Some aspects of a code division multiple access local area network

Pearce, Richard Sargon

PhD Thesis

"Some Aspects of a Code Division Multiple Access Local Area Network"

by Richard Sargon Pearce

Department of Applied Physics
& Electronics, University of Durham

1987

Thesis title - **Some Aspects of a Code Division Multiple Access Local Area Network**

Author - **R.S.Pearce**

This work is concerned with the design of a data communications network suitable for use within a Naval Shipborne Command and Control System. The use of Spread Spectrum techniques to allocate portions of the total network's bandwidth to communicating network users, resulted in the requirement for a new network architecture. This was in order to best utilise the inherent useful properties of such a "media access technique". Other considerations that the architecture had to take into account were the Naval demands on such a network and compliance with emerging International data communications standards.

In addition to the network architecture that was produced, the work also provides the specifications for the functional layer protocols and services required within the architecture. These are required for the interoperability of devices implemented which use the architecture as a reference model. The specifications are presented using descriptive techniques used by standards bodies to refer to established data communications networks. Also included are details of the design and construction of a testbed system for the analysis of the performance of the Spread Spectrum media access technique.

Acknowledgements

During the time in which the work presented here was undertaken, the contributions of many others were invaluable to its progress and eventual completion. Firstly, thanks must go to XCC division at the Admiralty Research Establishment, whose interest in the network led to the funding of this research work. In particular, thanks go to Dr Tony Mobbs, whose help enabled me to get an understanding of Naval Systems which was essential to this work.

During the time taken to undertake this work, the relaxed atmosphere and free exchange of information and ideas within the Department of Applied Physics and Electronics and especially the Digital Research Group, did much to make this period an enjoyable and inspiring one. I am greatly indebted to my supervisor, Professor C.T.Spracklen, not only for for his guidance and invaluable help during the period of this research, but also for inspiring and maintaining my interest in many fields of electronics and computing.

I would also like to thank Alistair Watt, the Managing Director of the company where I presently work, for giving me time off from that work in which to complete the writing of this volume. Finally, many thanks must go to my family and friends, whose constant support and understanding have made an immense contribution to the completion of this work.

Introduction

The work contained within this thesis is concerned with the design of a data communications network for use within a Naval Shipborne Command and Control System. The uniqueness of the network is in its use of completely digital Direct Sequence Spread Spectrum techniques for access to the transmission medium. Since this work entailed the bringing together of technical areas not usually associated with each other, the first four chapters include introductory material which clarify the meaning of the following chapters dealing more directly with the network's development.

The first chapter introduces the concept of a Local Area Network, the reasons why they exist and the properties associated with the various sub-types of such networks. The pro's and con's of the various existing types of networks is discussed The second chapter introduces the subject of Naval Shipborne C^2 Systems, including their overall objectives and the manner in which they have been implemented to date. The need for a new type of data communications network within future systems of this type is analysed, with a discussion on how the form of such a network will be dictated to by the naval requirements. This chapter is followed by an introduction to the emerging International Standards for data communications networks. Adherence to such standards is required if the interoperability of equipment from different manufacturers is to be included within the network. It was advised that such a situation would occur within future Naval networks. Since such standards were to have a large effect on the architecture of the Naval network designed later in the text, it was considered worthwhile to introduce much of the terminology associated with the subject.

The final introductory chapter deals with the subject of Spread Spectrum techniques, the range of techniques they embrace, as well as the properties that each exhibit. The reasons for selecting a particular technique from the available family for use as a medium access technique are discussed, along with the requirements that it places upon the network in which it is to be used. The digital schemes used to realise the Spread Spectrum access technique in the LAN are introduced, along with a discussion of how some basic communications problems are dealt with.

The next chapter is concerned with the testbed system which was designed and built in order to study the access technique in detail. The software developed for this task is included within Appendices C & D. Following this, the design of the network's architecture is discussed within the following three chapters, along with a specification of the logical and physical interfaces present within the system and the information passed across them. Finally, the concluding chapter summarises the results of the work presented and also discusses the direction in which future related work should be directed.

Contents

Page

Chapter1 - Introduction to LANs

1.1 - Introduction
1 - Definition of a Local Area Network 1.1.1
2 - Factors contributing to LAN development 1.1.2
3 - Types of LAN user application 1.1.4

1.2 - Types of LANs
1 - Methods of classification 1.2.1
a - Topologies 1.2.1
b - Media Allocation Techniques 1.2.2

1.3 - LAN Performance Characteristics
1 - Scope of performance study 1.3.1
2 - Data packet comparison 1.3.1
3 - Performance comparison 1.3.4

1.4 - Other LAN Issues 1.4.1

1.5 - References 1.5.1

Chapter 2 - Naval Shipborne Command and Control Systems

2.1 - Introduction
1 - Introduction 2.1.1
2 - Benefits of a Distributed C^2 System 2.1.2
3 - Military requirements for a distributed C2 System 2.1.3

2.2 - The Royal Navy's DDP System
1 - Hardware configuration 2.2.1
2 - Software description 2.2.1
3 - The ASWE Serial Highway 2.2.3

2.3 - LAN requirements for a future C^2 System
1 - The need for a new LAN 2.3.1
2 - Topology and MA scheme of the future LAN 2.3.1

2.4 References 2.4.1

Chapter 3 - Communications Standards for LANs

3.1 - Introduction
1 - The need for standards 3.1.1
2 - LAN related standards activities 3.1.1

3.2 - Open Systems Interconnection
1 - Basic definitions, aims and constituent elements 3.2.1
2 - The principle of layering 3.2.2
3 - Introduction to the seven layers 3.2.3
4 - Protocols and Services 3.2.5
5 - Service-access-points 3.2.7
6 - Communications structure and connections 3.2.8
7 - Management of an OSI network 3.2.9

3.3 - Refining OSI for LANs
1 - The need for refinements 3.3.1
2 - Connections and connectionless data transmission 3.3.1
3 - IEEE 802 committee standards 3.3.3

3.4 - Additional Issues
1 - Network interconnection 3.4.1
2 - Splitting the layers 3.4.3
3 - Relevance of issues to a C^2 system 3.4.4

3.5 - References 3.5.1

Chapter 4 - S-Spectrum Techniques for Multiple Access to Media

4.1 - Spread Spectrum Overview
1 - Introduction 4.1.1
2 - Types of Spread Spectrum modulation 4.1.2
3 - Types of code sequences used in SS systems 4.1.4
4 - Descriptive parameters of pseudorandom sequences 4.1.6
5 - Performance comparison between diff' seq' types 4.1.8

4.2 - Provision of CDMA for a C^2 System
1 - Requirements for the multiple access technique 4.2.1
2 - Direct sequence modulation analysis 4.2.1
3 - A generalised Direct Sequence SS system 4.2.5
4 - Provision of a digital DSSS system 4.2.5

4.3 - Digital CDMA System Requirements
1 - Determination of correlator threshold values 4.3.1
2 - Necessary properties of code sequence family 4.3.2
3 - Communication media requirements 4.3.3

4.4 - References 4.4.1

Chapter 5 - CDMA Testbed System

5.1 - Introduction
1 - The need for a testbed system 5.1.1
2 - Aims for testbed system 5.1.2

5.2 - Transmitter/Receiver Controller Overview
1 - Choice of hardware/software 5.2.1
2 - Controller-Tx/Rx interface 5.2.2

5.3 - Transmitter/Receiver Controller Software
1 - Software design philosophy 5.3.1
2 - Functional breakdown of software 5.3.2
3 - "Main" subroutine
a - Introduction 5.3.3
b - Code (seq') data and address generator 5.3.5
c - "Codeload " subroutine 5.3.7
4 - "Dataload" subroutine 5.3.7
5 - "File transfer" subroutine 5.3.8
6 - "Word error count" subroutine 5.3.8

5.4 - Use of System
1 - Controller debugging 5.4.1
2 - Results from testbed system 5.4.2

Chapter 6 - Full Scale System Design

6.1 - Introduction 6.1.1

6.2 - Major Network Features
- 1 - Choice of topology 6.2.1
- 2 - Choice of communications service provided 6.2.2
- 3 - Network node overview 6.2.4
- 4 - Int'n to node component use and interactions 6.2.6

6.3 - Issues Related to Individual Data Links
- 1 - Introduction 6.3.1
- 2 - Link addressing 6.3.1
- 3 - Types of data channel provided 6.3.2
- 4 - Provision of link privacy 6.3.3
- 5 - Data-link setup and termination 6.3.4

6.4 - Network/Node Management Issues
- 1 - Introduction 6.4.1
- 2 - Addition/removal of nodes to/from system 6.4.1
- 3 - Node resource allocation 6.4.2
- 4 - Network security
  - a - Introduction 6.4.3
  - b - Data link security/privacy 6.4.4
  - c - Red-Black seperation 6.4.7

Chapter 7 - Node Component Services

7.1 - LLC-Service Implementation Overview
- 1 - LLC services supported 7.1.1
- 2 - LLC operation component mapping to node blocks 7.1.2
- 3 - Specification of the Spreadnet Node Interface 7.1.4

7.2 - Interface Specifications
- 1 - Interface 1 - The LLC service 7.2.1
- 2 - Interface 2 - Control block service 7.2.5
- 3 - Interface 3 - Link to i/o block interface 7.2.6
- 4 - Interfaces 4 & 5 - Control to link block interfaces 7.2.6
- 5 - Interface 6 - Link block to Tx/Rx h'ware interface 7.2.13

7.3 - References 7.3.1

Chapter 8 - Node Component Structure and Protocols

8.1 - Logical Machines
- 1 - Introduction 8.1.1
- 2 - Connection-less channel link block 8.1.2
- 3 - Connection-oriented link block 8.1.6
- 4 - Input/Output block 8.1.9
- 5 - Control Block 8.1.10
- 6 - Related information 8.1.16

8.2 - Spreadnet Protocols
- 1 - Method of specification 8.2.1
- 2 - PCM protocol 8.2.1
- 3 - ACM protocol 8.2.2
- 4 - CPM protocol 8.2.3
- 5 - Link block operating modes 8.2.4
- 6 - CLM protocol 8.2.4
- 7 - COM protocol 8.2.4
- 8 - SMM protocol 8.2.7

Chapter 9 - Conclusion

9.1 - Conclusion 9.1.1

9.2 - Future Work 9.2.1

Appendix A - Data Encryption Measures

Appendix B - IEEE 802.4 Frame Control Fields

Appendix C - Transmitter Controller Software

Appendix D - Receiver Controller Software

Chapter One
Introduction to LANs

1.1 Introduction.

1.1.1 Definition of a Local Area Network (LAN).

The definition for this sub-class of data communications network has been attempted a number of times by many authors and organizations but none have yet received general acceptance even though they have become successively more general and vague in nature. The difficulty in developing such a definition was first identified by Freeman and Thurber who published a bibliography of all the existing networks at the time (1980) which bore the LAN title [1.1] and compared the varying LAN definitions in existence. The characteristics used in these definitions included the bit rate of the transmitted data, size of the network, transmission medium used, switching technology used, percentage of its total data traffic not sent to other networks via interconnection devices, single organization proprietorship, single function usage, use of distributed processing and the network's configuration.

Although the majority of these definitions, including one formulated by Thurber and Freeman themselves, were based on the consideration of the properties that existing LAN systems possessed, an alternative approach was first suggested by Metcalfe and Boggs [1.2] in 1976. They regarded LANs as filling a "logical gap" in a data communications spectrum which was bounded by remote networking and multiprocessing schemes, the data ranges associated with each type being summarised in Table 1.

Network Type	Seperation (km)	Bit Rate (Mbit/s)
Remote (long haul) networks	>10	<0.1
LANs	10-0.1	0.1-10
Multiprocessors	<0.1	>10

Table1. Data Communications Spectrum.

Multiprocessor schemes allowed the sharing of processors, memory or I/O devices one instruction or one data word at a time because of the low latencies and high throughput available, but the number of processors that could be effectively connected was low (typically three) without specific hardware/software optimization. At the other end of the spectrum, remote networks had high latencies and a much lower throughput which meant that the sharing of processor time, memory or I/O devices was not feasible except in units of coarse granularity (eg a file). However, a much greater number of computers could be interconnected (approximately

150) before network management posed problems. Therefore LANs included any system which offered some sort of compromise between the two sets of properties.

This seems to be the best type of approach to define the large number of diverse networks being produced under the "LAN" banner, even though the boundaries between the system types have proved to be hazy.

1.1.2 Factors contributing to LAN development.

The need for a localised communications network between computers arose from a continuing trend which had begun some years earlier in the 1960s. During this era, work destined for computers was sent to centralised data processing centres, containing large mainframe computers, in the form of either punched card or paper tape. This arrangement was based on the fact that the electronic hardware was very costly and so had to be shared among large groups of users. As the cost of the hardware decreased, remote terminals for interactive use of computing resources were introduced, thus decentralizing a small portion of the overall computing process. The continued fall in the cost of the hardware meant that this was no longer the overriding consideration when planning computer systems. Human and political arguments against centralization were now considered as well as the increasing cost of the phone lines used to link the remote terminals to the central computer. Grosch's law (cost of processing is inversely proportional to the square of the size of the computer) was eventually overruled in the 1970's and so the decentralization of more processing functions could occur.

With the growing sophistication of the end-users in computing systems, the centralized hardware systems could not raise processing speed fast enough to cope with the increased load placed on their CPU's by ever more demanding software, which was requested in increasingly large amounts. Clearly, distribution of some processing functions was necessary to off-load tasks previously undertaken by the central CPU. Indeed the same functions could often be done more economically with a set of interconnected, low-cost, special purpose or dedicated processors (as opposed to the general-purpose nature of the older larger computers which ran specialized software).

Over approximately the same period, developments in the field of data communications allowed this desire for computer interconnection to be acheived more efficiently. Until the early 1970's, most computer networks were built around traditional telecommunication systems designed to transport analogue speech signals. These networks carried data transmitted character-by-character, avoiding the possible confusion that a bit representation of the data could give if it contained sequences which would be interpreted as control characters. This was extremely wasteful of time and communication capacity.

It was noted that data traffic between or among computers had a high peak-to-average data rate requirement. These "bursts" of data could be encapsulated within blocks which contained special fields to encapsulate control and data information. Fields could also be specified to delimit the packet. Any bit sequence was hence allowable in the latter, variable length, field and so the transmission of data was sent in a more efficient bit representation scheme. The earliest published material on the subject of a new network switching method for sending data in this block format, was produced by Baran [1.3] at the RAND Corporation in 1964. The following year, independently of the work at RAND, D.W.Davies of the British National Physical Laboratory spoke of "packet" switching using terrestrial circuits and minicomputers (whose cost was falling) as the "store-and-forward system". By late 1969, the first four nodes attached to a country-wide data network in America, ARPANET, were in operation. This system used packet-switching techniques to route the transmitted data, as did units on the SITA network in France and the TYMESHARE Corporation network in the USA.

After rapid growth, the autumn of 1972 saw an important demonstration of the ARPANET network at the first International Conference on Computer Communications (ICCC). This established the credibility of packet switching among the computer communications community. The demonstration showed that the network could deliver very fast responses (less than 250ms) and could interconnect a wide variety of computers. In the early to mid-seventies, a number of packet-switched networks were in existence in Europe, Japan and Canada, each being developed and run by their respective PTTs (eg EPSS in the UK, run by the Post Office). The first public "value-added" packet switched service was offered by TELENET Communication Corporation, who successfully filed the Federal Communication Commision to do so in 1973, becoming operational by August 1975.

By the time that these networks were beginning to offer a communications service, the advances in hardware technology, especially the introduction of microprocessors, (Intel 4004 in 1972 was the first openly marketed device) had created a larger and substantially different communications market. By the mid-seventies, so much processing power was included into cheap devices such as typewriters and small personal computers, that their interconnection was seen to be desirable. However, the packet-switched networks in existence were tailored specifically for the reliable interconnection of medium/large computers over large distances, resulting in the high cost of the interface units to the network. Interface units which cost more than the transmitting devices themselves were obviously not attractive, especially when the majority of the communications taking place were over very short distances, as is the case in many organisations (eg in excess of 75% of all information is distributed within 600ft for office workers[1.4]).

This need for a low-cost, short distance communications system was shown to be solvable by networks such as the "Experimental Ethernet" (developed at the Xerox Palo Alto Research

Center from 1972 onwards [1.2]) which combined the use of the packet format for the data with the use of a cheap high speed bus, as had been used in multiprocessor communications. These networks provided transmission speeds and error rates comparable to the connected computers themselves, using the kind of technology whose cost was known to be dropping. Therefore, these LANs generated much interest among the rapidly growing number of organisations who were amassing large numbers of sophisticated processing devices with little or no interconnection between them.

1.1.3 Types of LAN user applications.

Three generic types of user applications have been identified [1.5] for LANs; resource sharing, distributed computing and communications multiplexing.

In a Resource Sharing application, the LAN is used to interconnect inexpensive computer units to more expensive resources whose purchase is not justified for each individual user. (eg. expensive units, such as line printers or mass storage devices, can be shared by many personal computers.)

Distributed Computing applications involve the partitioning of the applications software over a number of computers. Reasons for this partitioning could concern fault tolerance, off-loading of heavy CPU loading, utilization of special processors for seperate tasks or merely the economic use of the total system power available in any organization.

Communications Multiplexing applications are used to reduce system costs by the sharing of a data communications channel (as opposed to the first case when the resource terminating the link is shared). For example, a cluster of CRT terminals located at some distance from the computer processor running the applications software could use a LAN in order to provide a common, easily reconfigurable connection from each terminal to the computer without the need to install a seperate line.

1.2 Types of LANs.

1.2.1 Methods of Classification.

Since 1980, a number of publications have appeared which include methods for the classification of LANs [eg 1.6-1.13], none of which have been suitable for all circumstances. The methods used have included "degree of sophistication", bit rate supported, physical transmission medium used, network topology, resource sharing mechanism (media access method), degree of network control centralization and mode of transmission. Overlap between these classes is not uncommon, for example packet switching is both a transmission mode for the network as well as its resource sharing mechanism which causes problems when using more than one class to subdivide network groups. The most successful classification method to date considers the topologies and/or media allocation techniques employed by the networks with a coarser distinction being made between the older "centralised control" networks and the more recent "decentralised" networks.

The main types of network topology and media allocation techniques will now be briefly outlined before a more detailed breakdown is attempted.

1-Topologies

Fig 1.2.1 shows the various types of topology that have been used in inter-computer communications networks. The darkly shaded squares represent the "nodes" (hardware attaching computer(s) to communications line(s)) attached to the system, the more lightly shaded sqares representing a specialised network control node.

The Fully-connected topology (a) and the Hierarchical topology (b) interconnect nodes via dedicated point-to-point links and are the oldest types used for LANs due to the relative simplicity of the communications hardware required. Networks using centralised controller nodes are of three types, (c) shows the Star topology, (f) shows the Multipoint topology and (h) shows the Loop Topology. These are relatively old in design, since they were based on the assumption that the controlling hardware was too expensive to distribute.

The Mesh Network topology illustrated in (d) is really only suitable for long-haul networks (WANs) using expensive data communications lines between large expensive computers so that the complexity and costs of the individual nodes can be relatively high. Since LANs are used to interconnect nodes which can be of a low cost, using low cost in-house communications lines, the necessary expense for this type of node is not often justifiable.

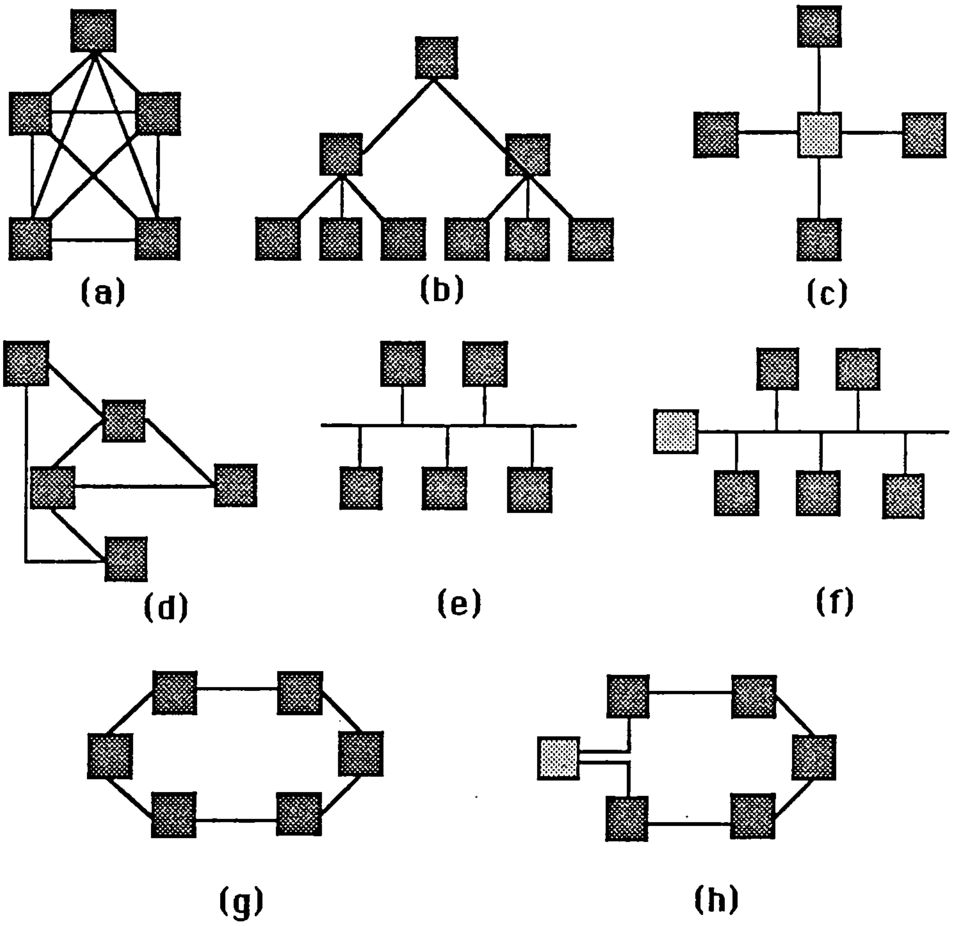


Fig 1.2.1 Network Topologies.

The bus topology (e) and ring topology (g) are the most recent types to appear and generally exhibit de-centralised control. These are the most important topologies for the future since they are the only ones considered in emerging international LAN standards. (also loop and multipoint now)

Media Allocation Techniques.

Given the limited transfer rate afforded by any transmission medium (as stated by Information Theory), this can be allocated in three basic ways, as illustrated in Fig.1.2.2. Time Division Multiplexing (TDM) involves the allocation of the entire bandwidth of the transmission medium to one transmitting node at a time. Frequency Division Multiplexing (FDM) involves the division of the transfer capacity of the transmission medium between a number of concurrent channels, each based on a unique carrier wave frequency (the digital data is modulated onto a particular carrier frequency before transmission over the network). The final scheme illustrated is a combination of both techniques, each FDM channel being shared using the TDM technique outlined previously.

TDM is an example of a "Baseband" signalling technique, that is the signal is transmitted in an unmodulated form. FDM and the hybrid technique use "Broadband" signalling, that is the available bandwidth is sub-divided into discrete channels allowing concurrent transmissions to occur over them.

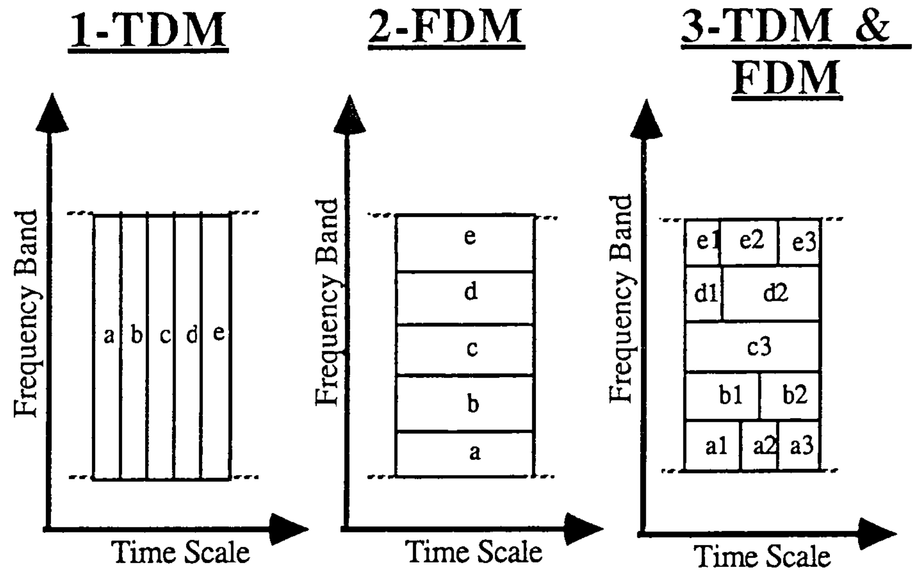


Fig 1.2.2 Network Bandwidth Allocation Strategies.

An overview of the major types of localised networks in existence will now be made, with reference to all of the properties mentioned above.

a) Non-switched Networks - involve the interconnection of all the network nodes via dedicated point-to-point links. The topologies used for this type of network are shown in Fig 1.2.1.a & b. This is the simplest approach to networking and was the first technique used, hence the network's high cost (communications lines and associated interfaces) and poor flexibility/adaptability (extensive re-cabling was necessary on the addition of a new node) was balanced against the high cost and small number of the computing devices in operation at the time. No media allocation techniques were used since the transmission lines could not be shared, resulting in considerable channel capacity wastage. Also, since the lines were dedicated, no flow control measures were necessary as the transmission rate could be arranged on connection. As the cost of the networked devices became cheaper, more numerous and more dynamic in placement and quantity, this arrangement became very unnattractive.

b) Circuit-Switched Networks - involves the setting up of a dedicated link between communicating nodes for the duration of the data transfer. Since these links can be between a number of nodes, this type of system is more flexible than the previously described non-switched networks. Since this type of network is of a basic nature, the "old" networks using this scheme are usually centrally controlled (see Fig 1.2.1.~~d~~c & ~~e~~f). The Star network is the most easily envisaged topology for this method, and is extensively used in telephone networks in which the lack of flow control, speed conversion etc are not important. When used in the multipoint configuration, the central controller employs polling techniques to allocate the bandwidth. The two most common polling strategies are "Roll Call Polling" (in which the controller invites each station to transmit in a pre-determined order) and "Hub Go-Ahead Polling" (where each station that has nothing to transmit passes the poll along to the next station in turn).

Circuit switching techniques ~~is~~ are also used in Loop networks (Fig 1.2.1.h) where the central controller polls the other nodes, but in this case the majority of applications involve sharing a communications line between terminals connected to a central computer, resulting in the destination of the data being inherent. These topologies have been in use for some time but rely on the fact that each node only has a small amount of information to send (or is given an upper limit to the quantity of data that it can transmit at any one time), to prevent any single node from "hogging" the line. These circuit switched networks result in two forms of TDMA (Fig 1.2.2.a) being used to allocate the network bandwidth. In earlier systems, a fixed amount of time was allocated to each node in which to transmit its data, resulting in a rigid TDMA system which was wasteful of bandwidth in cases when polled nodes were not ready for transmission. Later systems reduced this wasteage by enabling an unready node to inform the controller of this fact so that the next node could be polled, this being made possible by the increased intelligence of the nodes being interconnected.

c) Message/Packet Switched Networks - is the method normally used in WANs (Fig 1.2.1.c) (the switching processor receives data, stores it in memory whilst the intended route is determined, then retransmits it on an appropriate output line) but has been used locally with a star topology (the central node acting as a switching processor). Flow control and speed/code conversion can be provided by the central controller when interfacing devices of differing speeds. The star configuration also allows any line to be of a quality dictated by the requirements of the particular node using it. The network bandwidth in this case is allocated by a flexible form of TDMA, where the time slots are not pre-defined over the individual data links.

d) Contention Networks - are a relatively new idea to allocate bandwidth by a flexible form of TDMA without the use of a central network controller. This method requires the reception of data from one node by all the others , (although it may only be intended for some subset of these) and so the topologies most suitable are those which share a common communications line (Fig 1.2.e & g). Although techniques for the use of this method have been proposed and simulated on ring networks [1.14-1.16], all of the practical implementations have been associated with the bus topology. Since the technique differs slightly for the two topologies, the more established bus technique is described. This technique was originally developed for a radio network (ALOHA) but has since been developed for use in LANs.

The most successful of these developments is a technique named "Carrier Sense Multiple Access with Collision Detect" (CSMA/CD) which is illustrated in Fig 1.2.3. "Carrier Sense" is illustrated in the first two drawings which show that the node does not transmit when it senses the line in use (a) and broadcasts its packet only after detecting that the line was idle (b). Since no controller exists on the network, another node could also have begun to transmit during the time taken for Node X's signal to propagate along the line to be sensed by this other node (The time taken for it to be sensed by all other nodes is named the "Collision Window", the worst case being given by the time for propagation along the entire length of the bus when the nodes are at opposite ends of the line) and a data collision occurs (c). This collision is detected and then enforced by the broadcasting of a jamming signal which is of sufficient duration to make sure that all the parties concerned have detected the collision's occurrence (d). Node X then waits for a randomly selected time period before re-transmission (e). This process is repeated if a further collision occurs but the deferring of re-transmission is increased in scale according to some pre-defined algorithm (thus the nodes adapt themselves to the loading of the medium), otherwise the packet is transmitted and the process complete.

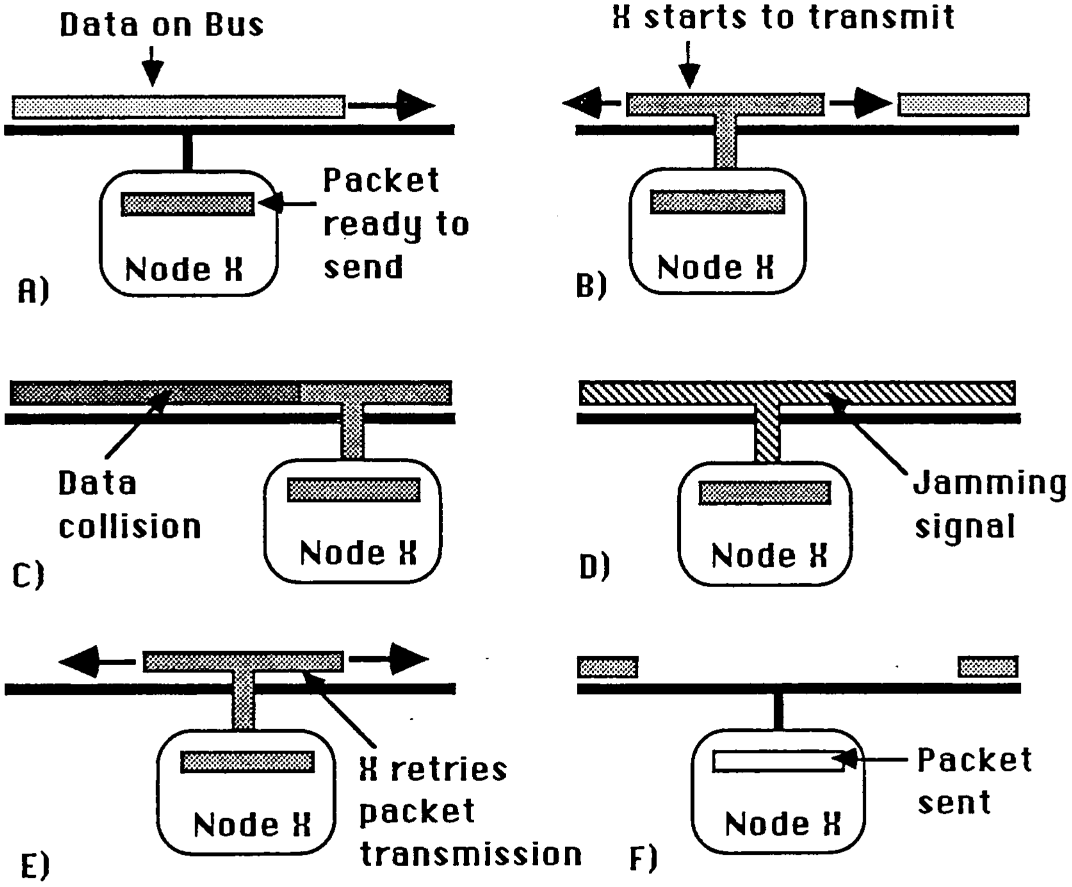


Fig 1.2.3 Outline of CSMA/CD technique.

Reception of data is less complicated, since all nodes listen to the network for data packets containing their particular network address in the destination field of the data packets, selecting those that do for storage within their hardware. The most famous network which uses this technique is "Ethernet", originally developed by Xerox.

Another, less famous type of contention technique uses collision avoidance measures in addition to and/or instead of collision detect, CSMA-Collision Avoidance (CSMA/CA). It is effectively a mixture of normal rigid TDMA techniques and CSMA/CD. After the line has been contended for in the usual way and the "winner's" packet transmitted, the network system changes from CSMA/CD to a fixed time slot allocation method. The "slotting" mechanism is effected by allocating different delay periods to each node which have to elapse after the completion of any other node's transmission before it can transmit itself. This results in a priority scheme being set up (highest priority-smallest delay). After the longest delay has elapsed since the last packet was transmited with no further transmission having taken place, the system effectively reverts back to CSMA/CD. This allocation technique is used in "Hyperchannel" developed by Network Systems Corporation.

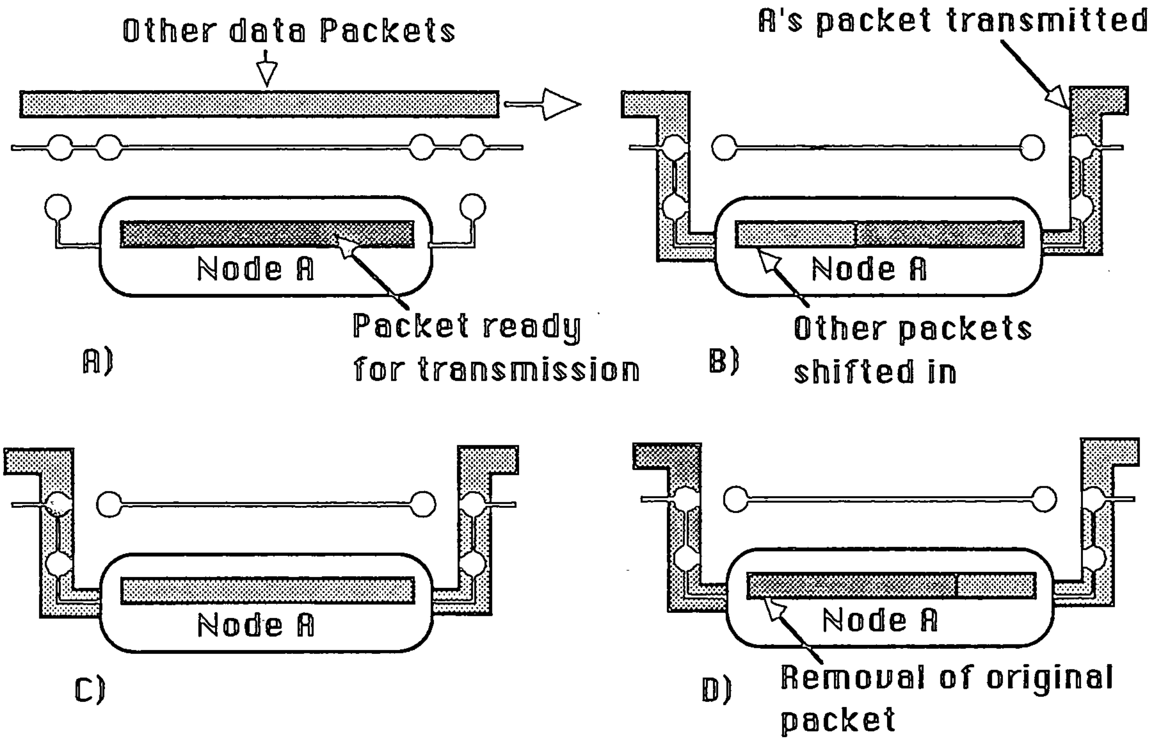


Fig 1.2.4 Idealised Register Insertion technique.

e) Register Insertion Networks - require data to pass through each active node on the network and that the data always returns to its source before removal from the network. The ring topology is obviously the most suitable in this case, in fact no examples of such systems using any other topologies have been found in the networking literature. The technique is illustrated by Fig.1.2.4 in idealised conceptual form and shows it to be another form of flexible TDMA.

Drawing (a) shows how Node A defers transmission whilst data is in transit on the line. When the line goes idle, the node switches the data stream so that it can shift its data packet onto the line. During this time, any data received by the node is stored within it for transmission after its own packet (b). The node remains in series with the network whilst it awaits the return of its packet, thus increasing the capacity of the network to cope with the increase in data traffic (c). When the original packet has been completely shifted into Node A (d), the data stream is redirected to by-pass the node again (as in (a)) and the packet is therefore removed from the network.

Reception of data is accomplished by shifting in the packet sent to it, copying it, then re-transmitting it to the sender. Practical implementations of this system differ in the number of registers used and whether or not the packet is returned to the sender. In the Hasler SILK implementation, three registers are used, one each for transmission and reception and one for placing in series with the line on transmission. Packets are not returned to the sender in this case.

f) Circulating Empty Slotted Networks - have their bandwidth allocated by the sharing of a time slot between the users, in effect producing a flexible TDMA system. The most famous application of this technique is the Cambridge Ring, whose operation is outlined in Fig 1.2.5.

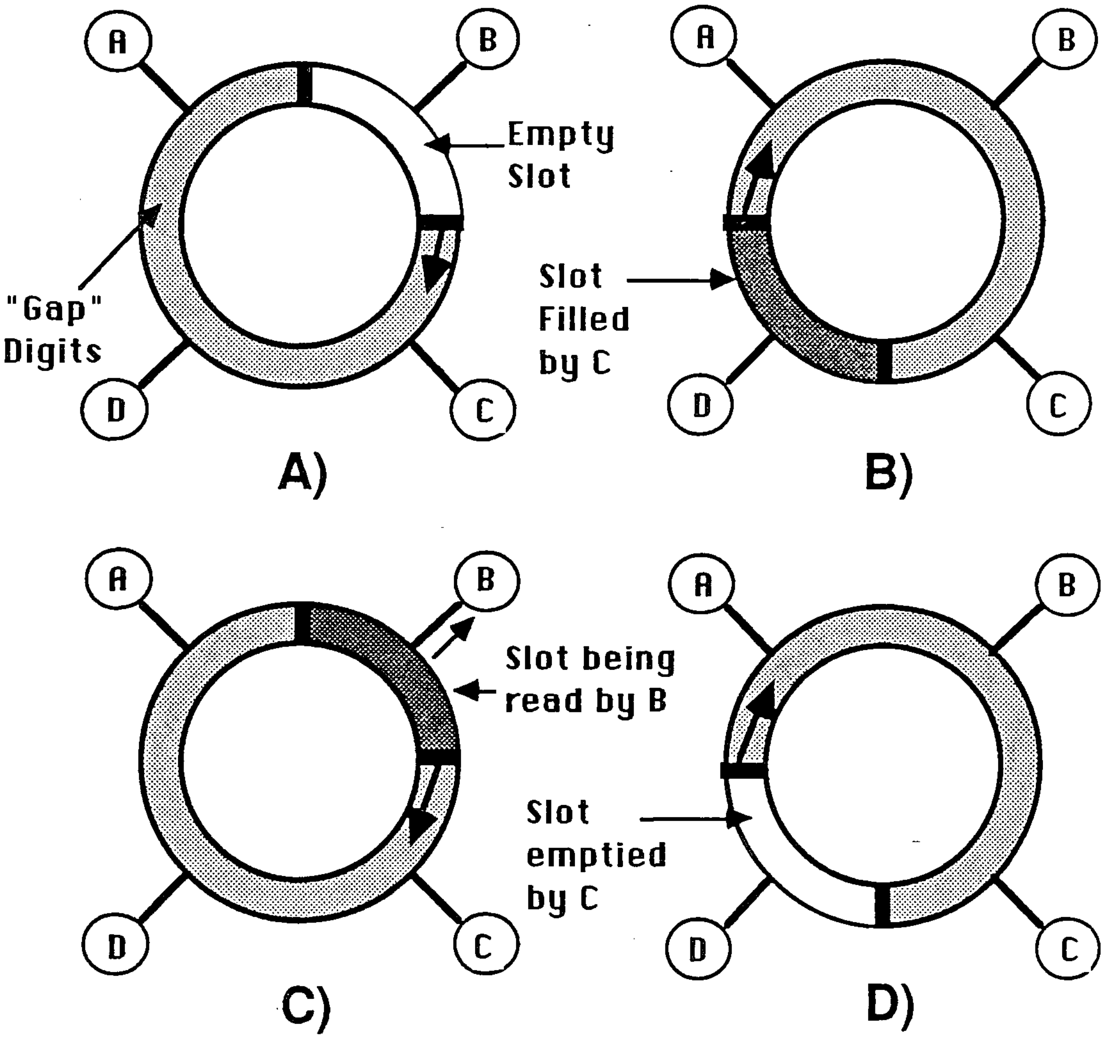


Fig 1.2.5 Circulating Empty Slotted Ring Network.

Drawing (a) shows the most important features of a simple one-slot system. The indicated slot is bounded by two dark areas, its header and trailer, the former containing a bit for indicating whether the slot is full or empty. It is easily seen that the slots size is dictated by the latency of the ring, since the slot is always present, the latency being predominantly due to delays encountered on passing through the network nodes. The rest of the ring is filled with "Gap" bits. Consider the case of Node C transmitting a "slot-worth" of data (or mini-packet) to node B. Node C inspects the busy/idle bit of each passing packet until an empty slot is encountered, when the busy bit is set and the slot filled with data (b). This full packet then continues around the ring until Node B is reached, when its destination address is matched to B's own and the data copied into it (c). The appropriate response bits are set in the slot's trailer indicating the acceptance/rejection of the data by B. This slot then continues on to C when it is reset to idle and the response bits are then checked (hence, even if the packet was rejected, immediate retransmission cannot occur), thus making the slot available to the next node in the ring that requires it (d).

This type of ring needs some degree of centralised control for the formation and maintenance of the slot(s) in circulation around the ring. For example, the busy/idle bit could possibly be corrupted to read busy when the slot is empty and so would circulate forever in this state if a monitor did not enforce that the busy slot circulates no more than twice without modification.

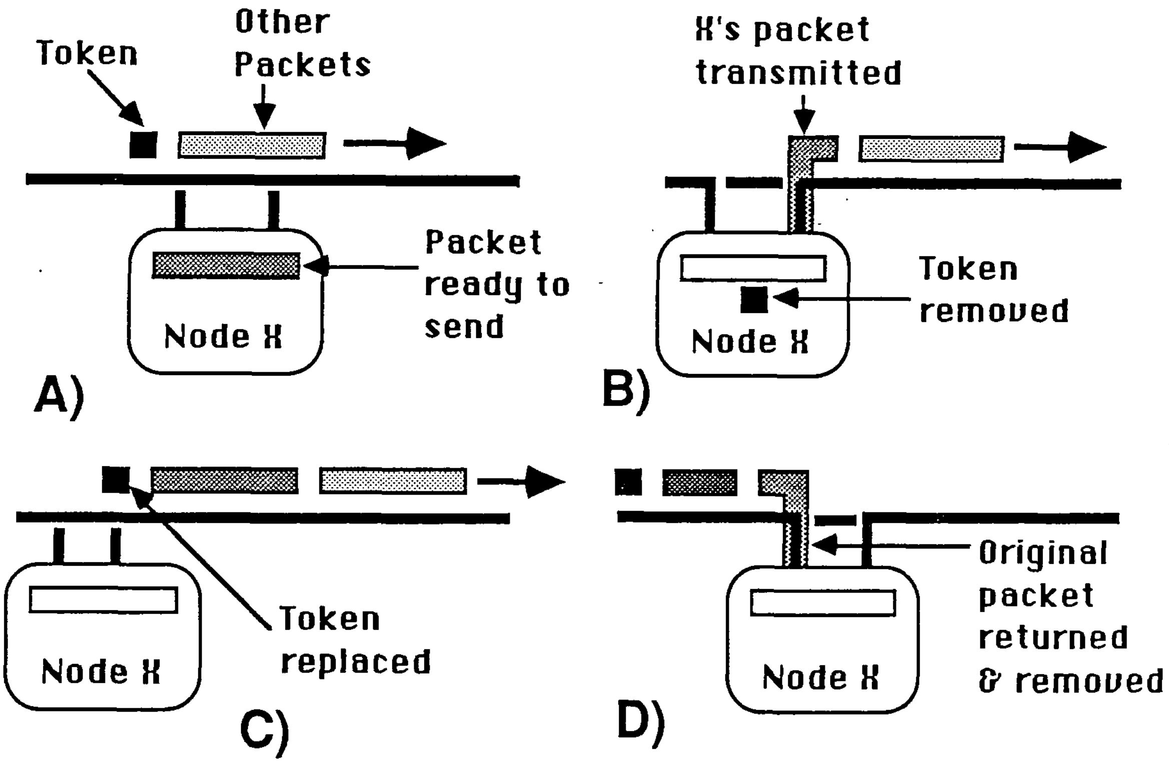


Fig 1.2.6 A Token Passing Bandwidth Allocation Scheme.

g) Token-Passing Networks - use special bit patterns transmitted over the data communications line (tokens) to enable a flexible TDMA scheme to be implemented. All of these networks require a node to obtain a token before transmission of their data onto the line. Two important differing schemes are outlined in diagrams Fig 1.2.6 and Fig 1.2.7.

Fig 1.2.6 shows a scheme which is used in ring networks (eg DOMAIN from Apollo Computer Inc). The first drawing (a) shows that Node X defers transmission whilst it detects other data on the line. On reception of a token on the line, the node removes it from the data stream and inserts its own data packet in its place (b). The token is re-transmitted directly after Node X's packet has been sent to its destination (c). The node then awaits the return of its packet, which should be the next packet of data received, as all of the other data would have been removed from in front of it (if its packet is not in this correct position, a central monitor is used to remove the offending preceding packet). Node X then removes its packet from the data stream and the process is complete. Reception of data is therefore seen to involve the copying of any packets addressed to it before retransmitting it to the original source node.

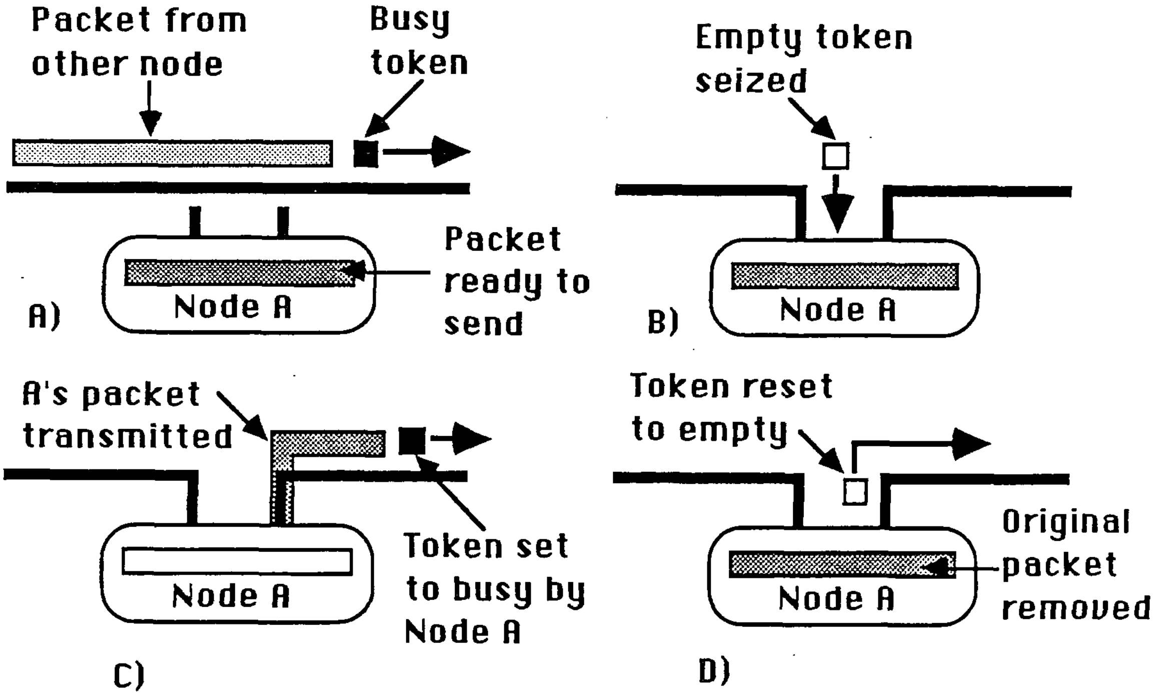


Fig 1.2.7 An Alternative Token-Passing Scheme.

Fig 1.2.7 illustrates another scheme, originally developed at MIT and subsequently at IBM Zurich for their ring topology networks. In this case, the token can be labelled as being either busy or empty, so passing busy tokens are ignored (a). When an empty token is received by a node wishing to transmit a data packet, it removes it from the line (b), marks it as busy, then retransmits it followed by its data packet (c). The node awaits this packet header's return when it removes it from the line. It checks the fields within this header and if the transmission is indicated as having been successful, marks the token as empty and then retransmits it. (If the transmission is noted to have been unsuccessful, the node will retransmit it at a later time dictated by the particular reason for failure, the empty token being retransmitted if the packet is not re-sent immediately). Reception of any data packet thus again involves making a copy of the packet and retransmitting it to the originator. Monitor functions are built into each node in the IBM system and so each is capable of acting as the network monitor if the established one fails, thus increasing the networks reliability.

Token passing techniques have also been specified for use on broadcast bus topology networks (see IEEE802.4, section 3.3.3), where the tokens are circulated around a logical ring of nodes, each node containing their predecessor and successor in the ring. The data packets are broadcast onto the bus by the node holding the token at the time. When use of the token by any node has finished, it is then responsible for the token's transfer to its successor node. Not all of the connected nodes are necessarily members of the logical ring that the token circulates around, enabling "read-only" nodes to be included in the network. This scheme has only recently been specified and so no examples of implementations have as yet been publicised.

g) Broadband Networks - at the present moment all use radio frequency modulation techniques to provide logically distinct networks, functioning on differing FDMA channels (Fig 1.2.2.b & c). Due to the high cost of the frequency agile modems used, ring topologies are too expensive since each node would need enough of these to receive and retranmit on all the channels in use. Use of the broadcast bus topology is the most suitable for this technique, which use either one or two cables for the main bus.

In the two-cable system, one cable is dedicated to carrying transmitted information whilst the other is for information to be received. Each node on the network is connected to each cable via an ~~R~~rf modem. One end of the dual bus is connected to a special transmitter/receiver called the "Headend", which receives all the data on the transmit cable (modulated within one frequency band) and retransmits it on the receive cable (modulated within another frequency band). These transmission and receiving bands are themselves sub-divided into a number of seperate channels so that different users and services (video, voice and data networks) can share the same physical medium. Networks using a single cable for the main bus seperate the transmit and receive channels by allocating them seperate frequency bands, the "Headend" doing the same task as before. An example of the two-cable system is the WANGNET developed by Wang Laboratories, whilst LOCALNET from Sytek uses a single cable.

In order to increase the number of data channels available on the system and to reduce the cost of the modems required (making them fixed frequency rather than frequency agile), many of the data channels are shared using the CSMA/CD technique. In this way, a number of different types of channels can be supported, dedicated multi-point or point-to-point links using specially assigned frequency pairs and general channels whose bandwidth is allocated by CSMA/CD. Also each channel can be assigned a different data rate capacity, giving great flexibility for tailoring networks to individual requirements.

1.3 LAN Performance Characteristics.

1.3.1 Scope of Performance Study.

Of the major network types identified in the previous section, the performance characteristics of Non-Switched Networks (a) and Circuit Switched Networks (where the length of the connections are "long-term" or where a rigid TDMA scheme is imposed) and dedicated broadband channels are easily understood. Since they involve the use of dedicated communications channels of fixed bandwidth, the data throughput characteristics between communicating parties is not dependent on the number of concurrent "conversations" supported. The only variance in performance for the circuit switched networks is that under heavy use, the amount of destinations required that are "engaged' will rise. For the circuit switched networks which impose a shorter upper limit on the duration of the circuit and those which use schemes such as "Hub Go-ahead polling", variance in the number of users wishing to use the line affects the network's throughput and transfer delay characteristics.

This performance change is also encountered in the other multiple access schemes; contention, token-passing and empty slotted networks since access to the line is related to the frequency of occurence of line idleness or empty slots/ idle tokens. It is this group which is the most important in terms of impact on local communications in the near future and so has received the most attention in performance measurement. Therefore these are the networks whose performance is to be studied in this section. Note that the study is also relevant to shared broadband channels, the only difference being the size of the worst case collision window (twice the duration) due to the two cables/channels used between the communicating pair.

1.3.2 Data Packet Comparison.

Before the performance comparison is given, an introduction to the formats of the data packets used will be given in order that the results can be more clearly understood. The formats shown for the CSMA/CD bus and token ring packets are those specified by the IEEE 802 draft standards (see section 3.3), whilst that for the slotted ring is given in the CR82 standard for the Cambridge Ring system.

