRANUM   ! CHECKSUM ERROR
      C
0131       170 DO 180 I=3,42
0132          IF( CODE(I).EQ.MEMRED(I) )GO TO 180
0134          WRITE(6,12)DMY,HMS,TRANUM,CODE(I),MEMRED(I),I
0135       180 CONTINUE
0136          GO TO 30
0137      9999 WRITE(6,13)DMY,HMS
0138          WRITE(6,14)SNTCNT,LCSERR,RCVCNT,PDPERR,TRANUM
0139          CALL CLOSE(6)
0140           1 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  LCS - PARITY ERROR')
0141           2 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  LCS - FRAMING ERROR')
0142           3 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  LCS - BLOCK ERROR')
0143           4 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  LCS - NAK ERROR')
0144           5 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - UNEXPTD RESPONSE',I6)
0145           6 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - PARITY ERROR')
0146           7 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - FRAMING ERROR')
0147           8 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - UNKNOWN ERROR')
0148           9 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - BLOCK ERROR')
0149          10 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - NO LCS RESPONSE')
0150          11 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - CHECKSUM ERROR')
0151          12 FORMAT(1X,9A1,2X,8A1,2X,'# ',F11.0,'  PDP - ',I4,2X,I4,2X,I4)
0152          13 FORMAT(1X,9A1,2X,8A1,4X,'  *** FINAL REPORT ***')
```

,DP6:W.LST=DP6:ERRMON

```
0153      14 FORMAT(1X,'PDP',F11.0,2X,F11.0,' LCS',F11.0,2X,F11.0,3X,F11.0)
0154      15 FORMAT(1X,9A1,2X,8A1,4X,'  *** INTERIM REPORT ***')
0155         STOP
0156         END
```

LOCAL VARIABLES, .PSECT $DATA, SIZE = 000512 (165. WORDS)

NAME	TYPE	OFFSET	NAME	TYPE	OFFSET	NAME	TYPE	OFFSET
ERR	I*2	000444	I	I*2	000504	INTVAL	I*2	000460
IONAK	I*2	000424	IORME	I*2	000426	IOSME	I*2	000422
IOSST	I*2	000430	KOUNT	I*2	000462	L	I*2	000464
LCSERR	R*4	000450	NXTCHR	I*2	000472	PIPERR	R*4	000454
RCVCNT	R*4	000500	RLEN	R*4	000506	SNTCNT	R*4	000474
THREE	I*2	000432	TRANUM	R*4	000466	TYPE	I*2	000446

COMMON BLOCK /DIAG /, SIZE = 000002 (1. WORDS)

NAME	TYPE	OFFSET	NAME	TYPE	OFFSET	NAME	TYPE	OFFSET
TRACE	I*2	000000						

COMMON BLOCK /COMMS /, SIZE = 000014 (6. WORDS)

NAME	TYPE	OFFSET	NAME	TYPE	OFFSET	NAME	TYPE	OFFSET
LUN	I*2	000000	EFLG	I*2	000002	SFRDF	I*2	000004
TWFAL	I*2	000006	TFRAL	I*2	000010	TFTMO	I*2	000012

LOCAL AND COMMON ARRAYS:

NAME	TYPE	SECTION	OFFSET	-------SIZE------	DIMENSIONS
CHARS	L*1	$DATA	000000	000200 (64.)	(128)
CODE	L*1	$DATA	000200	000052 (21.)	(42)
DMY	L*1	$DATA	000401	000011 (5.)	(9)
HMS	L*1	$DATA	000412	000010 (4.)	(8)
LCSADR	L*1	$DATA	000324	000003 (2.)	(3)
MEMRED	L*1	$DATA	000252	000052 (21.)	(42)
RESP	L*1	$DATA	000327	000052 (21.)	(42)

SUBROUTINES, FUNCTIONS, STATEMENT AND PROCESSOR-DEFINED FUNCTIONS:

NAME	TYPE	NAME	TYPE	NAME	TYPE	NAME	TYPE	NAME	TYPE
CLOSE	R*4	DATE	R*4	LRECV	I*2	LSEND	I*2	SETERM	R*4
TIME	R*4								

APPENDIX C

STATISTICS BLOCK FORMAT

Whenever a task issues a QIO$ with function code IO.GCS (Get Channel Statistics), parameter 1 equal to the address of the buffer and parameter 2 equal to the size of the buffer, the following block of data is returned to the task. All counter readings refer to the number of occurences since the counters were last read.

OFFSET INTO BLOCK	DESCRIPTION
0,1	number of SET PHASE messages issued.
2,3	number of STATUS RETURN messages issued.
3,4	number of SEND SYSTEM STATUS messages issued.
5,6	number of COUNTER SERVICE messages issued.
7,8	number of RESPONSE TO COUNTER SERVICE messages received.
9	total number of TIME CHECK, SET MEMORY, READ MEMORY and RESPONSE TO READ MEMORY messages received.
10	number of data concentrator timeouts
11	number of LCS timeouts

12	number of messages received with bad parameters
13	number of messages received with checksum errors
14	number of messages received with parity errors
15	number of messages received with framing errors
16	number of messages received with overrun errors
17	number of Rx loop breakages
18	number of Tx loop breakages
19	number of messages received which were unrecognisable