Fig 1.3.1 shows the packet format for the Cambridge ring. It is of fixed length and only 40 bits long, being so small because of the small number of bits that can be accomodated around the ring when small numbers of nodes are connected to it. Even when a shift register is used in the monitor station to increase the network's storage capacity, it is only in a few cases that more than one such slot can be accomodated within the ring. The small data field (mini-packet) results in the breaking up of data blocks into 16-bit chunks and so their reconstruction at the receiver places an extra overhead on the transmission of a single data block.

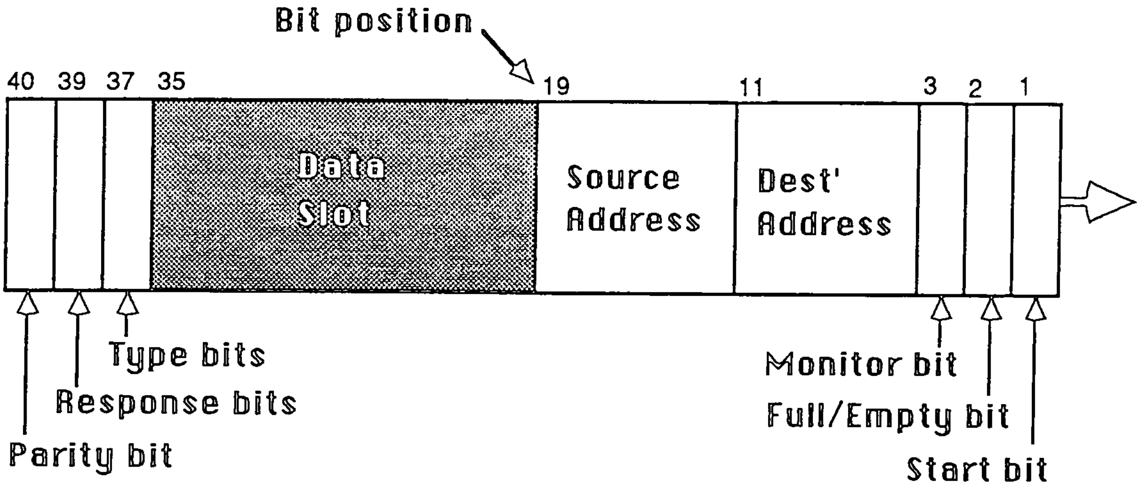


Fig 1.3.1 CR82 Packet (bit 1 rightmost).

The start bit (1) is used so that an unbroken sequence of "0"s cannot occur in normal operation, enabling error detection. The full/empty bit indicates whether the slot is full (1) or empty (0) whilst the monitor bit is used by the monitor station to detect and remove erroneous minipackets. The latter works since each transmitting station clears the bit whilst the monitor station sets it, therefore if the monitor sees the bit as set it knows that it has circulated more than once and so is in error (it is then marked as empty). The parity bit is used for error detection, being set for even parity by each station before being sent downsteam, therefore fault detection can be narrowed down to, at worst, two stations. The response bits are used by the receiving node to implement low-level acknowledgement of the packets, each of the 4 combination of the 2 bits indicating a different reply. The 8-bit address fields are adequate to cope with the local addressing needs, but for inter-networking between rings other measures are required (eg numbering each ring system). The two type bits are undefined at present.

Fig 1.3.2 illustrates the packet structure defined in IEEE 802.5 for token passing ring networks. The packet can be seen to be variable in length, as the data field can be of any integer number of octets up to the pre-determined maximum value. The start and end fields are unique patterns which cannot occur elsewhere in the packet due to the modulation method (ie not possible via Manchester encoding of data), the end field also containing an error bit which can be set by any node detecting one, hence informing the monitor station of its presence. The 48-bit address fields are considerably larger than the number of nodes supportable on the ring in order for inter-networking purposes, so that any node in any such network is given a unique address. The packet status field contains response bits (specifically "address recognised" and "packet copied' bits) so that low-level acknowledgement can be provided. The type field is undefined by the standard, but is designed for use by higher level software. The Frame check sequence field is used for detecting and/or correcting errors in the the data packet.

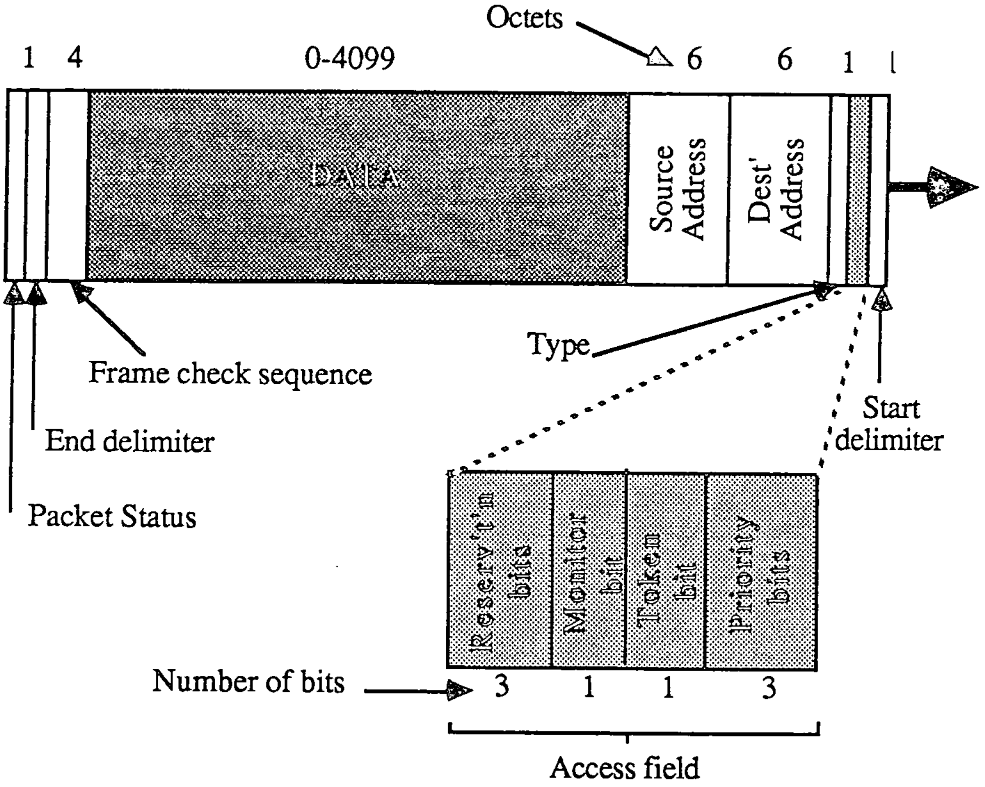


Fig 1.3.2 Token passing (IEEE 802.5) packet format.

The access field contains the token bit (0=empty, 1=busy), although the empty token which is passed around the network is actually the combination of the start, access and end fields (ie a packet with no data-oriented fields). There is also a monitor bit which is used in the same way as for the Slotted ring and two 3-bit fields used for providing 8 priority levels for the data and 8 reservation identifiers (used by IBM for mixing synchronous and asynchronous data). The standard is more complex than the packet used by IBM at present, but differs only in the absence of the status and type fields and its provision of only single bit priority and reservation fields.

Fig 1.3.3 illustrates the packet specified for a 10Mbit/s CSMA/CD network in the IEEE 802.3 standard. The most notable difference between this (variable length packet) and the other illustrated packets is the presence of a preamble field. This is necessary in such a broadcast bus system to allow the low level signalling circuitry time to synchronise onto the packet bit cells and to stabilize the physical transmission medium (ie in ring systems, data is constantly being clocked through the nodes, but in bus systems it can go "quiet", so a new transmission can occur at any time). The preamble is followed by a start frame delimiter and the two 48-bit global address fields (as seen in token ring). For the data field, a minimum as well as maximum length is required, the field being padded up to this minimum value if the actual data is too short. This minimum size requirement is necessary so that the packet is not confused with a collision fragment. The length field indicates to the receiving node the length of the data field following it. The packet is again terminated with a frame check sequence to enable error detection/correction.

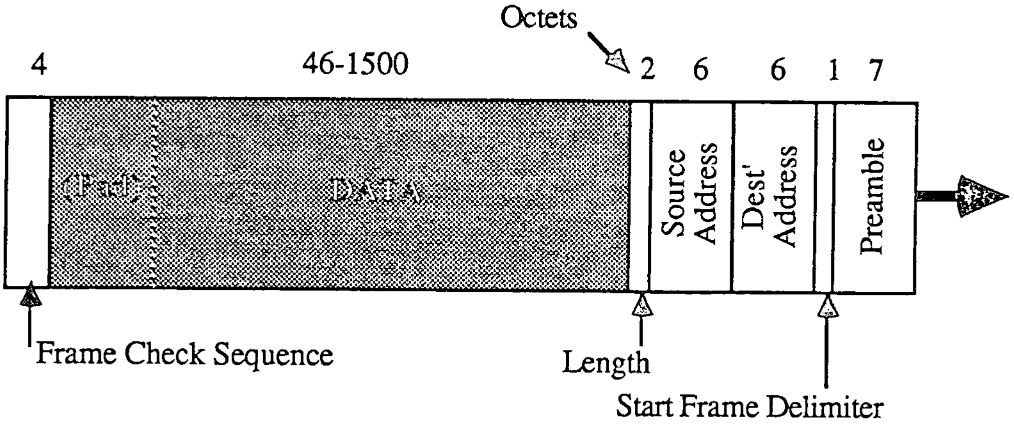


Fig 1.3.3 10Mbit/s CSMA/CD (IEEE 802.3) packet format.

1.3.3 Performance Comparison.

As mentioned above, the networks under consideration here are slotted and token passing rings ("IBM" method) and the CSMA/CD bus. The work from which much of this section's data is taken was undertaken by Bux [1.17-1.18]. Certain assumptions were made to obtain mathematical models relating to the network's properties and the reader is pointed to the references to examine these as they exceed the scope of this section.

Computer simulation was used for the comparison in order to compare the performance for varying conditions such as system bandwidth which would have proved expensive to obtain via hardware alterations. Also, the control of when and how much each node on the network transmits is made possible, which made for more meaningful comparisons. For the example results given in Figs 1.3.4 and 1.3.5 (the graphs enclosed are rough versions of those given in ref 1.17), the axes were defined as follows. For the ordinate, the scale is the mean transfer time of a data packet between nodes relative to the mean packet transmission time (defined in this case to be without headers), to give an indication of the amount of delay that a transmitting node encounters before successful transmission of a packet. For the mantissa, the scale is the ratio of throughput rate (bits/s) to the transmission rate used, giving the portion of the total channel load used for the (suççessful) transmission of data bits.

Consider Fig 1.3.4. The following network parameters were set for the investigation; transmision rate of 1Mbit/s, cable length of 2km (tending to worst case for these networks), number of nodes (stations) attached is 50, exponential distribution of packet lengths with a mean of 1000 bits and a header of 24 bits (low for CSMA/CD system). Finally, a station latency of one bit for the ring stations was assumed. The results show that under these conditions, the bus and token ring perform in a very similar fashion, the explanation being that in these systems the overheads imposed by the access techniques is almost negligible. The considerably worse performance exhibited by the slotted ring is attributed to the following pair of effects.

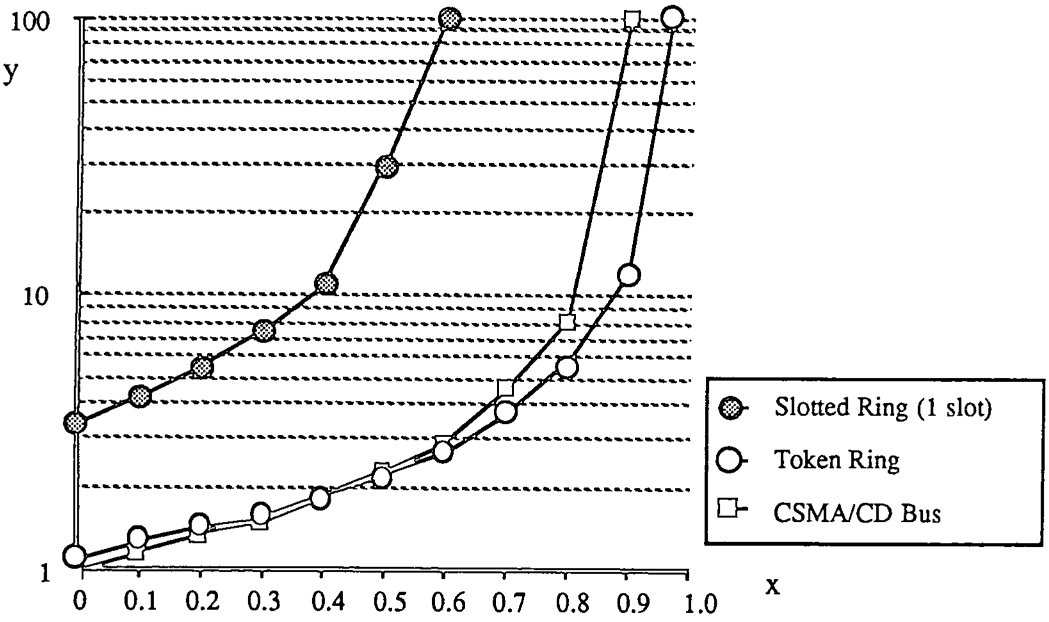
a) In this case, the bit storage capacity of the ring is only 60-bits (cable and node latency) which leads to a poor header/data field ratio, which in this case is the best possible at 24:36.

b) The time required for acquiring access permission is significant. For example, in the case of a single transmitting node the slot is marked as empty after use and passed on to guarantee equal access. This results in the wastage of half of the network bandwidth in this case.

For the results shown in Fig 1.3.5, only one parameter was changed from the previous case, the transmission rate now being 10Mbit/s, but the results are changed considerably. The most notable difference is in the performance of the contention bus network. Its position as the most efficient scheme is lost at loads above 0.2 and it becomes increasingly inefficient until a vertical asymptote is reached at loads of about 0.6. This is explained by noticing that for the higher transmission rate, more bits can be transmitted onto the line in the end-to-end propagation time and so the "collision window" is increased in size with respect to bit number. Therefore, as the throughput rate increases, the increased amount of collisions seen in Fig 1.3.4 will result in raising further the normalised delay value in the latter case. The performance of this scheme has been shown to be slightly improvable by modifying the retry algorithms but the general conclusion remains the same in that the scheme is inefficient if the ratio of the end-to-end propagation delay to the mean packet transmission time is too high (the values of which are investigated in ref 1.17).

The considerable increase in the performance of the slotted ring is due to the increase in the slot size offered (and used here) by the increased transmission speed, resulting in a reduced header/data ratio. In contrast, the performance of the token-bus is only slightly affected by the transmission rate change, its losing out to the contention scheme at low loads being due to the delay encountered whilst waiting for a free token.

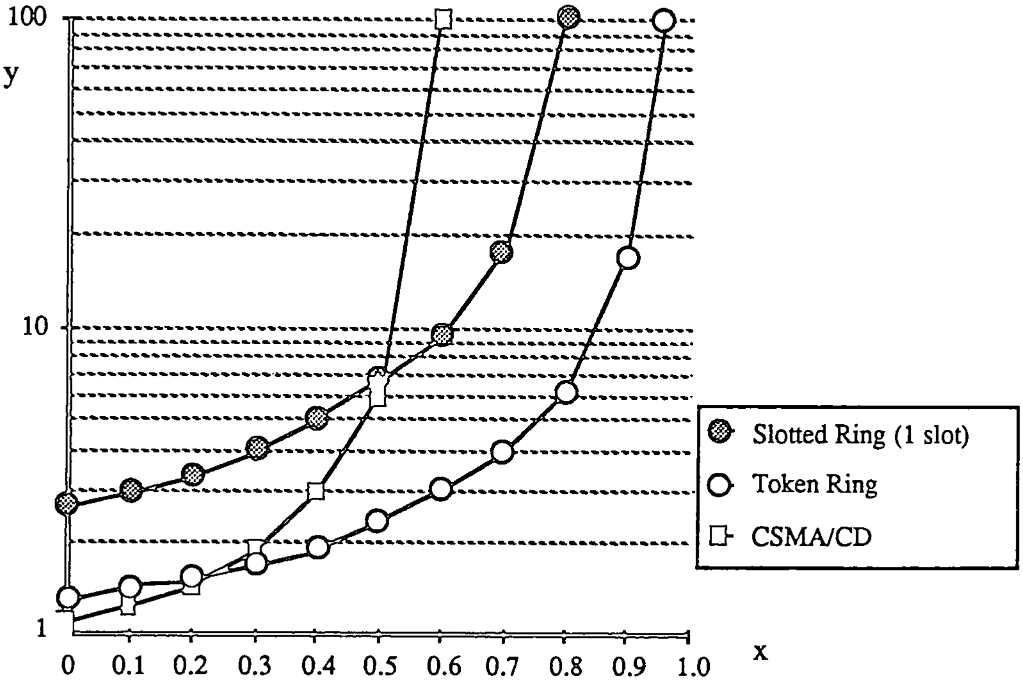
Results from further tests undertaken in this work produced these further conclusions. The greater the variance in the packet size, the greater the transfer delay for the token-ring and the contention-bus. The slotted ring showed no such variance as would be expected from using a rigid slot. However, the relative performances of the network types with respect to each other is unchanged. The Token-ring system was found to be very sensitive to the delay encountered by data when passing through a node, especially if a large of number of such nodes are attached or short packets are used. Therefore the station delay in practical systems should be as low as possible. For slotted rings it was found that the use of one slot was preferable to using two or more since the former solution gives a smaller header/data ratio. However, this assumes that the single slot can be expanded to fill the ring capacity. Further improvements were also seen to exist by increasing the delay in each node, the delivery delay actually falling due to the increased slot capacity which outweighed the accumulative station delay. Tests on the contention bus revealed the conditions under which it performed well and its limitations with respect to the



x=Throughput Rate/Transmission Rate

y= Mean Transfer Time /Mean Packet transmission Time

Fig 1.3.4 Transfer delay versus throughput at 1Mbit/s



x=Throughput Rate/Transmission Rate

y= Mean Transfer Time /Mean Packet transmission Time

Fig 1.3.5 Transfer delay versus throughput at 10Mbit/s

relation of propagation delay and packet transmission time.

This ratio, "a", (data propagation time/packet transmission time, for busses propagation is between the opposite ends of the bus, for rings it is the delay for a complete revolution) has been shown to have a much greater effect on contention busses than token rings (fig 1.3.6 obtained from [1.19] which is a basic study and assumes constant packet size, identical offered load from each node and no overheads.) and can lead to some interesting phenomena. In cases where the load offered to the network is too large to be coped with and all the transmissions are retries, then raising the transmission rate of each node does not help. Since the value of "a" will have been increased by this measure, the system can support still less throughput (due to more collisions) and the situation persists. Worse still, even if no more data is presented to the network by the nodes, this backlog will never be cleared. This instability of contention systems is in contrast to the token and slotted rings which can cope much better with overload conditions [1.19] and also can increase their throughput on raising their transmission rates. The token ring is slightly less efficient for a=1, since the transmitting node is unable to retransmit the token as soon as packet transmission has been completed, (ie for small "a", the packet header will have been returned and processed before completion of the packet's transmission) resulting in increased line idleness. Fig 1.3.6 also shows that although the throughput for contention systems decreases with the number of active users (more collisions), this is increased for the token system (proportionally less time is spent transferring the token).

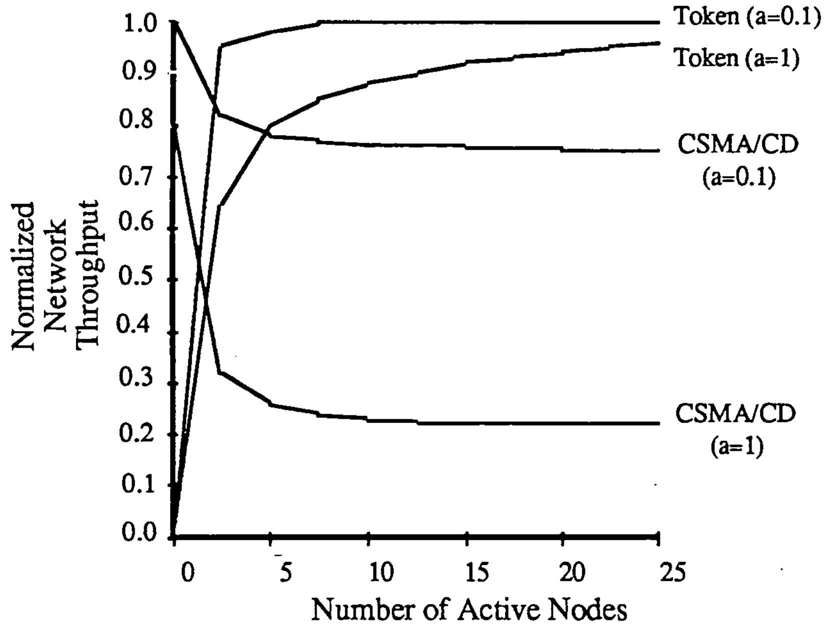


Fig 1.3.6 Throughput versus Active Node number and "a".

Other simulation results presented by Davies and Ghani [1.20] show that for smaller packets (64-bit constant) the slot and register insertion rings do much better as the packet size approaches the slot and register sizes, resulting in less packet fragmentation and hence less overhead. Due to the difference in the assumptions made some differences in the results are present, but the same general trends are apparent and the same conclusions drawn. Therefore the token-bus showed the best all-round performance, with the CSMA/CD bus being as good for systems where the ratio of propagation delay to mean packet transmission delay is sufficiently low. The slotted ring exhibits comparatively high transfer-delay values (given previously) but its advantages are that the packet delivery time is unaffected by the packet length distribution and the expected transfer time for a packet is proportional to its length.

The limitations of such studies have been examined by many authors (eg [1.21]), such as the automatic use of the worst case "Collision Window" in contention systems, which could cause significant changes in the results. Also, simulation studies of contention access schemes (such as [1.22]) have shown how the variance of certain parameters in the algorithms used in various schemes can have significant effects on performance. However, these comparative results are the best available at the present time as far as the author can ascertain. Such studies have led to the development of access protocols which attempt to combine the attractive properties of CSMA/CD at low traffic loads with the properties of Token Passing/TDMA systems under high load [1.21 & 1.23], but these have yet to undergo international standardisation and so have not been considered here.

1.4 Further Important LAN Issues

Apart from data throughput performance, many other issues arise when a choice for a network has to be made. These include issues such as cost (initial purchase and maintenance), upgradeability, reliability, size (geographic span and number of supportable users) and fault detection. The first part of this section will present the advantages and disadvantages for the most important network types and is followed by a more general discussion.

a-Star Network

Rather than the older type of networks radiating from and controlled by a large central processor , the newer breed of star networks which use a PABX (Private Automatic Branch Exchange) as the central switch are considered to be the systems under investigation, as these are more suitable for the interconnection of cheap devices. The primary advantage of these systems is their relative simplicity of installation and ease of maintenance, only requiring readily available skills obtained from phone installation. Since each spoke is independant of the rest, any failure is easily identified and localised to that link only, the addition of a new node also does not affect channels already in existence. Since these networks were primarily designed for interconnecting phones, the integration of voice/data is possible and indeed many could justify their cost for voice alone. Other advantages include the large number of user devices that can be connected and the large maximum distances between device and PABX (up to 3km). Since most of the required networking intelligence is available at the switch, the system is suitable for the interconnection of dumb user devices as well as intelligent ones.

The disadvantages for this topology include its vulnerability to central switch failure (although in practice they have exhibited high reliability) and its low data rate, typically only 64kbps, (due to the processing required at the exchange) over noisy lines, adequate for voice but not for demanding data transfer applications. Also, the expected network expansion has to be planned, since laying new connections may be impractical after the initial installation, a standard telephone system practice.

b-Ring Networks

Seperate consideration for the different kinds of ring based schemes is required, but some properties are associated with all of these. All use active repeaters in the user taps. These give the advantage of increasing the size of the network than would be possible without this signal regeneration and also allow the use of differing media for the different point-to-point links (eg using fiber-optic cable in electrically hostile environments). Another advantage is that each node can process the data passing through it "on-the-fly", enabling sophisticated management and control schemes to be constructed. If used in the pure ring format, the cabling requirements are very much less than for the star topology. The disadvantages for this topology include its vulnerability to single link or repeater failure which disables the entire network. Adding a new repeater to the network is not a trivial task, and may require the re-tuning of the two repeaters whose link has been severed if the length of cable between them and the new repeater is not the same as had previously seperated them (signal strength has to be optimized for the particular link). Also, care has to be taken to ensure that the circumference of the ring gives an integral number of bit times for its latency at the transmission rate used, thus any change in this frequency may require re-cabling. In order to ease the difficulty in adding new nodes, some systems now use a star-ring topology centered on one or more wiring cabinets, but this loses the benefit of reduced cabling costs.

Problems particular to the **Register Insertion Ring** are that node reliability is seen to be even more of a problem, a single register fault in one of the shift registers could have a catastrophic effect on the network's performance. Also, since the round trip time for packets is variable, worst case parameters always have to be assumed by network management software, thus making for poor responsiveness in error recovery etc. In its favour, however, the scheme offers the highest theoretical throughput, with little line idleness taking place.

For the **Slotted & Token Rings**, their guaranteed and rapid media access are their major advantages, whilst the requirement for a specialised Monitor station reduces reliability as its failure would bring down the entire network. Providing the monitor facilities in every node, so that each could take over in the event of the active monitor failure, requires complex hardware and software, thus increasing connection cost and possibly reducing network performance since each node's duties are increased.

Bus Networks

The general advantage of this topology is its small consumption of cable but its disadvantage is that any break in the line will disable the entire bus due to the reflection of data from these new ends (not providing the correct termination impedance for signal absorption). Other considerations arise from the type of signalling used.

In **Baseband** systems, decentralised control and use of a passive medium increases its reliability, no node failure would bring down the network. Also, node addition and removal is possible without affecting other channels in progress if the tap was made previously. However, signal balancing difficulties and media access schemes (CSMA/CD) require that the networks remain relatively small (up to about 2.5km). The use of active repeaters on the line could solve the former problem, but can affect the media acces schemes used as well as reducing reliability. Also, since the bus is shared by all the users, fault location is made more difficult since the whole of the cable may have to be inspected. Due to the presence of some DC components of the signal passing along the medium, care must be taken to ensure that it is grounded in only one location, whilst avoiding potential shock hazards and antenna effects. In **Broadband** systems, the geographic coverage can be increased (by an order of magnitude) as can the data rate. However, use of active taps and a headend reduce reli^ability and the cost of installation and repair is increased by the need for skilled technicians. Maintenance is required more often than with baseband systems with periodic testing and alignment of all network parameters.

A major issue concerning a prospective network buyer is the cost of the system. The increased maintenance costs for broadband networks and installation/alteration of ring networks were mentioned above, but consideration will now be directed to the initial cost alone. A study by the IEEE made into the cost of Local Networks [reviewed in 1.13] gave the following results. For networks serving around 50 people, the star (PABX) network was shown to be the cheapest, followed by baseband networks (bus and ring) with broadband systems being the most expensive. By the time this number of users is doubled, broadband becomes cheaper than baseband and these relative positions are maintained up to the maximum value investigated, around 5000 users. Baseband was most economical, on a cost per port basis, when supporting around 400 users, star networks around 700 users whilst broadband was best for the range 100 to 400. Extrapolation of the data at the upper end suggests that the base and broad-band systems could become even more economical for higher numbers of connected users. The costs included the wiring costs involved in the installation of the networks. Whilst the prices involved in the survey were in the band between $800 and $1300 per connection, the cost of high speed LANs (50MHz and upwards) such as Hyperchannel (mentioned in the previous section) were not included. Since they cannot at present support more than a very few users, their inclusion in the study would not be relevant, but their cost is known to be upwards of $45,000 per port.

Thus it can be seen that the many types of LAN that exist each best fulfill differing requirements. Since the marketplace for these networks is so wide, their does not seem to be any reason to presume the dominance of any particular type and this has been reflected in the adoption of many schemes for international standardisation (see section 3.3).

Chapter 1 References

[1.1] Freeman HA and Thurber KJ - "Updated Bibliography on Local Computer Networks" - *Computer Architecture News* 8th April 1980. (p422)

[1.2] Metcalfe RM and Boggs DR - "Ethernet: Distributed Packet Switching for Local Computer Networks" - *Communications of the ACM,* 19 1976. (p394)

[1.3] Baran P, Boehm S and Smith P - "Distributed Communications" - Memo RM-3097-PR, The Rand Corporation, Santa Monica, California, 1964.

[1.4] Edhart JL - "Understanding Local Area Networks", Seybold Report on Word Processing, June 1981.

[1.5] Wecker S - "DNA: The Digital Network Architecture" - *IEEE Transactions on Communications*, April 1980. (p510)

[1.6] Clark DD, Pogran KT and Reed DP - "An Introduction to Local Area Networks" - *Proceedings of the IEEE*, November 1978. (p1497)

[1.7] Joyce CC - "Communications Network transfer information inside and out" - *Word processing and Information Systems*, June 1981. (p 504)

[1.8] Weitzman C - "Distributed Micro/Minicomputer Systems: Structure Implementation and Application" - Prentice Hall 1980.

[1.9] Cotton IW - "Techniques for Local Area networks" - *Computer Networks* 4 1980.

[1.10] Tannenbaum AS - "Computer Networks" - Prentice Hall 1981 (Chap' 7)

[1.11] Hopper A - "The Cambridge Ring-a local network" - Advanced Techniques for Microprocessor Systems ed FK Hanna, Peter Peregrinus, 1980.

[1.12] GEE KCE - "Local Area Networks" - NCC 1982. (Chapter 1).

[1.13] Stallings W - "Local Networks-An Introduction" - Macmillan 1984 (Chap 3).

[1.14] Clark DD and Svoboda L - "Design of Distributed systems supporting local autonomy" -

IEEE Compcon, Spring 1980.

[1.15] (see [1.6])

[1.16] Davies P and Ghani FA - Access Protocol for an Optical Fibre Ring Network - *Computer Communications* August 1983. (p185)

[1.17] Bux W - "Local Area Networks: A performance Comparison" - *Proceedings of IBM/IFIP International Workshop*, Zurich - August 1980. (p302)

[1.18] Bux W - "Performance Issues in Local Area Networks" - *Proceedings of Advanced Course on Local Area Networks*, Glasgow 1983. (p 250)

[1.19] as [1.13].

[1.20] as [1.16]

[1.21] Stuck B - "Which Local Net bus acess is most sensitive to traffic congestion?" - *Data Communications* January 1983. (p107)

[1.22] Tagaki H and Kleinrock L - "Throughput Analysis for Persistent CSMA Systems" - *Transactions on Communications* July 1985. (p627)

[1.23] Dahod A - "Local Network Responds to Changing System Needs" - *Computer Design* June 1984. (p61)

Chapter Two

Naval Shipborne Command and Control Systems

2.1 Introduction

This chapter has been included so that the environment in which a LAN operates within a Naval Command and Control system can better be envisaged. In this way, the requirements of such LANs with respect to performance characteristics and topology are introduced. In addition, a comparison between these Naval requirements and civil LANs is made, to show how existing LANs are more oriented towards the needs of the latter group. It is not the intention to give a thorough understanding of such C and C systems in general, but to indicate the requirements that this specialised environment places on the local data communications network .

2.1.1 Introduction.

In a paper concerning a proposed shipborne Naval command and control (C^2), Gasden [2.1] likened them to a large and complex database system, in that information was taken from data providers, processed and organised and then passed on to data consumers. In this case, the "providers" are systems such as radar, sonar, log and compass, with the "consumers" being systems such as weapons and the command team. In a study prepared by Mobbs [2.2], the functions that such systems must perform, as well as their relationship to the "global" Naval environment, are explained. The separation of these functions into blocks enabled their interactions to be expressed graphically, the necessary data transfer between the physically separated systems being accomplished via a localised communication system. This system for transporting data within the confines of a single ship is the subject under investigation within this chapter, and its form has changed in a similar way (and for similar reasons) to civil systems. Firstly, however, a look at the system as a whole will be made in order to illustrate the requirements placed on such LANs.

The first computer systems used for C^2 consisted of large central computers which were directly connected to the sensors and weapons, resulting in a star topology network (reasons as in section 1.1). In this way, the required real-time links between system components throughout the ship were provided. However, a number of problems were soon encountered with this arrangement. Since the hull life of such vessels is around 20 years or so, the vessel is expected to undergo at least one major refit in order to benefit from significant weapon and sensor changes. During such refits, the system tended to be difficult to modify, however well the software and hardware was designed, since changes to the real-time interfaces required changes in the large real-time control programs. Also, many problems were encountered when renewing or altering the large amounts of cabling associated with such a topology. Therefore, it was found that the degree of system update was largely affected by the complexity, hence cost, of these associated modifications.

Furthermore, from studies of Naval exercises undertaken in the 1970's, it was noted that the data within the computer system was poorly used. This resulted in a decrease of the vessel's "action capability". Therefore, it was seen that an increase in the computing power brought to bear on the tasks of weapon management, data processing and presentation was necessary to obtain the C^2 system's full potential. Merely increasing the power of the central mainframe computer was not considered to be the most suitable approach. This was because such a system would require extensive software and hardware provision for upgrades (eg modular operating system), thus increasing the cost and complexity of the system. Even so, such measures could not guarantee the successful addition of all future equipment, whose form could not be accurately predicted for the vessel's hull life. This was one of the reasons why in subsequent systems, the required processing power was physically distributed around the vessel, a solution now possible due to the decreasing cost of computer hardware. Further attractive properties of such distributed systems to such C^2 applications are given in the folowing section.

2.1.2 Benefits of a Distributed Command and Control System.

Apart from the usual advantages offered to computer users by distributing the computer power (see section 1.1.2), such an arrangement has particular benefits for C^2 systems. These were grouped into seven groups by Gasden [2.1].

1- **Greater processing power and speed.** This is because of the greater ease of system update, so that additional processing power can be added (perhaps by direct replacement of a computer on the network, or the addition of a new one). This ease is especially relevant to Naval C^2 systems, where they are expected to have an operational lifetime of around 20 years, many times more than those used in the civil sector.

2- **Faster response times.** Since the system's data content can now be stored within multiple nodes around the network, the very rapid response times required by the real-time system are more easily acheivable. This is due not only to the fact that the information is physically closer to the associated user, but also because the potential bottlenecks within a centralised database are avoided (since not all of the required data is obtained from the same computer, or alternative sources for the same data are offered, prompter servicing of the request is accomplished).

3- **Flexibility.** Since the fitting of new or additional weapon and sensor systems is one of the major problems during vessel refits, the relative ease of adding functions/systems to a distributed network is a major benefit. In addition, it is also possible to do these tasks without disrupting the rest of the system.

4- **Reconfigurability.** Naval history has shown that many vessels have changed their role several times during their lifetime, in order to satisfy changing needs. Therefore, the more flexible distributed arrangement which allows the system's function to be changed without a complete re-design, is exactly what is needed.

5- **Reduced vulnerability.** The distributed system has an obvious advantage here, compared with the centralised system. The network is potentially less prone to single point failure due to damage in combat situations, or hardware and software "breakdowns", providing that the system design uses such network's relevant characteristics (eg replication of function and fault isolation within a single attached device).

6- **Reduced cabling.** If a suitable data communications system is used within the distributed C^2 system, then the amount of cabling necessary within a vessel can be drastically reduced. The centralised systems presented great problems due to the sheer number of cables used, their complex routing and the resulting impossibility of providing cable redundancy. The use of a LAN within the system reduced the cable count, simplified routing, therefore also allowing cable redundancy.

7- **Reduced data.** Since the centralised systems carried out the vast majority of the processing centrally, the data obtained from the sensors etc was transmitted in a raw, unprocessed form. By allowing processing functions to be embedded within the sensors themselves, the data can be drastically reduced in quantity, thus reducing the traffic over the communications media. This in turn means that more such information transfers can be supported over a given communications system.

2.1.3 Military requirements for a distributed data processing system.

In a report on the proposed use of a distributed data processing (DDP) system within US Naval Ships, Willis [2.3] compared the design and usage characteristics of existing commercial and Naval shipborne DDP systems. Table 2.1 summarises the differences noticed. For the Naval case, the users of the system are the shipboard personnel, not the Naval laboratory support personnel, which is why so little programming is done by the Naval system user. From the table, it can be seen that substantially different types of system could be used to fulfil each of the two individual requirements. This is especially true because of the heavy Naval emphasis on the properties of reliability and real-time capability. For the Naval case, the degree of CPU coupling was expected to become looser, the degree of program migration greater and the number of replicated programs greater. This was due to the anticipated availability of cheaper hardware. Use of these characteristics could then be made to ease the satisfying of real-time computational requirements.

Another major difference between the systems that is shown in the table, is in their configurations. Since most of the communicating systems in the Naval case are highly specialised, their intercommunication paths are fixed (ie radar sensors would never be required to communicate with sonar sensors). Therefore, the flexibilty often required by civil systems, in order that any system user can communicate with any other, is not necessary. This could make a big difference in the communications protocols and access techniques used in the local communications system.

Characteristi c	Commercial	Navy
Programming by user	Yes	No
User alters program	Yes	No
User involvement in application	High	Low
User needs	Variable	Fixed
Growth rate	High	Slow
Computations	Variable	Fixed
Configuration	Variable	Fixed
Reliability and backup	Desirable	Critical
Use in training	Low	High
Use in decision making	Medium	High
Data bases fixed/dynamic	30/70%	90/10%
Modifiability	High	Low
Software maintenance	High	Low
Real-time requirements	None	Critical
Fixed, preprogrammed dialogues	Few	Most
CPU coupling	Both loose and tight	Tight
Program migration	None to few	None
User input/system output ratio	Medium	Low
Replicated programs	Few	Few

Table 2.1 Established Commercial and Naval DDP System Comparison.

The solution adopted by the Royal Navy to provide a DDP system on board its ships is outlined in the next section.

2.2 The Royal Navy's DDP System.

2.2.1 Hardware configuration.

Fig 2.2.1 shows the expected overall topology of the naval C^2 system which is planned for use aboard the RN's new Type 23 Frigates, based on an article on the system by Rowland [2.4]. This is to be based on the experimental system, DIAS (Distributed Information Architecture for Ships) developed at the Admiralty Research Establishment at Portsmouth. It can be seen that the LAN used (the ASWE Serial Highway, see section 2.2.3) has a multidrop topology, using multiple redundant busses and controllers for increased reliability. Seperate highways are used for the Command System and the Combat System, the latter being concerned totally with the real-time communications between sensor and weapons systems. The individual systems used are interfaced to the highway via intelligent interfaces in order to remove the burden of communications processing from their main processors.

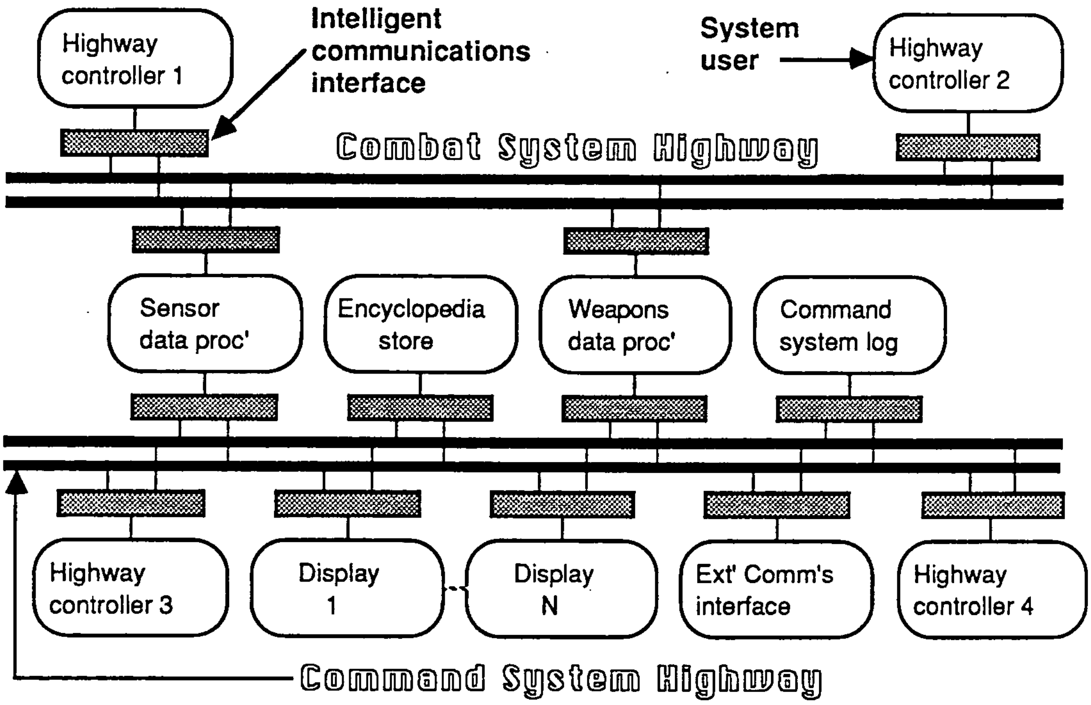


Fig 2.2.1 Distributed Naval C^2 System.

The communications interface is concerned with the translation of data carried over the LAN within a single vessel, to inter-vessel data sent over some other medium (eg radio) when required. Thus, the data from the sensors could be easily shared with that from another, enabling a task force commander to use his collection of vessels as a "distributed warship", with a higher degree of coupling than previously possible.

2.2.2 Software Description.

Fig 2.2.2 shows a functional block diagram of the software used within each of the communicating systems on the command highway (the software for communications over the ~~command~~ combat highway is optimized for these high speed, fixed links). The lowest layer of software is concerned with the MASCOT (Modular Approach to Software Construction, Operation and Test) operating system. A layer of communications software closely linked within this, is used to provide communications between between processes on different computers. This contains all of the high level protocols necessary for both point-to-point and multicast communications to take place.

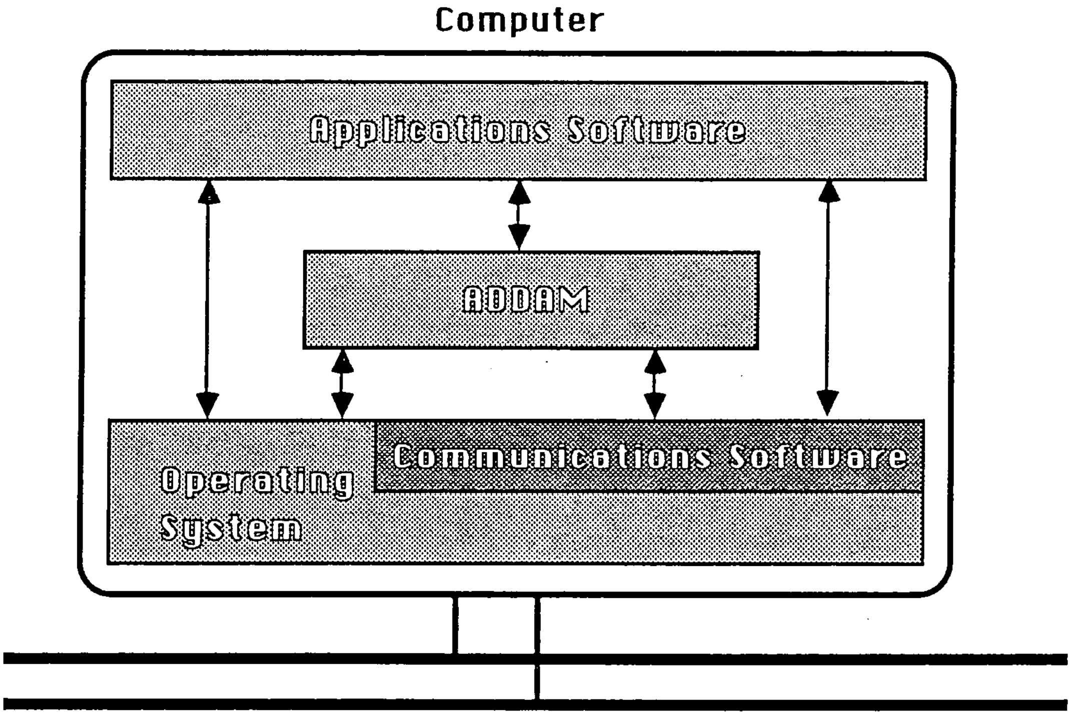


Fig 2.3.2 Software levels within a single network node.

Above these is the ASWE Distributed Database Manager (ADDAM) software, which alone contains the knowledge of the distributed nature of the system. However, the exact mapping of the processes to computers is not stored anywhere within the C^2 system, since it was felt that such fixed routings through the network were too vulnerable [2.1]. The database is organized into "pages", with each page existing as a number of copies held in multiple machines (see fig 2.2.3). One of these copies is designated as being the master copy, whilst the others are designated as slaves. The use of page replication enables the response times for bulk data users to be improved by providing local access to copies of the data required. The arrangement also aids the achievement of high system reliability, since the failure of individual nodes does not cause the loss of the data from the system. The physical location of database pages is dynamically controlled by a database manager, and is governed by the particular application's demand for data, as well as the number of active computers on the network. It is evisaged that almost all of the communications between applications software will be via ADDAM, since most of the software is concerned with the data held within this database.

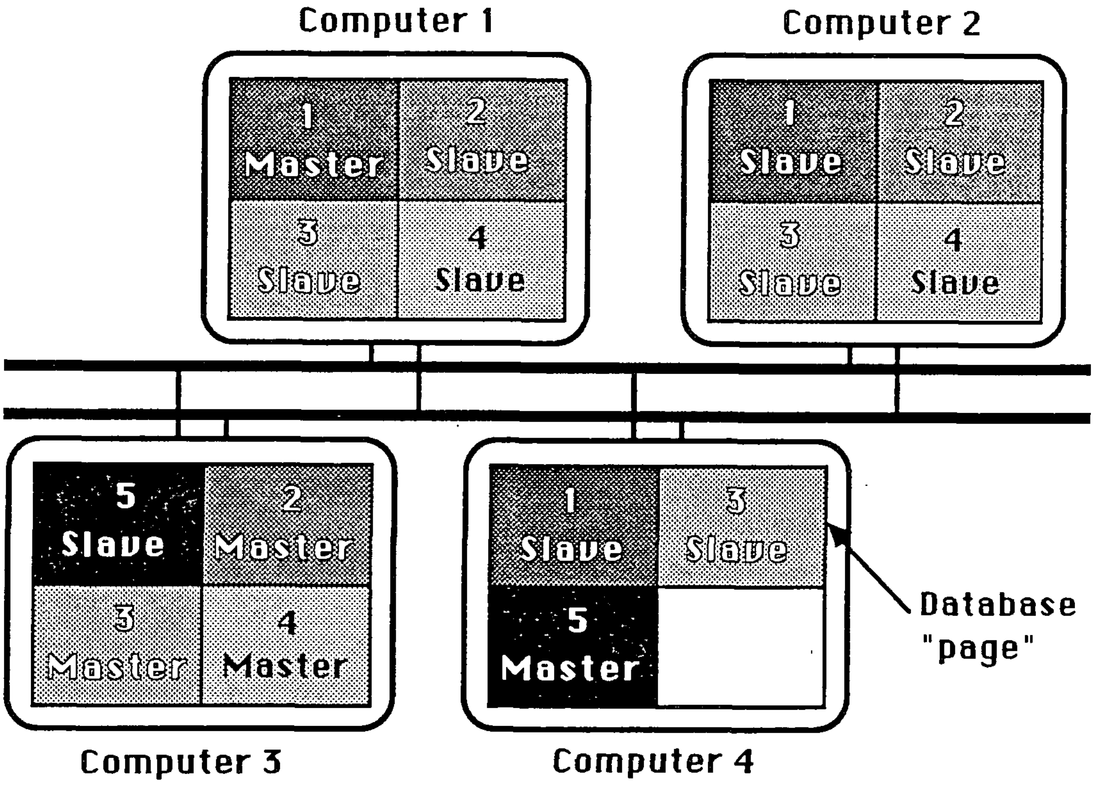


Fig 2.2.3 Page Partitioning and Replication in ADDAM.

The top layer of software is concerned with the particular applications oriented functions of that particular system. It has no notion of the distributed nature of the system as long as it communicates via the database manager. In this case, it sees a localised database and communicates with other processes as though they were also within the same machine.

2.2.3 The ASWE Serial Highway.

The LAN used within DIAS, named the ASWE serial highway (ASH), is a polled multidrop system. This entails the polling of the user systems by the network controller, which subsequently transmits its data (up to a maximum length) to the relevant destination user. Alternatively, it replies to the controller with a "nothing to transmit" signal. The controller then polls the next user in the sequence. Since, at the time of conception, no suitable data highway was available for the system, the ASH was specifically developed to meet the requirement for a robust, highly reliable communications system with the facility for supporting broadcast messages (consideration of the modes of operation showed that such messages would increase performance for many supported systems). The ASH design was then formally designated as an MOD standard, Def Stan 00-19 [2.5]. The polled system was selected for a number of reasons; it was simple, it had good potential for error recovery and at the time little had been published on the present popular schemes (eg CSMA/CD). It also gave a guaranteed upper bound for the delivery time of messages, independant of the load offered to it, which was necessary for the real-time application.

Of the network topologies then available, this was chosen since the active repeaters used in the Cambridge ring were seen to reduce the system's robustness (ie the failure of one node would bring down the entire network). The use of a bus allowed the user computers to be attached to the highway via transformers, giving them physical isolation from it if they failed. The media used was screened twisted pair cable, differetially driven, theoretically making it immune to common mode interference. The bit rate over the cable was 3Mbit/s, but the data rate was only 1.8Mbit/s after accounting for the overheads such as polling and message headers.

The highway controller plays a major part in the communication system's error recovery, by providing a means to recover "lost" messages. On polling one of the system users, that user is allocated a message number which it uses if it transmits a broadcast data message. Since all such messages are received by all of the users, a missing number can be easily detected. Since the controller keeps a copy of the last 128 messages transmitted, it can retransmit the missing one. This removes the need to use time consuming multiple acknowledgements for broadcast messages. Also, since the controller constantly polls each of the networks on the system, it can note any failure to reply, remove it from the polling table and alert the appropriate authority.

2.3 LAN requirements for a Future C^2 System.

2.3.1 The need for a new LAN

Since the ASH was developed before the great upsurge in interest given to LANs took place, it was unable to benefit from the knowledge subsequently obtained about such systems. Therefore it is very much a first-generation LAN, using a relatively inflexible MAC scheme and supporting only a modest network data rate. Recent studies, undertaken by the military in both the UK and USA, suggest that the amount of data traffic that will need to be supported within the next couple of decades will rise by an order of magnitude. This is seen to be primarily due to the increased volume of graphics data passed over the network. Since none of the proposed graphics transmission standards have satisfied the military requirements for their systems, it seems likely that this data will be sent in its raw, unprocessed (hence more lengthy) form. Therefore, the data rates supported by ASH will not be sufficient to support such graphics oriented systems.

Additionally, although the network controllers are replicable (one only being active at any one time), the system is essentially centralised and therefore prone to total failure if a small number of nodes (the controllers) fail. Clearly, a totally decentralised system would be more desirable in the rugged environments encountered by military systems. Another factor deserving consideration, is the delay encountered before access is obtained onto the network. Since this delay is exceptionally critical over the Combat System Highway, directly affecting the response times of the weapons to a sensor stimulus, the relatively large delay encountered before polling in the ASH could be reduced. Combining this with increased bit rates, a reduction in system response times would be obtained, enhancing performance.

2.3.2 Topology and Media Access Scheme of the Future LAN

Many of the determining factors encountered whilst designing the ASH still apply. (Whilst the properties of the different topologies were outlined in section 1.4, the relevant properties with respect to the Naval aplication will be made). The star network is prone to single point failure whilst the active taps used on ring networks are considered to reduce system reliability. Use of mesh-type topologies increases the propagation delay between users (since messages are received/retransmitted and routed by intermediary nodes) and requires a large amount of cabling if alternative message paths are provided. A fully connected topology also requires much cabling, and is expensive and difficult to update during refits.

This leaves the bus topologies. The disadvantages of using a multipoint topology have just been outlined, thus present day broadband networks cannot be used. This leaves the decentralised baseband bus. The use of multiple redundant highways to solve the vulnerability of a single bus is the most suitable arrangement devised so far, so it will be used within the future

LAN.

Of the media access schemes that can used over such a topology, the most important are CSMA/CD and token-passing. As pointed out in section 1.3, CSMA/CD's performance drops off rapidly at high loads (at the required data rate of 10Mbit/s or above), but more seriously, it is unstable. Since it is likely that the time period when the applied load will be greatest is also the time where data is the most time critical (eg the ship is under attack), then this scheme is obviously not suitable. Instability at critical periods would be catastrophic for the Naval application. Also, if during the course of a refit the data rate was raised, this would reduce the throughput characteristics of the system due to raising the 'a' factor. Therefore, major changes in topology would have to be made (ie adding more shorter busses) if such a performance upgrade was required. The token passing scheme could have its data rate increased without such problems (provided that the relevant faster hardware is obtainable) and is stable under high loads, but it does not offer instant access to the media.

Therefore it was decided to use a technique used in radio networks, known as Code Division Multiple Access (CDMA), and modify it for use within the LAN. This technique is described in Chapter 4, but its useful properties will be outlined here. Firstly, it allows instant access to the physical communications medium. It is also stable under high load conditions, indeed, depending on some of the relevant parameters, there need be no performance degradation. The data rate can be raised with no adverse effects on the data throughput and the data is recoverable in noisy environments (useful against jamming attempts in military systems).

2.4 References

[2.1] Gasden J - "ADNET: An Experiment in Computer Networking for the Royal Navy" - *Proceedings of the 3rd ICDCS* October 1982. (p351)

[2.2] Mobbs A - "Next Generation Command System - Logical System Description" - ARE Technical Note - July 1984.

[2.3] Willis P - "Data Buses and Distributed Data Processing in US Navy Ships" - September 1981. (p50)

[2.4] Rowland T - "British Navy still lagging in its military technology" - *Electronics Times* - May 1982. (p16)

[2.5] Def Stan 0019/Issue 1 - "The ASWE Serial Highway" - MOD 19th January 1981.

Chapter Three
Communications Standards for LANs

3.1 Introduction

3.1.1 The need for Standards.

For many years, the majority of the data communications literature has maintained the necessity for comprehensive standards to be formulated for all tasks associated with communications networks. These would benefit all parties involved with such systems in the following ways. The buyer of such equipment would obtain equipment which was more likely to be compatible with other equipment bought in the future and for which interfaces to existing equipment would exist. The suppliers of the equipment to be connected to the networks also benefit, because of the smaller set of interfaces that would need to be manufactured to enable compatibility of their equipment to networks on the market. The manufacturers of network hardware benefit because more equipment is available to be used over it, thus making it more attractive to the prospective buyer. A further reason for standardization is to accelerate the development of specialised VLSI circuits for the network hardware, thus reducing their cost and increasing their possible market penetration.

Whilst pre-emptive standards are obviously relevant in the world of WANs, where national and international agreements are necessary for compatability between user devices of many different sources , the need for the formulation of similar LAN standards has been challenged. Dahod [3.1] expresses the view that any necessary "standardization" would occur when a clear market leader emerges from the marketplace. At this time, when no competetive advantage could be gained by violating this *de-facto* "standard", formal standardization could then proceed with a great degree of confidence of its acceptability. This would prevent the stifling of design innovation that would occur if standards were formulated before this time, thus better serving the long-term interests of the user.

However, a set of pre-emptive LAN standards were formulated by the IEEE, necessarily based on methodologies presented by powerful companies in order that they gain widespread acceptance. Therefore, since these standards now exist and have been accepted by all the relevant major manufacturers, attention must be paid to them. The future of any innovative solutions to networking problems will be of little commercial value without some degree of compatibility with them.

3.1.2 LAN related Standards Activities.

The earliest standard specifically designed for a LAN was first released in 1973 by the US Department of Defense, for networks within the high noise environments inside military aircraft [3.2]. This standard, MIL-STD-1553, (a multidrop topology network with the option of multiple busses for backup purposes with stations attached to them via transformers for physical isolation) has since been applied for civilian use, for example in [3.3], for use in a wide range of hostile environments. Another early LAN standard was also released by the military, this time by the Ministry of Defence in the UK. Under the title DEF-STAN-00-19, it was released in 1981 [3.4] and has also been applied to a product suitable for civilian use. Its topology is similar to "1553", again using multiple redundant busses and the physical isolation of the users.

However, since this time, the major standards activity has been in the commercial world, focussed on various national and international standards bodies. This work has predominantly taken the form of a refinement of standards work previously undertaken by the International Standards Organisation for WANs. This work, begun in 1977 within Technical Commitee TC97, aimed to standardise all communications related tasks encountered in data networks. Since data communications was still in its infancy, rigorous standards were not formulated, but a framework for future standards was constructed. This was released in draft form in 1982 (not becoming a full standard until 1985) and was named the "Open Systems Interconnection- Basic Reference Model" (or OSI model for short). The definition of an Open System is given later, (section 3.2) but basically the model seperated the functions required in such systems into a number of groups and defined the basic means by which these groupings interacted. Future standards work was intended to refine the requirements for each of these groupings, but in such a manner as to give future implementors of the standards as much potential for innovation, in acheiving that group's functional requirements, as possible.

Since the OSI model was conceived before the rise in importance of LANs, the specified methods of interaction between the groups do not contain methods frequently used within LANs. Therefore, a number of addenda to the reference model were requested along with work regrouping certain low-level functions to aid the implementation of the OSI model for LANs. The functional regrouping was undertaken by the IEEE 802 commitee, who released the first batch of standards in 1982. These will be examined in greater detail in section 3.3. Other LAN standards which were released at about this time (for example, CR'82 for the Cambridge Ring LAN in 1982) have since been modified in order to increase their compatibility with OSI and parts of the 802 work. Work which had been progressing since the early 1970's on network standards for industrial and process control applications (Proway) was included in the IEEE work, as such networks were considered to be an important subset of LAN types.

In addition to the IEEE standards work, which covers networks which have data rates of about 40Mbit/s or below, the American National Standards Institute (ANSI) has released a draft standard for LANs operating at above this data rate [3.5,6 & 7]. Seperate standards are necessary for these higher data rate systems because it is the ease of processing , not the efficiency of data transmission that causes the bottleneck for the data flow. The two subcommitees within ANSI's X3T9 commitee presented seperate draft standards for two differing LANs. The first of these was named the "Local Distributed Data Interface" (LDDI) and the second, "Fiber Distributed Data

Interface" (FDDI). LDDI is concerned with the process of modifying and adopting a 70Mbit/s coaxial cable star topology network, originally proposed by DEC. FDDI defines a 100-Mbit/s fiber-optic ring, which uses a token passing access scheme. These standards cover the same functional groupings as those tackled by the IEEE 802 standards.

A number of early proprietary networks were seen to be potential *de-facto* standards. These included the "Xerox Network System", DECnet and IBM's System Network Architecture (SNA). None of these have received sufficiently widespread multivendor support, as yet, to become the international standard required. Indeed, each of these companies actively participated in the ISO's OSI and IEEE 802 standardisation work.

3.2 Open Systems Interconnection

3.2.1 Basic definitions, aims and constituent elements.

The definition of an Open System, as given in the OSI standard [3.8] is, "A system which obeys OSI standards in its communication with other systems". A system is also defined as, " A set of one or more computers, the associated software, peripherals, terminals, human operators, physical processes, information transfer means, etc, that forms an autonomous whole capable of performing information and/or information transfer". A concise definition of Open Systems Interconnection is included in a guide to the reference model by Gee [3.9]. It is there defined as "standardised procedures for the exchange of information among terminal devices, computers, people, networks, processes etc that are 'open' to one another for this purpose by virtue of their mutual use of these procedures".

In such systems, openess refers only to their potential to communicate with others, receive data and interpret it if both parties agree to do so. (ie there is no technical hinderance to such information exchanges). There is no compulsion for any system to receive everything that is sent to it, each system has the capability of imposing its own means of limiting access to itself from other systems and users.

The aim of OSI is to provide the ability for any network user to be able to communicate with any other, ideally without having to have any knowledge of the technical characteristics of the systems involved (ie brands or types of computer). All that has to be known is whether or not the target system observes the OSI conventions. In order to acheive this state of affairs, the reference model set out to provide a common basis for the coordination of all relevant standards development, whilst allowing standards already in existence to be put into perspective. It accomplished this aim by defining a structure for communications within an abstract representation of the real world, as depicted in fig 3.2.1. This required the internal structure of the member abstract systems to be defined, in addition to the structure of the communications between them. It should be emphasized, though, that this internal structural definition was made only to help clarify the external behaviour of real systems (ie OSI standardisation is aimed only at the external behaviour of real systems and not their internal structure).

The OSI reference model concerns itself with the relationships between four basic elements within the communications system (see fig 3.2.1). These elements are the **systems** themselves, the **data transmission medium**, the **application-processes** (carrying out the information processing for particular applications) and **connections** (which join the application-processes and enable their information exchange). These elements are abstractions from the real world, only addressing their aspects relevant to OSI.

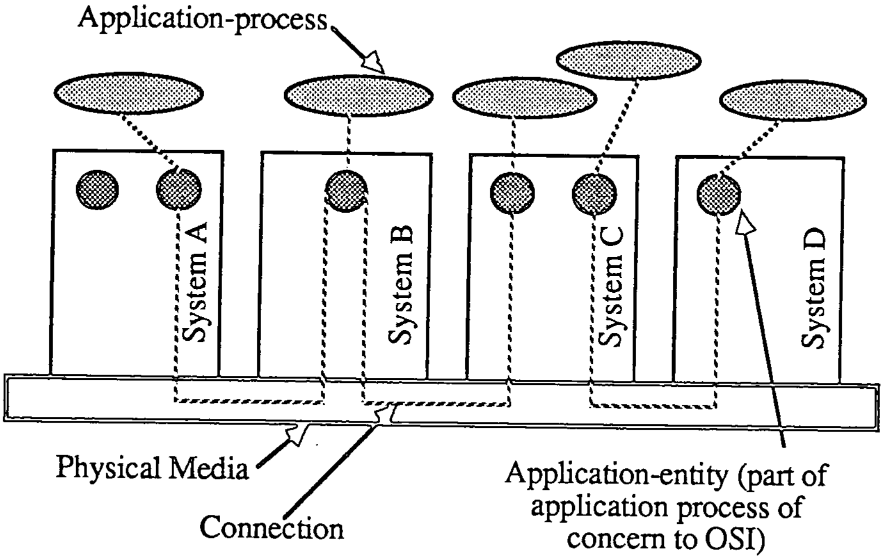


Fig 3.2.1 Basic Elements of the OSI Environment.

3.2.2 The Principle of Layering.

This is the means by which the grouping of functions associated with data communications is accomplished. The technique is a well established means of seperating communications-oriented functions from those associated with processing. Each system within the communications network is decomposed, logically, into **subsystems** which form a hierarchical sequence. This is done in such a way that the services provided by one subsystem are offered directly to the next higher subsystem in the sequence only. In addition, any subsystem only directly uses the services of the next lower subsystem. Each subsystem is said to be composed of a group of **entities** which perform the functions provided by it. Subsystems of the same rank collectively form a layer.

The entities of any given layer cooperate to provide services to the next higher one. In order to do this, they communicate using the services of the next lower layer, except in the case of the lowest layer where the entities communicate directly using the physical media. It was also noted, within the reference model, that further division of these layers may in some cases be necessary. These smallers structures, defined as groupings of functions in a layer that may be bypassed, are named **sublayers**.

The principles used to obtain the particular layered architecture for an open system are included within the reference model document. The principles addressed a number of issues, including not creating so many layers as to make the system engineering task of describing and integrating the layers more difficult than necessary. Boundaries were created at points where the description of services can be small and the number of interactions across the boundary are minimized whilst being at points which past experience has demonstrated to be successful. Also, boundaries were created at points where it seemed that a standardized interface could be defined at some future time (these would not be a requirement for the system to be considered open, but useful in constructing individual systems). Layers were created to handle seperate functions that were manifestly different in the process performed or the technology involved, or where a need existed for a different level of abstraction in the handling of data. They were also created in such a way as to enable their localized functions to be completely redesigned (to take advantage of hardware or software advances) without affecting the other layers.

Application of these principles resulted in a seven layer architecture being decided upon for open systems. The lowest four layers of the model are concerned with communications oriented functions, whilst the upper three layers embrace processing oriented functions. Therefore, the processing oriented functions are relieved from the need to know about the mechanisms involved in transporting the data being processed.

3.2.3 Introduction to the seven layers.

The lowest layer in the reference model, layer 1, is named the **physical layer**. This defines the interface to the physical media itself and also encompasses all of the functions necessary to acheive the transmission of a data bit across the physical media (or link or channel). The bits can be encoded in a variety of ways, depending on the transmission media employed. This layer does not guarantee that the bits it receives and then passes to the next higher layer are all correct. Therefore, layer 2, the **data-link layer**, has to check the data it receives. It does so by handling data in groups of bits, called frames, so that it can use various algorithms on the data to produce a checksum of some sort at the end of the frame. If the checksum is seen to not match the data within the frame, then an error has been detected. This can then be corrected immediately, or the retransmission of the incorrect data can be requested. Layer 3, the **network layer**, includes all of the functions required to route data from the transmitting system to its intended receiver. This can be across a single network, or across multiple interconnected intermediary systems if necessary. This includes a number of routing schemes, whether centralised in any one particular system in the network, in each system and whether the routing is fixed or adaptive (to take account of changes in network topology due, for example, to breaks in the physical media). The **transport layer**, layer 4, deals with all of the issues necessary to supply layer 5 with a full-duplex, bit transparent, error free pipe (or pathway) to the destination system. This may include further error detection correction when the data passed between the communications systems is sent over multiple data links (which contain their own error correction measures), due to such problems as frame duplication etc.

The lowest of the process oriented layers, the **session layer**, sets up, maintains and terminates communications "sessions" between the interacting systems. The most important function that this level contains is thus connected with remote log-on to any computer in the network. This enables the system user on one system to appear as a local user to a second remote system. Another important function contained within this layer is named "bracketing". This enables a receiving host to be able to tell when a particular transaction begins and ends. This is useful in applications such as database updating, where the system containing the database can now wait until the entire change has been successfully received before updating takes place. This avoids situations where only partial updates are made before the communications channel is lost.

Layer 6, the **presentation layer**, formats the data being delivered. Such activities are undertaken to accomplish data compression, data encryption, file translation (from the source system's data format on its machine to that used on the destination system's) and virtual terminal protocols. The latter define a hypothetical terminal to which mappings are then defined to any particular terminal in use. This is necessary in order to avoid problems that occur when some terminals are unable to accept a program because of their use of a differing sequence of control characters, for facilities such as cursor positioning. It is also the layer in which the functions involved with job transfer and remote job manipulation reside. Finally, the uppermost layer, the **application layer**, performs the mechanics of information exchange on behalf of the source and destination application processes. Many different types of application are relevant to OSI, from terminal to computer transaction processing to interconnected real-time process control programs. Network-wide applications also reside within or are integrated with this layer, such as distributed operating systems and distributed databases. This layer is sometimes thought to comprise of three sublayers, the lowest of which contains one or more "Common Application Service protocols" (eg a directory service). The second sublayer contains application specific protocols (eg file transfer), whilst the uppermost contains user-specific protocols which are agreed amongst the user set (perhaps within an industry).

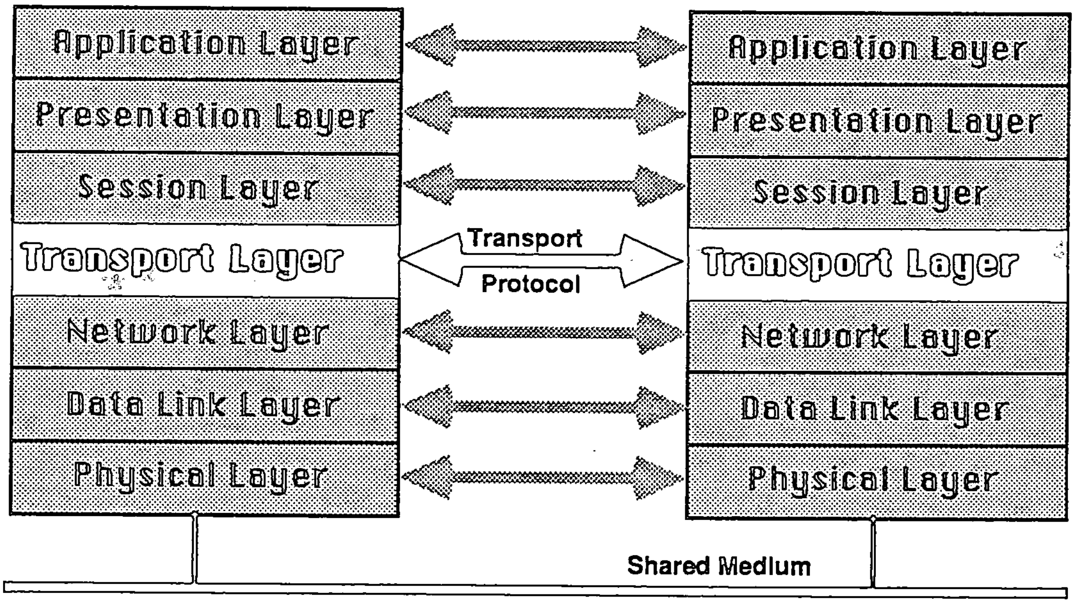


Fig 3.2.2 OSI Layer protocols.

3.2.4 Protocols and services.

It is clear from the previous section, that data has to be passed between peer layers of communicating systems and across layer boundaries within each particular system. The former is accomplished via the use of layer protocols, Fig 3.2.2 showing the case for the tranport layer protocol. A protocol can be defined as a set of rules agreed by two parties who wish to communicate with each other, in order that a mutually meaningful dialogue can take place between them. The reference model does not contain any specifications for these, this is left for specific layer protocol standards. Within these, the protocols are accurately specified so that a system following the specifications will be able to interact in the desired way with another at that layer.

The boundary between any two layers in the reference model identifies a point at which an OSI service standard is required. These are again not elaborated in the reference model, but are the subject of specific layer service standards. Fig 3.2.3 illustrates that a layer's service is defined as the service that the layer can provide to the layer above it, in this case the transport layer service interface is shown. These service standards define the service that can be used by the protocol of the next higher layer as well as the service to be supported by the protocol of the layer whose service interface it is. The functions provided by a particular layer protocol are defined in order to bridge the difference between the service offered to the layer above it and the service provided to it from the layer beneath it.

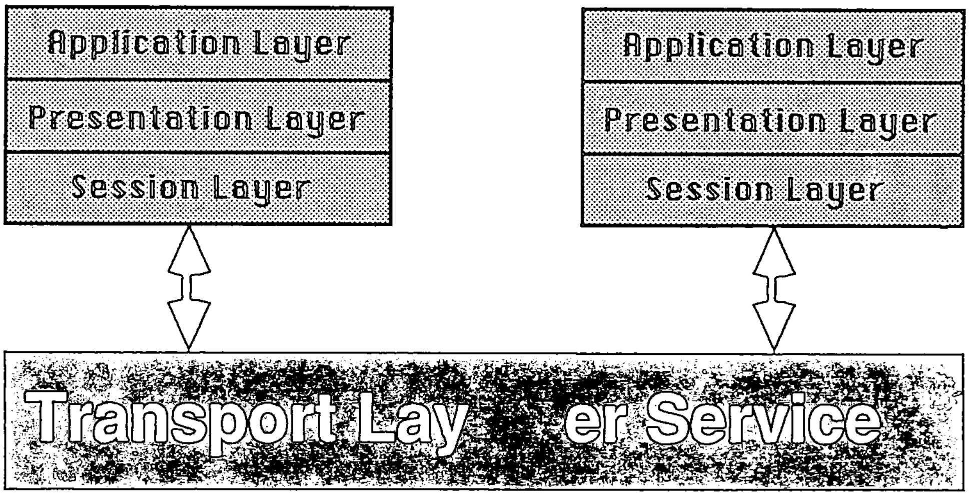


Fig 3.2.3 OSI Layer service.

These service interfaces are specified in a level of abstraction high enough to allow it to be implementable in a number of forms. Examples of such interfaces are;

1- The two sides of the interface are in different hardware systems interconnected by a reliable communications link.

2- The two sides of the interface are in different concurrent processes within the same processor, interconnected via an interprocess communication mechanism.

3- The two sides of the interface are different sets of procedures within one sequential program.

Therefore, since this allows a large amount of flexibility in the way in which an open system is implemented, any effort to lower the level of abstraction for a particular service interface necessarily restricts these implementation options. However, in certain cases, this sacrifice may be acceptable when different layers are supplied by multiple vendors.

Since the seperate systems in the communications network are only connected via the physical medium, then the data flow associated with the layer protocols has to be passed down through the layers before it is modulated within the physical layer subsystem. Figure 3.2.4 illustrates how a segment of user data is enveloped in data originating from all of the subsystems down to layer 2 (the data link layer also appends a trailer for error correction purposes, as outlined previously) before it is transmitted over the line. These headers and trailers are stripped from the data by the corresponding layer elements in the receiving system, so that the user data is received unchanged by the receiving application-process.

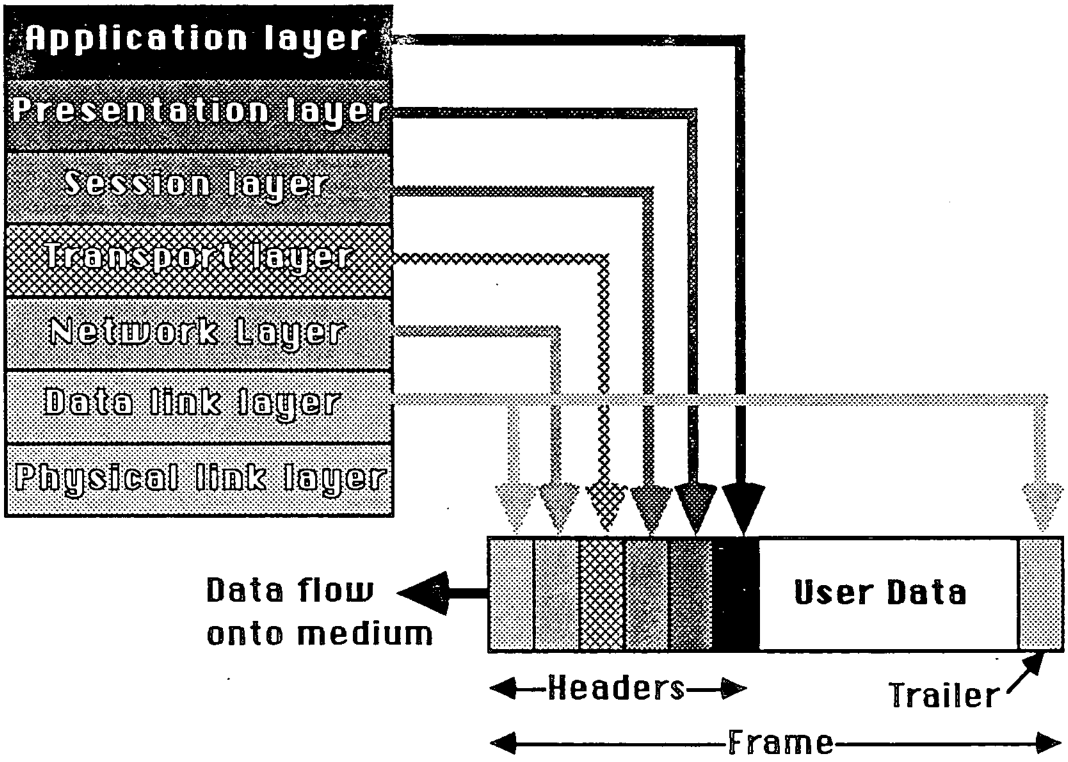


Fig 3.2.4 Headers and Trailer in data stream.

Protocol messages do not always have to be appended to data from the system user, but can originate from any layer which requires data transfer to fulfill its service to the next higher layer.

These messages will have appropriate headers and trailers appended to it as they are passed down the layers (or rather the subsystem portions of the layers) in the same way.

The units of data that are passed between subsystems or added to/subtracted from within them, are defined within the reference model. Fig 3.2.5 illustrates the relationship between these units for any given layer, n. (Note that layer 7 has no layer above it, so the data fed into it differs here in notation) The data unit passed between the two subsystems (from high to low in this case) leaves the upper layer in the form of its own "protocol data unit" (PDU). On reception by the lower layer, this same unit is regarded as a "service data unit". The subsystem itself adds a "protocol control information" data unit to the SDU (ie a header), thus producing its own PDU for downward transfer.

For the case of an upwardly travelling data unit, the names remain the same but the PCIs are removed from the SDUs. The simple case illustrated in the figure, where a single SDU produces a single PDU, is not always suitable. Therefore, the reference model includes the cases where a downward passing SDU can be "segmented" to form multiple PDUs (the receiving system "reassembles" the SDU), "blocked" together with other SDU-PCI pairings ("deblocked" at the receiver) or "concatenated" with other PDUs whilst sharing the same PCI ("seperated" at the receiver).

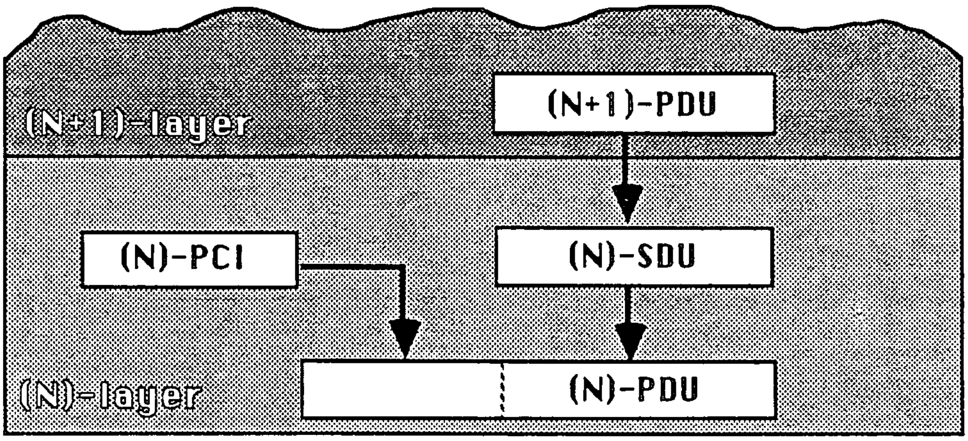


Fig 3.2.5 Relationship between OSI data units.

3.2.5 Service-access-points (SAPs)

The interaction of entities in adjacent subsystems takes place via SAPs which are located on the boundary between them. Therefore, the services requested by an entity from a lower layer, as well as the services supplied by an entity to a higher layer, are offered through SAPs. Fig 3.2.6 illustrates the abstract representation of a SAP, as given in the reference model. Each SAP is identified and located by its "address", enabling them to tie together the reference model and the communications environment it is modelling. They correspond to the addresses of systems on a physical network, to the logical communications ports within a system, to service programs etc (see 3.10 to 12 for Xerox's XNS usage).

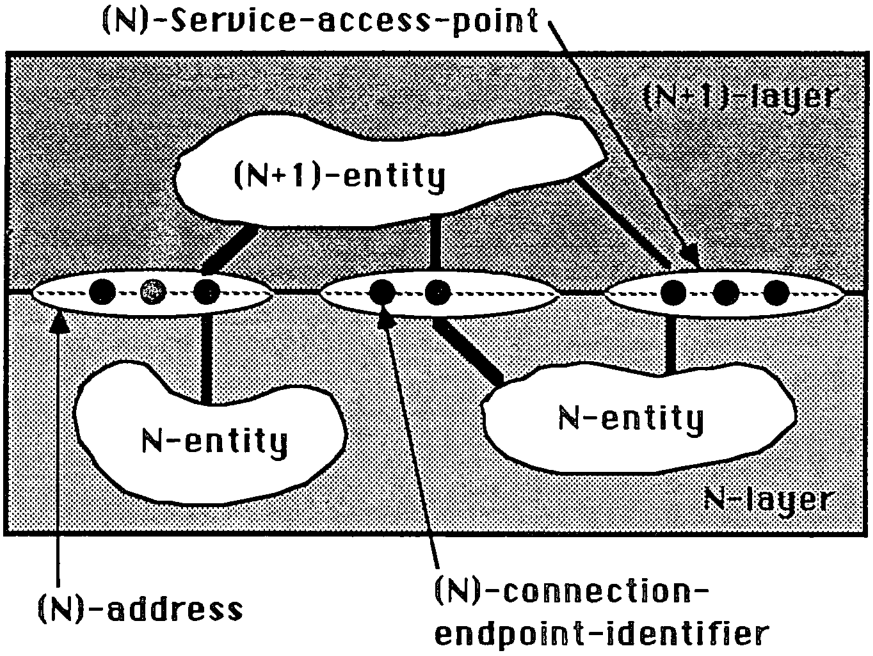


Fig 3.2.6 OSI Service-access-point.

An entity in a given layer is attached at a given time to a specific SAP on the boundary with the next lower layer, enabling it to be reachable via this for communication purposes. This relationship is not fixed, and the entity can be detached from it and re-attached to another (via network/mode management functions), the current relationships being noted by a layer directory function.

3.2.6 Communications structure and Connections.

In order for two application-entities to communicate, they must request the next lower layer to establish a "connection" (an association between two or more entities within one layer which is established and maintained by the next lower layer) between them, via the SAPs which they are each associated with. The entities in the next lower layer (accessed via the SAPs) establish and operate this connection (through the exchange of PDUs, governed by the layer protocol) over a connection which they, in turn, have requested from the next lower layer. Thus, a series of connections is set up, with each associated with a layer being of greater functionality than the one offered to it by the layer below.

In addition to providing basic data transfer on a connection, these "value added" connection services further functions, depending on which of the layers it is associated with. These include error detection and correction (including loss of entire PDUs), sequencing (misordering of PDUs), flow control, connection reset to a known state, multiplexing of connections onto a single connection provided by the next lower layer, as well as splitting of a connection to be supported by multiple connections provided by the lower layer.

3.2.7 Management of an OSI Network.

In order to deal with the special problems posed for initiating, terminating and monitoring activities, as well as assisting in their harmonious operation and handling abnormal conditions, the need for management functions and associated protocols was identified. The activities identified for attention by the reference model are those which imply actual exchanges between systems. Three categories of management activities were defined within the reference model; Application management (which deals with the management of OSI application processes), systems management (which deals with the management of the data processing and communications resources of concern to OSI) and layer management (concerned with particular layer activities and additionally acts as a subset of system management). The reference model makes no assumptions about whether these tasks are performed centrally within the network (ie by one system), or are accomplished in a decentralised manner.

Standardisation work in this area has been undertaken by the IEEE 802 commitee [3.16] (see section 3.3), which has produced a draft standard containing definitions for an abstract architecture, protocol and a set of commands pertaining to the manager and the entities which it manages. A closer look at this subject with respect to LANs will be made in the following section.

3.3 Refining OSI for LANs.

3.3.1 The need for refinements.

The OSI reference model was conceived at the time when computer networks were primarily of the WAN packet-switched type, which involved the use of one or more point-to-point links between communicating nodes and their intermediary relay nodes (if necessary). Therefore, the model assumed that one of the prerequisites for all open systems was the use of connection-based interactions between communicating entities, in order to allow for the sometimes complex problem of routing to be resolved. These connections proceed through three distinct phases; negotiation and setup, data transfer and finally termination. However, in the case of LANs, the most suitable model for the interactions used in existing systems was connection-less, where the transmission of a unit of data is accomplished by a single self-contained operation (see chapter 1). Therefore, the model was modified to include this mode of operation in addition to the established connection-oriented one.

Another major difference between the reference model and the existing LAN implementations, was that no provision was made for the media access techniques used within LANs for bandwidth allocation of the shared physical media. In order to include the emerging LAN techniques into the reference model, a re-definition of the lowest two layers was undertaken by the IEEE 802 commitee. These refinements are dealt with in greater detail in the following two sections.

3.3.2 Connections and Connectionless Data Transmission.

Chapin in [3.14], based on the ISO addendum for connectionless mode transmission [3.15], defines a connection and connectionless data transmission in the following way. An (N)-connection is a dynamic association established between two or more (N+1)-entities to control the transfer of (N)-SDUs between them (via the relevant (N)-SAPs). Connectionless data transmission is the transmission of independent, unrelated data units (sometimes called "datagrams") from a source SAP to one or more destination SAPs in the absence of a connection.

The characteristics of the two transmission schemes differ in many ways. Whilst a connection requires the agreement of three seperate parties (the two (N+1)-entities and the (N)-service) for its formation and for the acceptance of each susequent data unit sent over it (ie mutual willingness for the transfer is assured). The (N)-connection's formation involves the negotiation (and if necessary, renegotiation) of the parameters and options that govern the data transfer before any data is exchanged. In the case of connectionless operation, however, only two-party agreement takes place, (between the sending (N+1)-entity and the (N)-service) before the data is transmitted. The scheme relies on the assumption that the receiving entity is present on the network (information obtained from Network Management possibly) and able to receive the message. Higher level protocols are used to decide whether or not such data transfers are successful, or indeed accepted/rejected by the destination system. Therefore, as no negotiation takes place between the communicating (N)-entities before data transfer, the necessary parameters and options are pre-determined. However, negotiation can take place between the (N+1)-entities using appropriate (N+1)-protocols.

The data units transmitted using a connectionless service are entirely independent from each other, resulting in a number of characteristic differences of the data channel compared with those of a connection-oriented service. Data units transmitted over a connection are related to one another by virtue of their association with that particular connection. This enables problems such as detecting missing, duplicated or out-of-sequence data units to be tackled more easily, by methods such as numerical labelling of the individual units. It also allows the flow control of the data passing over the connection, so that both of the communicating entities are always capable of processing all of the data. For connectionless data transmission, such an association of data units is only possible, even when using a connectionless (N)-service including sequencing, if all of the data is passed through the same (N)-SAP. This will not always be the case.

However, the independence of these data units can improve the robustness of communications networks operating in a hostile or non-understood environment, since all of the necessary information is encapsulated in the data unit and not distributed within the service provider (ie along the connection in a connection-oriented system). Since all of this addressing data etc is present within each data unit, the overhead incurred during data transmission is greater than that associated with the data transfer phase of a connection, once established. Therefore, whilst the delay in setting up a connection before data transmission occurs is far greater than that associated with a connectionless service, it becomes more efficient for the exchange of a large amount of data over increasingly long periods of time. The almost universal use of connectionless data transmission in LANS is due mainly to their considerably simplified protocols, resulting in cheaper hardware and software. Also, the data unit's independence means that the removal/addition or failure of units within the system does not affect the ability of the data to reach its destination (the rate of change of the user systems on the network is much greater for LANs than for the more stable WANs). The services of a full connection are usually not required because of the relatively reliable physical transmission systems employed in LANs, as compared to those provided for WANs.

The addenda to the OSI reference model recognized the fact that OSI systems could contain layers operating in both the connection and connectionless modes. The layers in which the conversion between these modes was deemed acceptable were defined to be the data-link and network layers. In addition, such conversion was made permissible within the transport layer "only if limited additional functions are required". Fig 3.3.1 illustrates the possible layer combinations. The physical layer is not defined to be of either type, as the service's determining characteristics are too diverse to allow categorization into just the two modes of operation. The restriction of the number of layers in which the conversion was allowed was made in order to limit protocol complexity and to ensure the possibility of general intercommunication between systems.

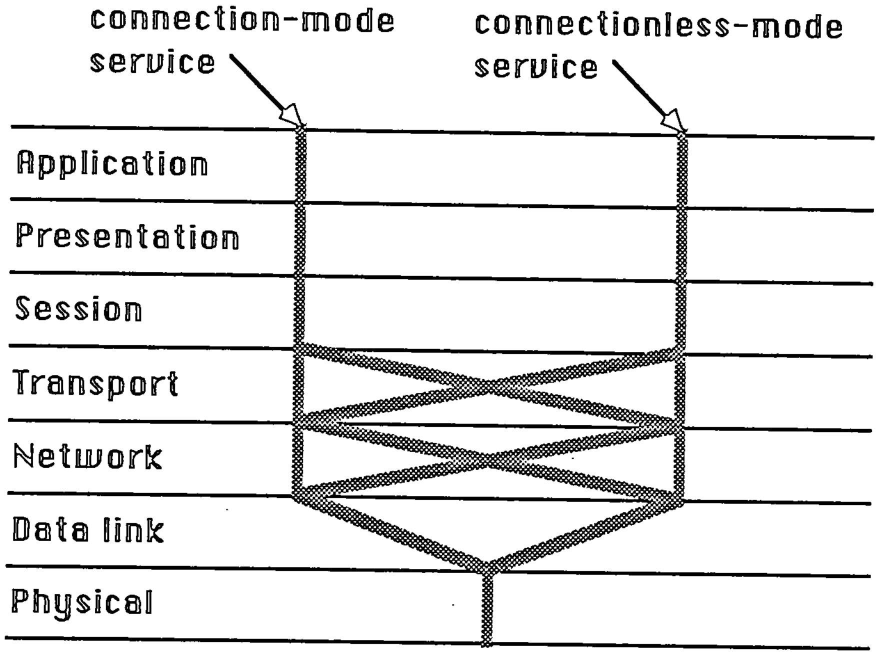


Fig 3.3.1 Layer Service Combinations

3.3.3 IEEE 802 Committee Standards

As already mentioned, the functions associated with the lower layers of the OSI model were not formulated with the techniques used in LANs in mind. Therefore, work was undertaken to re-define the lowest two layers of the model, where the majority of the differences between LANs and WANs exist. Although the original intention was to define a single standard for all LANs, it became apparent by the end of 1980 (the project was started in February of the same year) that this was not possible. Due to the large number of LAN application areas envisaged, as well as the divided industrial support for particular techniques, a number of alternatives were standardised. At the present time, the suite of standards are as illustrated in Fig 3.3.2, with the service provided by the LLC layer being equivalent to the OSI data link layer service. These standards [3.13 & 16] offer alternative methods for the media access technique, but present a standard interface to layers above the LLC (in fact, a choice of four, corresponding to four classes of LLC defined).

Of the four media access techniques presented as standards, only CSMA/CD (802.3) was well established before the work was completed. This was based heavily on the protocols used in Xerox's Ethernet LAN, but differed sufficiently so as not to make it compatible with the standards as it existed. After it had become clear that some applications required the deterministic properties of token passing systems, two techniques were standardised. One was the token-ring (802.5), favoured by IBM (it can support a mix of asynchronous and synchronous traffic), whilst the second was a token bus system (802.4) which drew on the work on "Proway" and which was supported by DEC. Neither of these techniques were available on commercial systems. The well established slotted ring technique was not included in this work, after performance studies showed that it could not perform as well as the token ring technique for the envisaged LAN systems. However, the CR '82 standards for the Cambridge (slotted) Ring have since been modified to appear in the same form as the 802 accesss techniques. The final technique, (802.6) is concerned with networks of the scale between present day LANs and WANs, Metropolitan Area Networks (MANs), whose media access is via the slotted ring technique. All of these media access techniques support a connectionless transmission service only.

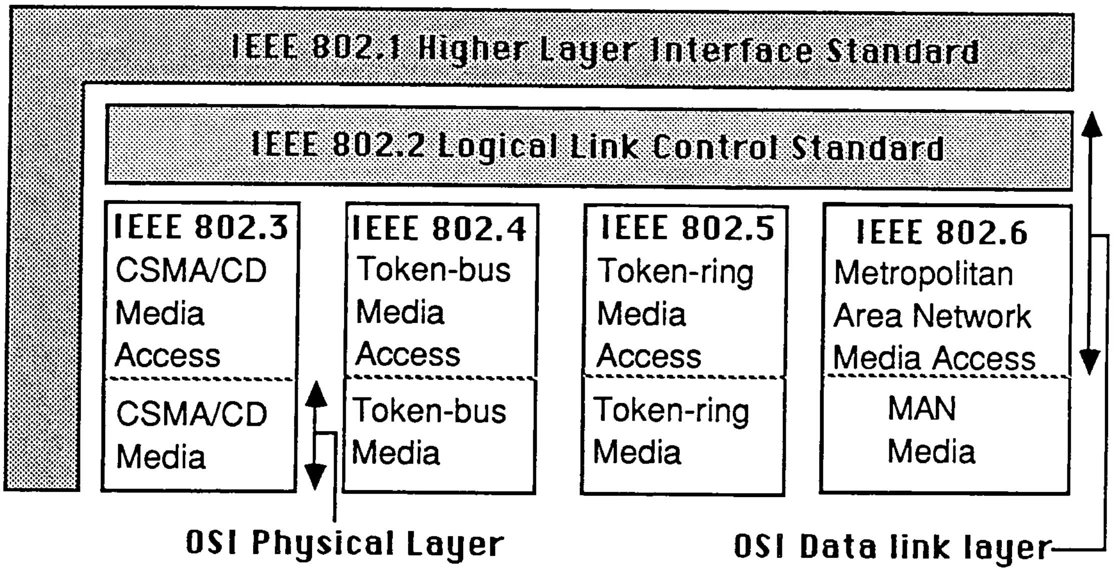


Fig 3.3.2 Suite of standards proposed by the IEEE 802 committee.

Although many LAN implementations have directly interfaced such MAC services directly to the network layer previously, a solution adequate for present day connectionless networks, the 802 committee identified a further sublayer, the Logical Link Control sublayer. The LLC builds upon the capabilities of the MAC to provide the following additional functions;

1- **Additional address selection** is provided so that multiple network layer entities can access the LAN. These multiple interfaces are Link-layer SAPs, which can be regarded as switches to permit the selection of various network layer protocols or functions (eg local communications or gateway functions(see section ~~x.x~~ 8.1)). It can also be used to address seperate users of the LLC service, in applications such as terminal concentrators.

2- **Enhanced connectionless service** adds the source and destination LSAPs before the data is passed to the MAC.

3- **Connection oriented service (optional)** creates connections between the LSAPs, enabling packet sequencing etc. Since the physical media is shared, each data packet still has to contain the source and destination addresses, so the data efficiency is not improved.

4- **Acknowledged connectionless service (optional)** provides a service required in certain real-time applications, where connection setup times are prohibitively long but where acknowledgement of the data's receipt is required.

5- **Polled response service (optional)** is provided for applications where the low-cost master-slave arrangement is sufficient (ie for interfacing low speed devices such as keyboards and sensors to LANs). This is especially suitable for token passing networks, where the largest cost reductions are possible.

6- **Exchanging ID with other LLCs** (and optional duplicate address check). The XID protocol allows a station to establish which other stations are active on the network and the level of features that they support. This protocol can also be adapted to check for other stations using its own address, by sending an XID request with its own address as source and destination. If any other node responds (apart from itself), then duplicate addressing is present.

7- **Downline loading (optional)** allows communications protocol programs to be kept centrally and loaded into the station on power up, (ie not MAC or certain essential parts of the LLC) allowing easier and faster software modification across the system.

8- **Loop-back test** is a simple test whereby an LLC receiving a test_c frame responds with a test_r frame (optionally echoing any information received on the original frame). This may be used as a test of the integrity of the link and underlying media (to establish bit error rates or to exercise part of the system).

The functions which are labelled as being optional do not have to be implemented in all systems. If any of these features are not present in a particular system, any commands associated with them must be ignored. Four types of LLC are defined within the standard; Type 1 supports the connectionless service only, whilst Type 2 additionally supports the connection oriented service. Types 3 and 4 are still under study, but involve the use of the Acknowledged connectionless and polled-response services respectively. They are intended to be used primarily in LANs for proccess control applications, in conjunction with the token passing bus standard.

Fig 3.3.3 illustrates the areas addressed by these standards as related to an actual LAN implementation (in this case, using the CSMA/CD standard).

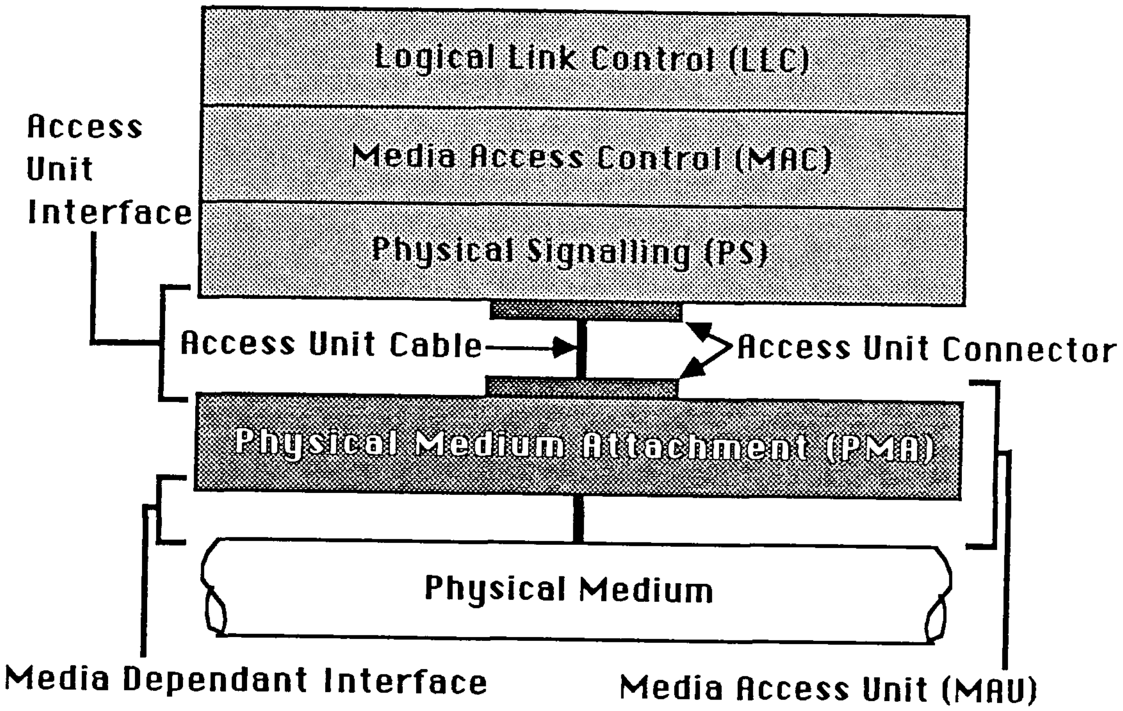


Fig 3.3.3 Example functional breakdown considered by IEEE 802

The final standard in the suite (802.1) is concerned with the relationship among these standards and the ISO model. It also explains the relation of the 802 standards to higher layer protocols and discusses internetworking and networking management issues. This is the most recent of the standards and is still under development.

3.4 Additional Issues

3.4.1 Network Interconnection.

When a collection of networks are interconnected to form an integrated communications system, the OSI terminology identifies the individual systems as "sub-networks". These "sub-networks" are interconnected via the use of "relays", an abstract concept consisting of an OSI system containing layers 1 to less than 7. These systems are known as "gateways" (or "bridges" or "interworking" units) in the real world, which are classified according to their function and named according to the sub-layer in which the relaying function is performed. The three major types of gateway , are the routing, converting and interfacing types. The routing type performs a mapping between two identical service interfaces and performs a routing decision, effecting a dynamic mapping between communicating SAPs. Such relays are used within packet switching networks where multiple routes are possible. The converting type of gateway performs the same mapping as the routing type, but performs no routing of the data. Finally, the interfacing type operates like the converting type but applies where a system has been divided at a service interface and physically distributed (this embodies the lower OSI layers).

Depending on the required function of the gateway, the relay function is placed in a sublayer somewhere between layer 1 and 5. The various types of function will now be listed with their associated layer position (ie they are present in a sublayer immediately above that layer, the sublayer not altering the layer service) and gateway type name as presented by Turner [3.17].

Where the relay functions are contained in a sublayer immediately above the **physical layer**, a Medium Extension Gateway (MEG) is formed. These are used to expand the topography of certain communications links, by allowing medium segments to be joined without signal degradation (eg. The MEG may regenerate clocking information and data waveforms). The two portions connected must be identical and belong to systems using the same low level mechanisms (eg CSMA/CD), since the physical signals are mapped one-to-one.

For the case where the functions are contained above the **MAC layer**, a Link Extension Gateway (LEG) is produced. Although the functions sit above the MAC layer, making them possibly media in dependant, the necessity for tight coupling to the lower layers (for efficiency) makes them suitable for a given sub-network technology only. It contains additional intelligence compared with the MEG. For example it is aware of the address space of each of the links that it interfaces and so can filter out messages not relevant to each link. This type of gateway is particularly useful where traffic within each of the link portions is high, but the traffic between them is low, enabling performance increases to be made.

With the relay functions placed above the **LLC layer**, an Inter Link Gateway (ILG) is formed, which is now independant of the network technology. Its efficiency makes it ideal for LAN to LAN interconnection, particularly where parameters such as data rate differ between the two. However, no mode conversion can take place (eg connection to connection-less operation) and so it can only interconnect like sub-networks. Inter-link routing is used between the communicating nodes.

The following two relay groupings are all placed within the network layer, which is itself divided into two sublayers. This is because of the wide range of services provided to the network layer by real systems, the lower sublayer (access network) is used to give a constant service above ~~to~~ the upper (harmonised) sublayer. The relay which contains its functions above the **access network sublayer** is known as a Network Extension Gateway (NEG). It is used to allow one network (the "lesser") to access another (the "greater") and so does not appear the same to systems on either side of it. The NEG appears transparent to systems on the "greater" network, but is visible to those on the "lesser" network, since communications must conform to the access procedures of the "greater". It supports mode conversion, using this to support the services of the greater network over the lesser (by using the same protocols or a suitable enhancement protocol over the lesser). The most likely application of an NEG is for connection of a LAN to an X.25 network, since the capabilities of the individual subnetworks can still be exploited at this level. When the relay functions are placed above the **harmonised sublayer**, an Inter Network Gateway (ING) is formed, which appears the same to systems on both sides of it. Thus it cannot use properties unique to any subset of the "sub-networks" used, but relies on the constant service provided to the harmonised sub-layer.

Finally, where the relay functions are placed above the **transport layer**, a Distributed System Gateway (DSG) is produced. This gateway looks like a single Open System from either side, since the transport protocol is supposedly end-to-end. This type of relay is not presently allowed in the OSI model, but could be used in certain circumstances when a LAN wishes to appear as an Open System to another external network. The transport entities apparently accessed within the DSG from without are actually distributed around the LAN (ie within the transport subsytems of the systems connected to the LAN), the high data rates used across it giving a suitably short response time. This allows the use of optimised solutions to communications problems within a particular LAN, and centralises the interfaces to multiple external WANs (the most likely application) into a single system, thus sharing their cost.

Fig 3.4.1 illustrates the case for a NEG, interconnecting a connectionless 802 LAN to an X.25 WAN (connection-oriented). In this case, the setting up of a connection requires the use of both the destination NEG address and the end-system "data-link" address. After this, only the end-system addresses are required. A more detailed appraisal of such an arrangement was presented by Folts [3.18]. All of the data has to pass through the network layer in the IWUs, as well as their associated relay and routing functions in the sublayer above it.

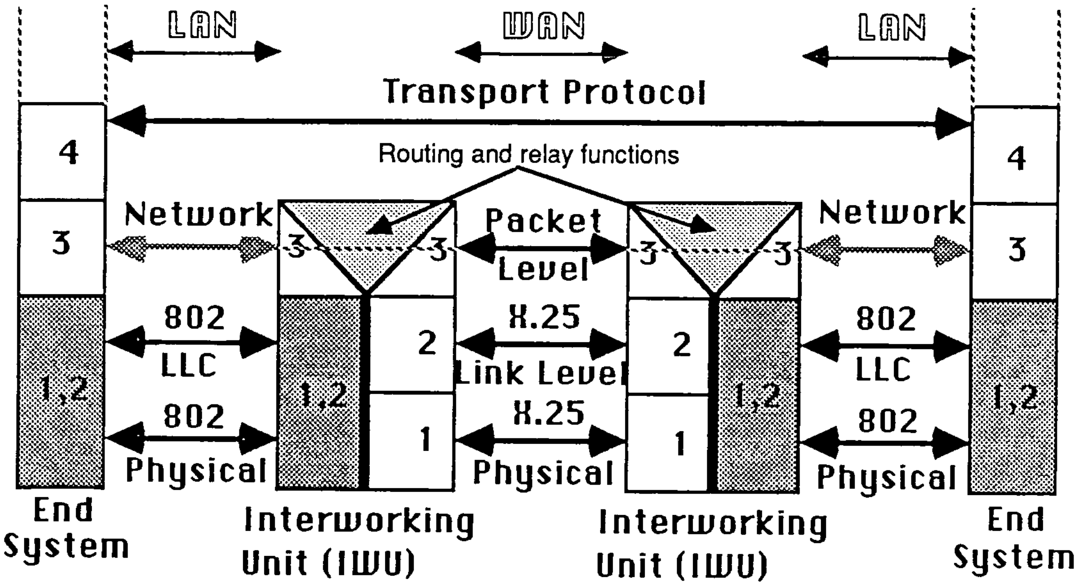


Fig 3.4.1 LAN to WAN Interconnection via NEGs.

3.4.2 Splitting the Layers.

There are many reasons for not having all of the seven layers run on the same processor. As experienced early on in computer networking, the processing power of computers involved with communications tasks is drastically reduced. Therefore, there is often a physical interface between portions of an end system dedicated to the processing of a specific subset of the seven layers. Much debate has taken place over the optimum location for the layer seperation to occur [eg 3.19], the position altering as more of the layers are accomodated within specialised hardware [eg3.20]. However, a number of factors, including the procurement conventions of the potential network customers, has led to three major network interfaces being offered to them.

The first interface is offered by homogeneous/single vendor networks where all of the necessary equipment to support the functions contained in the 7 layers is supplied by a single vendor. The user "interface" is therefore to layer 7, or the operating system enveloping it and the user is not concerned with any of the details of the network. This approach offers the user the highest assurance of equipment compatibility across the network, but makes the user very dependent on the products of a single vendor. The second interface is the one offered to users utilizing the "standards approach". The interface is generally presented as a tap to the physical media, and the user has to specify the protocol and signalling standards to be used by all equipment attached to the medium. This results in the possibility of interworking systems from multiple vendors, but care must be taken to specify the precise options to be used for the various ISO, 802 , ANSI X3T9.5 etc layers if incompatibity is to be avoided. Therefore the user has to be concerned with all levels of the OSI model, but will probably buy complete end systems containing all seven layers.

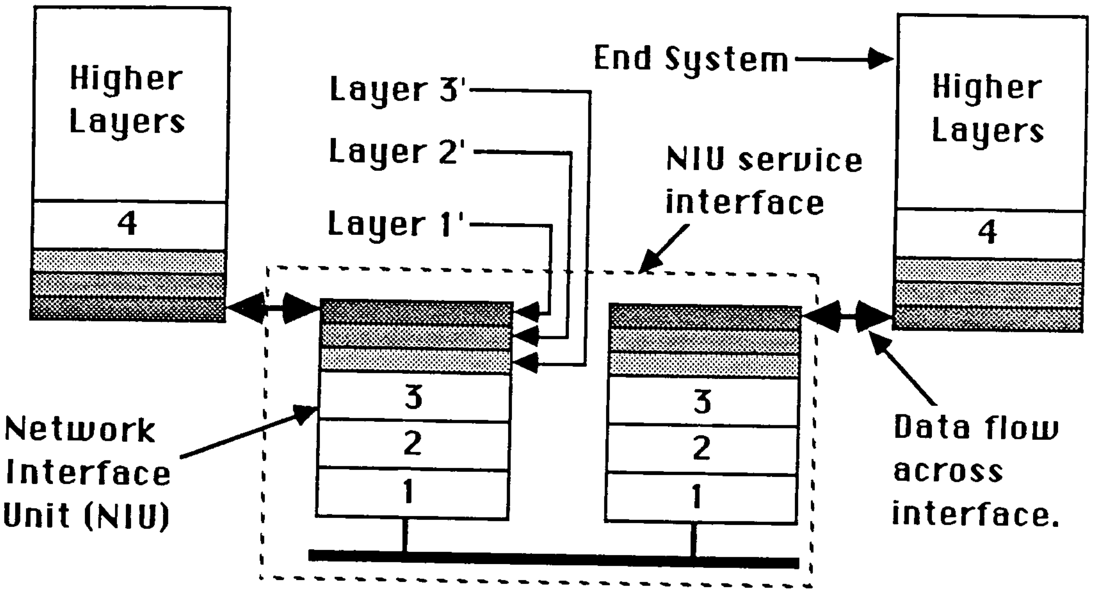


Fig 3.4.2 Interfacing to a Network via NIUs.

The final interface now commonly encountered, is that offered by intelligent network interface units (NIUs). For WANs, the units contain the first three layers, but for LANs this is often reduced to only the lower two. This is due to the fact that end-to-end links within the LAN itself are not made up of multiple links, and so the network layer's routing functions are not required. Therefore, the possibility exists to buy end systems containing the functions defined in layer 4 and above from multiple vendors, in addition to using NIUs from multiple vendors (although the latter may be less likely if a high assurance of equipment compatibility is required). This arrangement offers the possibility of changing the network used without having to change the devices connected to it, as long the service interface is sufficienly well defined and adhered to by the NIU suppliers. Examples of NIU interfaces in use at present are X.25, TCP/IP and RS232C. Fig 3.4.2 illustrates the auxiliary layers (or tiers of logic) necessary between the two seperated layers, in order for the transport of data between them to take place. Depending on the distance between the seperated layers, the complexity of these auxiliary layers varies, but are usually much simpler than the corresponding layers in the NIU.

3.4.3 Relevance of Issues to a Command and Control System.

The C^2 system under investigation is required to support devices from multiple vendors. Also, due to the cost and long development time for the systems used on the network, it would be greatly beneficial for any necessary alterations that have to be made to them (on changing from the old network to the new) to be minimised. Therefore, a standard interface or layer service should be used. These requirements point to the suitability of producing a new NIU for the C^2 system, whose functionality is sufficiently great as to make any peculiarities of its internal operation transparent to the user systems.

The use of gateways within the system is also necessary, since no single C^2 (or ship in our case) unit operates in isolation. It is desirable for end-to-end properties such as data encryption and routing to be handled by a dedicated system in order to offload this task from other specialist real-time end-systems, especially since the possible environment may require much adaptive routing. Also, since the network is likely to use specialised techniques for meeting the particular requirements of the application, their benefits should not be compromised by the use of a harmonising protocol. Thus the gateway should appear asymmetrical to data approaching it from differing sides. These requirements are best met by the use of a gateway whose relay functions are within the transport layer, a DSG. However, for applications where the end-systems are required to be visible as independent Open Systems from the "outside world", then a ING is the best solution This is because the lower two layers of the C^2 system are unlikely to be directly compatible with other communications systems.

Chapter 3 References

[3.1] Dahod A - "Local Network Standards: No Utopia" - *Data Communications* March 1983.(p173)

[3.2] Sanders L - "Interfacing with a Military Data Communications bus" - *Electronic Design* August 1982. (p205)

[3.3] Mandelkern D - "Rugged Local Network follows Military Aircraft Standard" - *Electronics* April 1982. (p147)

[3.4] Def Stan 00-19/Issue 1 - "The ASWE Serial Highway" - MOD 19th January 1981.

[3.5] Joshi S and Venkatraman I - "New Standards for Local Networks push upper limits for Lightwave Data" - *Data Communications* July 1984. (p127)

[3.6] Burr W and Carpenter R - "Wideband Local Nets enter the Computer Arena" - *Electronics* May 1984. (p145)

[3.7] FDDI "Token Ring Physical Layer Draft Proposal for American Standard" - ANSI X3T9.5 August 1985.

[3.8] ISO "Information Processing Systems - Open Systems Interconnection - Basic Reference Model" - ISO 7498 1983. (p4)

[3.9] Gee K - "An Introduction to Open Systems Interconnection" - NCC Publications 1980. (p6)

[3.10] The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications - DEC, Intel and Xerox - November 1982.

[3.11] Xerox System Integration Standard - "Internet Transport Protocols" - December 1981.

[3.12] Xerox System Integration Standard - "Courier: The Remote Procedure Call Protocol" - December 1981.

[3.13] Draft IEEE Standard 802.1: Part B "Systems Management" - February 1985.

[3.14] Chapin A - "Connections and Connectionless Data Transmission" - *Proceedings of the IEEE* December 1983. (p1365)

[3.15] Addendum to ISO 7498 covering Connectionless Data Transmission" - DP 8524 October 1983.

[3.16] IEEE Standards, 802.2 Logical Link Control, 802.3 CSMA/CD Access Method and Physical Layer Specifications, 802.4 Token Passing Bus Access Method and Physical Layer Specifications -1985. Draft 802.5 Token Passing Ring Access Method and Physical Layer Specifications and 802.6 Metropolitan Area Network.

[3.17] Turner K - "Gateways for Networking in the Framework of Open Systems Interconnection" - *Proceedings of the ICCC'84* October 1984. (p686)

[3.18] Folts H - "802 LAN/X.25 WAN Internetworking- A Pragmatic Approach" - *Proceedings of the ICCC'84* October 1984. (p572)

[3.19] Wale D - "Front-End Processors smooth Local Network Computer Integration" - *Electronics* February 1984. (p135)

[3.20] Larson K and Chestnut W - "Adding Another Layer to the ISO Net Architecture reduces costs" - *Data Communications* - March 1983. (p215)

Chapter Four

Spread Spectrum Techniques for Multiple Access to Media

4.1 Spread Spectrum Overview.

4.1.1 Introduction

"Spread spectrum is a means of transmission in which the signal occupies a bandwidth in excess of the minimum necessary to send the information; the band spread is accomplished by means of a code which is independent of the data, and a synchronized reception with the code at the receiver is used for despreading and subsequent data recovery."

The definition given above, presented by Pickholtz et al [4.1], differentiates the techniques used in spread spectrum (SS) systems from other modulation schemes, such as FM and PCM, which also spread the spectrum of an information signal. Use of these SS techniques date back to the Second World War when SS signals were seen to be useful for radio frequency transmissions and sensitive phone links. The suitability of these techniques for military purposes resulted in the classification of the majority of the subsequent development work [4.2-4.4] until the 1970's. Therefore, possible civilian uses of these techniques have only recently been considered.

The major useful characteristics that the SS signals possesed for the military applications were;

1- Low probability of interception- the unpredictable nature of the signal carrier makes surveillance (and even detection) of the tranmission difficult, also resulting in an increased degree of difficulty in ascertaining the position of the transmitter.

2- Antijamming- the unpredictable nature of the carrier results in the inability of the jammer to use observed signals to aid the jamming (eg narrowing the bandwidth to be jammed). Therefore techniques which are independent of the signal to be jammed are required. This property also results in good noise rejection.

3- High time resolution- differences in the time of arrival of the wideband SS signal (of the order of the reciprocal of the signal bandwidth) are detectable, resulting in the possible suppression of multipath effects (signals travelling between the Tx/Rx pair by multiple routes).

4- Cryptographic capabilities- which result from the fact that the data modulation is indistinguishable from the carrier modulation in the transmitted signal. Also, the carrier modulation is effectively random to an unwanted observer, therefore the signal is encrypted.

In addition to these, a further characteristic was identified. Multiple access to a common medium bandwidth was possible when Tx/Rx pairs transmitted using independent random carrier sequences. Systems using this property are identified as code-division-multiple-access (CDMA) spread spectrum systems. This multiple access technique is unique among those discussed in Chapter one because multiple users can use the same frequency band concurrently. This differs from TDMA systems where users can only access the medium at specified times. It also differs from FDMA in that, whilst both offer the provision of concurrent communication channels, only one transmitter can use a particular frequency band (channel) at any instant in the FDMA case.

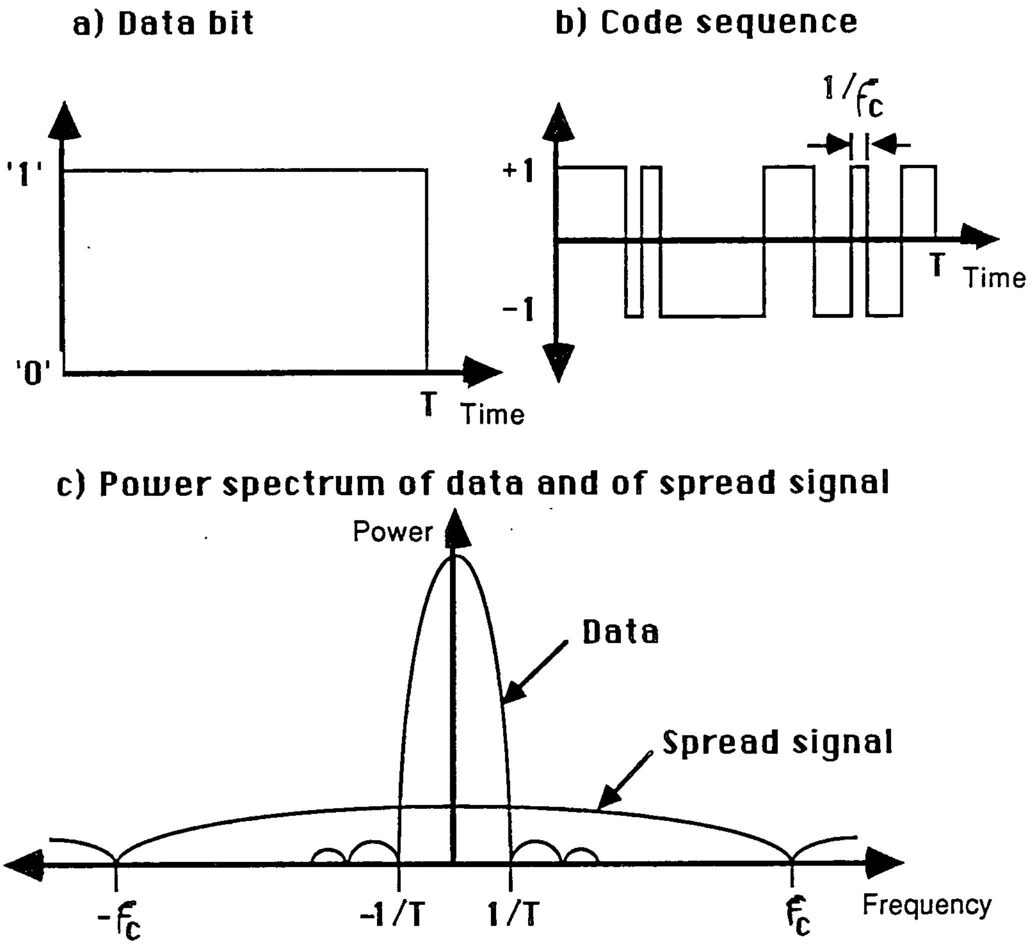
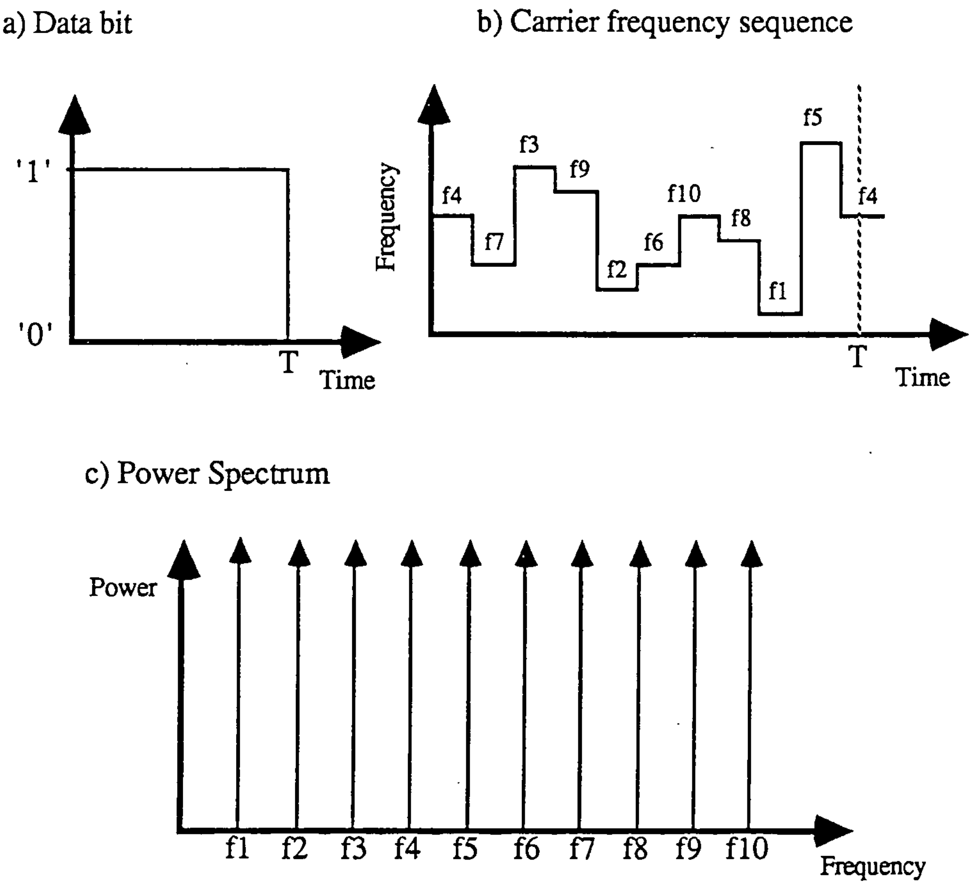


Fig 4.1.1 Direct Sequence Spread Spectrum Modulation.

4.1.2 Types of Spread Spectrum modulation.

There are three major types of SS modulation techniques that comply with the definition given in the previous section; direct sequence, frequency hopping and time hopping, with one which does not (due to its "codeless" method of operation), chirp modulation. However, since the latter is generally accepted to be a form of SS modulation [4.5-4.6], it will be included in this section.

Direct sequence modulation is accomplished, as the name suggests, by the direct multiplication of the data or the carrier by a code sequence. Fig 4.1.1 illustrates the spreading of a data bit's power spectrum by this technique. The data bit (a) is modulo-2 added to an integer number of code sequence cycles(b, in this case only one cycle). The period of each code sequence bit is named the "chip period" (1/fc where fc is the code sequence frequency) and this determines the amount of spreading of the power spectrum (c). The power spectra shown in the diagram are the shaping functions for the (average) discrete power peaks seperated by fc/L Hz, where L is the bit length of the sequence. The centres of the shaping functions lie at the carrier frequency used for the signal transmission in broadband systems (but at 0Hz for the baseband case) and both are described by the sinc^2 function. This results in about 90% of the transmitted power being contained in the central lobe, (the side lobes for the spread signal not shown on the diagram are of width fc) so that the bandwidth of the SS signal can be considered to be twice that of the code sequence frequency.


a) Data bit

b) Carrier frequency sequence

c) Power Spectrum

Fig 4.1.2 Frequency Hopping Spread Spectrum Modulation.

Frequency hopping modulation involves the transmission of the data on multiple carrier frequencies as illustrated in Fig 4.1.2. A number of such frequencies are chosen and assigned a channel number. A code sequence, of the type used in direct sequence modulation, is used to provide the sequencing of the frequency "hops". This is done by partitioning it into blocks whose values have a range equal to the number of assigned channels. Each block is read and the binary sequence used to identify the new channel, resulting in a signal such as that shown in (b). The sequence length must be an integer multiple of the block size. A suitable choice of both results in each channel being used only once during the sequence. The power spectrum for this type of modulation (c) shows peaks at the assigned channel frequencies only.

Two forms of this modulation technique have been used. The first, "fast frequency hoppers", transmit on multiple carrier frequencies during the transmission of a single data bit (as shown in b). The second, "slow frequency hoppers", transmit one or more data bits on a single frequency between "hops". The first type will give greater jamming immunity to the signal, but requires more complex receiver circuitry in the SS system.

Time hopping modulation uses a code sequence to switch the signal transmitter on and off in an apparently random manner. The swiching occurs when a one-to-zero or zero-to-one transition occurs in the sequence, one of the logic levels being associated with each of the "on" or "off" states. Since this scheme transmits only on a single carrier frequency, it is susceptible to jamming. Therefore instances of combining this technique with frequency hopping, to produce time-frequency hopping, have been observed. In this scheme, the frequency is changed only on code sequence level transition times, as opposed to the regular time intervals used in pure frequency hopping.

Chirp modulation spreads the transmitted signal's bandwidth by changing its transmission frequency during a certain "pulse" period. This is often acheived using a common sweep generator, with the receiver containing a filter which is matched to the angular rate of change of the signal. This technique is mainly used in radar applications, where it can result in significant power reductions for the transmitter.

Of the modulation methods outlined above, the most popular is direct sequence, whilst the time hopping and chirp techniques are only rarely used. It should also be noted that these techniques can be combined to form other hybrid methods such as direct sequence/frequency hopping modulation.

4.1.3 Types of Code sequences used in Spread Spectrum systems.

The major classes of the pseudorandom sequences used in SS modulation techniques have been defined by many authors [4.5 - 4.7], resulting in multiple names for the same code groupings. The classes considered here are m-sequences, combinational sequences, composite sequences and bent sequences. The generation of all of these code types from a single shift register of variable length and feedback taps, has been shown to be possible by Shaar [4.10]. However, in order to easily distinguish the classes in this section, their generation will be considered to be via the use of one or more m-sequence generators (as specified in their original definitions).

M-sequences (maximal-length sequences) are the longest codes which can be generated by a given shift register. In the binary shift register sequence generators considered in this section, the maximal sequence length is 2^n-1 bits, where n is the number of stages in the shift register. Examples of this type of generator are given in Fig 4.1.3, both of which function in the same way. Data in the register stages is shifted to the right, providing one bit of the code sequence as well as new data bits dictated by the modulo-2 sums of the previous register stage values. Different sequences are produced by varying the arrangement of the feedback taps.

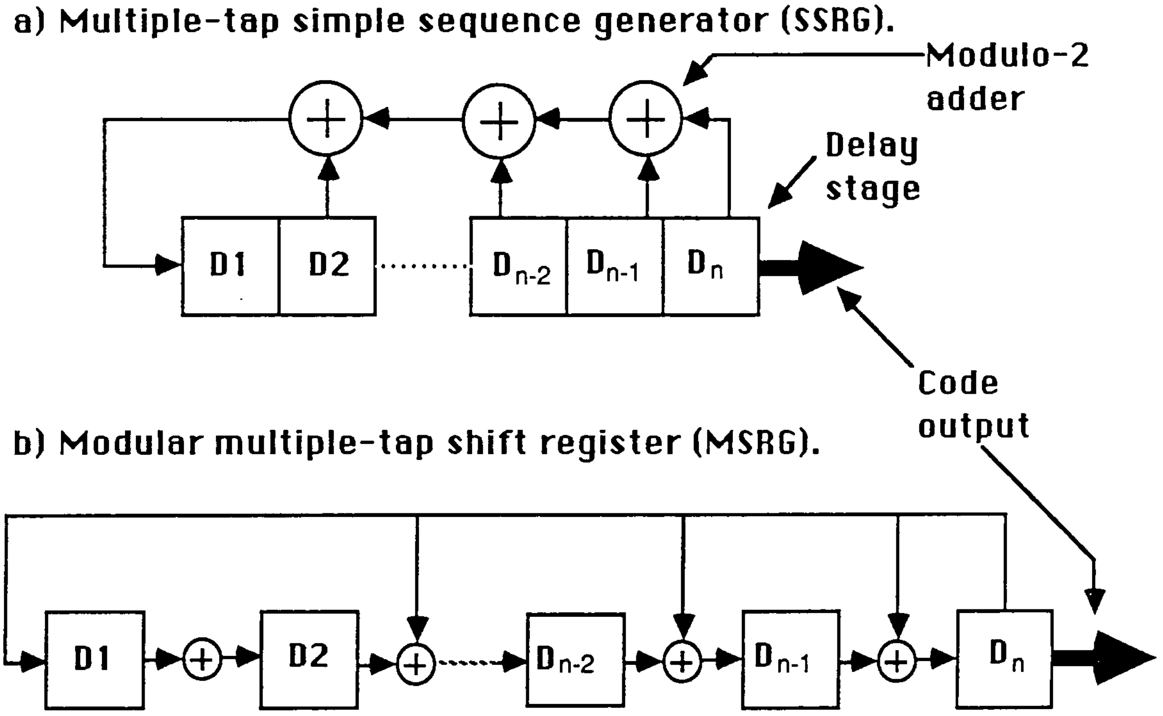

a) Multiple-tap simple sequence generator (SSRG).
Modulo-2 adder
Delay stage
D1
D2
Dn-2
Dn-1
Dn
Code output
b) Modular multiple-tap shift register (MSRG).
D1
D2
Dn-2
Dn-1
Dn

Fig 4.1.3 M-sequence Shift Register Generators.

Appropriate feedback connections have been tabulated for maximal code generators from 3 to 100 stages, so the sequences are well known. The generator arrangement shown in (b), is used in hardware implementations of code generators for two reasons. Firstly, the hardware is easily divisible into repeatable modules, easing implementation problems. Secondly, since the shift delay consists of only that for a single-tap sequence generator of the type shown in (a), (the delay rises linearly with the number of taps in a) resulting in higher possible sequence frequencies.

Combinational sequences are formed by the multiplexing together of two m-sequences according to a particular algorithm. The most well known type of sequences belonging to this class are the "Gold codes", which are produced by the modulo-2 addition of two m-sequences of the same length. This produces a new, non-maximal, sequence with the same bit count as each of its constituent pair. The difference in phase between the two source codes defines the particular Gold code which is generated. This results in a family of codes, whose size is equal to the length (in bits) of the source codes. For all of the codes of this type, no information is included to enable the reproduction of the constituent code sequences.

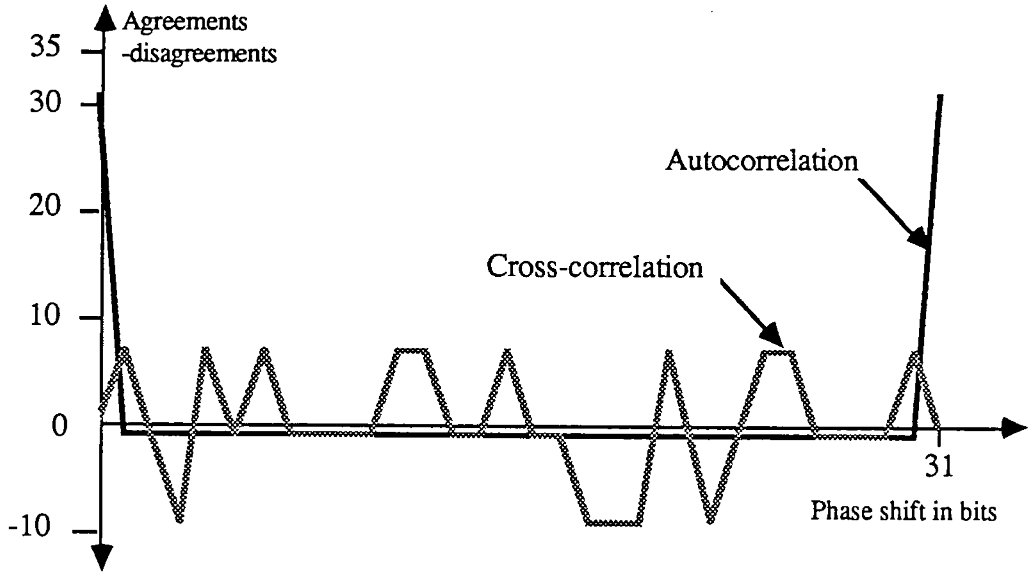


Fig 4.1.4 Autocorrelation and Crosscorrelation plots for 31-bit m-sequences.

Composite sequences (concatenated/syncopated sequences) are similar to combinational sequences in that they also use the multiplexing of m-sequences in their generation. However, in this case, sufficient information is resident in the composite sequence to allow the regeneration of the source sequences.

Bent sequences also use the multiplexing of m-sequences in their generation, but in this case the multiplexing operation is non-linear (ie the mathematical definition of the operation cannot be expressed by additions alone, but must include multiplication).

4.1.4 Descriptive parameters of pseudorandom sequences.

The numerical value of a number of parameters determines the suitability of certain code types for use in particular SS systems. Before a comparison is made between the code types outlined in the previous section, the meaning of each parameter will be explained.

The **Sequence Imbalance** is the difference between the number of bits containing a logical '1' and a logical '0' in the sequence. Since all of the sequences are odd in length, this value cannot fall below zero. The importance of this parameter is twofold. Firstly, the smaller the imbalance, the smaller the DC component in the signal, which eases some system implementation problems. Secondly, the smaller the imbalance, the greater the signal's similarity to random noise. This is important in CDMA systems, since the desired signal's recoverability requires such a random noise background to exist. Therefore the sum of multiple sequences being concurrently transmitted onto the shared media bandwidth, should always approximate to random noise, which will occur if each sequence does so.

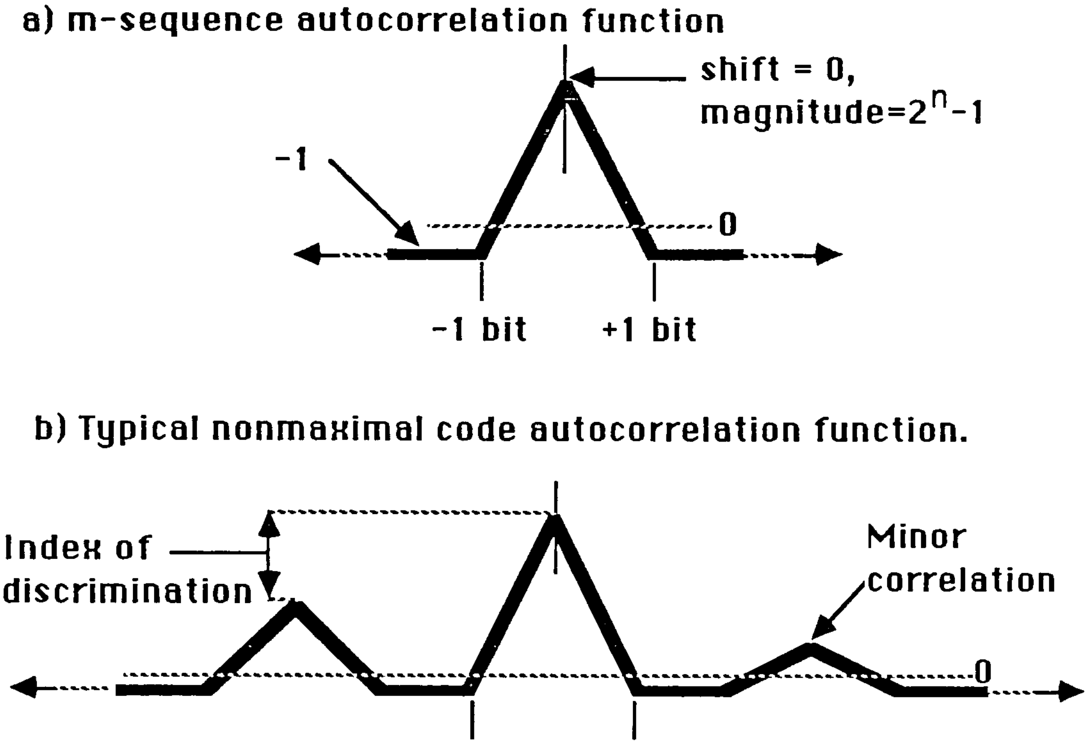


Fig 4.1.5 Autocorelation of Maximal and Non-maximal sequences.

The **Autocorrelation Function** (ACF) reflects the degree of correspondence between a sequence and a phase shifted version of itself. Autocorrelation plots, such as fig 4.1.4, show the number of agreements minus disagreements for the overall length of the two codes being compared, as the codes assume every shift in the field of interest. Whilst the maximum in-phase autocorrelation is obviously the sequence length in bits, the out-of-phase values differ according to the sequences. For m-sequences, these values are always -1 (see fig 4.1.4 and 4.1.5.a), but for all others this uniformity is not present (fig 4.1.5.b). The difference between the in-phase and maximum out-of-phase values is named the **Index of Discrimination**. The larger this value is, the easier it is to synchronise a locally generated version of a sequence to the same sequence in a noisy CDMA environment.

The **Crosscorrelation Function** (CCF) reflects the degree of correspondence between two different sequences. This is plotted in the same way as for autocorrelation (fig 4.1.4), the example given being for two 31-bit m-sequences which are not images of each other. Sequence images (ie same sequence but generated in reverse order) give a smaller index of discrimination and produce a larger number of differing cross-correlation values. (Note the three value nature of the example). The smaller the crosscorrelation function is, the easier it is to discriminate between different sequences in a CDMA environment.

The **Correlation Parameter** is a measure of the most significant interference component encountered within sequence families. It takes the larger value exhibited by the maximum out-of-phase ACF value (considered for all family members) and the maximum CCF value (considered for all possible pairings within the sequence family). A low value for this parameter enables easier discrimination of the sequences within the family in a CDMA environment. The theoretical lower bound for this parameter is called the **Welch lower bound**, which is given by the equation;

$$W.L.B = L((f-1)/(Lf-1))^{1/2}$$

where L is the code sequence length and f is the size of the sequence's family. (A recent, more accurate bound was formulated by Sidelnikov, especially for binary sequences.)

The **Synchronisation Time** is the number of sequence bits required for synchronisation between two instances of the same sequence can be acheived (a low value is preferable for CDMA systems). The **Linear Span** of a sequence is the number of bits required before the rest of the sequence can be accurately predicted (secure systems prefer large values).

4.1.5 Performance comparison between different sequence types.

With reference to the parameters defined in the previous section, the performance of the differing sequence types will now be compared. The results presented in this section are based on data taken from comparative studies presented by Smythe [4.11], Shaar [4.10] and Lin[4.12].

The autocorrelation of m-sequences has already been mentioned, the resulting index of discrimination being the maximum possible. Bent sequence families have a slightly reduced IOD, but this is greater by a similar amount to that exhibited by families of Gold codes. Values for composite sequences are very dependent on the source codes used, and so a representative figure for this sequence type is unavailable. The maximum crosscorrelation function for the sequence types reveals that Bent sequences exhibit the smallest value by far, especially for long sequences (again, no data is available for composite sequences). The highest value is exhibited by m-sequences. These results enable correlation parameter values to be calculated, which again favour the bent sequences with the other two types exhibiting higher values. However, comparison of the Welch lower bound with the experimental results shows that, in fact, m-sequences have the same lower bound as bent sequences. This is a result of the fact that both exhibit the same family size for a given sequence length. Therefore, the data merely shows that considerable variation can occur above this lower bound for differing sequence families.

The synchronisation time for composite sequences is very low (about 10% of sequence length for short sequences of less than 1000 bits in length, falling to below 1% for longer sequences of around 10,000 bits) whilst for all other types of sequences, this time is equal to a complete sequence cycle. Bent sequences exhibit the highest values for their linear span (although the values are dependent on the particular sequence, thus the figures are less predictable for users), with m-sequence families exhibiting the lowest. The values for the composite codes are dependent on the sum of the lengths of the source codes and so vary greatly.

Another parameter, not mentioned in the previous section, is the size of the family from which the sequence is a member. For m-sequences, the size is given by $1/(n\mathit{E}(2^n-1)$, where n is the number of stages in the shift register and *E* is the Euler function. Gold codes have a family size of 2^n-1 whilst the figure for bent sequences is $2^{n/2}$. Finally , for composite sequences composed of source codes of lengths a and b, the family size is given by $1/(ab\mathit{E}(f(a))\mathit{E}(f(b))$, where f(x) is the family size of the m-sequence of length x. This gives the result that gold codes give by far the largest family size for a given sequence length, with bent sequences having the smallest. Finally, the sequence imbalance for m-sequences and Bent sequences are both 1, the lowest possible value. Gold codes exhibit higher values, but comparison with composite sequences is again difficult.

It is clear that more work is required in this area for a comprehensive comparison to be made, especially between codes of the same type but differing families, and between the code types themselves. However, it is hoped that the basic characteristics for each type have been illustrated, which wil be used in section 6 when discussing the construction of a CDMA system.

4.2 Provision of CDMA for a C^2 system.

4.2.1 Requirements for the multiple access technique.

SS techniques were employed in the multiple access scheme for the new LAN under development because of the resultant system properties that they provide (see section 4.11). All of these properties are useful in military networks and cannot all be provided by any other known technique without the use of much extra equipment. Fig 4.2.1 shows a schematic diagram of the general form that was required for the network. Concurrent communications channels would be provided by the use of mutually orthogonal (or sufficiently so) code sequences over the same media bandwidth. This is what was considered to be CDMA in this instance, although other authors have termed it " Spread Spectrum Multiple Access".

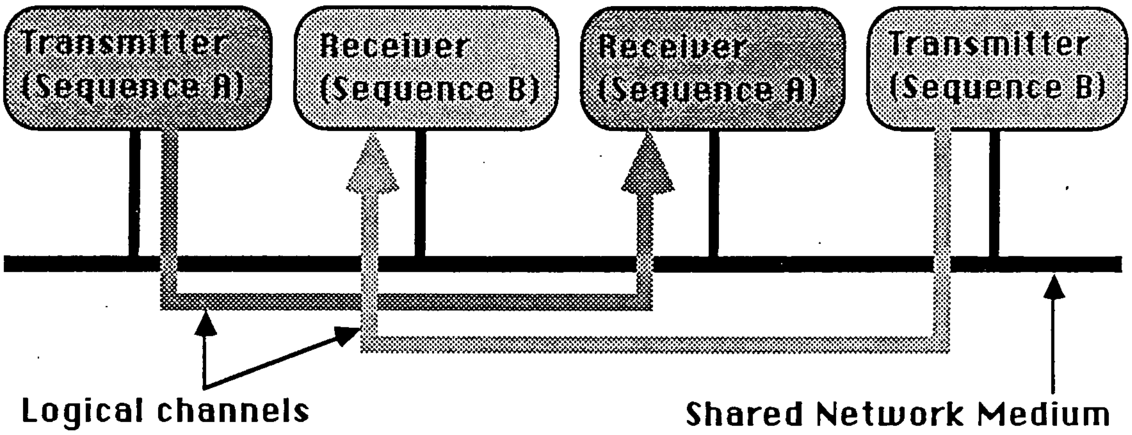


Fig 4.2.1 The required form of the CDMA system.

The use of Direct Sequence modulation for the network was decided upon because it offered the potential to be realised by a completely digital system. The envisaged hardware was to be comparatively simple (thus more reliable) and compact (attractive for the Naval application), with the added benefit that the component cost should fall in future years. The means by which DSSS systems in general provide the useful properties outlined in Section 4.1.1 is explained in the following section. The completely digital solution to the provision of this technique then follows.

4.2.2 Direct Sequence modulation analysis.

The analysis that follows is based on that provided by Judge [4.13], which was subsequently refined by Smythe [4.11]. The system considered transmits the signals without a carrier (ie baseband) as the new LAN is intended to, but for the present, the signal is assumed to be analogue in nature. Therefore the demodulation is considered to be accomplished by a method known as "integrate and dump", which is the nearest approximation to the digital technique to be used. The multiple user system under consideration is illustrated in Fig 4.2.2.

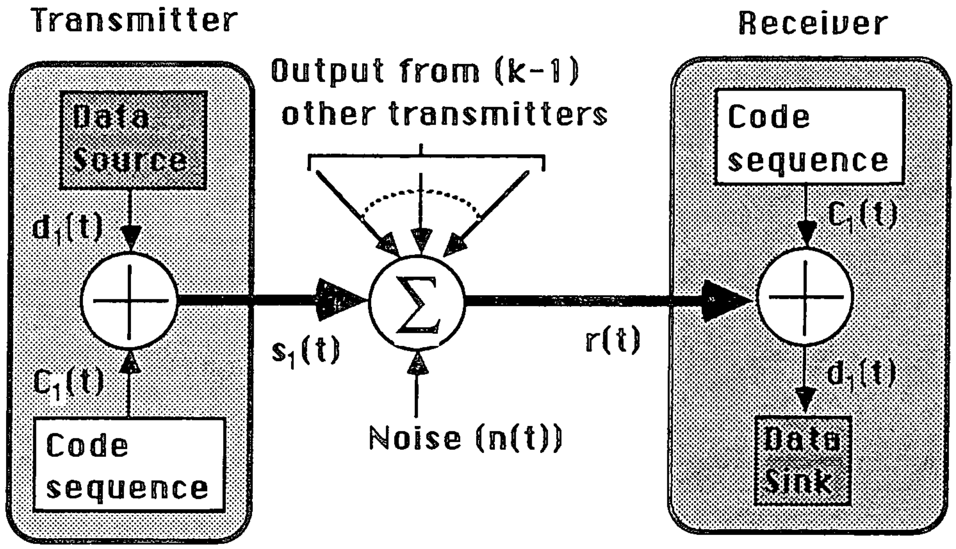


Fig 4.2.2 Schematic diagram of a synchronised DSSS system.

The system considered consists of k simultaneous users, the data originating from the k^{th} user taking the form $d_k(t-t_k)$, where t_k is the time displacement from a system reference code. The code sequence of the same user takes the form $c_k(t-t_k)$. The transmitted signal is represented by equation 4a, and is illustrated by fig 4.2.3.a.

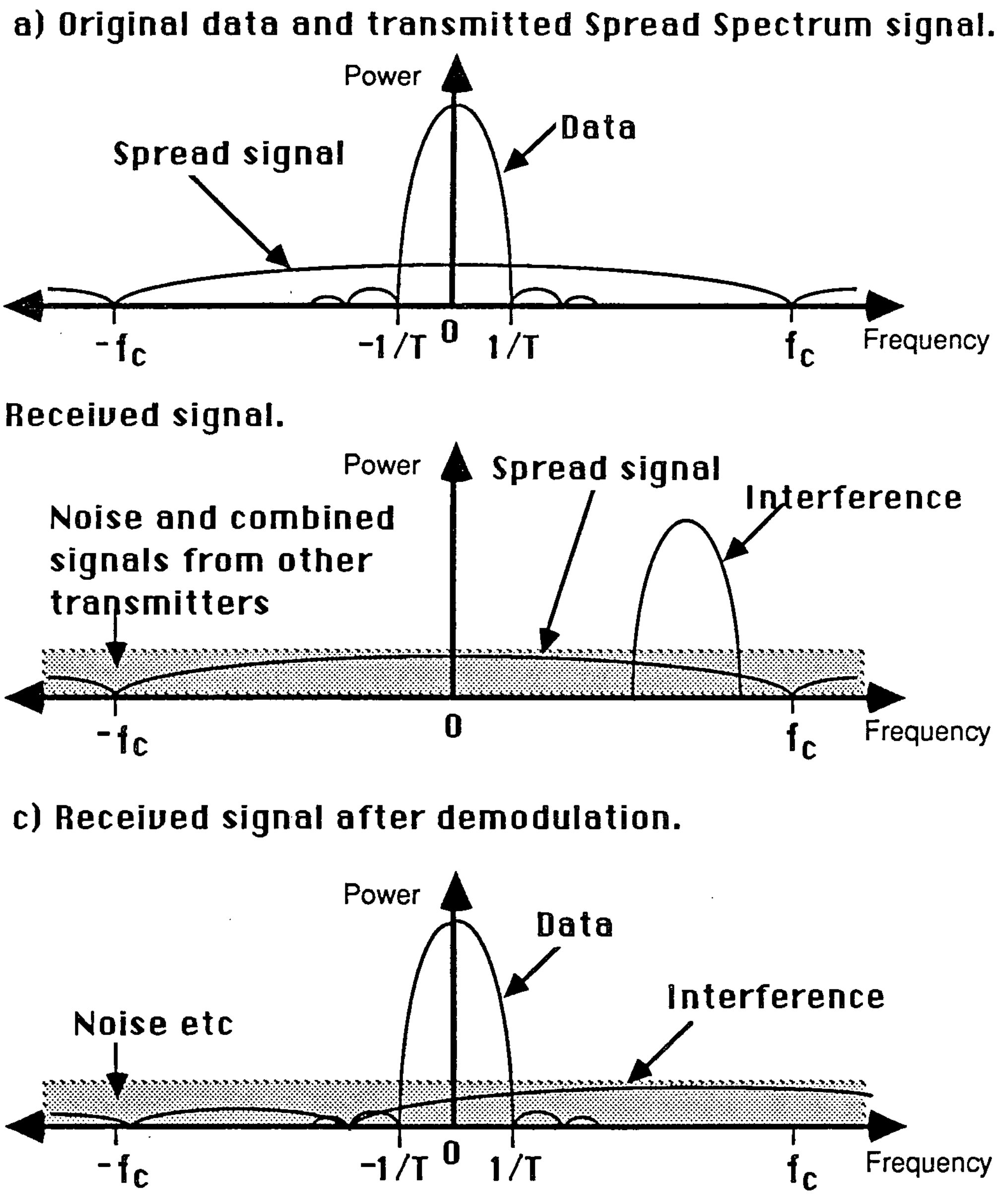
$s_k(t-t_k) = C_k(t-t_k)d_k(t-t_k)$ -------------(4a)

If it is assumed that the network medium is linear and additive, that the noise and interference not originating from other users is defined by n(t) and that all of the system users are transmitting simultaneously, then the signal observed by the receiver is given by equation 4b and illustrated by fig 4.2.3.b.

$r(t) = n(t) + \sum_{i=1}^{k} s_i(t-t_i)$ -------------(4b)

If it is assumed that the code sequences used in the transmitter and receiver have undergone synchronisation (ie t_i=0), then after the incoming signal has been multiplied by the local code sequence (which in this instance is the same as that used in the transmitter), the signal is described by equation 4c.

$r_1(t) = s_1(t)^{.C_1(t)} + n(t)^{.C_1(t)} + \sum_{i=2}^{k} s_i(t-t_i)^{.C_1(t)}$ ------(4c)



a) Original data and transmitted Spread Spectrum signal.

b) Received signal.

c) Received signal after demodulation.

Fig 4.2.3 Power spectra of signals in a baseband DSSS system.

The signal is then demodulated by the process outlined previously, to give a signal of the form (4d). (note. Td is the data bit period).

$$r_1(t) = (1/Td)\int_0^{Td} s_1(t)C_1(t)dt + (1/Td)\int_0^{Td} n(t)C_1(t)dt + (1/Td)\sum_{i=2}^{k}\int_0^{Td} s_i(t-t_i)C_1(t)dt$$

--------------(4d)

If we substitute the equation describing the signals form (4a) into this output (4d), then the equation takes the form of 4e.

$$r_1(t) = (1/T)\int_0^{Td} C_1{}^2(t)d_1(t)dt + (1/T)\int_0^{Td} n(t)C_1(t)\cancel{d_1(t-t_1)}dt +$$

$$(1/T_d)\sum_{i=2}^{k}\int_{0}^{T_d} C_i(t-t_i)C_1(t)d_i(t-t_i)dt \text{ ----------(4e)}$$

If the signal $C_1^2(t)$ is formed to become unity (ie in phase autocorrelation) and the signals $C_1(t)C_i(t-t_i)$ are arranged to give zero (ie low crosscorrelation values) then the signal collapses to give 4f, illustrated in fig 4.2.3.c.

$$r(t) = d_1(t) + (1/T_d)\int_{0}^{T_d} n(t)C_1(t)dt \text{ ----------(4f)}$$

Since the received noise and interference underwent spectral spreading before being integrated over the bit period, its energy is spread over a wide bandwidth. Therefore, after filtering the signal about the data bandwidth, the signal effectively becomes that shown in equation 4g and the desired data signal is received.

$$r_1(t) = d_1(t) \text{ -------(4g)}$$

It should be noted that if the sequence used by the receiver was not used during the integration period, then the received signal would fall to zero. Therefore, to avoid confusion between this case and the reception of a '0' data bit, bipolar signalling is used so that a negative voltage signifies a logical '0'.

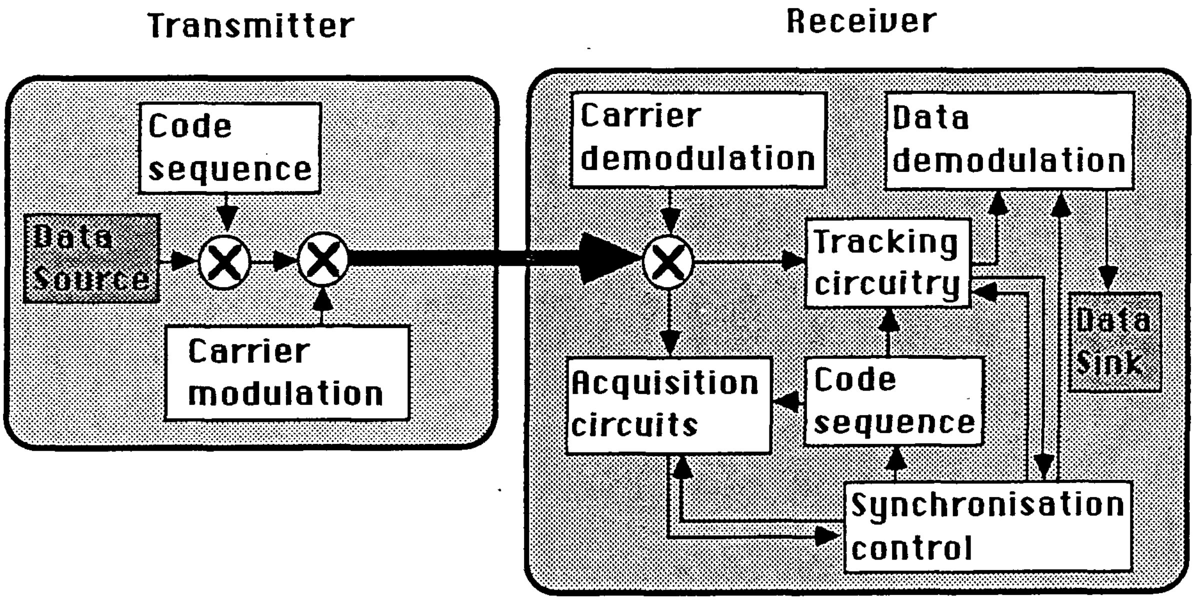


Fig 4.2.4 A general DSSS system.

This analysis indicates a property required from the code sequence family used in DSSS systems. The need for low cross-correlation values is illustrated by the requirement that the signals associated with other sequences fall to zero. However, other properties are also required to enable the code synchronisation assumed in this analysis. These are outlined in the following section.

4.2.3 A Generalised Direct Sequence System.

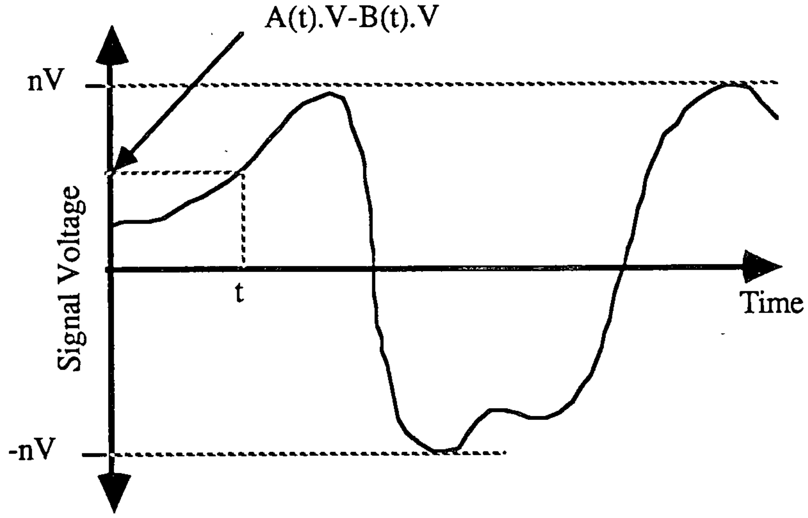
Before an analysis of the new all digital system is undertaken, consideration of a generalised DSSS system is made in order to ascertain the functional requirements such systems present. Such a generalised system is illustrated in Fig 4.2.4. in terms of interconnected functional blocks.

The transmitter illustrated uses the modulo-2 addition of data and code sequence as outlined previously. In addition, the signal is modulated onto a carrier in some way, this being a "null" function in baseband systems which transmit the digital data directly. The carrier demodulation function in the receiver recovers the SS data from the modulation technique used. Before the actual data is obtainable, the sequences have to be synchronised (ie chip aligned). This process is split into two tasks, initial coarse synchronisation (acquisition) and continued fine-tuning (tracking). The need for acquisition is a result of the asynchronous nature of the network (ie transmission can begin at any time), doppler shift in non-static systems and the instability of the clocks used to generate the sequence rate. Solutions that have been used in the past include "sliding correlators" (where the data stream is continually autocorrelated with the local code sequence, used at ever decreasing frequencies, until a match is detected) and Surface Acoustic Wave devices (which are tuned to the individual waveforms produced by particular sequences).

Tracking of the sequences is necessary because of the doppler shift and clock instability factors mentioned previously. The circuitry used normally monitors the autocorrelation peak so that if the value drops, the receiver's clock is advanced or retarded to regain the peak value (ie the autocorrelation peak is triangular in nature in practical systems). If synchronisation is lost during tracking then the signal is re-acquired from scratch. As illustrated in the figure, the tranfer of control between the two functional units is undertaken by a synchronisation control unit. The data is obtained from the tracking cicuitry when the latter is in use and then passed on to the data sink. The use of the autocorrelation peak in the acquisition and tracking techniques results in the need for a large index of discrimination in the code sequence family used. This is especially true in noisy environments when some of the sequence bits will inevitably be incorrectly read from the channel.

4.2.4 Provision of a digital DSSS system.

The digital system described within this section is based on work presented by Ismail [4.14] and subsequently developed for use within LANs by Scott [4.15]. It assumes the use of an additive channel (ie two logical '1's on the channel give twice the signal voltage on the channel than for a single '1'). The multilevel signal that is therefore produced on the channel is illustrated in Fig 4.2.5. The signal is closer to an analog signal than a multilevel digital one in practice, however, due to the attenuation of the signal over distance and the non-uniform attenuation and velocities of the frequency components of the digital signal.



A(t)= Number of nodes transmitting '1' at time t.
B(t)= Number of nodes transmitting '0' at time t.
n= Total number of nodes.
V= Signalling voltage (V=1, -V=0).

Fig 4.2.5 Additive channel signal waveform.

A functional block diagram of this system is illustrated in Fig 4.2.6. The transmitter functions in a manner as outlined before and the outputs of all such transmitters are added together on the channel along with the noise present on the channel. In order to obtain a binary signal from the received waveform, the simplest solution is to perform a 1-bit A/D conversion (ie if the sampled waveform has a negative voltage, a logical "0" is assumed. Conversely, a positive voltage is interpreted as a logical '1'.

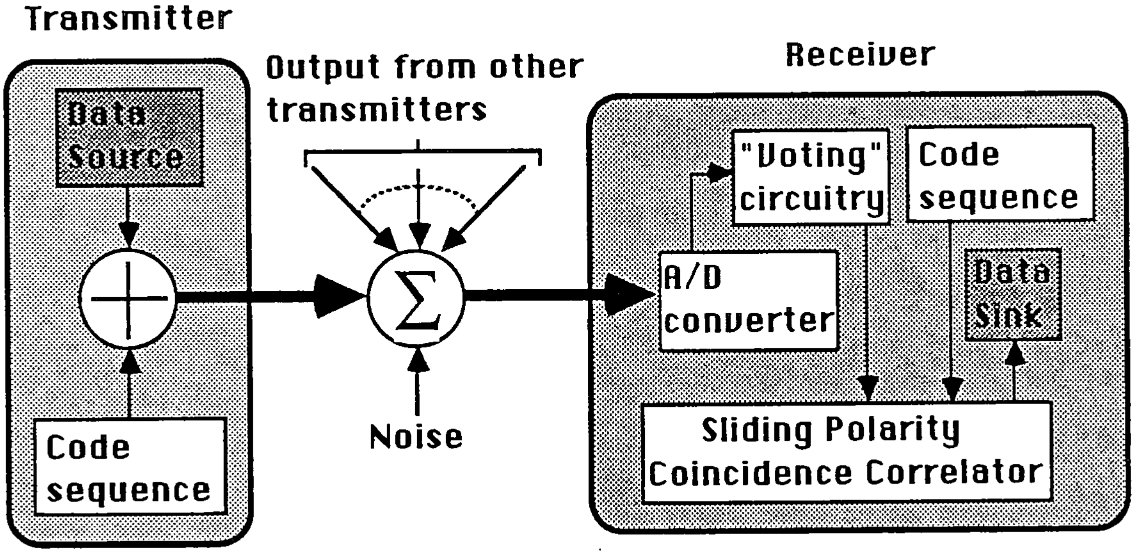
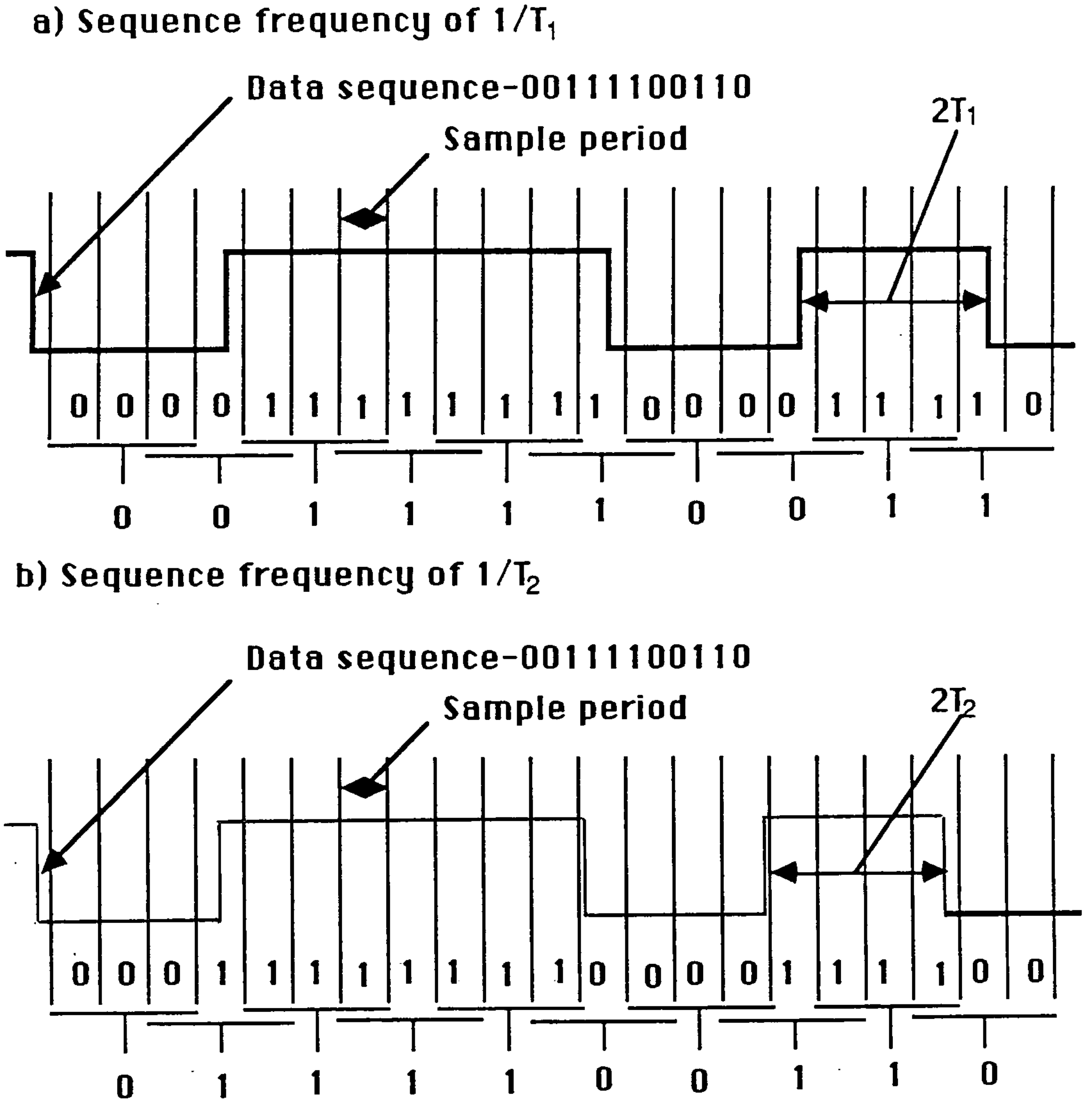


Fig 4.2.6 Functional block diagram of digital DSSS system.

In order to obtain this binary information without any initial synchronisation, the signal on the channel is sampled at twice the nominal sequence rate, with the value occuring most over three sampling periods being considered the correct value. This scheme is illustrated in Fig 4.2.7 Diagram a) shows how the correct sequence is recovered from a binary waveform whose sequence rate is correct, but whose phase is non-aligned with the sampling periods. The case illustrated in diagram b) is similar to that in a), but the sequence rate is slightly faster than the nominal value, resulting in a phase shift of the recovered sequence, which is nevertheless still correct.


a) Sequence frequency of $1/T_1$

b) Sequence frequency of $1/T_2$

Fig 4.2.7 "Voting" system used to determine sequence bit.

This binary signal is then passed to the "Sliding Polarity Coincidence Correlator", which performs a continuous acquisition on the data stream, thus no tracking circuitry is necessary. The correlator works by continuously comparing the local sequence with the input stream and counting the number of agreements or disagreements that occur over the entire sequence length, calculating the value of their difference divided by the sequence length ((A-D)/L). These values are updated each time a new data bit is read in, the process being illustrated in schematic form in Fig 4.2.8. If the (A-D)/L value exceeds one of a pair of threshold values, synchronisation is assumed. If the upper threshold is exceeded, then a logical '0' is passed to the data sink. Conversely, a '1' is passed on if the lower threshold boundary is exceeded. This is because of the modulo-2 addition of the sequence and data at the transmitter, resulting in the sequence itself on adding '0' and the sequence's complement on adding '1'.

The reception of data modulated on any other code sequence results in the value of (A-D)/L being within the threshold values and so no data is fed to the correlator. The calculation of this threshold value is dependant on a number of factors and will be investigated in the next section.

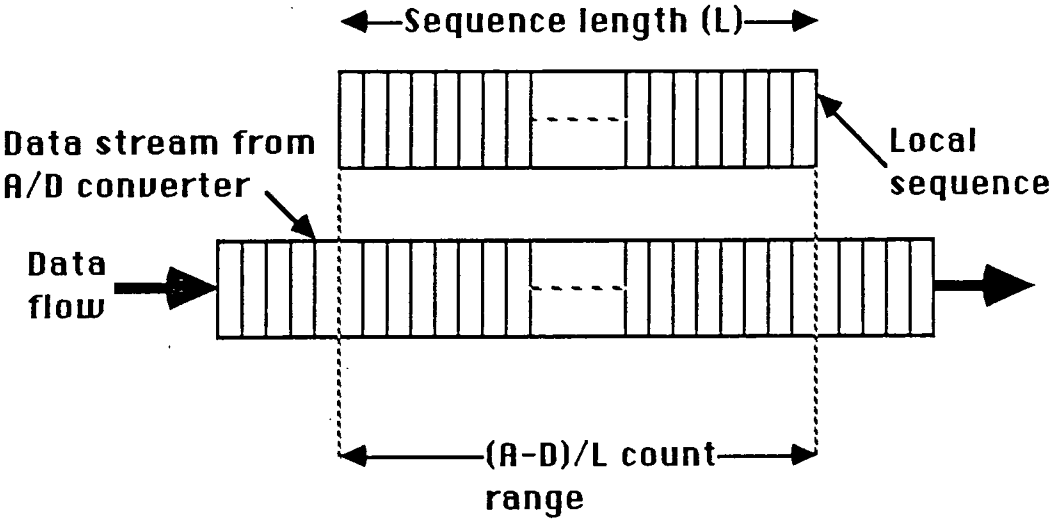


Fig 4.2.8 Schematic diagram of "Sliding Polarity Coincedence Correlator".

An improvement in the bit error performance of the receiver is possible by increasing the quantisation of the input waveform. Two methods for doing this have been suggested by Smythe [4.11] and Freeman [4.16]. The first involves the use of multibit A/D converters in the following way. A converter providing n bits of data identifies n^2 signal levels, n parallel correlators being used to correlate the binary signals from the different data bits in the the n-bit word. The results from these are then combined, using appropriate weighting factors for the most significant bits, to obtain the data bit. The second method involves the introduction of a dither signal to the received signal, which helps when regulated noise is present on the line. The first method can improve performance by about 10-20%, but at great additional complexity and hardware expense. The particular benefits obtained from the second method are obtained from less additional complexity at the receiver.

4.3 Digital CDMA system requirements.

4.3.1 Determination of Correlator threshold values.

One of the requirements for any system using the digital CDMA technique outlined in the previous section, is the allocation of the threshold boundary values to the receiver's hardware. These values have been shown to be dependant on the signal to noise ratio of the received signal, denoted by SN_{rec}. This dependance is described by a pair of equations, 4.3.a&b, (where L is the sequence length in bits) and is illustrated by Fig 4.3.1. It should be noted that the equations were derived using the assumption that $SN_{rec}<1$, so that the point where the threshold reaches unity (at $SN_{rec}=\pi/2$) is beyond their limit of application. However, it should be noted that in the cases where SNrec>1, the performance of the system will be improved, since the randomizing of the data by noise and other users is removed. This results in the applicability of the use of threshold values approaching unity.

Agreement threshold= $L\sqrt{(SN_{rec}/(2\pi))} + L/2$ -------(4.3.a)

Disagreement threshold= $-L\sqrt{(SN_{rec}/(2\pi))} + L/2$----(4.3.b)

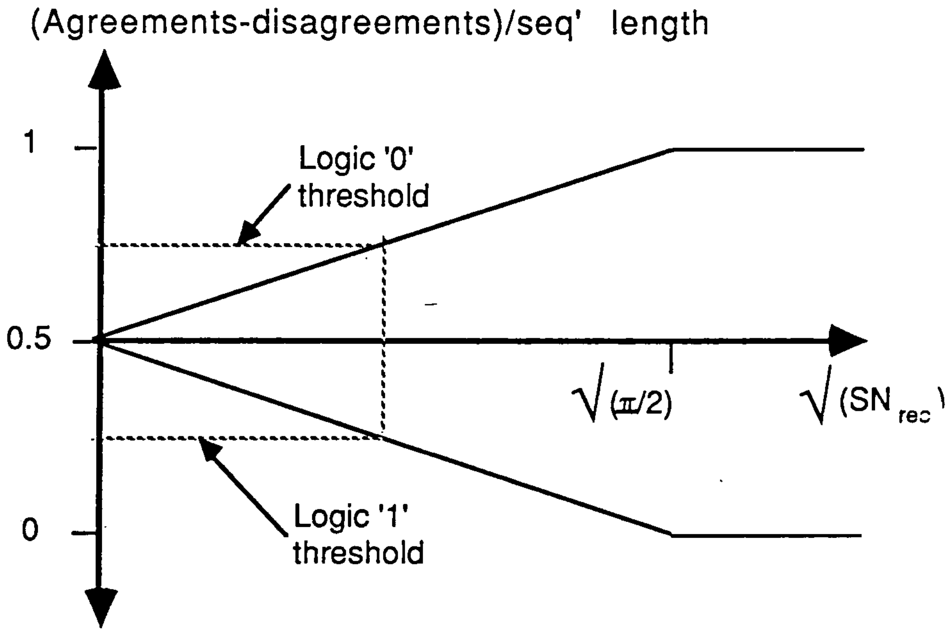


Fig 4.3.1 Digital correlator threshold plot.

It can be seen that if the threshold values are too low, the receiver may fabricate data from non-valid seqences of bits. However, if this value is too high, the receiver will miss valid sequence bits. Therefore, these values have to be set at a high degree of precision to enable a low bit error rate from the receiver. It seems that the best way to tackle this problem is via the calculation of SN_{rec} by the receiver, using a formula which was again presented by Smythe, eq

4.3.c.,

$SN_{rec}=1/((k-1)f(p) +1/SN_{one})$ ---------(4.3.c)

where $f(p)=((V-vL_n)/(V-vL_s))^2$, v=voltage loss/metre, V=signalling voltage and Ln and Ls are the mean distances (from the receiver) of the noise and signal sources respectively. Finally, SN_{one} is the signal to noise ratio of the signal if it had been passed over a noiseless channel, which is constant for all possible transmitters.

Of these parameters, only L_n Nd L_s are unknown. SN_{one} can be found experimentally and V,v andL will be known. The number of concurrent transmitters in use, k, is obtainable from Network Management software (see chapter 8), but must be stabilised in the following way. Network management can only know how many "conversations" are in progress and cannot know when pauses between messages occur. This could give rise to large variations in the SN_{rec} value if the transmitters stopped transmitting the code sequence during this time. Therefore, for the complete duration of the "conversation" between users, data of some sort must be transmitted. If a lull in data to be transmitted occurs, then idle bits of some form are coded up and transmitted over the channel using a sequence reserved only for this purpose. In this way, no other receiver is affected.

The values of L_s and L_n will vary with the particular combination of users transmitting and the position of the noise sources, both of which could alter during the duration of a "conversation". Therefore, these values cannot be calculated exactly, but could be estimated. In this way, a coarse setting for the threshold values can be made for the system at any given time, with the fine tuning being done by examining the output data. If higher level circuitry and software detects many errors in the received data, the threshold values are altered slightly until the bit error rate is optimized at its lowest rate. The correct value for f(p) can then be computed and stored for use in future coarse adjustments. (Note. The fine tuning mechanism is constantly used for the duration of the conversation and not just during channel setup).

4.3.2 Necessary Properties of Code sequence family.

From the previous sections, the necessary properties of low crosscorrelation and a high IOD were pointed out. In addition, for a useful system, the family size of the code sequence must be reasonably large, for a given sequence length. This is because the shorter the sequence is, the higher the possible data rate of the system. Another property that is important in this DSSS system is that the cummulative signal from multiple users approximates to White Gaussian Noise, an assumption made in the theory behind the performance of the receiver (ie mean value is equal for all frequencies), therefore a low sequence imbalance figure is required. Also, if the privacy of data transmitted over the media is to be provided by the code sequence, then it requires a large linear span.

Bent sequences provide the best overall performance in all of the above areas, being inferior only in their small family size. Therefore, for systems that do not require high data rates but expect that a large number of the available sequences will be used concurrently for most of the time, then the use of Bent sequences is seen to be the best choice. The large family size, but larger cross-correlation values exhibited by Gold Codes, seem to make these more suitable for systems requiring higher data rates between nodes which do not frequently all require to transmit concurrently. A compromise can be obtained by the use of m-sequences, which have the additional benefit of being the easiest family of sequences to generate.

4.3.3 Communications media requirements.

The requirement for high quality co-axial cable is determined by the "near-far" effect. This addresses the problem that a transmitter near to a given receiver may swamp the data transmitted to it from a user much further away. Therefore, the attenuation per unit length of the cable should be as small as possible and the cable length minimised. This problem could also be tackled by the use of active repeaters on the cable, but this reduces the media's reliability and some possible designs would increase the end-to-end propagation delay for the medium, degrading performance. The medium should also posses linear attenuation characteristics over the frequency range used for signal transmission.

4.4 References.

[4.1] Pickholtz RL, Schilling DL and Milstein LB -"Theory of Spread Spectrum Communication-A Tutorial"-*IEEE Transactions on Communications Special,* May 1982. (p855)

[4.2] Scholtz RA - "The Origins of Spread Spectrum Communications" - *IEEE Transactions on Communications Special,* May 1982. (p822)

[4.3] Scholtz RA - "Notes on Spread Spectrum History" - *IEEE Transactions on Communications,* January 1983. (p82)

[4.4] Price R - "Further Notes and Anecdotes on Spread Spectrum Origins" - *IEEE Transactions on Communications,* January 1983. (p85)

[4.5] Dixon RC - "Spread Spectrum Systems" - (book) Wiley Interscience, 1976.

[4.6] Cook C and Marsh HS - "An Introduction to Spread Spectrum" - *IEEE communications magazine,* March 1983. (p8)

[4.7] Maskara SL and Das J - "Concatenated Sequences for Spread Spectrum Systems" - *IEEE Transactions on Aerospace and Electronic Systems*, March 1981. (p342)

[4.8] Lin S and Costello DJ - "Error Control Coding: Fundamentals and Applications" - (book) Prentice-Hall 1983.

[4.9] Milston LB and Ragonetti RR - "Combination Sequences for Spread Spectrum communications" - *IEEE Transactions on Communications*, July 1972.

[4.10] Shaar AA - "A Study of Code Division Multiple Access with respect to fibre Optic Local Area Networks" PhD Thesis, University of Kent, April 1985.

[4.11] Smythe C - "Direct Sequence Spread Spectrum Techniques in Local Area Networks" - PhD Thesis, University of Durham, xxxx 1985.

[4.12] Lin D - "The Generation and Comparison of Linear and Nonlinear Binary Sequencs for Spread Spectrum Communications" - Project Report, University of Durham, April 1984.

[4.13] Judge WJ - "Multiplexing using Quasi-orthogonal Binary Functions" - Reprint, Spread Spectrum Techniques, Ed R.C Dixon, IEEE Press, 1976.

Spread Spectrum techniques for multiple access to medium.

[4.14] Ismail NA - "Application of Bit-Slice Microprocessors to Digital Correlation in Spread Spectrum Communication Systems" - PhD Thesis, University of Durham, 1982.

[4.15] Scott M - "xxxxxxxx" - PhD Thesis, University of Durham, xxx.

[4.16] Freeman JJ - "The Action of Dither in a Polarity Coincidence Correlator" - Reprint, Spread Spectrum Techniques, Ed R.C Dixon, IEEE Press,1976. (p56)

Chapter Five
CDMA Testbed System

5.1 Introduction.

5.1.1 The need for a Test-bed system

Since the use of Direct Sequence Spread Spectrum in LANs to obtain CDMA is a new idea, no data was available at the start of the project to enable parameters associated with the OSI Physical Layer to be set with any degree of confidence. In order to get such data relating to a large scale system without building one and then continuously modifying it to vary the parameters, a simulation of the system was made [4.11]. Since assumptions had to be made during its writing, comparison of its results with a small-scale hardware system was seen to be a suitable method of checking their validity. Suitable refinements to the simulator could then be made which would then be used with greater confidence in the simulation of large scale systems.

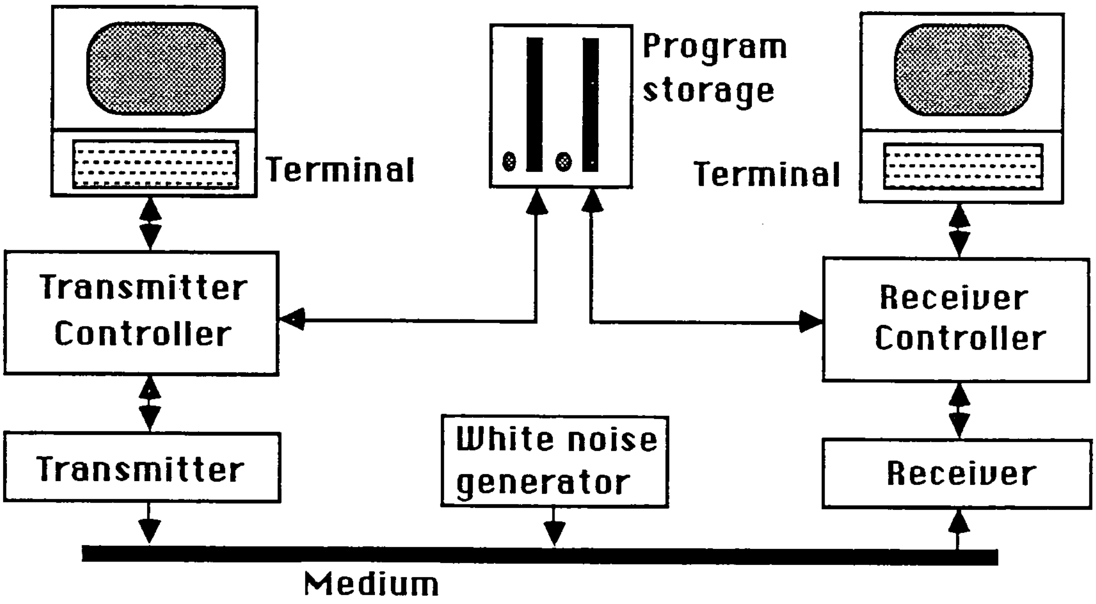


Fig 5.1.1 CDMA Testbed System.

The small-scale hardware system to be built (the CDMA Testbed) was to consist of a single Spread Spectrum (SS) transmitter and a single SS-receiver communicating over a medium of some sort onto which white noise was also transmitted to simulate the presence of many other users (see Section 4.3.2 on the cumulative properties of SS signals). The communicating hardware was functionaly divided into "dumb" coding and decoding/selection units and control units which would supply the former with all the necessary PN sequence information, the test data and run the necessary system tests. Figure 5.1.1 illustrates the topology of the test-bed system.

Production of the prototype Transmitter/Receiver hardware was undertaken elsewhere [4.15], the work covered in this chapter being that involved in the production of the Transmitter and Receiver controllers along with the additional software needed to run the system tests.

5.1.2 Aims for Testbed System.

Since the aims for the system are closely associated with those for the simulator, the latter's will be outlined. The major parameter under investigation was the bit-error-rate and how this was affected by variations in the signal/noise ratio at the transmitter and the distance between the Tx/Rx pair. Also, an investigation into the "near-far" effect was made (ie the signal from nearby transmitters would have a greater amplitude than that from a more distant one) to determine what conditions would result in the non-recoverability of a signal by a receiver for a given value of the parameters discussed previously. A further investigation was made into how variations of the "Data Definition Factor" (the number of simultaneous users divided by the code sequence length in use) affected network performance w.r.t data throughput and bit-error-rate.

The major assumptions made by the simulator was that the medium over which the communications took place was perfect and had an infinite bandwidth, thus being able to carry a perfect digital signal. In practice, since the data would be transmitted at a very high frequency (approx 10MHz) then the signal's shape would be distorted and be non-uniformly attenuated due to the non-uniform properties of the various frequency components of which it is composed.

Therefore the expected differences in the results from the testbed and simulator would indicate what modifications were necessary for the latter to model the signals travelling along the medium. The parameter most needed to be measured by the test system for comparison was hence the bit-error-rate for a variety of sequences of differing lengths over media containing differing noise levels for varying distances between the Tx/Rx pair.

5.2 Transmitter/Receiver Controller Overview.

5.2.1 Choice of Hardware/Software

Since the passing of data between the Tx and Rx would be of the order of 1Mbit/s, the microprocessor used in the two controllers had to be of the 16/32-bit generation. Of these, the only processor for which adequate hardware and software support was envisioned to be easily available for the period before the end of the work was the MC68000. Access to two makes of design module using this processor was possible, Motorola's own "MEX68KDM" board and Force Computer's "Profi-kit". Whilst the latter offered a larger on-board RAM and an on-board assembler/disassembler in ROM (both of the boards contained a monitor in ROM), it contained only a single PIA (Peripheral Interface Adaptor) whilst the KDM had two. Since these adaptors were to be used for the interface between the controllers and the Tx/Rx hardware, use of more than one of these was seen to be desirable.

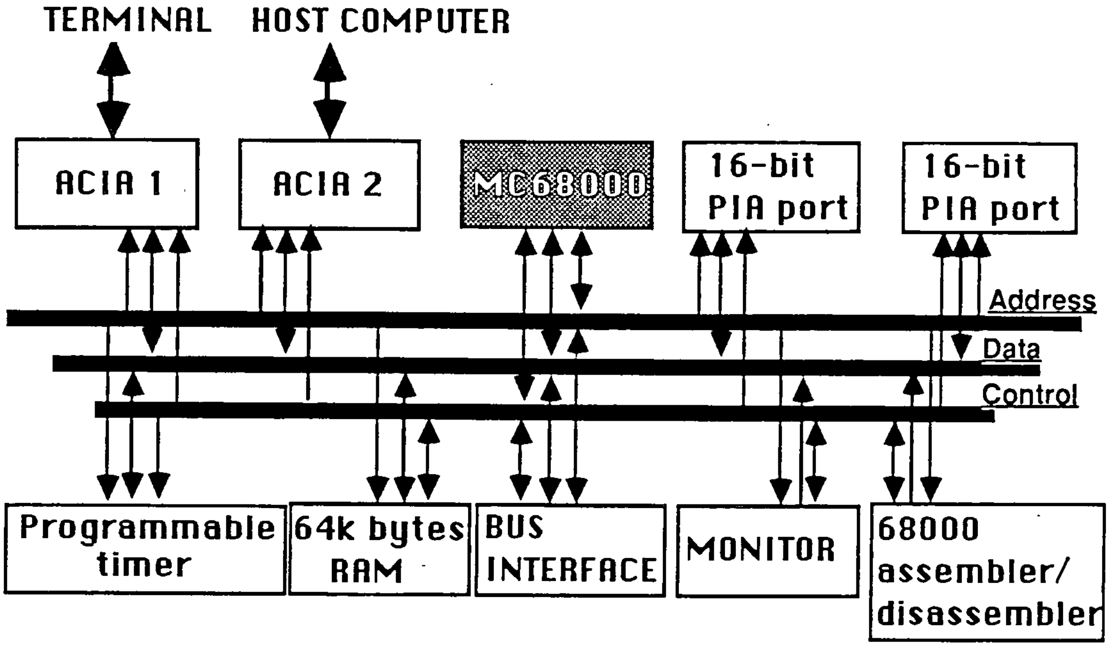


Fig 5.2.1 Modified KDM Functional Block Diagram.

Although the KDM lacked its own assembler/disassembler, one was available on the University mainframe computer which was designed for use with the board. However, due to difficulties experienced using this remote system, an on-board ass'/dis' was later added along with a modified monitor. Since the source program was now written on the board itself, the board's memory was deemed inadequate and so was doubled in size. These modifications involved some re-wiring of the board itself, replacement of some passive components and the modification of its address decoding PROMs. A functional block diagram of the modified KDM board is given in Figure 5.2.1. This figure shows the presence of a backplane bus interface on the board. This was seen to be useful for future work when more than one controller would be under the control of another processor in increasingly sophisticated testbed systems.

The software used in the controllers was required to be fast in certain areas such as data transfer and so these portions had to be written in a low level language. Initially, the idea was to write the code in the "C" language on a 68000 Unix system, transfer the compiled code to the KDM board and then optimize it if necessary using the assembler/disassembler. Due to the chronic unreliability of the Unix system, however, all of the code had to be written in assembler code, thus drastically increasing the necessary amount of work required to complete a suite of debugged software.

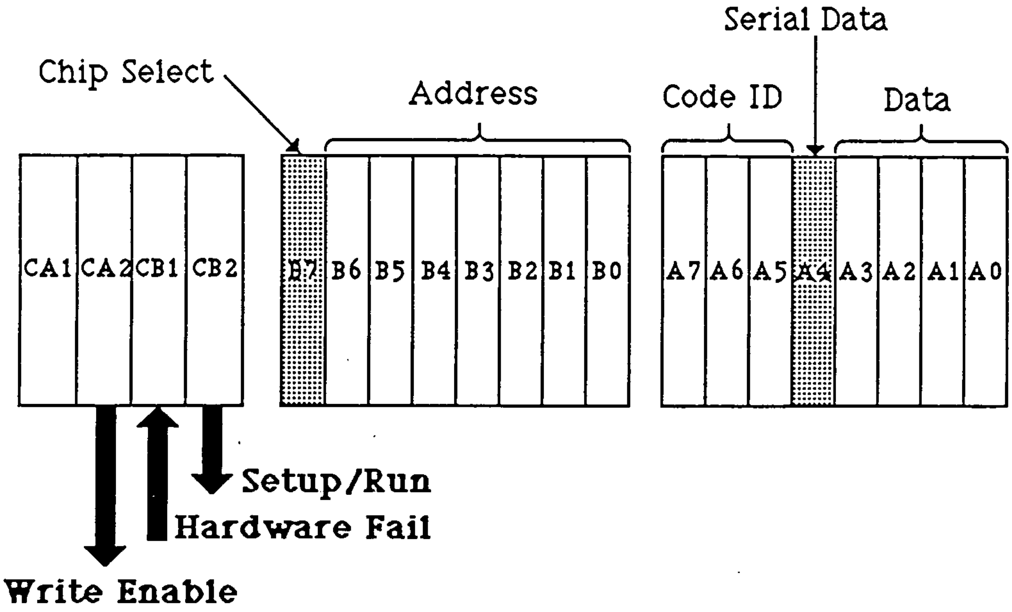


Fig 5.2.2 Data arrangement in PIA1 in the transmitter controller.

5.2.2 Controller to Tx/Rx Interfaces.

As mentioned in the previous section, these interfaces were to be implemented using the KDM's PIAs. It was decided to identify two modes of operation of the interfaces, "Setup" and "Run". The former mode is entered when PN sequence information is loaded into the hardware, whilst the latter is entered when data is to be transmitted to or received from it. For the first testbed system it was decided to use one of the PIAs to transfer data associated with each mode. PIA1 was to be used to transfer the PN sequence data, whilst PIA2 would be used for data transfer.

Consider first the case of PIA1, which is the same for both the Transmitter and Receiver Controllers. Figure 5.2.2 shows how the data was to be arranged in PIA1 before transmission to the hardware. Three bits were reserved for "Code ID" (A5-A7) because sufficient memory was to be made available in the hardware to store eight PN sequences of up to one hundred and twenty-seven bits in length. Four bits were allocated for "Data" although this is simply a one bit value. The actual data bit value is carried in bit A0 , the result of the modulo-2 addition of the generator tap contents in A1, whilst A3 contains a one only when the data bit is the first in the PN sequence. A2 is used in the same way but to signal the final bit in the sequence.

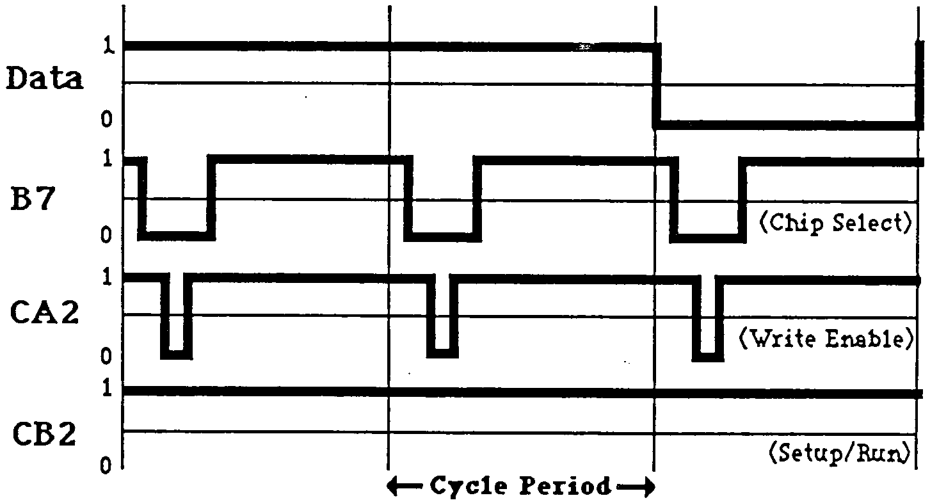


Fig 5.2.3 Setup Mode Transaction Timing.

An address field is included because the data is not stored sequentially in the hardware's memory. Although each code is stored within a block of memory one hundred and twenty-seven bits long (identified by the "Code ID" which forms the most significant three bits of the address), within these blocks it is scattered in a manner determined by a combination of the address presently accessed and its associated value of the modulo-2 addition. In this way, the non-consistent memory access times caused by the necessary resetting of a counter used in the sequential addressing method was avoided. Since consistent access to the next sequence bit is necessary when coding the data, the addressing method used (which automatically resets itself-see section 5.3.3.b) allowed higher data rates to be achievable. Thus an address field was necessary so that the correct data was located in the correct memory address. The "Chip Select" bit and the "Write Enable" control line were used to synchronise the loading of the data to the hardware whilst the "Setup/Run" control line was used for mode identification. The "Serial Data" bit was not used in the first system but was reserved for possible future use. The "Hardware Fail" line signals if the power to the Tx/Rx hardware has been cut off, if so an error message was required to be sent to the user terminal.

The timing of this data transaction is shown in fig 5.2.3. In this case, no "handshaking" occurs across the interface since the hardware's response time was specified and was easily fast enough to accept the data sent to it. Firstly, the new data word was placed into the PIA and then subsequently B7 and CA2 were sent from high to low (in that order) to select the appropriate chip and signal that the hardware was now allowed to read the new word. These then return high ready for the next cycle. CB2 is set high to signal setup mode.

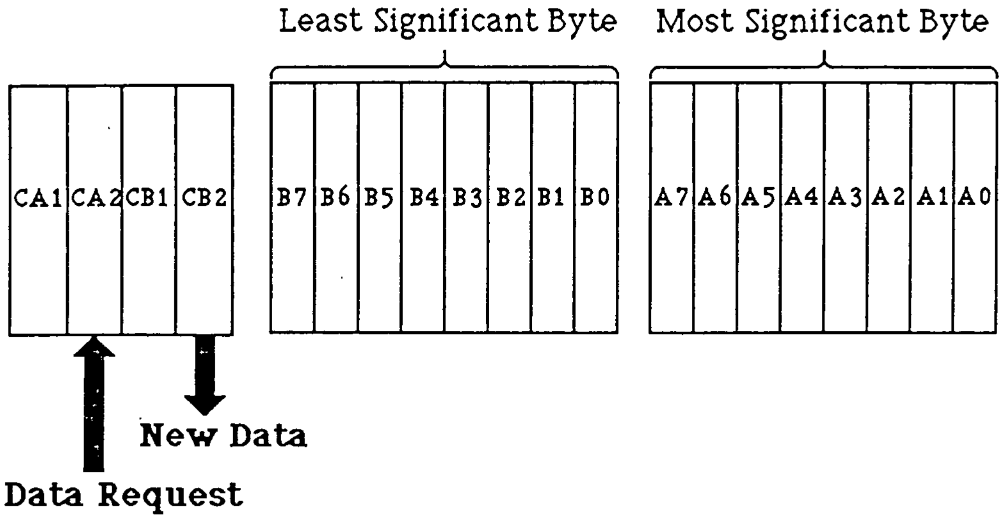


Fig 5.2.4 Data arrangement in PIA2 in the Transmitter Controller.

Figure 5.2.4 shows how the data is organised within PIA2 in the Transmitter Controller before transmission to the hardware. Bits A0-A7 and B0-B7 contain the most and least significant bytes of the data word respectively. The "Data Request" line is used by the hardware to signal that it is ready to accept another data word whilst the "New Data" line signals that the new word has been placed into the PIA and can be read by the transmitter hardware.

Figure 5.2.5 shows PIA2's required data on reception from the receiver hardware. The only difference between this and the case for the transmitter is in the direction of the control lines. In this case, "New Data" signals the arrival of a new data word over the network to be received. "Data Request" signals the readiness of the controller to accept another new word, enabling the receiver hardware to clear the old word to make way for the new.

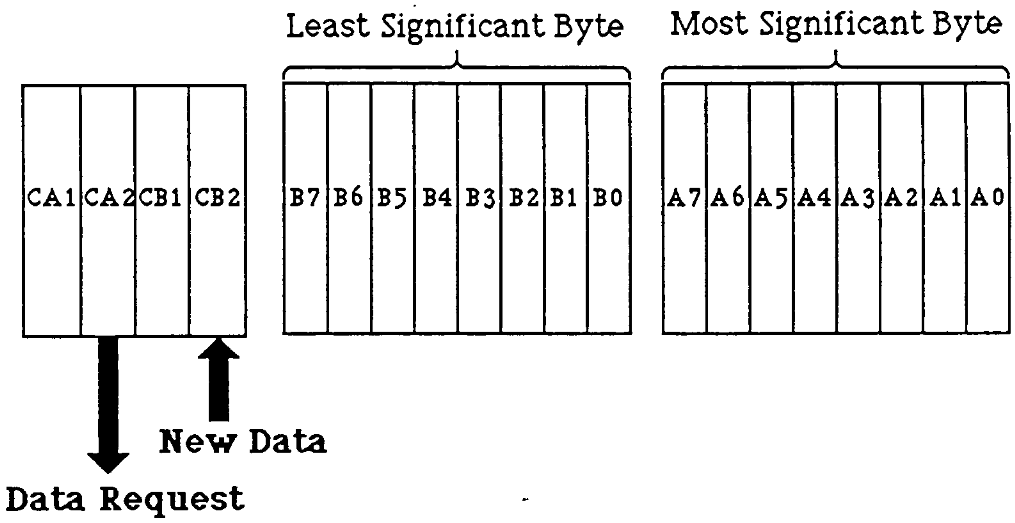


Fig 5.2.5 Data arrangement required from receiver hardware.

Figure 5.2.6 shows the timing of the data transactions over these interfaces for the transmitter/receiver, but the sequence is interpreted differently for each. In the case of the transmitter, the sequence begins with the reception of the signal on C~~B~~A2. This causes an interrupt and the next word is loaded into the PIA. As soon as it is loaded in, the signal on CB2 is sent to signal the presence of the new word. Therefore the transmission rate is decided by the transmitter hardware. In the case of the receiver, it waits for a signal on CB2, reads the new word after receiving the signal and then sends a signal on CA2 after it has stored this word in memory. Therefore the reception rate is governed by the receiver hardware. CB2 is held low by the controller signalling the data transmission mode.

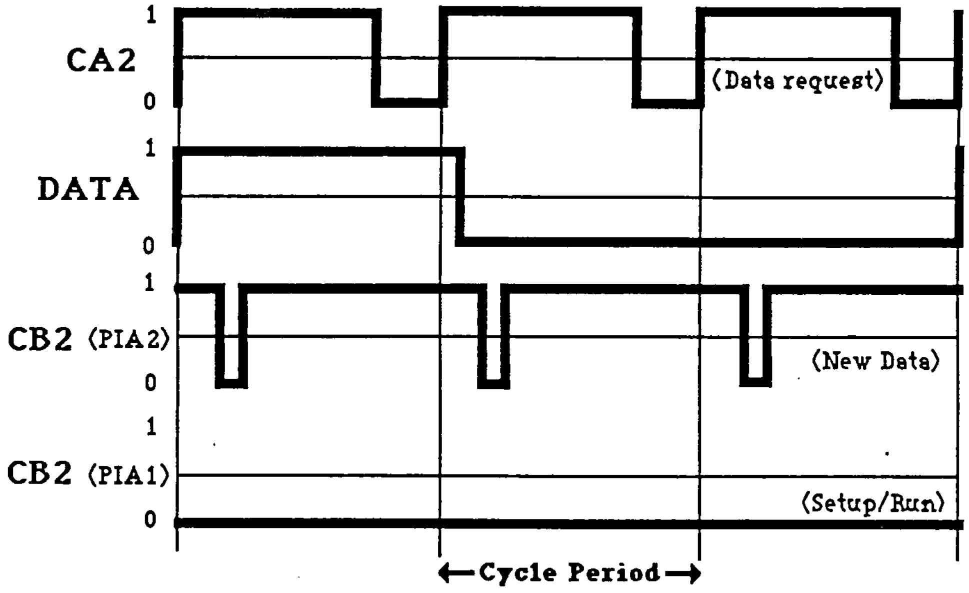


Fig 5.2.6 Timing of the data transactions during data transfer.

5.3 Transmitter/Receiver Controller Software

5.3.1 Software Design Philosophy

The major features of the software written were its modularity and ease of use. The former feature was important for the following reasons. Firstly, the upgrading or modification of parts of the program was made easier since all changes were localised within a fixed number of routines. Secondly, since many of the functions required by the Tx and Rx controllers were the same, these could share common routines, saving time in the production of the software. Thirdly, since some of the functions met in the test-bed system were also present in the proposed full-scale system, some of the real-time routines written could be used in the latter, again saving time. The only portions of code which were not made relocatable in the software were the tables of data, which were kept at constant address values in order to make their examination easier for debugging purposes. Since they were all given labels, however, these could also be made relocatable by altering a single line of code.

User-friendliness was considered to be an important feature because the amount of similar data to be entered was large and could therefore be confused. Also, the testbed system was envisaged to be used by a number of people who may not know the internal workings of the software and who therefore had to be aided by good documentation and a user-friendly system.

An analysis of the software follows, in which some of the routines are looked at in detail whilst the more routine ones are noted in a more superficial way. Two appendices are also referenced, Appendix ~~X~~ C&D being a listing of the assembler code ~~and Appendix X being an example illustration of the user interface~~.

5.3.2 Functional Breakdown of Software

Figures 5.3.1 and 5.3.2 show the functional make-up of the software for the transmitter and receiver respectively in the form of a high-level flow-chart. It can be seen that both are menu-driven at this level in a similar way and contain some common routines. The major routine "Main" is present in both and is responsible for the input, generation and storage of the PN sequences to be used in the tests (see section5.3.3). Some of the routines present on the figures will not be examined in detail and so a brief mention of their functions follows. "Which" sets the code ID bits in PIA1 to the code selected for use in the test, whilst "Setup" and "Run" set and clear CB2 (PIA1) respectively to indicate the mode of operation to the Tx/Rx hardware. The routine "PIAS sets up the PIAs present on the boards and are different in the Rx and Tx cases only in the values placed into the PIA control registers (it was seen to be more efficient to change 3 lines of code within this board specific routine than to call external labelled values). The messages output are to the user's terminal and are included to increase user-friendliness.

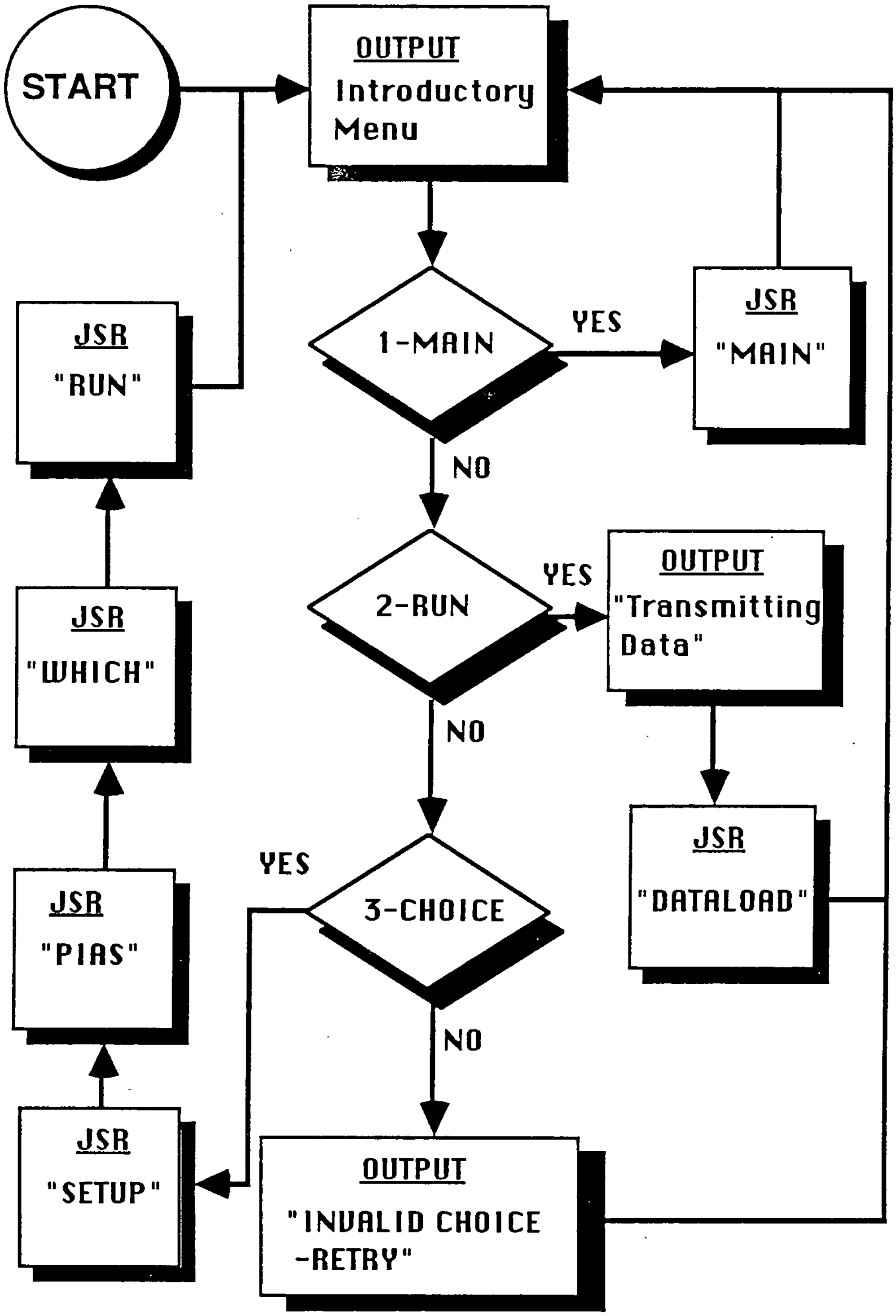


Fig 5.3.1 Transmitter Controller Software Overview.

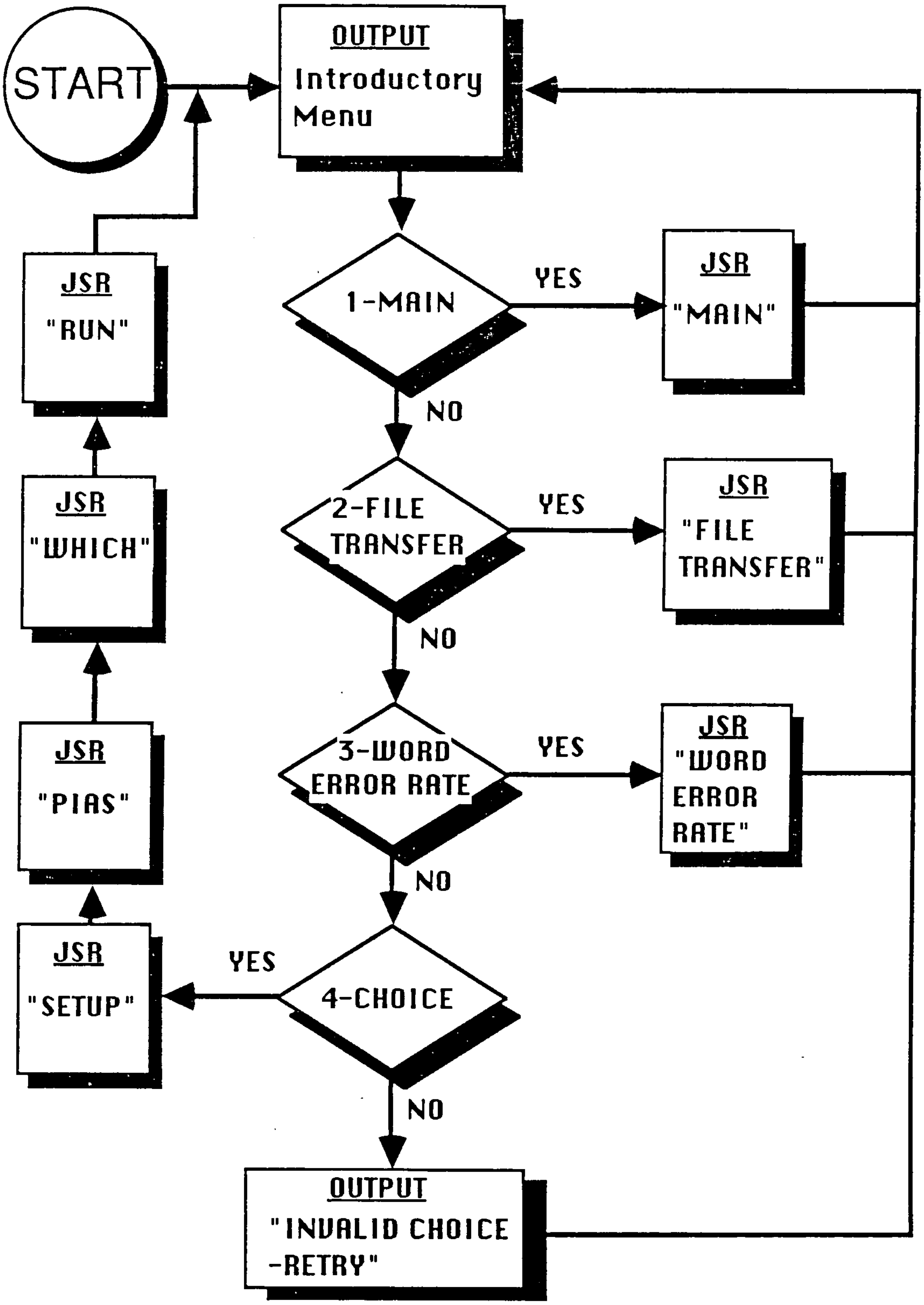


Fig 5.3.2 Receiver Controller Software Overview.

5.3.3 "Main" Subroutine.

a-Introduction

The function of this routine (see Fig 5.3.3) was outlined in the previous section. It is split into three major sections. Firstly, the sequences are entered and stored in the controller's RAM as a pool of codes for future use. Secondly, eight of these codes are selected to be loaded into the controllers associated Tx/Rx hardware and lastly they are loaded into it.

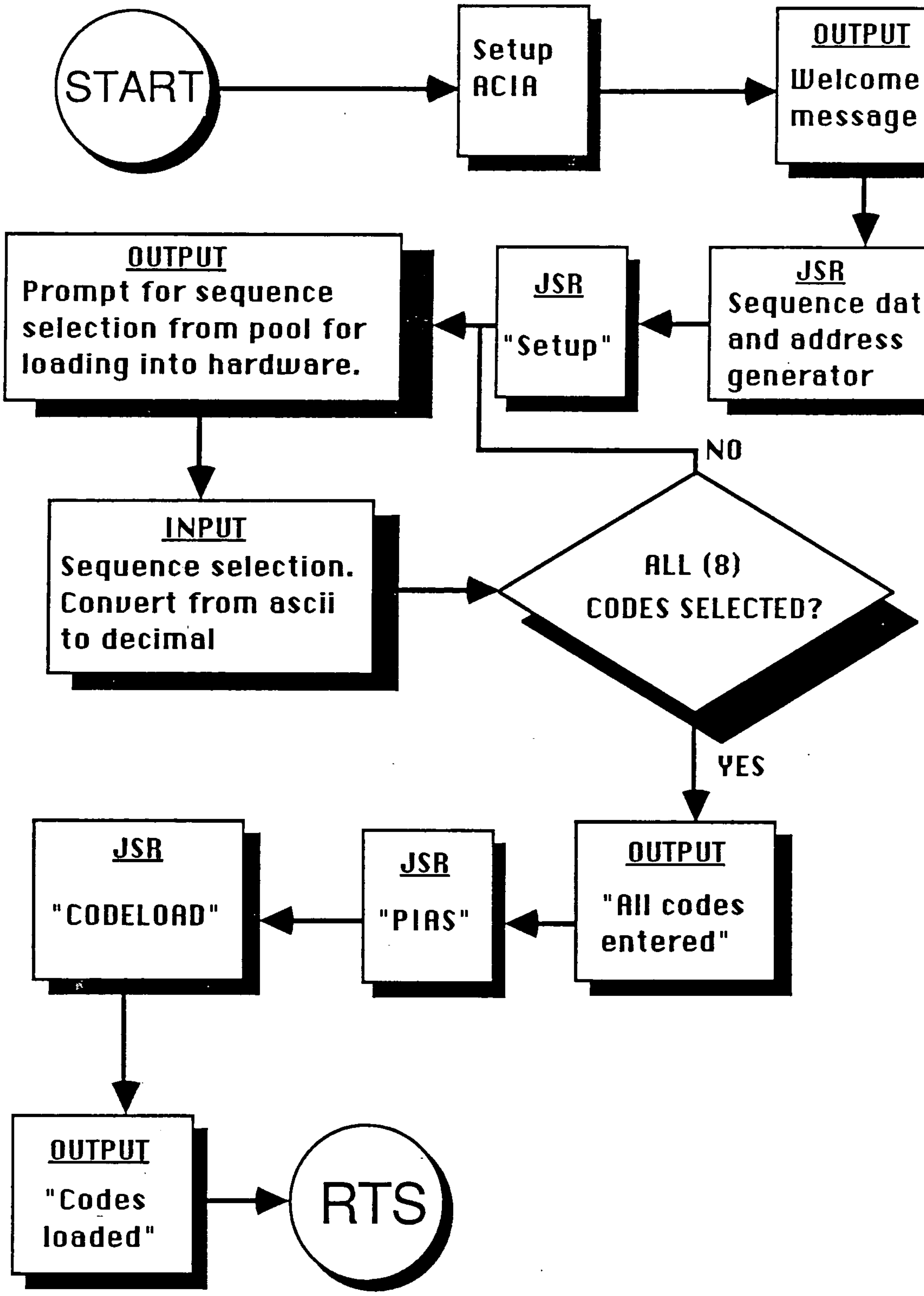


Figure 5.3.3 "Main" subroutine

The reselection of a new set of eight codes from the pool is carried out at present by re-entering "Main", immediately passing through the first section to the following ones. The first section was written in such a way as to be easily seperable from the others, but it was found to be more convenient to combine them within the same menu selection in this case. A more detailed look at the routines within this one is now made.

b-"Code (sequence) data and address generator" subroutine

The two halves of this routine are illustrated by the flow charts (Fig 5.3.4 and 5.3.6) and are concerned with the input of the required data and generation and storage of the sequences respectively. As shown by Fig 5.3.4, the first data required from the system user is the identity that the code is to have in the pool of codes held within the controller's memory. This pool consists of a block of memory which is segmented into blocks of the same number of bytes as the maximum allowed number of bits in the generated sequence. This maximum length of sequence is stored within the variable "Length" and in the case of our system was dictated, by the limited size of the Tx/Rx hardware RAM, to be 128 bits (127-bit sequence plus a "safety-byte"). Code sequences of all lengths were allocated blocks of memory of this size (as they were to be in the Tx/Rx hardware), the excess bytes being filled with a reset byte (both A3 and A2 are set) so that if one of these locations were accessed (after power-up perhaps) the hardware would recover in a rapid pre-determined manner (ie the generator would reset itself to the first value of the sequence in its memory).

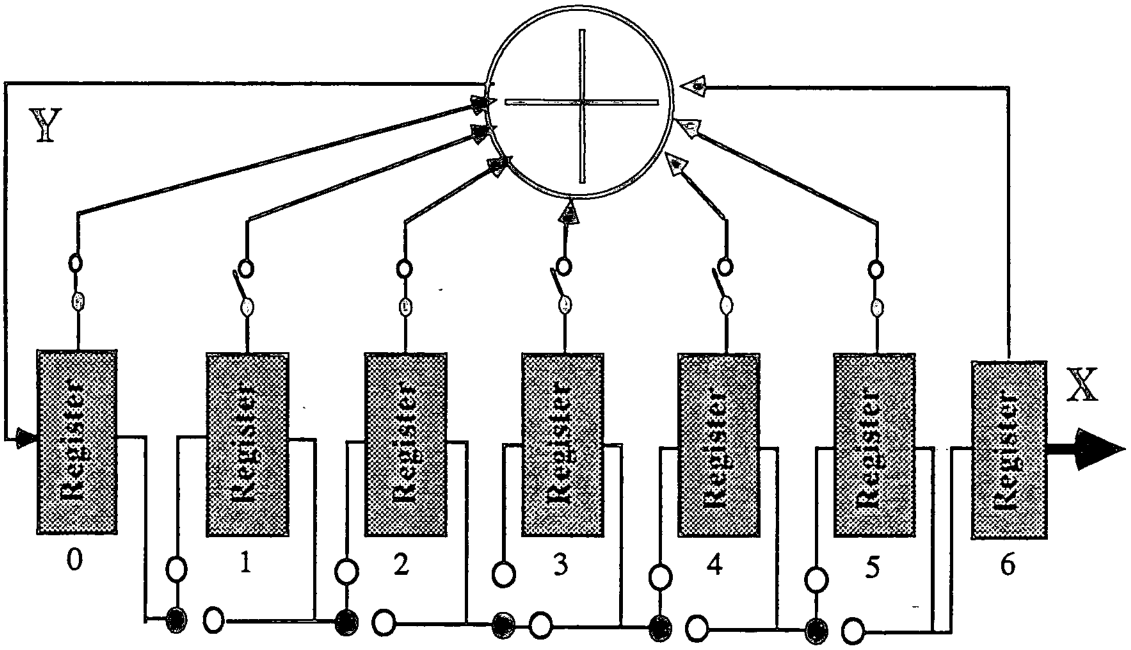


Fig 5.3.5 Adaptable Shift Register Generator used in Testbed System

The second set of data to be entered into the system concerned the location of the feedback taps in the generator (see section 4.1). The form of the generator required by the system is depicted in Fig 5.3.5. This depicts a generator which is selectable in length (up to seven shift-registers giving a maximum sequence length of 127) with individually selectable feedback

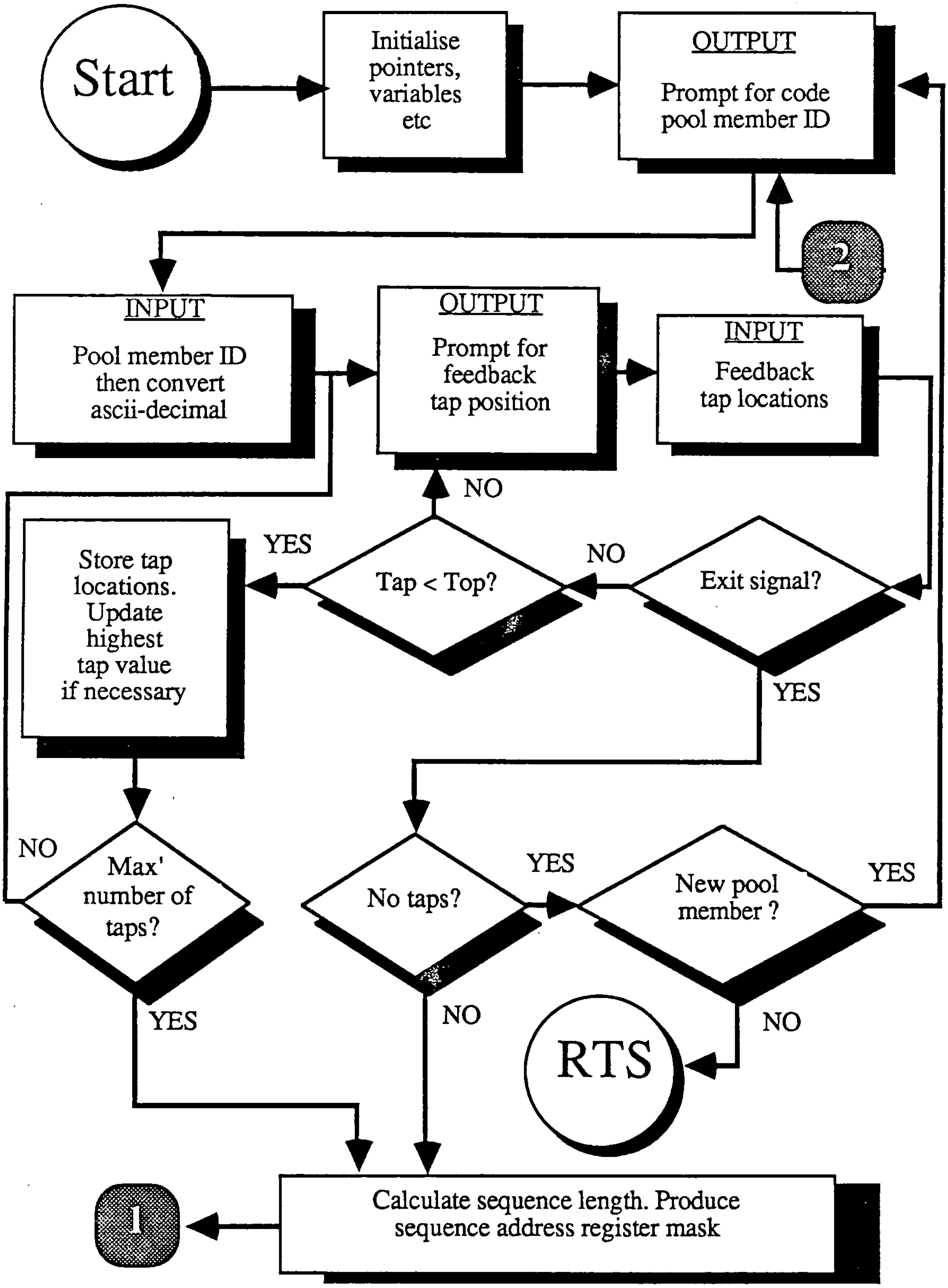

Start
Initialise pointers, variables etc
OUTPUT
Prompt for code pool member ID
2
INPUT
Pool member ID then convert ascii-decimal
OUTPUT
Prompt for feedback tap position
INPUT
Feedback tap locations
NO
YES
Store tap locations. Update highest tap value if necessary
Tap < Top?
NO
Exit signal?
YES
NO
Max' number of taps?
No taps?
YES
New pool member ?
YES
YES
NO
NO
RTS
Calculate sequence length. Produce sequence address register mask
1

Figure 5.3.4 Input stage for sequence generation routine

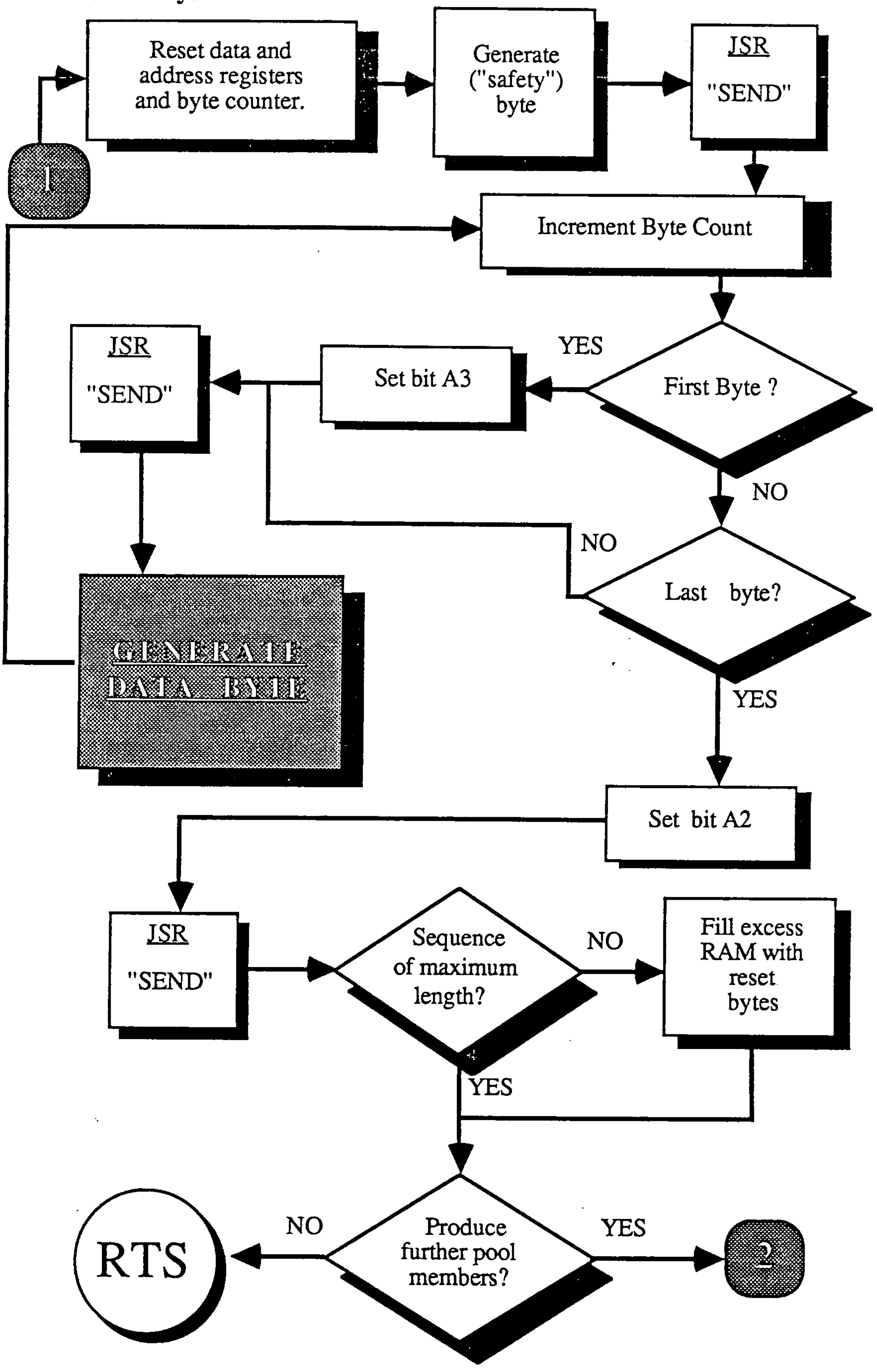


Fig 5.3.5 Generation stage of sequence generator routine

taps. The illustrated example is of a six-stage register with taps from registers 0,2,5 and 6. A modular shift register generator was not used since although it is the fastest in hardware, due to reduced propagation delays through it, it would be less efficient when modelled in software due to the larger number of modulo-2 adders involved (which would have to be updated sequentially rather than concurrently in hardware).

"Safety Byte"

First byte in sequence

Register & address value	X	Y	Data
0000000	0	1	0010
0000001	0	0	1000
0000010	0	1	0010
0000101	0	1	0010
0001011	0	0	0000
0010110	0	0	0000
0101100	0	0	0000
1011000	1	0	0001
0110000	0	0	0000
1100000	1	1	0011
1000001	1	1	0011
0000011	0	1	0010
⋮	⋮	⋮	⋮
0010000	0	0	0000
0100000	0	0	0000
1000000	1	1	0111

Last byte in sequence

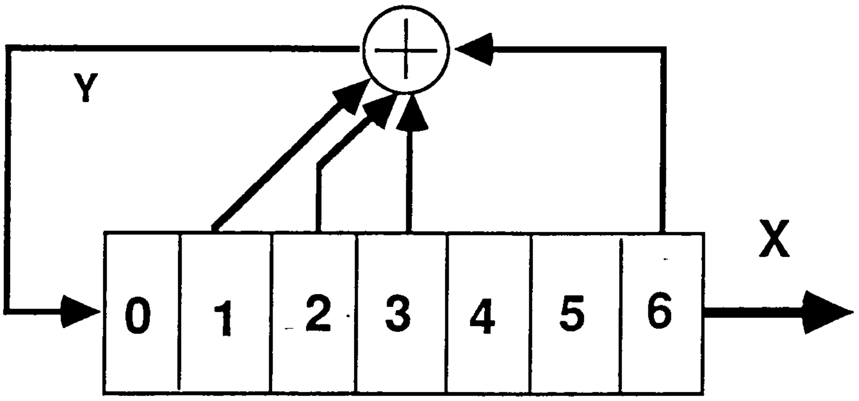


Fig 5.3.7 Example of sequence data with associated generator config'.

In the case of our system, the upper limit of the number of shift registers in the generator was 7 , resulting in the highest value for a tap being 6, a test being made to check that this was not exceeded. Also, since no more than 6 taps could be used, no more than this number could be entered. This phase of data entry is left in one of two ways. Either after 6 taps have been entered or after an exit signal is typed in. In the case of the latter, the specification is checked for the null case (no taps) and the option of re-entry offered if this is so. If the null case was not present, then the program proceeds in the same way as after the former case. Using the highest value of the feedback tap data entered, the sequence length and the code's associated address register mask are calculated. The former is needed to calculate the number of reset bytes to be entered into memory whilst the latter is needed to model the variable length generator used.

The section of code illustrated by Fig 5.3.6 is entered after this point. The "safety byte" is generated in order to recover from the case where the sequence generator contains all zeros, a possibility on power-up or after a transient power fault. It is illustrated in Fig 5.3.7, the first row in the table. It can be seen that it is not included in the cyclic 127-bit PN sequence generated in this example, but will lead into it after the addition of its associated "Y" value to its address, as would occur in the Tx/Rx hardware. Therefore, bit A3 is not set until the following byte is generated.

The subroutine "send" stores the address and data values in the appropriate tables within the pool of such sequences. The shaded box encompasses the functions related to the modelling of the generator, enabling the generation of results such as those shown in Fig 5.3.7. The test of the sequence length is made to find out how many reset bytes need to be generated (if any) to fill the allocated memory block. These bytes are given sequential addresses since the "scattered" bytes fill up a complete segment of the total (number of shift registers used is smaller so the addresses cannot be larger than a fixed maximum).

c- "Codeload" subroutine

The second and third sections of "Main", the selection and loading of codes, proceeds after the generation subroutine. The codes required are selected, placed onto a stack and passed in this way to "Codeload" (see fig 5.3.8). This series of IDs is converted to the base addresses for each code sequence's data and address tables and again stored on a stack. The assembly of the data word is then undertaken for each data bit for each sequence and the control sequence followed as outlined in Figs 5.2.2 and 5.2.3. No handshaking was needed for this data transfer due to the low response time specified for the hardware.

5.3.3 "Dataload" subroutine.

This transmitter controller routine sends data words from a table of test data held in the controller's memory, to the transmitter hardware for the purpose of the tests being made at the

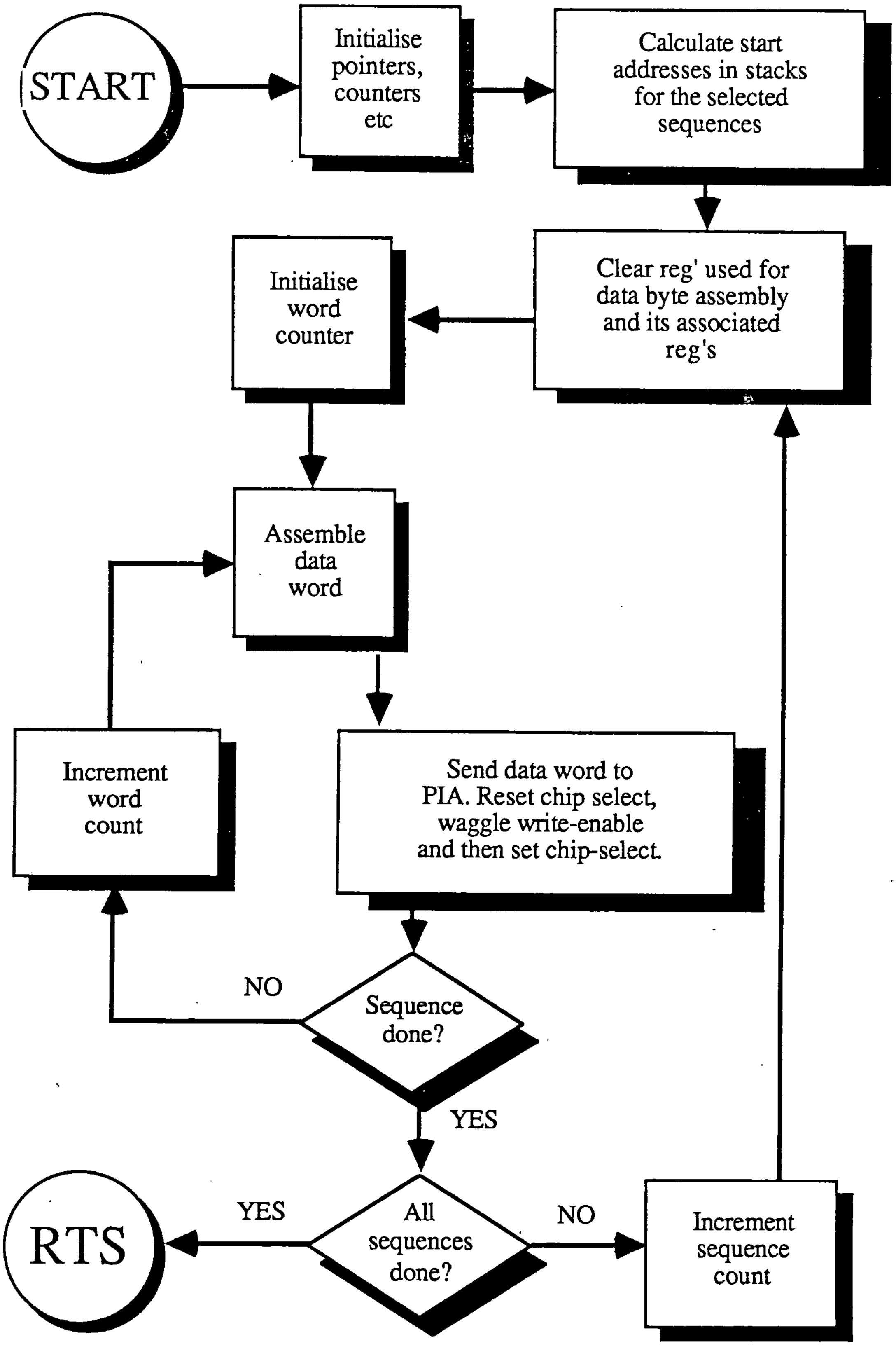

START
Initialise pointers, counters etc
Calculate start addresses in stacks for the selected sequences
Clear reg' used for data byte assembly and its associated reg's
Initialise word counter
Assemble data word
Send data word to PIA. Reset chip select, waggle write-enable and then set chip-select.
Sequence done?
NO
Increment word count
YES
All sequences done?
YES
RTS
NO
Increment sequence count

Fig 5.3.8 "Codeload" subroutine

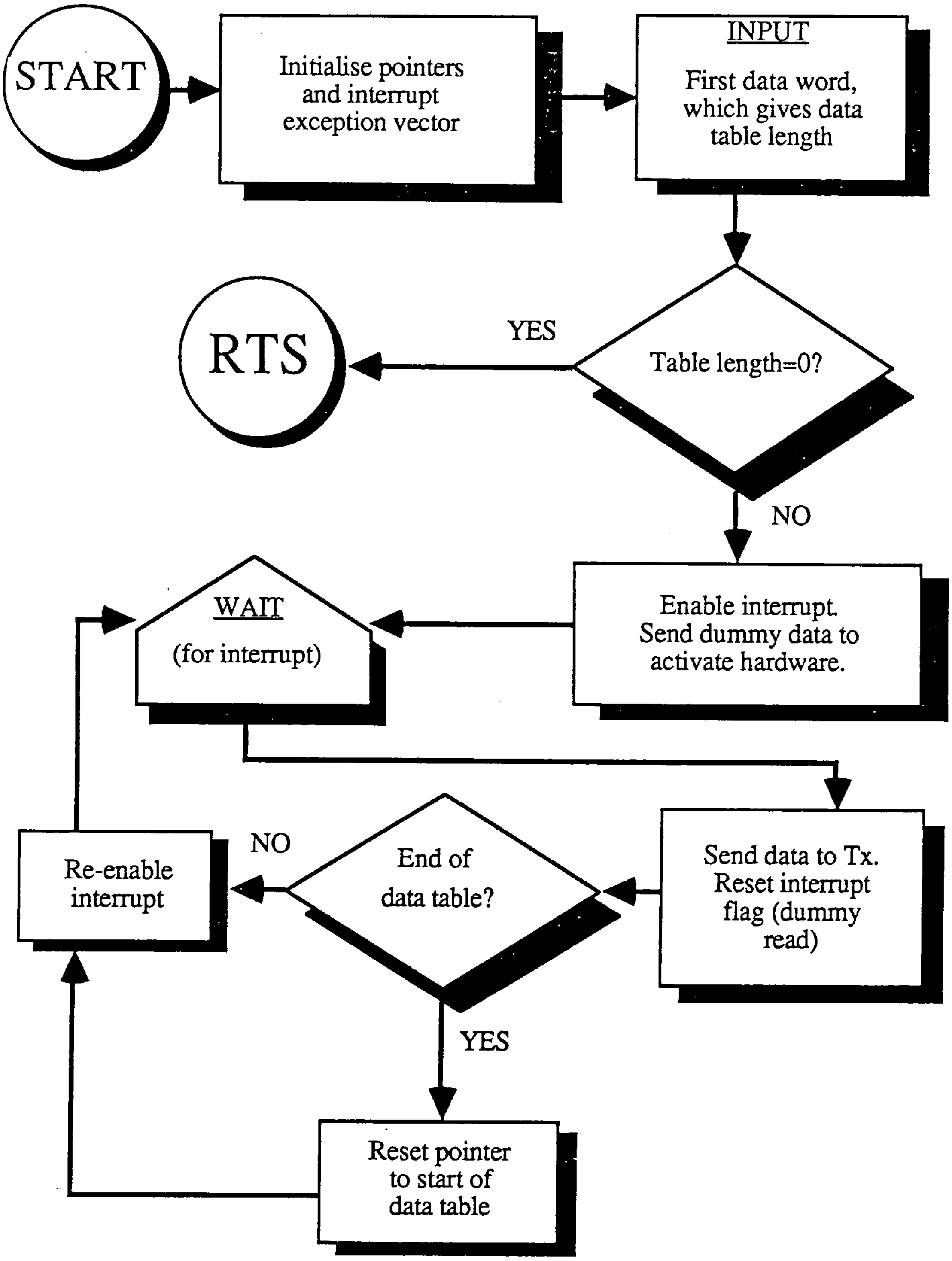

START
Initialise pointers and interrupt exception vector
INPUT
First data word, which gives data table length
Table length=0?
YES
RTS
NO
Enable interrupt. Send dummy data to activate hardware.
WAIT
(for interrupt)
Send data to Tx. Reset interrupt flag (dummy read)
End of data table?
NO
Re-enable interrupt
YES
Reset pointer to start of data table

Fig 5.3.9 "Dataload" subroutine

receiver (see Fig 5.3.9). The first data word in the table gives its length in words, this is checked for the null case and if this is not found it is then loaded into a register for future use. The interrupt used to request more data to be sent to the Tx hardware is enabled (this was configured to be a level four interrupt by suitable wire-wrapping on the KDM board) and a set of dummy data is sent to activate the transmitter (ie this causes a pulse on CB2 to be sent). The routine then waits for the next data request. On its arrival, it sends the data to its PIA and then resets the interrupt flag, thus making sure that each word is placed there before the next request is made (ie the Tx cannot tell if it has read the same word twice or not). The word sent is checked for its being the last one in the data table and the pointer to the next correct word updated. For the purposes of the tests required, this routine was left by means of the controller reset button thus enabling long term tests to occur without needing to test for large numbers of loops around the test, slowing the routine down.

5.3.4 "File Transfer" subroutine.

This receiver controller routine is used with a particular data table at the transmitter. The routine reads in characters from its PIA, checking for the start-of-file character. After reception of this, the following characters are stored within the controller's memory until the end-of-file character is also received (see Fig 5.3.10).

5.3.5 Word Error Count subroutine.

In order to find the error rata between Tx/Rx pairs, a file containing just the "correct" character for the test was used in the transmitter. As long as this character was received, error free transfer was confirmed, but when any other was encountered , an error was logged. The length of each test is selectable as is the number of times this test is repeated. When each test is complete, the error count is stored on a stack and these are all printed out after all of the tests are completed (se Figure 5.3.11). For longer term tests, a further routine was written which outputted the results as soon as they were available, thus requiring less memory to store the results. However, the printing of the results onto the terminal slowed the routine down, and so characters could be missed.

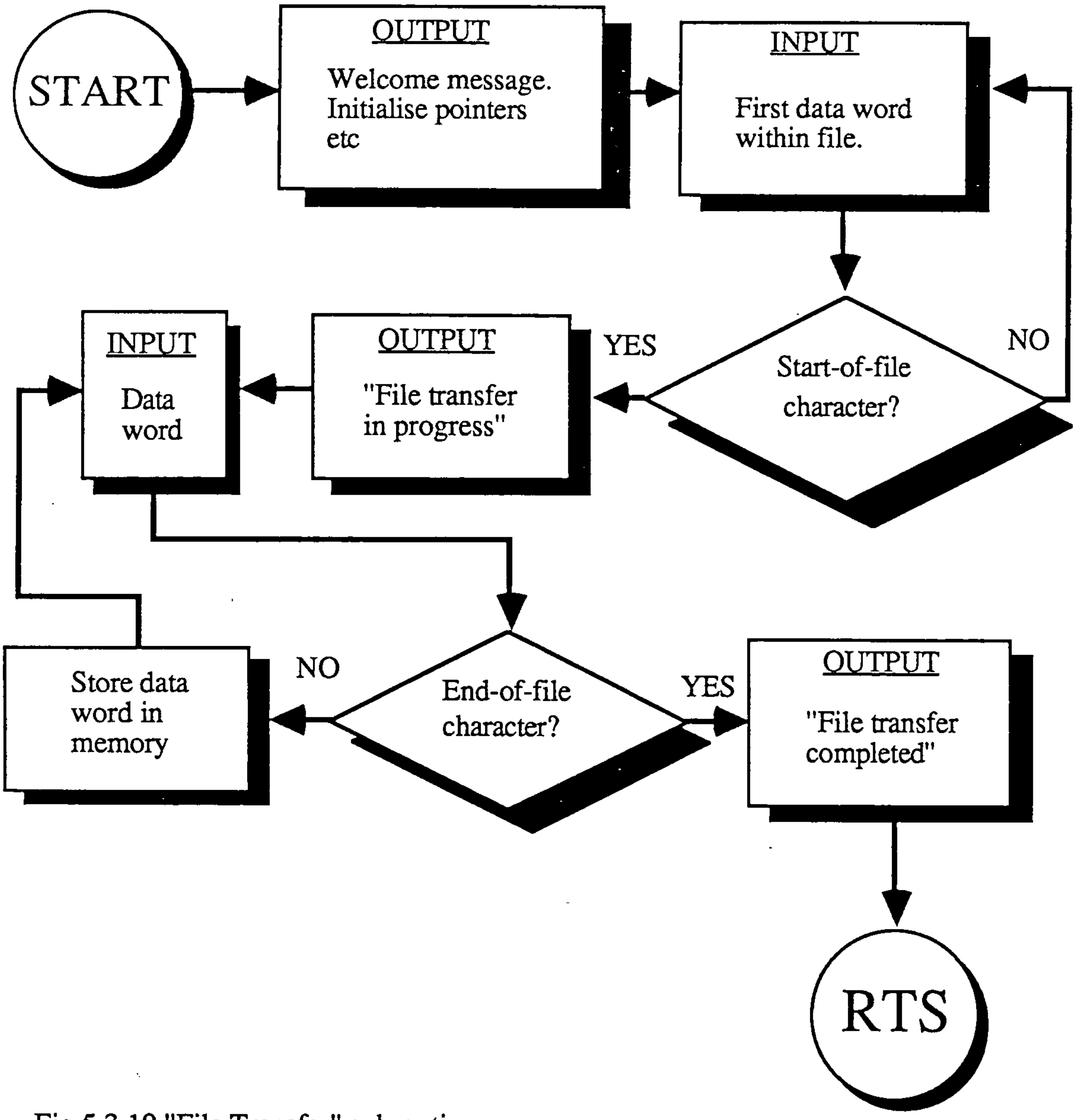


Fig 5.3.10 "File Transfer" subroutine

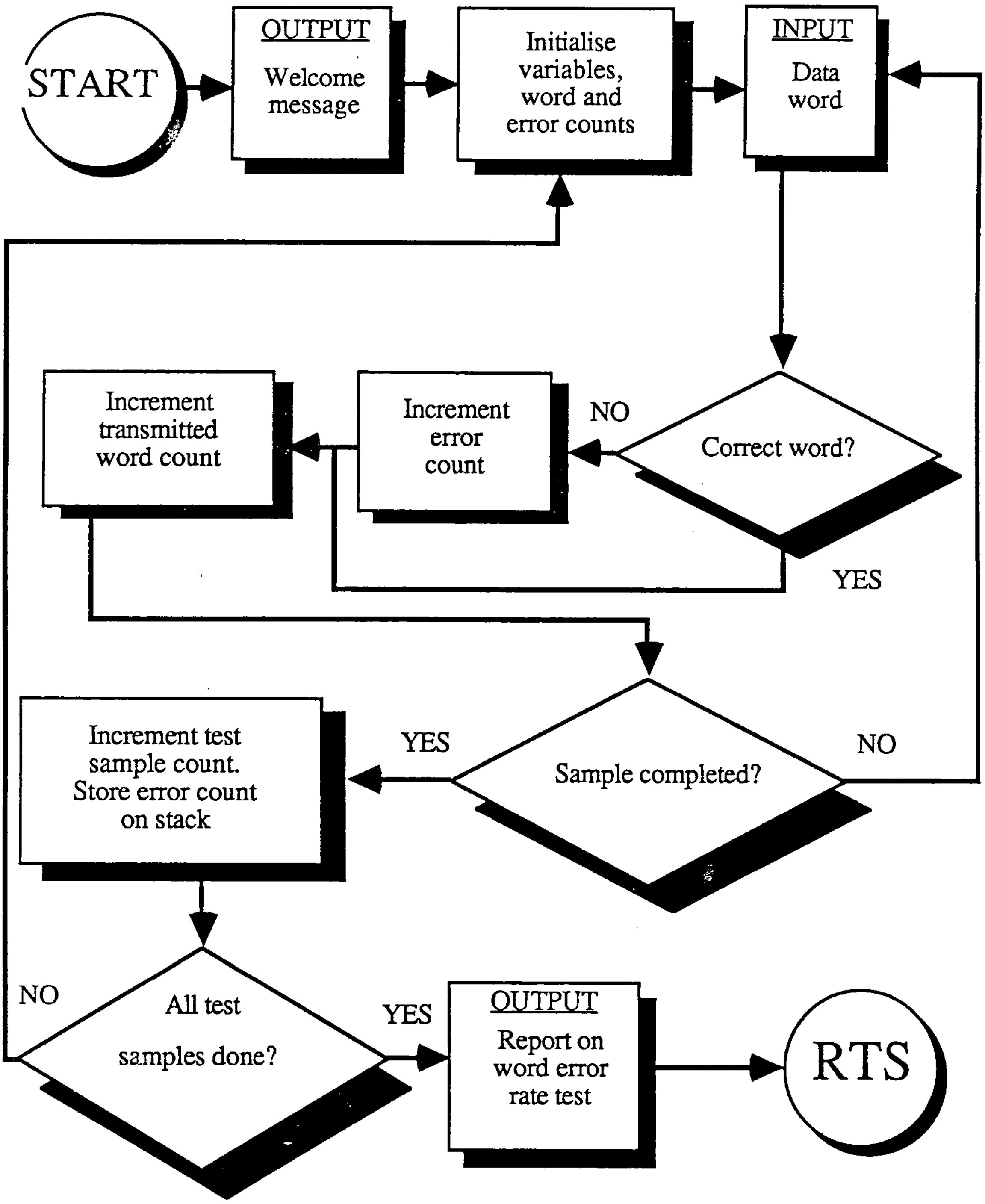

START
OUTPUT
Welcome message
Initialise variables, word and error counts
INPUT
Data word
Correct word?
NO
YES
Increment error count
Increment transmitted word count
Sample completed?
YES
NO
Increment test sample count. Store error count on stack
All test samples done?
NO
YES
OUTPUT
Report on word error rate test
RTS

Fig 5.3.11 "Word Error Count" subroutine

5.4 Use of System.

5.4.1 Controller Debugging.

Due to the non-completion of the Tx/Rx hardware development undertaken elsewhere [see ref 4.15], the debugging of the controllers was done without those units in the following ways. The code generation function was checked by examining the data tables produced in the controller RAM. The code loading function was checked by the use of a logic analyser which examined the data sent to the controller's PIAs. The Tx and Rx specific functions were checked in this way, but additional checks were accomplished by interconnecting the two controllers directly via their PIAs. The interface protocol was suitable for this purpose, the only restriction being that the receiver's software had to be run before the transmitter's in order for the dummy data signal sent on CB2 (Tx) to begin the transaction.

In order to determine the maximum rate of "Data Requests" that the transmitter could respond to, the receiver was replaced by a signal generator connected to the appropriate pin on PIA2. By looking at the interrupt line within the KDM board, it was found that the maximum rate of the "Data Requests" possible before the interrupt requests from the PIA became continuous (ie. limited by the response time of the interrupt service routine) was 18khz. This gave a data transfer rate of 288Kbits/s using a 4Mhz clock.

The maximum data rate that could be handled by the non-interrupt driven Receiver test software was obtained after linking the two controller's PIA2s together via a ribbon cable. By running the word error rate test on the transfer between the two controllers, the data rate was measured by again measuring the "Data Request" frequency, this time initiated from the receiver controller. This rate was found to be 14.9 KHz, a drop from the maximum possible rate of 3.1 KHz. This was mainly due to the nature of the test programs used in the receiver controller. Although the "Data Request" signal was automatically sent on reading the new data word received (see "PIAIN" subroutine, Appendix C), the programs using this data did not examine the PIA to see if further information had arrived until after completion of the processing and/or storage of the present data word. This was done to avoid the situation of the program being continually interrupted before completing its task, resulting in no storage or completed processing of data at all in a continuous long-term test, such as needed for determining the word error count. This repetition rate for the "Data Request" was nearly doubled to 25KHz when an 8MHz version of the same processor was substituted into the KDM boards, giving a data rate of 0.4Mbits/sec.

The successful transfer of every data word was confirmed by the use of the "File Transfer" routine, the data stored in the Rx controller's RAM being easily compared with the data file held within the Tx controller's.

5.4.2 Results From Testbed System.

The results for which the system was designed to yield cannot be included here due to the unavailability of the prototype transmitter and receiver hardware packages under development elsewhere. However, the controller hardware and software was completed and extensively tested for correctness. No errors were recorded for the transmission between the controller pair, so that any encountered during future work with the completed system are known to be associated with the system interconnecting the controller units. The work produced routines that would be useful in future systems (eg code generation and dataload), since these would fill requirements presented by any larger scale test-system.

Chapter Six
Full Scale System Design

6.1 Introduction.

The design of a full scale LAN using CDMA as its access mechanism was undertaken which took into account the OSI communications standards and the needs of a C^2 system as well as the requirements placed on the system by the access mechanism itself. This chapter gives an introduction to the system and an explanation of the working of some of its features. More detail is given in later chapters along with a more formal definition of the protocols mentioned.

Section 6.2 is an investigation of the effect that such considerations had on the choice of major system features such as its topology, the service it provides and the functional architecture of the system.

Section 6.3 investigates how the proposed system would tackle the problems associated with the provision of individual data-links and why a particular method was used when options presented themselves.

Finally, section 6.4 investigates how the proposed system would tackle important issues associated with a complete communications system providing a service to users, again explaining how choices were made when necessary.

6.2 Major Network Features

6.2.1 Choice of Topology

Since the use of CDMA as the media-access method was a pre-determined decision, the topology of the network constructed around it (henceforth to be known as Spreadnet) had to make the maximum use of its useful properties to justify the additional complexity (and hence expense) of the hardware and software involved. Therefore, in addition to the restraints imposed by the environments in which the C^2 systems operate (see chapter 2), consideration of the scale of improvement that the introduction of CDMA would make to the performance of the various types of LANs w.r.t the increased cost was made.

Use of a star topology would not utilize the ability of the access method to superimpose many logical channels upon a single medium. Use of SS in the dedicated links radiating from the central controller to the nodes would only be useful in increasing their end-to-end link privacy. This does not seem to be a sufficient reason to warrant the extra investment in equipment.

Use of a mesh network, although being the best choice for path redundancy (and hence survivability), is unsuitable for a number of reasons due, as in the above case, to the use of inter-node dedicated links. Although the use of multiple channels between nodes could be used to increase the data throughput of these links, the same improvement could be more cheaply obtained by the use of, say, fiber-optic technology which is sufficiently developed for such point-to-point links. If a unique code were to be assigned to a particular session of communicatons between a pair of nodes using suitable network management protocols, some use of the inherent addressing of CDMA could be made. Recognition of the incoming data's destination, without having to examine the data itself, would reduce its end-to-end delivery time. However, as the number of such allocated codes increases, the number of code sequences that each node may receive and then retransmit (the latter if it is acting as a relay) will rise until unnacceptable penalties in throughput performance or required hardware cost occur. Since estimates that at least 100 channels would be required in the application considered, the cost of accommodating worst case situations without incurring drastic system degradation in performance was calculated to be unacceptably large. Data privacy would also be enhanced between each node in any logical link, but the inherent slow delivery times of such a network would not support the real-time applications needed for our applications, especially if many relay nodes existed in any data path.

The use of a ring topology raises similar problems. All the ring protocols presently used rely on the use of individual dedicated links (as in mesh networks) and the shepherding of data by tokens or slots of some kind , to solve the following problems. These concern the removal of data from the medium and the possibility for data to reach its desination via two routes, often with different delivery times, if the data were simply broadcast onto the medium. Therefore the use of CDMA will once again only be useful in the ways outlined in the mesh topology case.

The bus topology type of network is the only one able to accommodate an access method which expoits broadcasting of data onto the media at any time. CDMA is just such a method and so the bus is the natural choice of topology. The properties of superimposing many channels onto the bus and the inherent addressing in this case are easily utilized. Data channel privacy is also obtained as is low internode message delay.

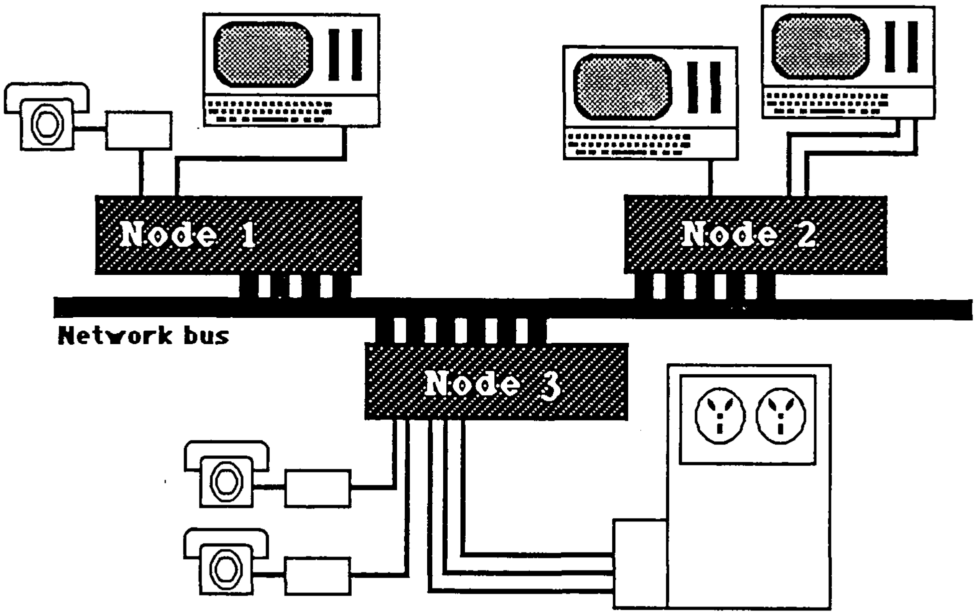


Fig 6.2.1 Example of a possible system configuration.

A refinement of this topology is the sharing of the occasionally used, but complex, network management software within a node by multiple links. By using a distributed star topology, this is acheived giving the following benefits. The cost of adding a new link to the network is reduced, making the interconnection of relatively cheap equipment a viable proposal, increasing the range of applications that Spreadnet will support. In the worst-case situation, where the main bus and all its duplicates (used for fault tolerance) are severed, communications within the local group can still take place. With suitable contingency planning, this could improve the survivability of the system.

Fig 6.2.1 shows a possible system configuration using this kind of topology, the main bus using CDMA, whilst the host-node link uses one of the established short range communications techniques.

6.2.2 Choice of Communications service provided.

Although it was always the intention to make the Spreadnet system as compatible as possible with the important emerging communications network standards, the sacrificing of CDMA's particular advantages was to be minimised. Thus, although the layer services outlined in OSI and 802 were applicable, the protocols were decided not to be. This was due to the fact that these protocols, especially the lower level ones, were specifically tailored to a particular type of network (see chapter 1.2). Since the kind of network needed to exploit the characteristics of CDMA is very different to those which have undergone the standards process so far, a new protocol suite was seen to be necessary. There seemed to be little benefit in adhering to these established protocols anyway, since the use of spread spectrum and CDMA techniques would render Spreadnet incompatible with any other at the physical layer. Interoperability with other networks is a problem solved by gateways, as is shown by the work presently being undertaken on LAN to WAN interconnection (see chapter 3.4).

The Logical Link Control service was considered by 802 to be the correct standard interface to a variety of LANs whilst the variations in network technology were all contained in the Media-Access-Control sub-layer. Since Spreadnet was also a type of LAN, this seemed the most suitable service to be provided. It was considered to be inappropriate in our case, however, to attempt to contain all the unique features of the LAN below the MAC service interface. This was due to the drastic effect it would have on the performance of the communications systems envisaged, inevitably so since it was designed to contain various techniques for time sharing a single transmission medium. Also, since the software needed to support even this low-level interface for a CDMA system involved the addressing of issues tackled by the LLC (such as provision of connection-oriented as well as connection-less services), the amount of wasteful functional replication was considered too great.

It was seen that the lowest service interface which the network could provide, without a significant amount of functional replication occuring with layers higher up and whilst not imposing performance penalties on users not requiring OSI facilities, was the LLC interface itself. Use of this interface had further advantages. Since this was an established International standard, many computer manufacturers were either producing equipment which conformed to it, or adapting it to fit. This was envisioned to reduce the task of LAN replacement with Spreadnet, since this interface is often exposed, and in this case is often between seperate pieces of hardware.

Since research is already under way into the way a LAN supporting LLC can interwork with WANs, the necessary software for Spreadnet to communicate with other systems will be already developed and tested by the time it is envisioned for its need. Also, the LLC layer provides a suitable service for Intra-network communication, without automatically passing on the performance penalties of the use of a higher service to clients not wishing to use it. For example, use of the Network layer is unneccessary for communications within Spreadnet only, whilst many of the other higher level protocols are not needed by such clients as specialised sensors and voice communications.

6.2.3 Network Node Overview

With the level of service to be provided by the node now established, the general architecture of the node could be considered. The mechanisms needed within the node were divided by function. The routine processing needed during data transfer (such as error checking) was seperated from the less used functions controlling channel setup/closure and internal node re-configuration. Thus a block diagram seperating differing functional units for a node satisfying our requirements was drawn. (see Fig 6.2.2)

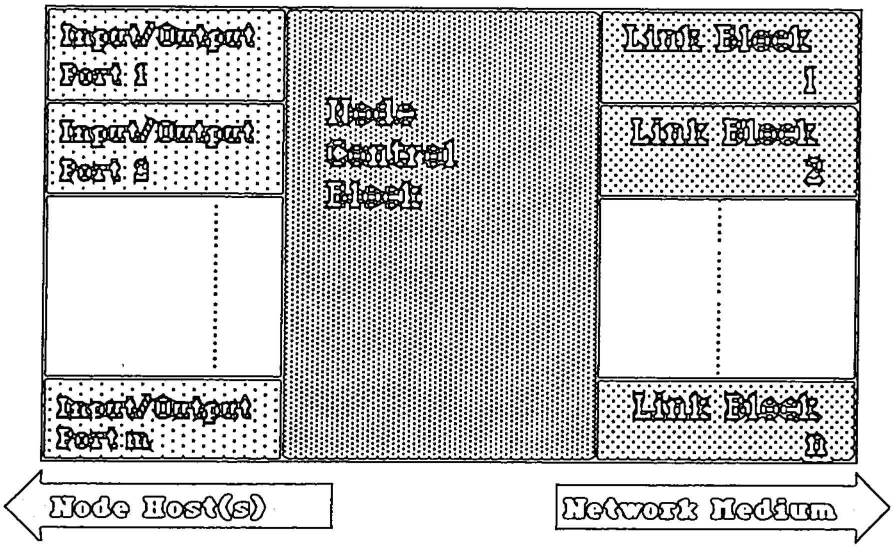


Fig 6.2.2 Block diagram of Spreadnet node.

Multiple input/output channels were provided to enable the node's role as a local control centre to be realised, the number being unspecified and variable for added flexibility. Multiple link blocks are used so that each channel allocated one code is used by only one communicating pair, utilizing the ability to send many messages simultaneously over a single channel. The numbers n and m are not necessarily the same. A single control block is shared by the communications channels due to its complexity and expense coupled with only occasional use.

Sufficient detail of specification was now established to enable the re-definition of the lowest two layers in the OSI model to more closely meet our requirements. Fig 6.2.3 shows the differing functional division below the same service interface. The unusual shape of the link-block software is due to the fact that during normal data transfer through the node, the services of the control block are not required and so it is by-passed. The Spreadnet Tx/Rx hardware service is slightly more complex than the standard physical layer due to its acceptance of a byte rather than a bit-stream (see chapter 5.2). Both the link and control blocks communicate with higher levels via the I/O blocks which are not present on the diagram due to their functional simplicity.

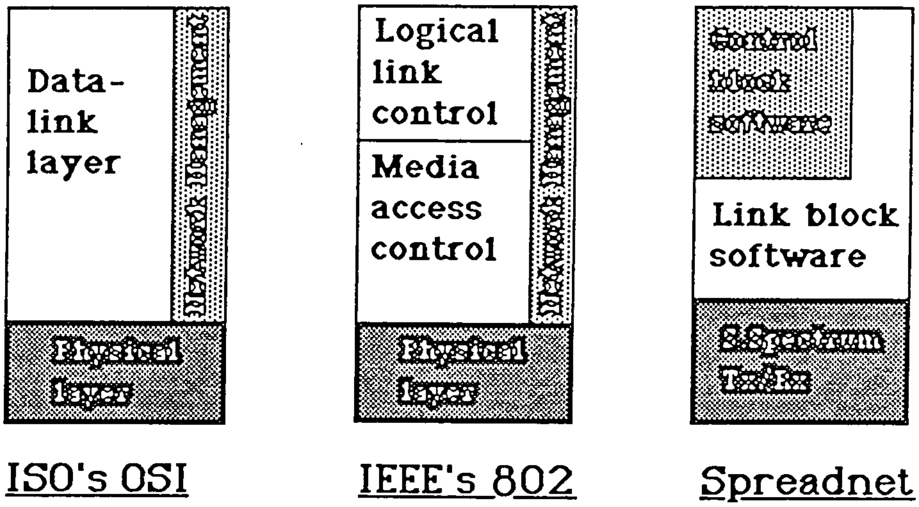


Fig 6.2.3 Varying functional division for the provision of the same service.

The functional separation also provided a guideline to the physical division of the node's hardware. The hardware architecture chosen for the node is illustrated by Fig 6.2.4.

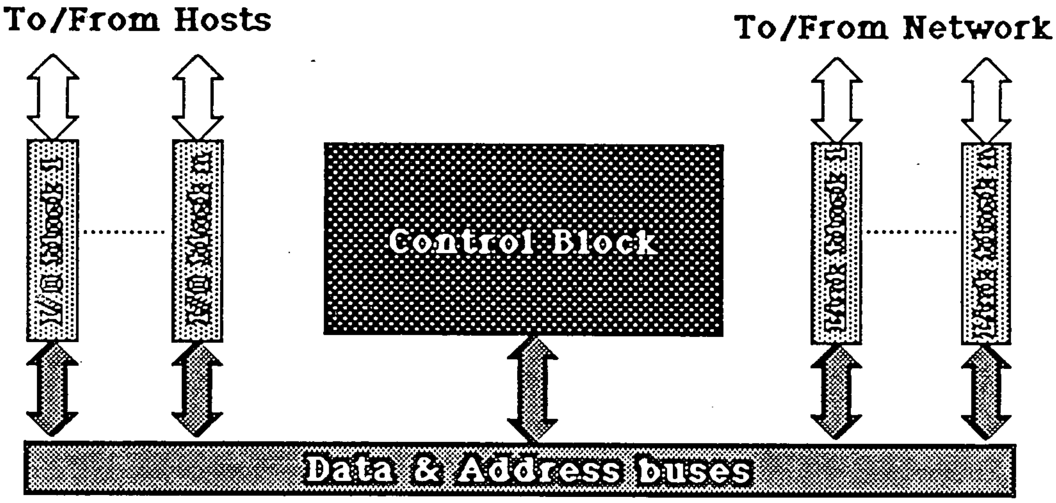


Fig 6.2.4 Hardware distribution of Node's functions.

This architecture had additional advantageous properties to those already mentioned. By separating the hardware onto seperate cards plugged into a central bus, replacement of faulty components is made easier. By centralising the control functions in the node, these can be modified more easily either by replacement of ROM chips or by its replacement by an extensively modified board. Since this control board is addressable on the bus, the control software could also be down-line loaded accross the network from another node or from one of its own associated hosts. If a standard backplane bus was used in the node, this would enable the node to be embedded in specialised equipment on the network with associated gains in performance and reliability.

The main advantage of using a central bus, however, is the increased fault-tolerance of the node. Since any I/O port can access any link block (subject to approval by the control block), then if one of the latter fails during a session with another node, the faulty node can reconfigure itself to allow the data to pass through another working link block. Similarly, for applications needing very high reliability, the control block could be duplicated so that if the active board goes down, the second recognizes this and takes over.

A further advantage of this hardware architecture is that, as previously mentioned, it is easily upgradable by the addition or removal of standard cards of hardware. In fact the property of dynamically altering the number of each type of block without affecting the data communications already under way is critical when important information could arrive at any time, a possibility if the system is used for command and control. Exploitation of this advantage was an important consideration in the later development of the node software.

6.2.4 Introduction to Node component use and interactions.

The process of functional seperation of tasks was again applied w.r.t the use of the available system bandwidth by the interconnected nodes. It was seen that although information related to distributed network management was required to be broadcast to all other nodes, data transfer was more commonly of a point-to-point nature. Hence it was decided to have a seperate dedicated broadcast channel for the exchange of data of the former kind, with the latter being allocated dedicated channels for the duration of a particular communications session. Therefore one link block per node was allocated for the exclusive use of the control blocks, the "Management Channel" (MC). The broadcast nature of the channel was possible by the allocation of a single code sequence to all of the transmitters and receivers associated with it (see next section).

This seperation of data according to type enabled increased network performance for the following reasons. Since the data channels no longer had to carry such information, their throughputs were increased and made more uniform, both being qualities central to the provision of real-time channels. Indeed, the gains of instant access to an allocated data channel would be greatly negated if real-time data had to be stored until transmission of control information had been completed. Also, if the control information was carried within data chanels, such links would have to be established between members of every other node on the network. Since it would be impractical to have all of these channels continuously pre-allocated (hardware limitations), then the time needed to set up the required number of these and send the information would result in a system with a very slow response time to change. For applications such as C^2, this was not suitable.

This separation of data by type required the node architecture to support the routing of data from a Spreadnet user to differing parts of the node when required. Hence it can be seen that since issues, such as the allocation of code sequences to channels, must be handled by network management, the logical connections between node components changes during the lifetime of a given logical link between two Spreadnet users. Consider, for example the case of a connection-oriented half duplex link. Firstly, a request for a link to network user B is made by user A. This request is passed to A's control block via its own associated I/O block, and then broadcast onto the MC. The control block which serves user B then processes this request and then broadcasts an acceptance or denial of the requested channel. If a denial is sent, A's control block passes this on to A via its I/O block and the matter is complete. However, if an acceptance is received, A's control block allocates a link block to A for the session, sends the information to A (reconfiguring the I/O block en-route) and updates the tables of all the nodes. Data transfer between the users then takes place, by-passing both of their control blocks, until one of the users or control blocks wishes to terminate the link. The users are then linked again to their respective control blocks and the link terminated. A more detailed examination of component interactions is made in the next section.

6.3 Issues related to individual Data-links.

6.3.1 Introduction.

The purpose of this section is to explain how the basic facilities required to provide useful data-links between communicands were realised using a CDMA system. Due to the unique properties that this access method possessed, an investigation into whether new ways of solving old problems were now possible was made. In some cases, the adaptation and combination of existing techniques was undertaken to solve problems unique to this network.

6.3.2 Link Addressing

Use of the inherent addressing property of CDMA was naturally to be exploited, but different ways of doing so existed. The following options were considered;

a- Each transmitter-link-block is allocated a unique code sequence and uses only this. The receiving-link-block would have to search the codes it is expecting to receive until a match is found and the data received.

b- Each receiver-l-b is allocated a unique code sequence and uses only this. The transmitter-l-b would select the necessary sequence associated with its intended target receiver-l-b and then transmit using it.

c- A code sequence would be allocated to each transmitter/receiver-l-b pair every time a request for a connection is made.

Whilst options a & b seem attractive from the point of view of offering link availability with no associated setup negotiation overhead, a number of major difficulties were seen to arise if they were used in a LAN system. Consider option b. If more than one transmitter-l-b wishes to communicate with a single receiver-l-b, then use of a protocol such as CSMA would have to be used to handle possible contentions. Hence link access could be denied for the first transmitter-l-b mid-way during a session if a break in data-flow occured, since the second transmitter-l-b would not know if the session had terminated or not. Thus long term real-time channels could not be maintained. If we now consider option a, other problems arise.

In order to search the many code sequences that it could receive on, two types of receiver-l-b were envisaged. The first would consist of a single receiver-l-b loading a succesion of code sequences into its hardware and testing to see if the sequence was in use. The second would consist of a bank of dedicated receivers each searching for a particular code. The second type would be extremely expensive due to the cost of the large numbers of correlators required whilst the first would require the use of long pre-ambles preceding the data to overcome the possible long delays before the correct code sequence was selected and loaded into the receiver hardware. Also, since both methods were based on the assumption that the receiver-l-b would know every transmitter-l-b to whom it may be communicating, then in order to get reasonable economy in the first case and acceptable pre-amble length in the second, the number of such sequences would have to remain small, thus limiting system flexibility.

However, the major drawback is due to the fact that many receivers could be searching for the same sequence and to enable the correct destination to be specified from within this group would require additional addressing measures to be taken by the transmitter. Since each of the potential receivers would have to receive sufficient data to check the required destination address, this would entail a lot of wasteful processing on the network. Also, if the required destination is already communicating with another transmitter on a different code sequence (in the second case) then this sequence is not even received, and since the receiver cannot know that this connection attempt has failed, this fact must be realised by the transmitter alone. Although this could be accomplished by the use of time-outs, the transmitter does not know when the receiver will be available and so must continuously re-transmit the same data stream until an acknowledgement is received. This of course could result in extremely inefficient use of the transmitter-l-b.

Therefore, in spite of the overhead involved in the negotiation of the code to be used, option c was considered to be the most suitable as it did not display any of the problems associated with options a & b. The negotiations would take place over the Management Channel which was considered to be able to support such data transfer in addition to other functions. The broadcast nature of this channel also enabled an efficient method of locating the required destination user among the nodes on the network.

6.3.3 Types of Data Channel Provided

The two major types of data channel to be provided were the point-to-point and point-to-multipoint channels (the broadcast channel being a special case of the latter kind). The method of addressing in both of these cases is the same (as outlined in 6.3.2) except that in the latter case, more than one receiver-l-b is loaded with the same sequence. Both are arranged over the Management Channel.

A range of differing link qualities was also seen to be desirable to provide between two communicating link blocks, each being suitable for a particular application. These were;

a- Full Duplex Channel with error correction.
b- Full Duplex Channel without error correction.
c- Half duplex channel with error correction.

d- Half duplex channel without error correction.
e- Simplex channel without error correction.

The inclusion of channels b, d and e, which contain no error correction, was justifiable for the following reasons. Firstly, the error rate encountered in LANs is low compared with other types of communication, the estimated bit-error rate for Spreadnet being no exception [4.11]. Also, in C^2, there exist a number of applications in which speed and consistent throughput of data delivery are more important than the guarantee of its error-free nature, such as the transmission of voice. This is due to the large amount of redundant data exchanged in these situations.

Channel type e was justified in spite of the fact that all link blocks can support two seperate channels, because point-to-multipoint links are inherently simplex in nature. Also, the increased throughput obtained in such links, as well as within half-duplex links, was seen to be desirable in some cases, even if two such links were used to enable high performance full duplex channels to be provided. The performance increase for half -duplex channels was due to the fact that larger frames could be used, (since storage of frames from both directions is not required) as well as the fact that responses could in some cases be despatched more promptly (waiting for data frame end not necessary).

Therefore some mode-dependency within the link-block was seen to be necessary to support these various types and qualities of data links. A more detailed look at the resultant necessary block is made in Section 8.1.

6.3.4 Provision of Link Privacy

Although the use of Spread Spectrum modulation for the data over the common medium is used, which gives the data stream the appearance of random noise, this alone was not considered to give adequate protection against intrusion into the privacy of data flowing between two communicating link blocks in all cases. This was due to the following reasons. Although in normal operation the network user has no direct control over the sequence to be used in any particular session, modifications could be made to the hardware to enable a user to receive any code sequence and hence intercept any information without the intended recipient(s) knowing. Also, in order to maximise the data throughput of the links, as short a sequence as possible is used for the number of links presently in use, resulting in only a few codes through which to search to listen in on any given "conversation". Therefore, it was essential to encrypt the data passing over the network media in the cases where data privacy is an important consideration.

Due to the existence of LSI implementations of the Data Encryption Standard (National Bureau of Standards), which were designed to handle data rates of up to around 10MHz, the encryption algorithm used in the Spreadnet data channels was this standard. Use of its CFB mode (see appendix A) was the most suitable in this case due to its greater flexibility, avoiding the problems associated with the padding of cleartext to match the block size in use. A performance penalty is imposed by the use of this mode, but it was calculated that it was still more than adequate to support our envisaged maximum link data rate of around 1MHz.

Since the reception of one enciphered bit in error causes the garbling of a constant sixty-five cleartext bits before the ciphers automatically re-synchronise, it can be seen that if the medium was unreliable then a channel requiring full error correction could suffer a notable degradation in performance. This would seem to suggest that the standard's OFB mode would be more suitable since errors are not propagated even though it does not posses as many desirable features as CFB. However, since Spreadnet contains many features that make its data links reliable even in adverse conditions (due to the use of Spread Spectrum techniques), this problem did not assume the importance that it would in other types of networks. However, due to the nature of the networks possible C^2 environment, use of encryption over the channel was made a user selectable option.

In order for this method of encipherment to work, both of the communicating link blocks must share the same cipher key and the initial values that the shift registers contain before data transfer occurs (see Appendix A). This data would be sent via the management channel when a link is being set up. Since the DES relies on the secret nature of the cipher keys in use, this information would itself be encrypted using a public-key technique. This will be examined in greater detail in the section on Network Security (see section 6.4.4).

6.3.5 Data-link setup and termination.

An examination of the intra and inter-network interactions involved in the seting up and termination of data links will now be made. This will include the case briefly mentioned in section 6.2.4 but in greater detail. A detailed look at the LLC interface protocols and a formal analysis of all the other protocols involved is deferred until later (see Chapter 8).

Consider the setting up of a full-duplex, encrypted point-to-point link with error correction between users A and B as depicted in Figs 6.3.1. to 6.3.5. With reference to Fig 6.3.1, the procedure begins with a request for a link (1) containing the destination's identification number (user B), details of the type of link needed and its priority wrt the other links associated with the node. This information is passed to the control block (2) where it is processed. Firstly, the resources of the node are checked to see if such a link can be supported (and if its priority enables it to displace another user from using some). If it cannot be supported or if the destination node is not present in its system resource tables , a reject message is returned to the user explaining the reason why the request was refused (via the i/o block), but if sufficient resources can be allocated then suitable information is passed to the link block allocated for Management Channel use. This information, which is subsequently broadcast onto the network (4), contains B's ID and the type of link required (as this would affect the amount of resources required).

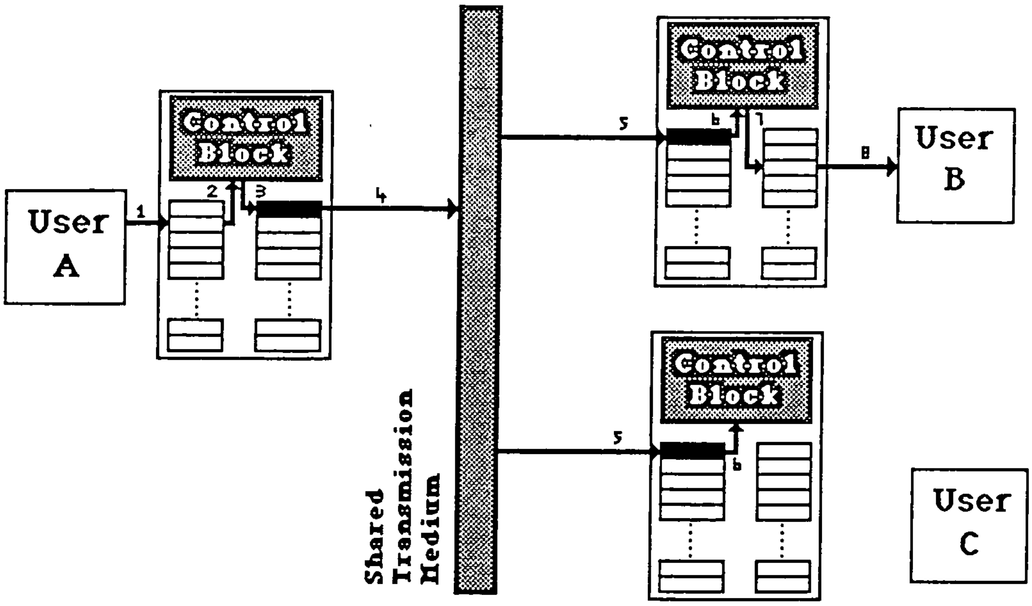


Fig 6.3.1 Link setup/termination request.

This message is received by all the other nodes on the network (5) and then passed on to their respective control blocks (6). The control block then checks to see if the destination ID refers to a user that it supports and whether or not it could support the proposed connection. If the answer to the former search is negative (as in the case of user C's node) then the information is ignored, but if the latter case is the one encountered then a rejection of the proposed link is broadcast with reasons. This will be received by A's node and A will be notified of the rejection. However, if the searches give positive results, then the link request is passed onto user B for consideration.

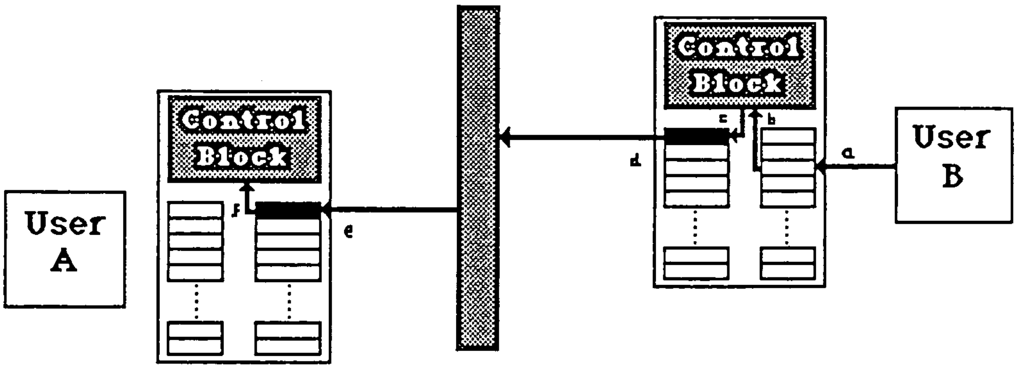


Fig 6.3.2 Reply to link setup request.

The outcome of the request for a connection to user B from user A is then returned to A's control block (see Fig 6.3.2). This message is labelled as a reply to A's request so that no confusion arises with other nodes expecting similar replies. This is important since the source of the reply is unknown to the requesting node. The system's ability to function despite this ignorance is useful in two ways. Firstly, the node can function without the need for a large updateable routing table of the users connected to the network (all nodes would then contain data they would never need, such as the routes between telephones and radar sensors). Secondly, the desirability for the non-existence of such routing tables, for the purposes of security, has been stated by the Navy (see section 2.2.2).

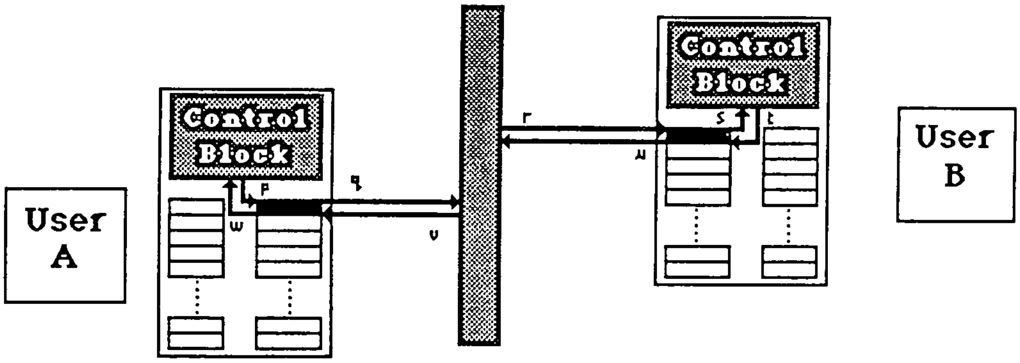


Fig 6.3.3 Exchange of encrypted messages.

This exchange of messages is necessary so that the subsequent messages encrypted using Public-key cryptographic techniques (see section 6.4.4) are not computed if the link is not able to proceed. This is because of the large computational resources needed to encrypt such information which will thus only be undertaken if the link is to be used and the current public key for any given node having been pre-arranged over the MC (see section 8.2). The exchange of the encrypted messages is illustrated in Fig 6.3.3.

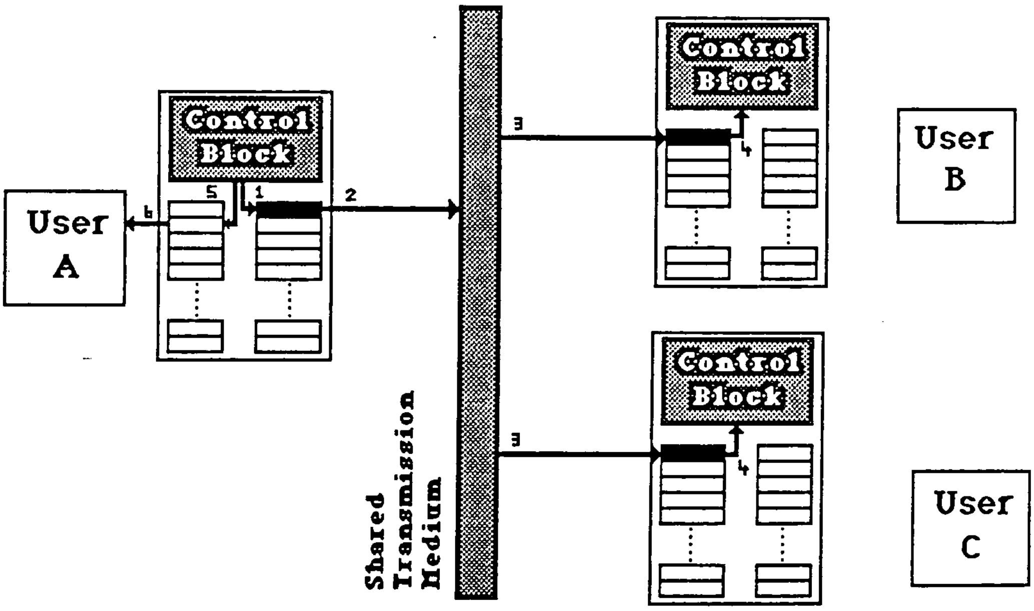


Fig 6.3.4 Sequence use and link confirm notification.

The message from A's control block to B's contains the key and initial fill to be used in the DES chip associated with the data channel from A to B as well as the code sequence to be used by the hardware (p to s). Similarly, B's control block then replies with the relevant information for the corresponding channel from B to A. It should be noted that although all of the other nodes also receive this information which is not initially known to be encrypted, their control blocks will recognise its irrelevance since it will not correspond to expected messages for their protocol state or will be seen to decrypt incorrectly if that node is expecting a similar message (see Appendix A).

Now that the details for the link have been arranged, a further message is broadcast onto the network (see fig 6.3.4) which informs all nodes that the pair of codes is selected and so cannot be chosen for any other link. This also acts as a confirmation to B's node that the arrangements have been finalised (1 to 4). If a node receives such a message and is mid-way through negotiating a channel using one or both of these codes, then the procedure for that link has to be restarted using another set of codes, thus avoiding the multiple allocation of a single code sequence. A's control then confirms to user A that the channel is set up and that transmission of data can commence (both nodes have configured themselves accordingly by the processes explained in Chapter 8). The logical bi-directional link depicted in Fig 6.3.5 is thus available for use.

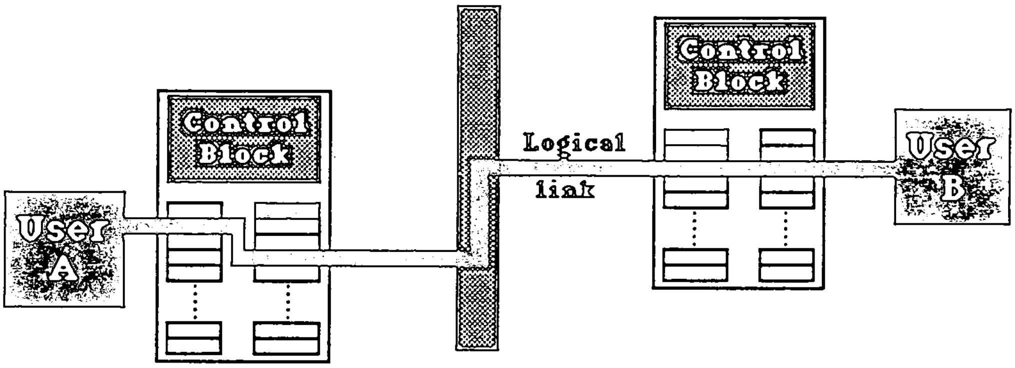


Fig 6.3.5 Completed link.

The termination of a point-to-point link can be initiated by either user or by either control block (the latter for reasons of resource allocation). If the case of initiation by A is considered, reference to Fig 6.3.1 can again be made. In this instance, the message transferred between the nodes is a disconnection request (1 to 6). This contains the code sequences which were used, so that all the nodes can update their sequence tables and allowing the pair to be used within another link. User B is informed of the request (7 & 8) and the control block broadcasts an acknowledgement of receipt of it which is passed back to user A along the same route. If the request was initiated by A's control block, then the procedure would commence with information passed between it and its MC link block (3).

The setup/termination of a half-duplex point-to-point link is the same, only the utilization of the channel pair is different. In both cases, if use of encryption or error correction is not needed, the selection fields in the messages are left empty (ie. they are default settings).

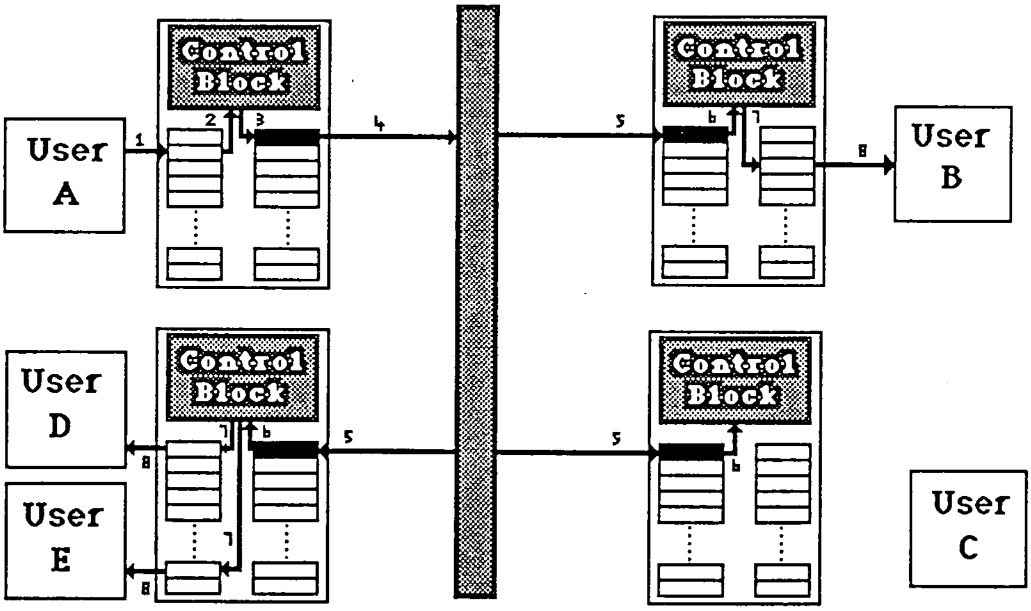


Fig 6.3.6 Point-to-Multipoint link request.

The setting up of a point-to-multipoint link does not follow the same pattern since the multiple receivers cannot all reply to the transmitter (the latter does not know how many of the former exist, it is only supplied with their group ID). Consider the case of user A wishing to broadcast to users B, D and E. Fig 6.3.6 shows the path followed by the request to all nodes (1 to 6) and in particular to the intended users (7 & 8). This request also contains the sequence to be used so that all the nodes can update their sequence tables. The control blocks interpret the request as being of a point-to-multipoint type and hence a search of the group as well as individual IDs is made. A user can have any number of such IDs (subject to restrictions imposed within the users themselves and the memory allocated for this purpose within the control block) and can thus be a member of many point-to-multipoint groups. If more than one of these groups becomes active and requests a connection to a given user, the response is user selectable. Either the present link is maintained until completion or the link with the highest priority rating is selected.

This request is not answered and the sender merely assumes that all the prospective destination receivers have received it. This also means that encryption cannot be used over such links unless pre-arranged to occur within the users and also precludes the use of error correction at the Spreadnet level. The channels are therefore set up by the sending of this message only (see Fig 6.3.7). Therefore it is necessary to inform the user that the link is set up after the request has been sent, even though no guarantee exists that any users are receiving it.The termination of such links also involves the sending of one message only, again containing the sequence used, to enable table updating.

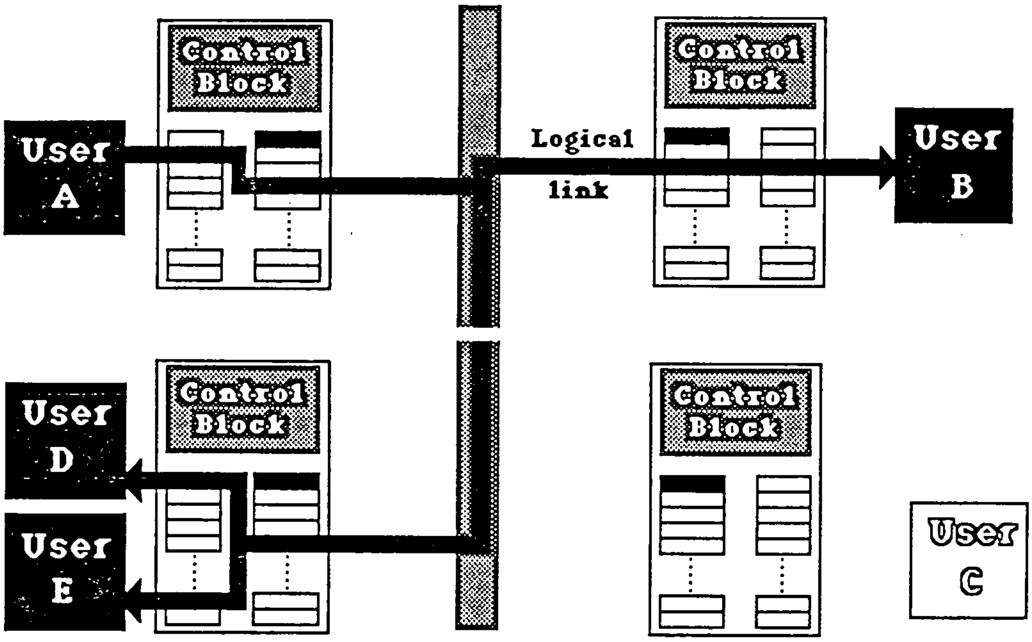


Fig 6.3.7 Completed point-to-multipoint link.

6.4 Network/Node Management Issues.

6.4.1 Introduction.

Whereas the previous section dealt with issues relating to single data-links across the network, the issues concerned with the provision of services assumed in the former will be discussed within this section. The formal presentation of the protocols and the description of any specialised data packets used is deferred until Chapter 8.

6.4.2 Addition/ Removal of Nodes to/from the Spreadnet system.

This issue of the addition and removal of nodes also addresses the issue of network initialisation. Since the system was required to be of a decentralised nature, then the updating of the relevant tables within the nodes had to be accomplished among that same group with no master node overseeing the operations involved.

Node removal could be notified by the broadcasting of a unique message over the MC. The information enclosed in this message is the node ID and any code sequences now released by the termination of data channels still open at the time of node removal. This is called the "Graceful Removal" of a node where the removal is known to all. If the node is removed from the system due to reasons such as hardware failure ("non-graceful removal"), then this notification may not take place. Thus, the system will remain ignorant of this change until a link is requested or until the operation of an associated data channel is noticed to be incorrect. In the case of the former, if the requesting node does not receive a reply within a certain time, it repeats its request. After a pre-determined number of failed retries, this node then terminates the connection attempt.

In the case of the disrupted communications channel, then the link-block associated with the link informs its control block and the latter informs the node at the other end of the questionable channel over the MC. If it obtains no reply after retries, then the link is terminated. (If a reply is obtained then link checks are made, see section 8.2). No notification of the node's fault/absence is made since this message could be forged and even if nodes check to see if false information is passed about themselves and the message exposed as a forgery, the malevolent user would not be easily located and so could repeat the process (the user is assumed to have built a special control block). The penalty for this approach is wasted MC and control block time, but since the duration of the links was assumed to be long then this was not seen to present a potential bandwidth problem.

Node addition is accomplished in the following way. Each node is provided with its identification by unique hardware/firmware within its Control block and therefore has to establish what other control blocks are attached to the network. To do this*, a message is broadcast over the

* Information is requested from another node (see section 8.2.8), then

MC which contains all of the necessary information about the new node to allow all the others to update their tables. This "Hello" message contains the node's ID, the node's current Public Key (see Appendix A) and the class of service that it can support (eg LLC class, see sections 8.1.5 & 8.1.6). This message is received by all other nodes ~~but is answered only by the node with the lowest ID number (or perhaps the highest, every node knows who these are) excluding that of the new node~~. This reply confirms the acceptance of the new node to the system and contains all the relevant information about all of the other nodes presently connected to the system.

If the new node is the first present on the system, then it will obtain no reply after a number of retries and so knows this fact. To prevent confusion in the situation when many nodes are connected at the same time, a time delay related to the node's ID is introduced before the node broadcasts its "Hello" message. Also, if any node receives such a message, then it defers the broadcasting of its own message for a pre-determined period allowing the former to complete its initialisation. Since the rate of addition of new nodes is assumed to be low in the shipborne application, all of the data transfer involved could take place over the MC.

Since the MAC protocol specified for the IEEE 802.4 Token bus network [3.16] fulfilled many of the functions required by Spreadnet, it was decided that this would be used within the MC Link-block, with the specialised data concerning codes etc being generated within the control block and sent as data across the MC. All of the MAC functions were placed within the link block in order to offload the necessary processing from the control block. Also, since the interface was originally designed to minimise the data primitives passed across it, this was seen to be beneficial, decreasing the traffic on the node's internal bus. Since greater functionality is enclosed within the link-block in this case, a small change in Fig 6.2.3 is required for this special case, raising the level of the link-block functionality and removing the by-pass channel.

The major reason for the adoption of the token-bus protocol, rather than using CSMA/CD, was the determinism of the former, which was felt to be of paramount importance in the C^2 environment. It also allowed message priorities, a property regarded as being of great value in the MC. Some changes had to made to the specification, however, since it was originally designed for use over a traditional broadband bus topology (see section 1.2.1(g)). Therefore, the values of the end-to-end propagation delays had to be be re-defined, with its interface to the physical layer being totally scrapped, owing to the byte-level nature of this interface in spreadnet.

6.4.3 Node resource allocation.

The most important resource that the node supplies to its users is communications bandwidth, or more specifically link blocks and code sequences. In previous sections concerning the allocation of link blocks to requested data channels it was assumed that these were allocated for the entire lifetime of the required connection. However, the possibility that the supply of link blocks could not meet the demand would suggest action that could invalidate that assumption.

When a request is made for a connection (either remotely or from a local user) the availability of link-blocks is checked. If none are available then the request's priority rating is checked against those of the data-links currently in use. If it is greater than one of these then the control block informs the two users of that connection of its deferment/termination (the local user via its i/o block and the remote user via the MC) and subsequently allocates the link-block to the higher priority channel. To prevent the unauthorised use of high priority codes to gain access to the link-blocks, some checking procedure must be employed in the Network Management software within the Spreadnet users. However, if strict priority ratings could be appplied to the users, as is possible in our C^2 case, then these ratings could be entered into the control block firmware and the ratings checked when any request is made. This re-allocation of link-blocks could also occur in the case of link-block failure, thus increasing the reliability of the system.

It should be noted that the MC possesses the highest priority rating and hence its link-block cannot be used for any other purpose.

The changing of the code sequence during a connection was also possible for certain types of Networking applications. This could result in optimised throughput of channels by changing the sequence length to the minimum able to support a family of codes large enough to cope with the current number of users. However, in order to change the sequences at the same time, all of the connections would have to be temporarily held up whilst the hardware is loaded with the new codes. If the variation in the number of connections was frequent, this could lead to unnacceptable loss of performance of the links. If use of this sequence changing was considered appropriate then it would be accomplished via the use of the procedures summarised in Section 8.1.6.

To prevent the need for an excessive numbers of code changes for small variations in connection numbers, the upper and lower link-number thresholds between sequence lengths would not be the same. Use of an overlap between the two, the boundary for length reduction being less than that for an increase in length, enabled the required increase in sequence stability.

7.4.4 Network Security.

a--Introduction.

Consideration of network wide security issues has led to the identification of eight essential features that such systems should posses.

1-- Positive identification of users must be made before use of the system is granted.

2-- The systems and possibly also the network management must be able to check that their actions are authorized.

3-- Users actions should be monitored so that errors or misuse can be traced.

4-- Data, hardware and software should be protected from fire, theft or other forms of destruction and locked to prevent unauthorized use.

5-- Data should be reconstructable in the face of accidents etc and be auditable.

6-- The network and systems should be tamperproof.

7-- Transmission should be failsafe so that errors or failures do not cause message loss, double-processing etc.

8-- Transmissions should be private.

Since the design work is only concerned with providing a logical-link service, not all of these issues can be dealt with. The issues relevant to Spreadnet are numbers 6 to 8.

Item 6 relates to the protection against the bypassing of security measures by programmers, this is done by not providing any type of interface to secure information (ie the messages passed to Spreadnet are of a pre-defined type allowing only particular interactions to take place between the network and its users).

Item 7 is dealt with by the use of error checking mechanisms over the data links (see section 8.1.2).

Item 8 is partly dealt with by the use of encryption over the data-links as outlined in sections 6.3.4 and 6.3.5. The rest of the section looks at this item in more detail and explains the kinds of threats encountered on the links that it is used to combat.

b-- Data-link Security/Privacy.

The established security goals for data passed over general data-links are as follows;

1-- The detection of spurious node and channel connection initiation so that information is not passed to the wrong node.

2-- The detection of message stream modification.

3-- Detection of data corruption caused by the removal of an entire data unit.

4-- Prevention of release of data contents.

5-- Prevention of traffic analysis.

These general goals were defined at the time when such channels only existed over WANs where data was passed between many nodes during its transmission, therefore not all of them are valid in the case of LANs, and particularly Spreadnet.

Goal 2 is important in networks where the data is read in by an intermediate relay node during its path between the communicating parties. The opportunity to modify the data thus exists. In Spreadnet, due to the broadcast nature of the messages, each node only passively taps the information on the network and so it cannot be modified. Goal 3 arose because of similar considerations and so also does not apply over the inter-node data-link itself. Loss of a complete data unit by jamming is countered by the inherent properties of Spread-Spectrum. For both of these cases, such modifications could only occur between the link and I/O blocks and so if other users of a particular node are not trusted then suitable measures within the users protocols must be included.

Goal ~~3~~ 1 is accomplished in the following way. Since each node's control block maintains a table of the nodes currently connected to the system, then no other node can request connection to the system which gives an ID already in use (ie first come first served). If the false node obtains entrance to the system before the real node, then the subsequent refusal for entrance to the system will signify the problem. No release of secure data will occur even if the false node is successful in penetrating the system since it cannot decrypt any existing secret data flow and would not receive any, as such channels would utilize further authentication in the user's software (eg passwords) before such a data exchange is made. Only unencrypted data could be received and only then if the correct code sequences are known or found, the latter requiring specialised hardware.

A brief definition of traffic analysis now follows before an examination of Spreadnet's relevant properties is made. Traffic analysis is the examination of the length of messages and their frequency of transmission along with all observable message headers in order to gain knowledge of the data being exchanged and between whom it is taking place. Origin-destination along with frequency and length are normally obtained by examining a single channel. Each packet sent on a normal LAN contains its source and destination adresses and is terminated by a special flag. In Spreadnet, the data channels do not contain the address information, this would have to be obtained via the MC on link setup. Therefore any intruder would need to monitor *at least* two channels in order to gain the same information previously obtained by monitoring only one. Thus traffic analysis is not prevented, just made to be more difficult.

The final goal, prevention of release of data information contents, is met by the use of encryption. At the time of writing the best choice for the encryption algorithm was the DES standard. Although the code has been shown to be breakable by the use of large dedicated computers, this was not seen to present a problem in a LAN environment, such a system being labelled as being "practically" secure (as opposed to being "theoretically" secure). An introduction to the method of secret-key distribution was presented in section ~~7~~6.3, further detail follows;

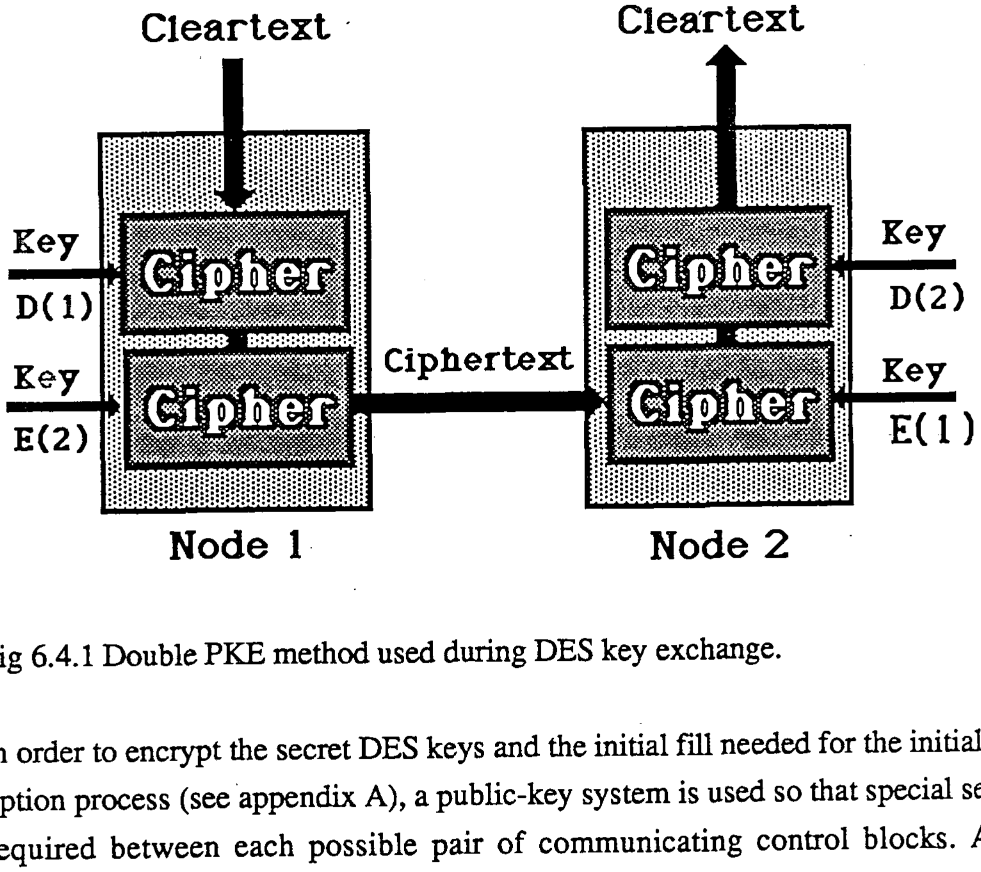


Fig 6.4.1 Double PKE method used during DES key exchange.

In order to encrypt the secret DES keys and the initial fill needed for the initialisation of the encryption process (see appendix A), a public-key system is used so that special secret keys are not required between each possible pair of communicating control blocks. A number of public-key algorithms exist, but only one, the RSA algorithm, can use the keys reversibly. ie the cleartext message can be recovered by decrypting a message encrypted with the public-key or by encrypting a message "encrypted" by its corresponding secret decryption key. Thus message authentication can be provided (since only user x is assumed to know x's secret key) as well as encryption using the same algorithm. The algorithm's drawback is its computational complexity, resulting in long encryption times and so it is only used for encrypting short non-time critical messages.

The Spreadnet system uses the double encryption method illustrated in Fig 6.4.1 to provide both encryption and authentication of the secret information passed during communication channel setup (see section 6.3.5). Authentication was seen to be desirable to prevent the loss of secret data to another node supplying a false DES code whose presence may not be detected if, say, Node 1's hardware became faulty during the transaction and so would not receive the fake message and thus raise the alarm.

c- Red-Black seperation.

This technique is used in dealing with the release of message contents by attempting to construct a barrier between the secure (red) and the insecure (black) facilities in a communication network. The goal is to prevent the transmission of cleartext across the barrier from the red side to the black side by forcing all such data to pass through an encryption unit. The flow of information in the reverse direction has to pass either through a decryption unit or through a bypass around it. In the network context, complete red-black seperation is not generally acheived because address and control information must be passed around the sender's encryption unit from the red side to the black side for routing and flow control purposes.

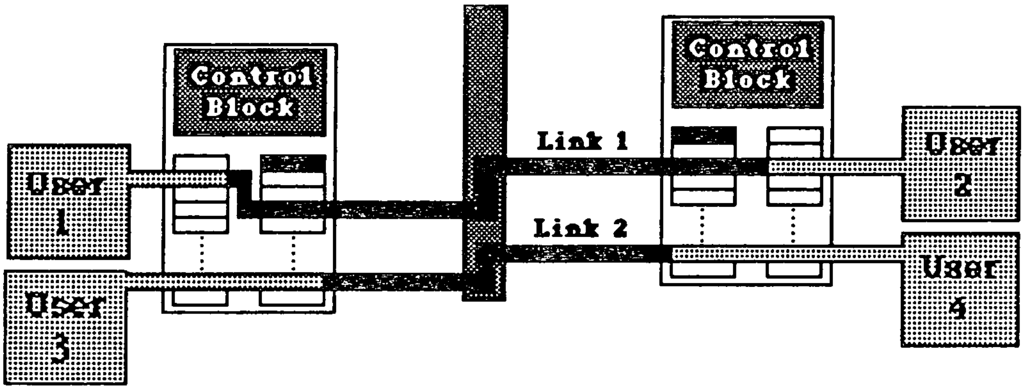


Fig 6.4.2 Possible Red-black seperation points.

In the case of Spreadnet, the problems are different. Although the data channels can be encrypted, the MC cannot due to its broadcast nature and so total red-black seperation cannot occur. However, this seperation is possible during the transmission of data between two users. Figure 6.4.2 shows the possible forms of red-black seperation for a Spreadnet logical channel. The choice of which form to use and hence where to locate the DES chip was made after consideration of the following factors. Link 2 assumes that its own node is secure and so its data can pass in cleartext form along the node's own internal bus. If this information is secret then the data would be encrypted within the user. Link 1 gives additional privacy within the node itself but at great expense. Since the DES key has to be passed from the control block to the I/O block during link setup, in order for this to remain secret within the node, it would have to be encrypted. Therefore a complex security mechanism would be needed which would add greatly to the expense and increase the response time of the node. The method used in Link 2 was seen to be the most suitable for the system. The DES chip was thus installed within the link-blocks in the system. A more detailed look at the internal workings of the node components is given in the following chapter.

Chapter Seven
Node Component Services

7.1 LLC-Service Implementation Overview

7.1.1 LLC Services supported.

The relevance of the various optional services and functions provided by the LLC sublayer, outlined in section 3.3.3, were considered for the Spreadnet application. These are listed below, with the reasons for their inclusion (or rejection) in the Spreadnet specification.

a- **Connection oriented service** - this was considered to be the most suitable service to be provided by a CDMA system such as Spreadnet, since both are primarily designed to support long-term and reliable data links. Therefore, this service was supported and was indeed to be the major service, since the majority of C^2 applications required long-term links.

b- **Acknowledged connectionless service** - although this service was seen to be usefull in certain applications, since it is still under consideration by the standards bodies, the provision of this service was not included at the present time.

c- **Polled response service** - although primarily designed to reduce LAN costs, this service could be used by Spreadnet to reduce the number of sequences in use, if these are in short supply. This could be done by allocating a particular sequence to the polled group, over which they would all converse. However, since this service is also still under consideration, the decision to use this service is again deferred.

d- **Exchanging ID with other LLCs** - Although the response to such a request is mandatory, the ability to initiate one is optional. The ability to do both was considered useful within Spreadnet, so the facility is supported.

e- **Loopback test** - this was seen to be useful for a variety of necessary node functions, so was supported.

f- **Downline loading** - use of this feature to enable the loading of altered LLC software remotely, was seen to pose many serious problems in a decentralised system where security is an important issue. Reliability requirements suggest that such software should be resident within the individual nodes, being held within ROM. Due to the localised nature of the network, as well as the static nature of its user topology and the large length of time betwen software updates (see section 2.1.2), the relative ease of modifying the necessary ROMs suggests that this is the better solution. This facility was therefore not supported by Spreadnet.

From the above information, it can be seen that the LLC provided to the Spreadnet user is of Class II (ie supporting c-o and c-less operation). The provision of the c-less service to the user is made in the interest of conformance and to preserve the performance required by certain transaction-oriented applications (eg. those associated with network management).

7.1.2 LLC Operation Component Mapping to Node Blocks.

The operation of the LLC is logically divided into components. Each component characterizes a set of protocol operations performed by an LLC entity, describing it as perceived by another LLC entity in a remote station or by a higher layer protocol in the local station. The three components defined within the LLC standard [3.16] were as follows.

1- **Station Component** - This component is responsible for processing the events which affect the entire LLC entity. The station Component handles PDUs addressed to the null DSAP address (ie for station protocols such as exchange ID and loopback) and processes the duplicate address check (if implemented). One station component shall exist for each MAC-SAP present on the LAN.

2- **Link SAP Component** - This component is responsible for processing the events which affect a specific operating SAP. One of these components shall exist for each LLC-SAP within the LLC entity.

3- **Connection Component** - This component is responsible for processing the events which affect a specific data link connection for Type 2 procedures only. One such component shall exist for each data link connection supported in the LLC entity.

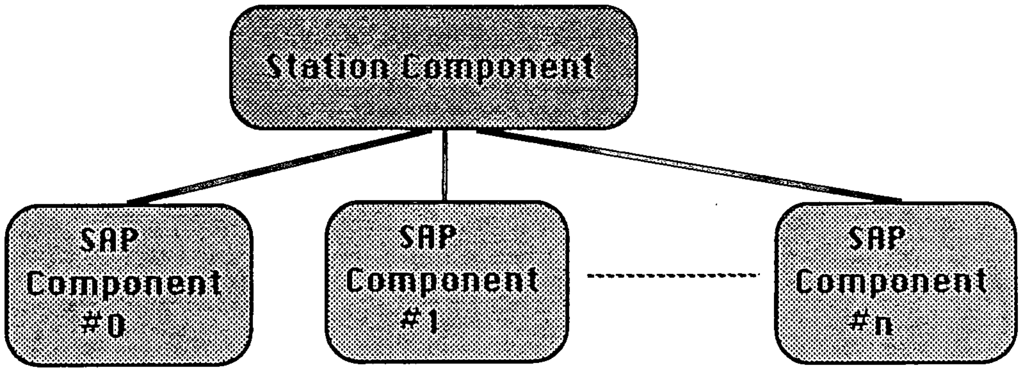


Fig 7.1.1 LLC Class 1 Component Relationships.

These three components are hierarchically related. The Station Component is the "parent" of the SAP Component, which in turn is the "parent" of the Connection Component. These relationships are illustrated in Figs 7.1.1 & 2, for the two differing classes of LLC operation. Each "parent component has a state which provides the enabling condition for the "child" component(s) to operate. If the "parent" component is disabled, then the associated "child" processes are as well.

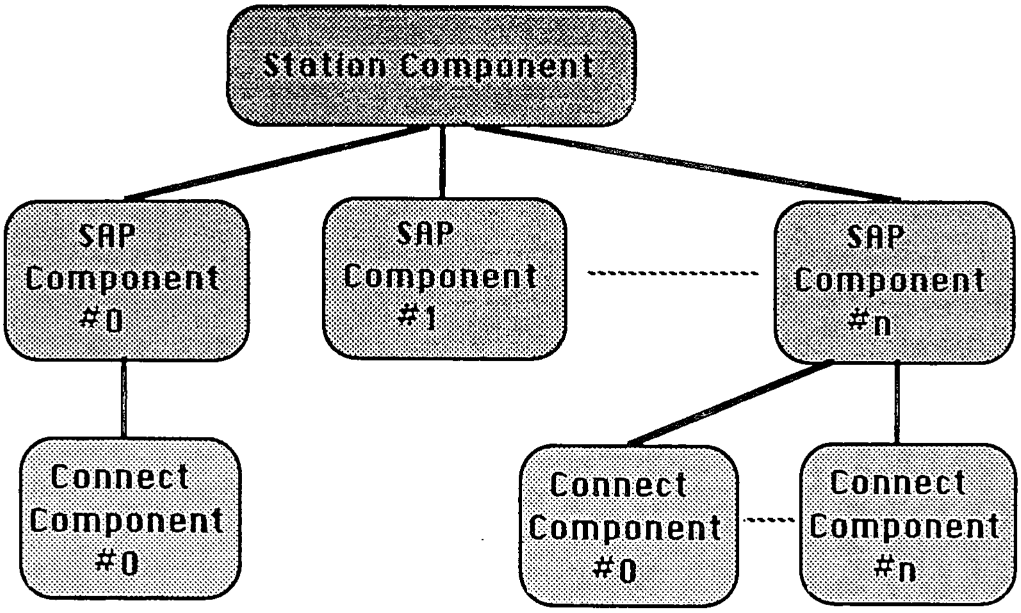


Fig 7.1.2 LLC Class 2 Component Relationships.

In addition to these, a further component was defined, a **Network Management Component**, since many functions relating to the operation of Spreadnet as a whole could not be considered relevant for inclusion into the Station Component. As well as the issues associated with Spreadnet alone, functions associated with this layer have been identified for the management of an OSI network [7.1]. These include setting system parameters, initialisation and closedown, diagnostics, exception reporting and the gathering of statistics about network performance. The exact requirements in these areas on the LLC sublayer are still under study, so those not deemed essential for Spreadnet are not specified at present. One of these components is required within each of the Spreadnet nodes. This component was considered to be the "parent" of the Station Component, since it is desirable for it to control the latter's state.

The description of the Station Component has to be reconsidered for Spreadnet. Since a given connection is comprised of data flow across the MC and an associated communications channel, then it could be argued that more than one "MAC-SAP" is present, each supporting a channel using differing protocols etc. However, since this condition was made in order to be able to accomodate connection to multiple LANs, then this was not seen to be a compatibility problem. If a given Spreadnet node is required to be interfaced to a number of LANs, then the same number of Station Components would be necessary, each interfaced to their associated multiple "MAC-SAPs". The processes comprising the Station Component in a Spreadnet node reside within the Control-Block, so that the null LSAP addresses part of this hardware unit. Therefore, if multiple Station Components are required within a single node, a choice between housing them all within a single control block or within multiples of these must be made by the system operator. this decision will be dictated by cost, node performance/loading characteristics and possibly allowed node size.

Within the Spreadnet node, L-SAPs are used to select the service required and not to select individual users of the node. Therefore, at present, four exist in ordinary stations, with an extra one for the selection of functions associated with a station providing both a local communications capability and a gateway to another network (plus a possible further two to support LLC service types 3 and 4, when they are fully specified). The four always included are associated with the Station Component (null L-SAP), the Network Management Component, a SAP Component interfacing to users requiring a connection-less service and a further SAP Component associated with Connection-oriented data tranfer. Therefore, since each of these components is concerned with multiple data channels, the processes involved with L-SAP Components are housed within the control block containing their associated Station Component.

The processes included within the Connection Component are housed within all three of the node's functional blocks. The connection -endpoint -identifier within a LSAP associated with a c-oriented data channel, is the identifier of the I/O & link block pair used by the appropriate node user (ie incoming data addresed to that c-e-i is passed directly to the appropriate i/o block. Similarly, outgoing data is passed directly to the appropriate link block). This by-passing of the control block does not occur with c-less data transfer, since the associated SAP is within the control block (but the i/o block it passes through is still the representation of its associated c-e-i). Part of the processing involved during connection setup/termination or modification etc is accomplished within the control block, in order for the Network Management Component to monitor and control the node and network more effectively.

The Network Management Component is linked closely with the Station Component, being responsible for the same set of communications channels. Thus its processes are resident within the same control block as its associated Station Component (ie. each supported LAN is controlled by a seperate Network Management Component).

7.1.3 Specification of the Spreadnet Node Interfaces.

Within the Spreadnet node, a number of interfaces exist between the three types of functional blocks, which will now be identified and later specified in the manner adopted by ISO and the IEEE [7.2 & 3]. By specifying these interfaces in this abstract way, an implementor of the system is not bound to divide the hardware along these lines or to use methods dictated by present technology. Figure 7.1.3 illustrates the 6 interfaces which have to be defined. Interface 1 is the LLC service interface and so must support the services detailed in its specification [3.16]. Interface 2 is between any I/O block and the Control Block, whilst Interface 3 is between an allocated I/O-Link Block pair. Interface 4 is between the Control Block and a Link Block supporting a connection-oriented data channel, whilst Interface 5 is similar except that the service supported by the Link Block is connection-less. Finally, Interface 6 defines the service provided to the Link Block by the Spreadnet Tx/Rx Block.

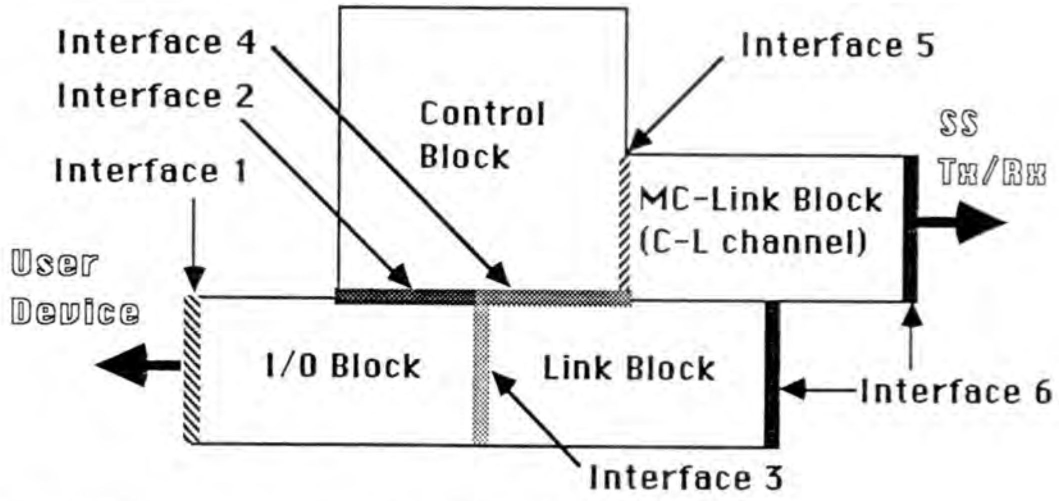


Fig 7.1.3 Interfaces associated with Spreadnet Nodes.

As previously mentioned, these interfaces which describe the services provided to each block to another are described in the form now popular within international standards. Using the abstract model of a data communications system comprising an n-layer service provider and two n-layer service users (see Fig 3.2.3), the service is defined by the interactions occuring over the interface, described in the form of "service primitives". These service primitives are abstract, implementation independent interactions, which are divided into several categories or types by ISO. The **request** primitive type is issued by a service user to invoke some procedure within the service provider (eg connection establishment). The **indication** primitive type is issued by the service provider to invoke a procedure in the service user, or to indicate that a procedure has been invoked by the service user at the peer SAP. The **response** primitive type is issued by a service user to complete, at a particular SAP, some procedure previously invoked by a request at that SAP. The **confirm** primitive type is used for the same reason as the response primitive, but is issued by the service provider (the generation of such a primitive to be passed to one user is always as a result of a response primitive from the other user).

The service primitives used by IEEE 802 are of three different types. The request and indication primitive types are the same as those defined by OSI, the response primitive type does not exist and the confirmation primitive is redefined. In this case, the confirmation primitive type is defined to be a service provider confirmation only (ie. the services provided by the IEEE protocols do not provide for associating a confirmation with the actions of another service user). Therefore, no guarantee is provided that the receiving user actually received the data, only that it was delivered to the appropriate place (SAP). Therefore, if the conveyance of information about the actions of a remote service user is required, then this has to be handled as data by the service provider. Since the service specification for the LLC does not include response primitives, the

IEEE primitive set was used within the Spreadnet interface specifications.

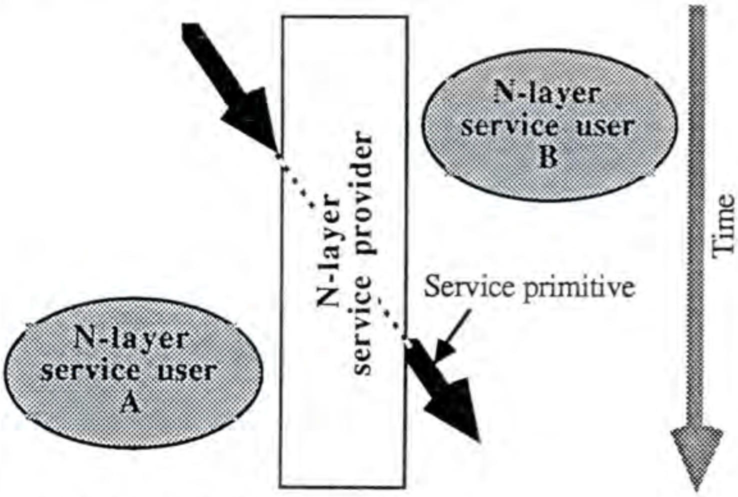


Fig 7.1.4 Primitive Sequence Description Diagram.

Within this chapter, the primitive sequences used across the interfaces are illustrated by diagrams based on those used by ISO and IEEE. Fig 7.1.4 is an example of the diagrams to be used, which shows how the three communicating parties and the primitives are represented. The vertical lines represent SAPs, whilst dashed lines indicate definite cause-and-effect relationships between the primitives (if any). These diagrams help clarify the service primitive list (with associated parameters and a description of their use) which constitutes a complete specification of the services offered by a given layer (or functional block) to its user.

The LLC specification uses eight different parameters associated with the various primitives. A brief explanation is given here before the specifications are given in the next section. The parameters **"local address"** and **"remote address"** specify the transmitting and receiving LSAPs respectively. This is usually done by providing, at a minimum, the logical concatenation of the physical address of the nodes and the required SAP within it. In the case of Spreadnet, this parameter consists of a unique user identifier (this includes the SAP components within each node describing the Station Components and Network Management Components). This is broadcast over the MC, resulting in each Control Block searching its "User Table" so that it can be located. In this way, no routing tables are required by the network users, only a directory of device identifiers. The **"l_sdu"** is the specification of the link service data unit to be transferred by the service provider. In the case of Spreadnet, this parameter consists of the data itself, since the network is presumed to occupy a seperate hardware entity to the users.

The **"status"** parameter is used in indication and confirm primitive types to signify if the requested service was successfully completed, or gives the reason why it failed. The **"reason"** parameter is used to identify the cause of link disconnections, whilst the **"amount"** parameter is used to specify the amount of data that the service user can pass, without data loss, over that paticular data channel. (used for flow control purposes). The final pair of parameters are named

"service_class" and **"quality"**. These indicate the quality of the connection service that the user requires. Within the LLC standard, the only item of quality selectable by "service class" (used in c-o service) is the priority that the connection has over others (the "quality" parameter, specified in the c-less service, is less explicitly defined). This reflects the fact that the larger range of "Quality of Service" parameters considered in layers 7 down to 3, are primarily used to select a particular subnetwork, whose properties are well defined in these areas. Since Spreadnet is able to offer a number of services (see section 6.3.3), so that the upper layers can select services within a single subnetwork, Spreadnet, the use of the priority parameter is expanded.

The "Quality of Service" parameters, specified within layer 3, that can be used to select one of the "Spreadnet Services" are "residual error rate", "protection" and "priority" (with "resilience" having a direct relationship with priority in Spreadnet, see section 6.4.3). Also, "throughput" is selectable, since the half duplex channels do not have to interleave messages and acknowledgements over the channels used. Also, the channels not using error correction exhibit higher throughputs, due to smaller overheads in noiseless conditions and due to the absence of retransmissions in noisy conditions (these channels are used where the timeliness of the messages arrival is of greater importance than its correctness). In order to to this, the priority parameter is split into four fields;

a- "Error correction" - either selected or not.

b- "Encryption" - channel encryption selected or not (not applicable to c-less service, where either all of the shared channel's data is encrypted or not, therefore not affording privacy to other nodes. This is because the encryption would have to be done by software, or by switching of the data stream between an encryption unit and a corresponding unencrypted line for the addressing information etc. Both would give rise to unacceptable overheads within the node, so the task must be undertaken at a higher layer in this case).

c- "Channel type" - full duplex or not (ie half duplex).

d- "Priority" - the channel is assigned a priority level from a range of such values.

In this way, either the user or the higher layers can select the desired service merely by choosing a particular value of connection priority. Within the specifications, some further parameters will be introduced which were not specified within the LLC specification. These will be explained as they arise in the text.

7.2 Interface Specifications

7.2.1 Interface1- The LLC Service

This interface is the level at which the LLC service is provided to the service user. Therefore, it uses the primitives specified in the IEEE LLC service specification. However, the way in which these generalised primitives are to be used in a communications system which utilises Spreadnet is outlined. Since this level of service supports two differing types of data links (c-o & c-less), the primitives are divided into two groups.

a- **Connectionless service primitives** - there are just two primitives for this service;

L_DATA.request - this includes parameters specifying the IDs of the data's source and destination for the data transfer, a parameter specifying the link-service-data-unit to be transferred and a quality parameter which specifies the service class desired for the data unit transfer.

L_DATA.indication - includes the same parameters as above.

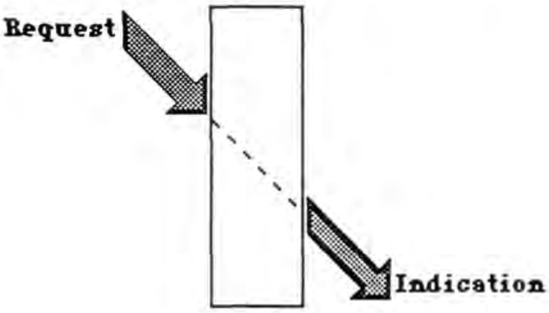


Fig 7.2.1 Primitive sequence for successful C-L data transfer.

The request is sent when the user requires a c-less data transfer to occur. On receipt of the indication primitive type, the action of the receiving entity is unspecified in the LLC specification. If the request is unsuccessful, no primitive exists to signal this to either of the users. In this case, the quality parameter can only select the data's relative priority over other such messages on the shared channel. All of these priority levels are lower than network management transactions if the data is transmitted over the MC.

b- **Connection-oriented service primitives** - there are fourteen primitives associated with this service falling into five groups;

L_CONNECT - this set of primitives is used when a logical channel is desired to be setup.

.request---- this includes parameters for the source and destination users required and a "Service Class" primitive.

.indication-- this includes all the parameters included above, but in addition also contains a "Status" parameter which signals if the connection was successfully formed or why it failed.

.confirm---- this contains the same parameters as .indication, giving the requesting user details of the result of the request.

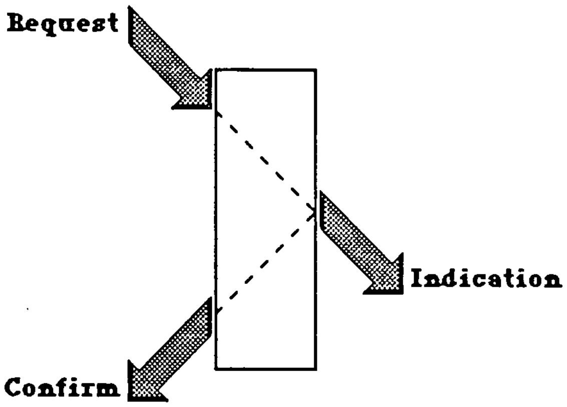


Fig 7.2.2 Primitive sequence for successful C-O link setup

In this case, the priority parameter is used to select the range of options outlined in the previous section. These are granted subject to the approval of Network Management within both of the associated nodes, which may be capable of deciding what priority assignments are allowable from differing user IDs and/or whether the requested services can be met by available node resources. This information would be present in the control block software of each node.

L_DATA_CONNECT - this set of primitives is used for the transferrence of a LSDU over the link.

.request---- this contains the source and destination addresses of the data (as in previously described primitives) as well as a parameter specifying the LSDU to be transferred over the link.

.indication-- this contains the same parameters as the request.

.confirm---- this contains the standard address parameters and a status parameter which indicates the success or failure of the transfer.

These primitives are used as standard, in our case the LSDU would be transmitted to the i/o block instead of parameters describing its position in memory.

The primitive sequence for the successful transfer of C-O data is shown in Fig 7.2.2.

L_DISCONNECT - this set of primitives is used to terminate the data channel.

.request---- this primitive just contains the source and destination IDs.

.indication-- in addition to the IDs, an additional "reason" parameter is included which differentiates between local and remote reasons for the termination.

.confirm---- in addition to the IDs, an additional "status" parameter is included which informs whether or not the remote LLC entity acknowledged the disconnection or not.

The primitive sequence for the successful termination of a link is shown in Fig 7.2.2.

L_RESET - this set of parameters is used when the resetting of the connection's state is required. ie Every channel passes through a number of states during its lifetime, changing on the receipt/transmission of certain primitives. At certain times, the return to the connection's initial state is desirable.

.request---- this contains the address parameters only.

.indication-- as well as the address parameters, a "reason" parameter is included which differentiates between a remote and a local request, and gives the reason for the local request.

.confirm---- this contains the address parameters only.

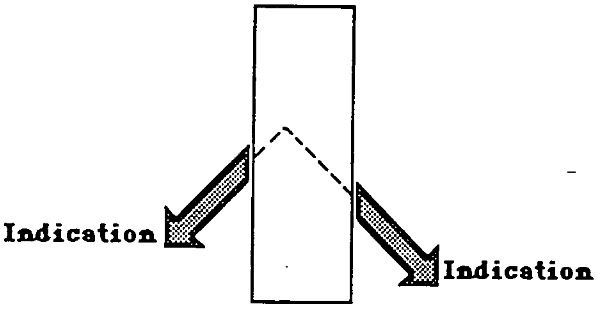


Fig 7.2.3 Primitive sequence for the successful resetting of a link via Network Management.

This is an abortive service and all unacknowledged LSDUs are dicarded. Thus data may be lost unless appropriate measures are taken at higher layers within the spreadnet users. The primitive sequence for the successful resetting of a link via a remote request is shown in Fig 7.2.2. However, reset due to Network Management is shown in Fig 7.2.3.

L-CONNECTION-FLOWCONTROL - this set of primitives is used to control the amount of data passed across the LLC service interface.

.request---- this contains the address parameters to identify the data link in question and also an "amount" parameter which specifies the amount of data that be passed out of the LLC service interface (without data loss) in implementation specific units. If the amount parameter is set to zero, then the associated flow is stopped. The amount can also be specified as infinite. The amount of data permitted to pass is updated by each request.

.indication-- This contains the same parameters as the request primitive, but in this case the flow controlled is that into Spreadnet.

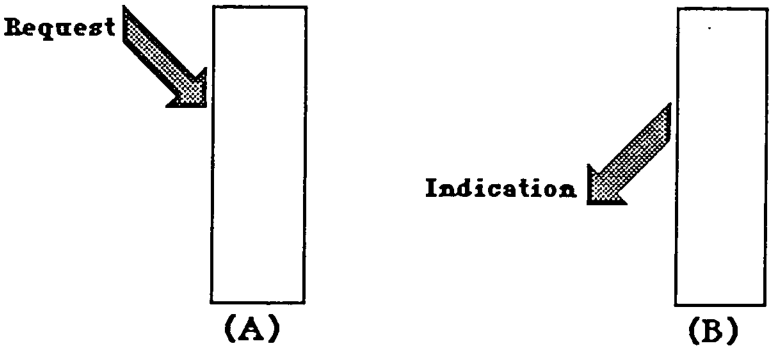


Fig 7.2.4 Primitives used in provision of interface flow-control.

These primitives are the only ones detailed within the LLC specification. Since work on the layer management functions is still incomplete, the primitives associated with them are not available. Therefore, most of this area of work is undefined for Spreadnet until the relevant standards are available. However, one parameter can be identified, which is associated with the loading of the network user's ID into the Control Block's user table.

USER_IDS - this parameter set is concerned with the addition of the IDs associated with a network user, to the spreadnet system.

.request---- this includes a compulsory individual ID (unique throughout LAN) parameter, as well as a parameter listing the group IDs associated with the user (if any).

.confirm---- this includes a parameter indicating if the individual ID is unique within the user table of that particular mode (ie global uniqueness cannot be guaranteed in the absence of a global user table held within spreadnet).

The request primitive type is issued by the network user to the Spreadnet node when it desires to be added to the system. The primitive sequence associated with this process is illustrated by Fig 7.2.6.

7.2.2 Interface 2 - Control Block Service.

The service provided to the user above this interface is very similar to that offered above Interface 1. This is because the I/O block acts mainly as a routing device, dividing the primitives between other components within the node. Therefore, instead of re-stating the primitives already presented in the previous section, the correspondence between them is stated.

a- **Connectionless service primitives** - these are passed to the control block, since they have to be directed to one of the shared communications channels supported by the node (across an instance of interface 5). The primitives correspond in the following way, the parameters associated with them being unchanged;

L_DATA.request ------> CB_DATA.request
L_DATA.indicate ------> CB_DATA.indicate

b- **Connection-oriented service primitives** - not all of the fourteen primitives associated with this service are present at this interface. This is because once the connections are established, the data flow by-passes the control block. However, if the state of the connection is required to be changed (eg resetting), then the appropriate request is routed to the control block so that its relevant state table can be updated. The primitives correspond in the following way, each associated with the same parameters as supplied at Interface1;

L_CONNECT.request ---------->	CB_CONNECT.request
L_CONNECT.indication ------->	CB_CONNECT.indication
L_CONNECT.confirm ---------->	CB_CONNECT.confirm
L_DISCONNECT.request ---------->	CB_DISCONNECT.request
L_DISCONNECT.indication ------->	CB_DISCONNECT.indication
L_DISCONNECT.confirm ---------->	CB_DISCONNECT.confirm
L_RESET.request --------->	CB_RESET.request
L_RESET.indication --------->	CB_RESET.indication
L_RESET.confirm --------->	CB_RESET.confirm
USER_IDS.request --------->	CB_USER_IDS.request
USER_IDS.confirm --------->	CB_USER_IDS.confirm

CB_LB_ALLOCATE - In order to indicate to the I/O block which of the link blocks has been allocated to receive the data from its attached user, the Control Block issues a primitive containing suitable addressing information for the data flow between them to occur.

.indication -- this includes a "lb-address" parameter to enable the routing of data to the link block allocated to the connection.

The primitive is issued after a successful request for a channel has been concluded, (the confirm primitive having been sent) or when the Control Block wishes to alter the allocation (in the event of a Link Block failure etc). The primitive "sequence" is illustrated in Fig 7.~~4~~2.5.

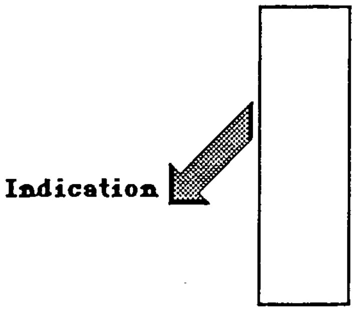


Fig 7.2.5 Primitive sequence for I/O-Link Block allocation.

This interface specification is incomplete in the same way and for the same reason as Interface 1.

7.2.3 Interface 3 - Link-I/O Block Interface.

The primitives passed across this interface are concerned with issues related to the data stream passed over an established data connection. Since this interface supports the Interface 1 primitives not passed across Interface 2, the primitives are again directly related to those encountered at Interface 1. Therefore a list of the correspondence between these primitives follows;

L_DATA_CONNECT.request ---------->	LB_DATA_CONNECT.request
L_DATA_CONNECT.indication -------->	LB_DATA_CONNECT.indication
L_DATA_CONNECT.confirm ---------->	LB_DATA_CONNECT.confirm
L_DATA_FLOWCONTROL	LB_DATA_FLOWCONTROL
.request ---------->	.request
.indication -------->	.indication

7.2.4 Interfaces 4 & 5 - Control-Link Block Interfaces.

Whilst both of these interfaces are between the Control Block and a Link Block, 4 is associated with the c-o service and 5 is associated with the c-less service. The selection of the interface type is made by the passing of a primitive to the Link Block, which specifies the software suite to be used. This primitive could include a pointer to software already resident in all of the Link Blocks, or could contain the software itself (this process would occur at node power up, where the large amount of data traffic is acceptable). The second option reduces the amount of memory/logic required within each of these blocks, whilst allowing the greatest flexibility in the services to be provided (including those under development within 802, supported by further similar interfaces).

LB_SERVICE - this set of primitives is used for the specification of the service selection process outlined above.

.request---- this includes the service parameter which contains the specification of the service to be supported below the interface.

.confirm---- this includes a service status parameter which indicates the service now supported by the block.

These primitives are passed during the initialisation of the node (when the number of channels supporting each of the service types is likely to be known, or set to default values). When a block is selected to support either service, it is informed of the Control Block's address, allowing a redundant CB to alter this if it is required. When a c-less service is selected, the message routing of all data to the CB is implicit. The primitive sequence associated with this process is illustrated in Fig 7.2.6.

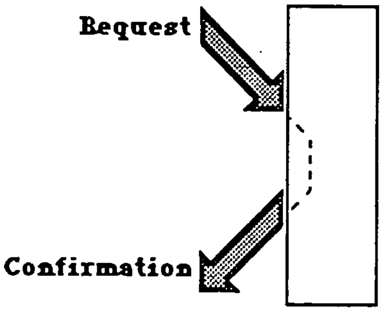


Fig 7.2.6 Primitive sequence associated with service selection.

LB_RESET - this primitive is used to reset the link block to one of a choice of states, when this is deemed to be necessary.

.request---- this contains a type parameter specifying the degree of reset required. (ie purge of all data within the block and re-initialising of any connection state tables in use, or complete purge of all software within the block.

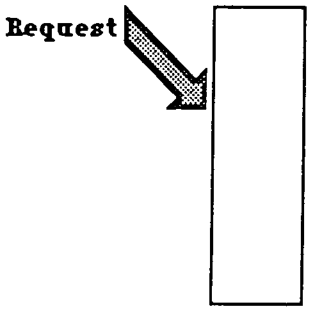


Fig 7.2.7 Primitive sequence associated with link resetting.

This primitive is used to rectify detected problems concerning a particular channel and also to prepare the block for the loading of different software when necessary. The primitive sequence associated with Link Block resetting is illustrated in Fig 7.2.7.

SS_SEQ - this set of primitives is used to load spread spectrum code sequences into the Link Blocks (for the associated Tx/Rx hardware) for future use. More than one sequence can be included within the primitive, selection of one occuring at a later time.

.request---- this includes parameters specifying a code sequence and another its associated local (within the Link Block) ID.

.confirm---- this includes the ID parameter (used to signify which of the sequences were received) and a reason parameter specifying if each was accepted, or why it was rejected.

The addition of sequences to the list of those associated with a particular Link Block can be made at any time during a data connection. Reasons for the failure of the transfer include a sequence ID identical to one of those in use. This primitive is also used to flag errors related to the sequence (see Section 8.1), so more than one confirm primitive can be associated with a given request. The primitive sequence associated with the loading of the sequences is shown in Fig 7.2.8.

SS_SELECT - this set of primitives is used for the selection of a particular sequence for use over a data connection arranged over the MC.

.request---- this includes the ID parameter of the code to be selected (or the sequence length ID in the case of the MC) and a threshold parameter indicating the coarse value of the correlator threshold to be used by the receiver hardware (see section 4.3.1). There is also an initiate/not parameter which indicates whether the local node is to initiate the change, or just respond to the commands from a remote node.

.confirm---- this contains the same ID parameter(s) as in the request primitive, whilst also including a status parameter indicating the acceptance of the selection, or the reason for its rejection.

Over a p-t-p link, the node with the lowest ID is the initiator. For multipoint links, the initiator is the transmitting node. For changes requested over the MC, the node which is requesting the change acts as the initiator. Usually, a sequence pair is specified within each primitive, one each for the transmit and receive channels (only one is allocated to a Link Block supporting c-less data transfer, since the same sequence is used for transmit and receive. If one of these sequences is unsuitable for use (eg not present in memory, or both different in the c-less case), then the pair is rejected. The primitive sequence associated with the selection of the sequence(s) is illustrated by Fig 7.2.6.

The remainder of the primitives are divided into two groups, one associated with each of the two interfaces defined so far.

a- **Connection-oriented service parameters** - the primitives passed across Interface 4, including those in common with Interface 5 are concerned with the formation, termination and management of the data connections. These issues include those present within any communications system, whilst others are unique to Spreadnet.

CB_IO_ALLOCATE - In order to indicate to the Link Block which of the I/O Blocks has been allocated to receive the data that it is receiving over the link, the Control Block issues a primitive containing suitable addressing information for the data flow between them to occur.

.request -- this includes a "io-address" parameter to enable the routing of data to the I/O block allocated to the connection.

The primitive is issued after a successful request for a channel has been concluded, (the confirm primitive having been sent to the user device) or when the Control Block wishes to alter the allocation (in the event of a Link Block failure etc). The primitive "sequence" is illustrated in Fig 7.2.7.

LB_MODE - this set of primitives is used to select which of the currently supported c-o service quality choices is required for the connection.

.request---- this contains a quality of service parameter, used to identify the service quality required over the link. It also contains a size parameter, which indicates the octet number in the frame to be used over the link. The final parameter is the error rate parameter, which specifies the upper limit of the error rate (number of frames received in error divided by the reference number of frames received) before the Control Block is to be notified to take appropriate action(eg reduce the frame length and raise the channel's efficiency).

.confirmation---- this contains the same parameters as the request primitive type, but in addition contains a status parameter which signifies the acceptance of the requested parameters, or the reason for their rejection.

One of the reasons for rejection is that the error rate is too high. This can be used by an unsolicited confirmation primitive to indicate that modification of frame size is required, or that failing this the service cannot be provided over the channel (using the sequence selected, mode selected etc). The primitive sequence associated with this mode selection is illustrated in Fig 7.2.8, with the second confirmation indicating the exceeding of a threshold.

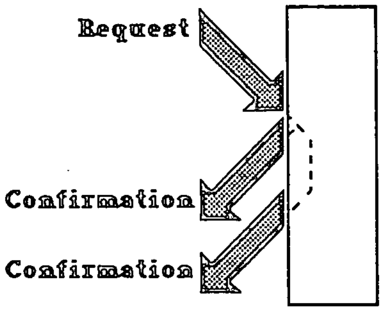


Fig 7.2.8 Primitive sequence associated with mode allocation.

b- **Connection-less service primitives** - these parameters are passed across Interface 5 and are predominantly taken from the IEEE 802.4 Token ~~Ring~~ Bus MAC service specification [3.16]. This is because the protocol specified for this type of MAC was considered to be suitable for use within Spreadnet, resulting in the service that it supports being largely unchanged (with the addition of the primitives shared with Interface 4). The standard specifies three primitives that are passed between the LLC and MAC sublayers, with another six being passed between the station management entity and the LLC. An explanation of the usefulness of each of these primitives is now presented.

MA_DATA - this set of primitives is used for the transfer of data between the communicating Control Blocks.

.request---- this contains a destination address parameter (user ID in the case of Spreadnet), a data unit parameter (the data itself in this case) and a desired_quality parameter (indicating the messages priority only in this case).

.indication---- this contains both destination and source address (ID) parameters, a data unit parameter (as in request) and a quality parameter (which indicates the priority that the data was allocated).

.confirmation-- this contains a quality parameter (indicating in this case the priority assigned to the data) and a status parameter (indicating if the data was successfully transmitted or not, it does not signify successful reception of the data).

The primitive sequence associated with this passing of connection-less data is illustrated in Fig 7.2.9.

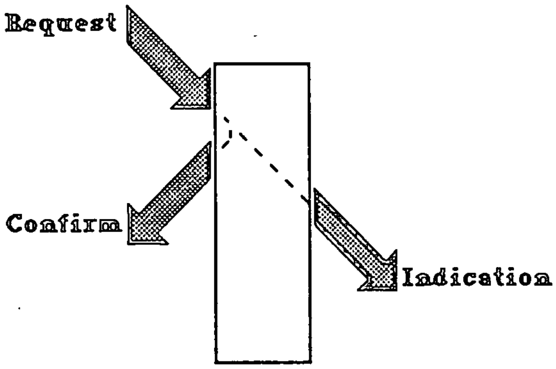


Fig 7.2.9 Primitive sequence associated with c-less data transfer.

MA_INITIALIZE_PROTOCOL - this set of primitives is used during the initialization of the MAC entity, or can be used to reset it.

.request---- this contains a local MAC address parameter and a desired protocol parameter (ie distinguishes between a head-end station and the rest, the former is required to repeat all data).

.confirmation---- this contains a status parameter which indicates the success or failure of the initialization request.

This set of primitives is not used by Spreadnet. The code sequence to be used over the channel, as well as the "MAC" reset commands and Link Block ID, are supplied by other primitives outlined within this section. Also, since no headends exist in Spreadnet, such a distinction between node types is unnecessary.

MA_SET_TIMER_LIMIT - this primitive is used for the setting of the timers used within the MAC entity.

.request---- this contains two primitives, timer class (identifying the particular timer to be set) and timer value (the maximum value allowable for that timer).

Since these values cannot be negotiated over the network (they are a pre-requisite), they must be set by the network administrator (manager etc) and either loaded by a node user on setup, or stored in ROM. The classes of timer used in the Link Blocks are the same as those identified in the IEEE standard, although their actual values differ due to the network's topology (ie no headend in Spreadnet, see section 6.4.2). The primitive sequence used is illustrated in Fig 7.2.7.

MA_DESIRED_RING_MEMBERSHIP - this primitive is used to specify whether or not membership of the particular ring-network is desired.

.request---- this contains a desired status parameter, which signifies the states in_ring or out_of_ring.

Receipt of this primitive by the Link Block results in it taking appropriate actions to join the logical ring (specified in the standard). The primitive sequence used is illustrated by Fig 7.2.7.

MA_ADDRESSES - this primitive is used to notify the Link Block of the addresses (IDs) to be associated with itself on the logical ring. These addresses must include the unique Control Block ID, the broadcast address (used by all of the nodes on the network) and as many group addresses as desired.

.request---- this contains a parameter which specifies the node's unique Control Block ID, with another specifying the remaining set of addesses to be associated with the node.

On the reception of a data header, the Link Block identifies the destination address. If this does not correspond to any in its address table, the data is not passed to the Control Block. This primitive sequence, not specified by IEEE 802, is illustrated by Fig 7.2.7.

MA_DUPLICATE_ADDRESS_DETECTED - this primitive is used to indicate the presence of other nodes having the same address as the local node.

.indication---- this primitive is parameterless.

This parameter is sent to the Control Block when a duplicate address is detected, an error which is reported to higher layer protocols. The CB then provides an alternative ID if this primitive is received during entry to the logical ring. In this way, the unintentional occurence of this problem is resolved. The primitive sequence associated with this process is illustrated in Fig

7.2.5.

7.2.5 Interface 6 - Link Block - Tx/Rx Hardware Interface.

As can be seen from Fig 7.1.3, this interface is common to both connection service types. The primitives used in all of the IEEE standards are not directly applicable in the case of Spreadnet, due to the unique nature of some of the parameters passed and that the data is transmitted at the word level (bit level in the standards).

TXRX_DATA - this set of primitives is concerned with the passing of data between the communicating Link Blocks.

.request---- this contains a data parameter, comprised of a "word" of data, whose octet number is implementation dependent.

.indication-- this contains the same data parameter as the request primitive.

The primitive sequence associated with this operation is illustrated in Fig 7.2.1.

TXRX_SS_SEQ - this primitive is used to load the spread spectrum sequence pair into the Tx/Rx block.

.request---- this includes a transmit sequence parameter and a receive sequence parameter, which contain the code sequences to be used within the transmitter and receiver circuitry respectively.

The primitive sequence associated with this operation is illustrated in Fig 7.2.7.

TXRX_SS_THRESH - this parameter is used to initialize or modify the threshold parameters associated with the Rx hardware.

.request---- this includes a threshold parameter (coarse or tuned) whch updates the value to be used.

The primitive sequence associated with this operation is illustrated in Fig 7.2.7.

TXRX_RESET - this primitive is used to reset the Tx/Rx hardware block to a known state.

.request---- this includes a type parameter which selects the degree of reset required.

The reset can be of two degrees. The first of which involves the purging of all data within the buffers, correlators etc and the resetting of any counters used. In addition to these measures, the second degree involves the resetting of the memory containing the code sequences in use. The primitive sequence associated with this operation is illustrated in Fig 7.2.7.

TXRX_DELIM- this set of primitives is used to add delimiters to the outgoing data stream and to alert of their reception.

.request---- this includes a field signifying if a delimiting flag or an abort sequence is to be transmitted.

.indication-- this includes a field signifying if a flag, abort or idle sequence has been detected on the channel.

The meaning of these data sequences is discussed in Section 8.1. The primitive "sequence" associated with this operation is illustrated in Fig 7.2.7.

Chapter Eight

Node Component Structure and Protocols

8.1 Logical Machines

8.1.1 Introduction

The specification of the internal structure of a data communications layer has been accomplished recently by partitioning it into loosely coupled functions (see IEEE 802.4). In this way, a number of asynchronous logical "machines" were identified. Fig 8.1 shows the partitioning that resulted within the 802.4 standard. It was seen to be convenient to use such a scheme within the specification of Spreadnet, since it allowed the inclusion of machines defined elsewhere, as well as the use of machines in a number of components. Each of the machines within the node component will be specified by a description of the tasks that it is required to perform, with a specification of any data frames used in communications with peer machines in other node components. A specification of any protocols used during such communications will be made in the following section. A detailed explanation of the working of the IEEE 802 protocols and logical machines is not given here [see ref 3.16], this section being concerned with the presentation of machines and frame structures unique to spreadnet, along with their inter-relations (including the inter-relations with machines defined and/or promised within the IEEE 802 work).

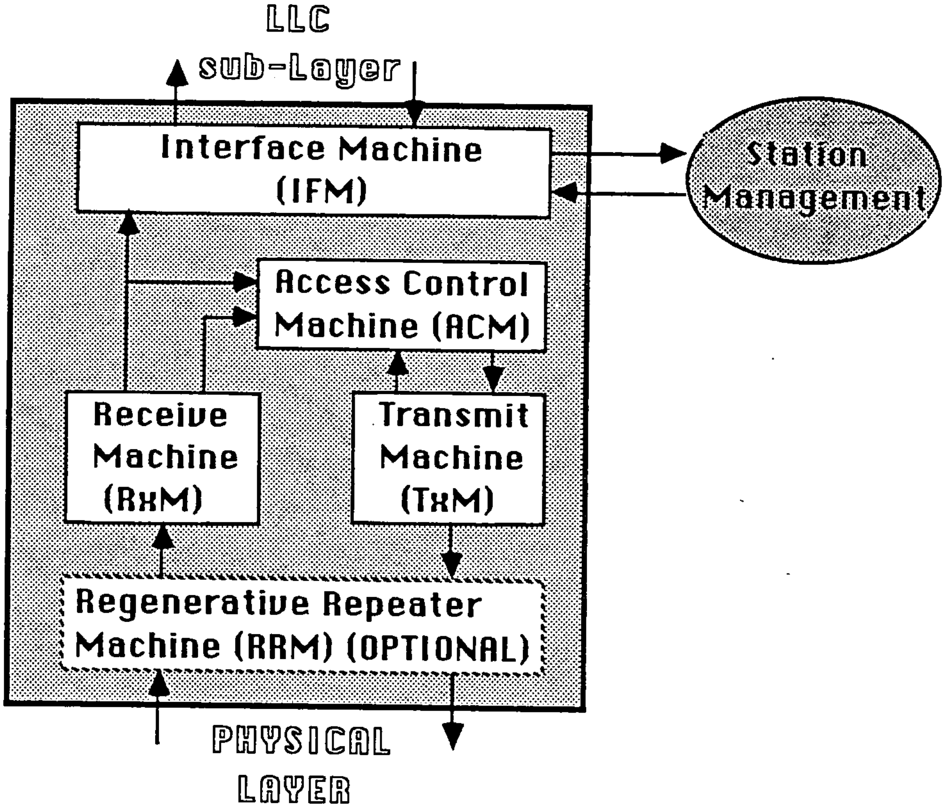


Fig 8.1 Logical Machines within IEEE 802.4 MAC Specification.

8.1.2 Connectionless Channel Link Block

As mentioned in sections 6.4.2 and 7.2.4, the protocol defined within the IEEE 802.4 Token Ring standard were seen to be ideally suited to the connectionless channels used within Spreadnet. Since the service presented to interface 5 is comparable with that provided by the 802.4 MAC sublayer, the basic internal structure of the link block (see fig 8.2) is similar to that of the 802.4's MAC (see fig 8.1). However, only the Access Control Machine is the same in both cases, since tasks associated uniquely with Spreadnet are undertaken by each of the others.

The description of the **Access Control Machine** (ACM) given here is derived from that given in the 802.4 text. By co-operation with other such machines associated with a given channel, the handling of the token used to control transmission access onto the shared channel is accomplished. The optional capability of supporting multiple levels of "quality of service" is not included in Spreadnet, the use of priority levels being considered sufficient for its requirements. The ACM is responsible for the utilization and maintenance of the logical ring around which the token is passed. This includes the admission of new stations to the ring, and a special case of this, the initialisation of the ring. It is also responsible for the detection of, and where possible, recovery from faults and failures in the token-bus network.

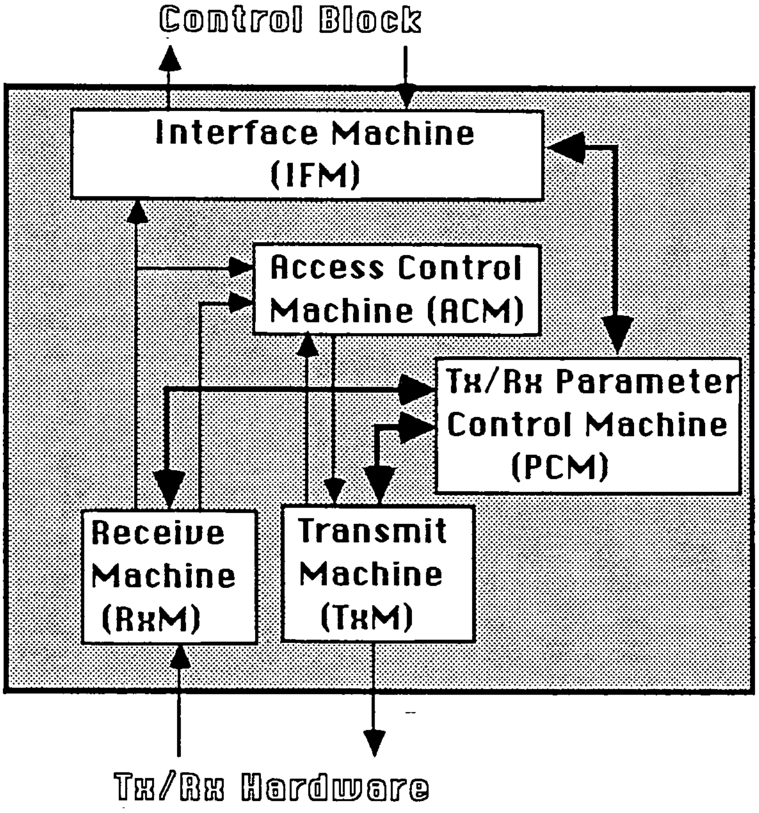


Fig 8.2 Logical Machines within the Connection-less Link Block.

The format of the frames used for the transmission of the data passed between the ACM machines is also similar to that defined by the 802.4 standard, but a number of important differences exist. Fig 8.3 illustrates the general format for the 802.4 frame. Spreadnet does not use the preamble field, since synchronisation is accomplished by the correlator. It does not use the same form of frame delimiters because the correlator within the Spreadnet receiver only signals the reception of detected data, so that the use of non-data signals (ie signalling patterns which cannot be formed by data streams) on the medium is not suitable (since the correlator will merely continue the search for a valid data bit). Therefore, use of a delimiting flag of the type used in SDLC/HDLC communications systems (the same at the start and end of the frame) has to be used. Since the flag consists of the octet 01111110, the transmission of more than five consecutive 1's is not allowed elsewhere within the portion of the bit stream representing data. This is accomplished using the technique of "zero bit insertion", where a 0 bit is inserted into the bit stream whenever necessary to prevent such a flag sequence occuring (the receiver strips this 0 bit from the stream to recover the original data).

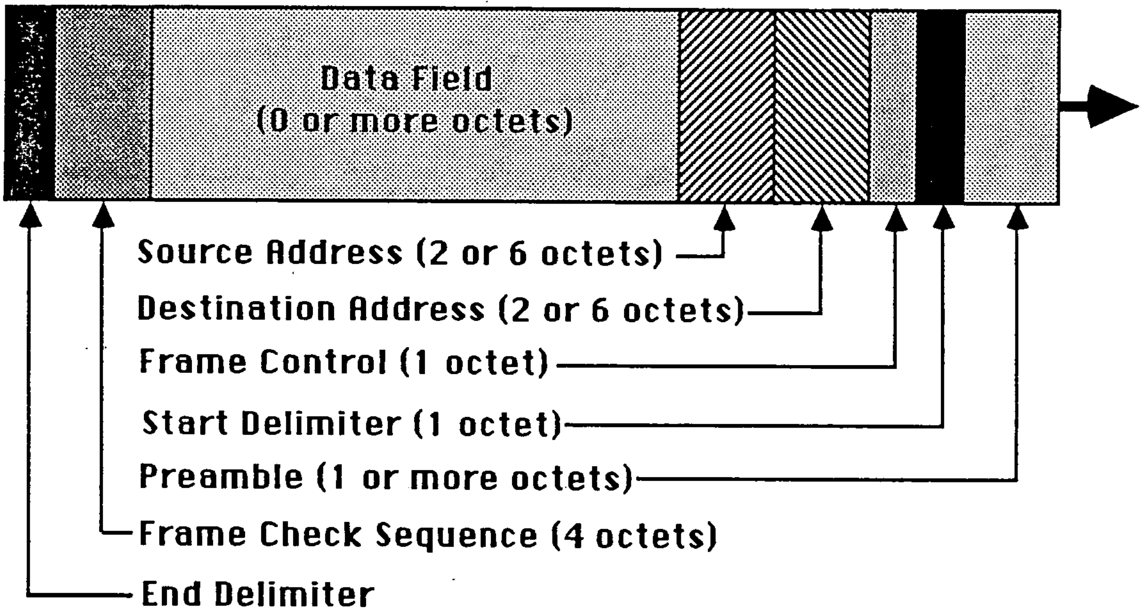

Data Field
(0 or more octets)
Source Address (2 or 6 octets)
Destination Address (2 or 6 octets)
Frame Control (1 octet)
Start Delimiter (1 octet)
Preamble (1 or more octets)
Frame Check Sequence (4 octets)
End Delimiter

Fig 8.3 IEEE 802.4 Generalised Frame Structure.

This technique also allows the signalling of an abort command, by transmitting seven or more consecutive 1's onto the line, which cannot be mistaken for data. This is used because the 802.4 abort sequence also uses non-data signals. The final "non-allowed" sequence used is comprised of a stream of more than fifteen consecutive 1's, which is used to indicate the idleness of the line (this is necessary in Spreadnet in order to stabilise the noise environment, so that threshold values do not have to be changed frequently). Within a Spreadnet node, zero-bit insertion is accomplished within the Tx/Rx hardware, since it is only within this part of the node that the data is formed into a bit-stream. The hardware recognises the flag, abort and idle sequences and passes a signal to the link block of their occurence. If they are detected off an octet boundary, then bit loss has been detected. The incomplete data word that is under assembly for later transmission to the link block, is padded with 0 bits and transmitted. The error will then be detected by software within the link block. Therefore, the generalised frame used within

Spreadnet is as illustrated in Fig 8.4. The schemes used for network addressing are detailed in Section 8.6.

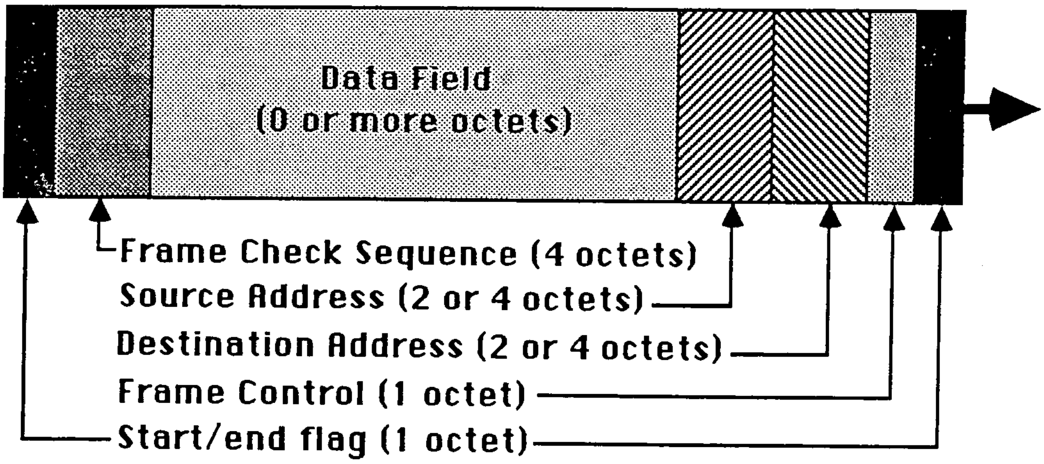

Data Field
(0 or more octets)
Frame Check Sequence (4 octets)
Source Address (2 or 4 octets)
Destination Address (2 or 4 octets)
Frame Control (1 octet)
Start/end flag (1 octet)

Fig 8.4 Generalised Spreadnet Connection-less Frame.

The Frame Control field is used in the same way as specified within 802.4 (ie identifying LLC Data Frames and MAC Control Frames), the field identifying MAC Management data frames remaining undefined (see Appendix B). However, use is made of the Control field type which was reserved for future use, in order to identify messages associated with the **Parameter Control Machine** (PCM) in systems where the variation of the SS sequences during a connection is required. Frames containing this field are passed via the IFM. The format for this control field type is illustrated in Fig 8.5. The absence of a priority field within the octet is due to the automatic allocation of the highest priority to all such messages. Bits 2 and 3 within the field are used to identify the command. Within CHANGE_SS_SEQ and STOP_CHANGE FRAMES, bits 4 through to 7 identify the sequence length selected (ie the sequences are divided into sets , which enable 15 sequence lengths to be used over the network for the purposes of throughput optimization.).

Within the SYNCH_CHANGE frame, these 4 bits are used to notify the time limit imposed for the completion of the change (in pre-defined multiples of the slot time) after the end of the control frame itself (ie this node stops transmissions during this time interval, as does the receiving node(s)). The STOP_CHANGE frame is defined in order to prevent the possible case where the thresholds used to define when sequence lengths may/must be changed are crossed more than once during the period between the transmission of the other two primitives. Therefore, processing time is saved during such occasions, since multiple changes will not be required in rapid succession. It should be noted that this situation is unlikely, since the thresholds used for lengthening and shortening the sequences are not the same, the latter having a smaller value.

Apart from the selection and loading of the SS sequences into the Tx/Rx hardware, the PCM is concerned with correlator threshold tuning and the initial search for the channel sequence in use. The latter is necessary for the MC link-block in Initialisation mode, because the node has no knowledge of which of the pre-assigned set of sequences, exclusively assigned to the MC, is in use. This is solved by loading a sequence and monitoring the medium (for a pre-assigned period) for its presence. If the sequence is not detected, then the next sequence in the set is tried. Since this new node has no knowledge of the number of users on the medium, it initially sets its threshold level to the sequence length itself (ie 100% correlation required) on the first search through the sequences. If no correlation (hence no valid frame) is obtained on the first pass, the percentage value is lowered and the process repeated until the sequence is found. If the entire search is completed with no correlation obtained, the process is repeated a further n times (n may often be 0), after which the node assumes that it is the only active member on the network. It therefore proceeds with the 802.4 initialisation process using the shortest pre-assigned sequence and a required correlation value of 100%.

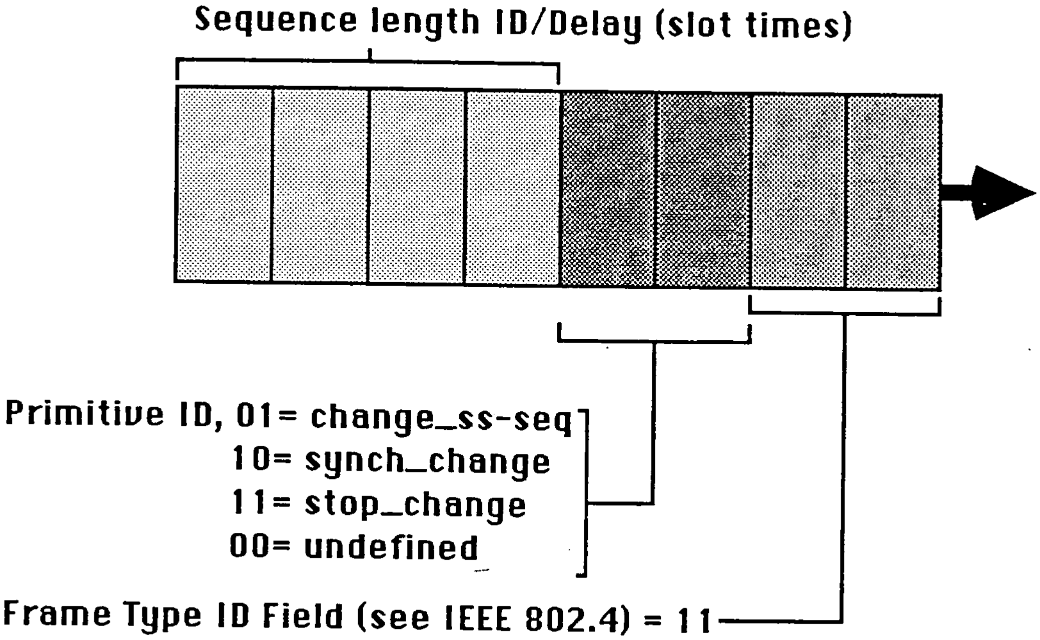

Sequence length ID/Delay (slot times)
Primitive ID, 01= change_ss-seq
10= synch_change
11= stop_change
00= undefined
Frame Type ID Field (see IEEE 802.4) = 11

Fig 8.5 Control Field Passed between peer C-less PCMs.

The tuning of correlator threshold values is accomplished in the following way. When the receive machine detects a frame in error, this result is passed to the PCM and logged. If this condition remains for more than a pre-defined time, the threshold value is "scanned" (ie lowering then raising the value by the same, ever increasing, amount relative to the original value) until a valid frame is received. This new value is sent to the Control Block. If tuning does not rectify the problem, an error message is sent to the Control Block (ie within status parameter of the SS_SELECT.confirm primitive).

The **Interface Machine** (IFM) within this link block acts as an interface and buffer between the Control and Link blocks. It also routes data to the relevant machines within the LB (eg information within SS_SEQ .request primitives to the PCM). It carries out all of the signalling required to enable the successfull passing of the interface primitives between the Control and Link blocks. It is required to interpret all of the incoming service primitives, commanding and/or passing data to the various machines when relevant. Conversely, it is required to assemble data for the .indication and .confirm primitives from these machines when appropriate. It is also required to organise the data into multiple queues, according to their priority, for selection by the ACM.

The **Transmit Machine** (TxM) is responsible for the reading in of frames from the ACM and then passing them, in the form of data "words" to the Tx Hardware. It is required to provide all of the signalling associated with the successful completion of the .request service primitives passed across Interface 6. It does not add delimiters to the frame as in the 802.4 case, since this has to be done within the Tx/Rx hardware after passage through the zero-bit insertion circuitry. However, it does supply the timing signals required for this action to occur. In common with the machine of the same name in the 802.4 specification, the machine is responsible for the calculation of the FCS sequence and its addition to the end of the frame. The primitives passed to the TxM directly from the PCM in Initialisation and Setup mode do not undergo the same manipulation, since they are for purely local use.

The **Receive Machine** (RxM) is responsible for reading in data "words" from the Rx hardware and assembling it into a frame. It determines the validity of the frame by examination of the FCS field, discarding it if it is found to be in error. This information is signalled to the PCM (and IFM if the CB requires such information). The address fields are also examined; If the source address is equal to the node's own address, an appropriate error message is sent. If the destination address does not correlate to the block's individual (in the case of the MC channnel, these represent the node's addresses or group addresses), the frame is discarded.

8.1.3 Connection-oriented Link Block.

The logical machines used within this type of link block all differ from those used within the c-less type, although most of their descriptive names are the same (see Fig 8.6). The PCM is again used for threshold tuning and sequence loading, but not for sequence searching (as the sequences and coarse thresholds are supplied by the CB). Therefore, a section of the software used in the c-less Link block is not included here, but is replaced with software dealing with possible sequence contentions. The possibility of sequence contentions arises due to faulty node hardware/software (on any of the connected nodes) which could result in the improper use of a sequence, or give rise to conditions that compromise the orthogonality of the sequence selected (ie the correlator decodes data from signals not associated with that channel). The latter could arise since the additive nature of the received signal could be affected by faulty sequence transmissions onto the line, resulting in false correlation at some correlation thresholds. In order to test for this condition, the PCM loads the sequence into the hardware and then monitors the line (for a pre-assigned length of time) to see if any data is obtained from it (ie test to see if the receiver has assembled a word of "data"). Since this process is undertaken before this channel has been set up via the MC, then such data indicates that the sequence selected (at the given threshold) cannot be used (this is signalled within the reason parameter of the SS_SEQ.confirm primitive). Since the threshold was calculated from network parameters held in the CB, then checks have to be made there.

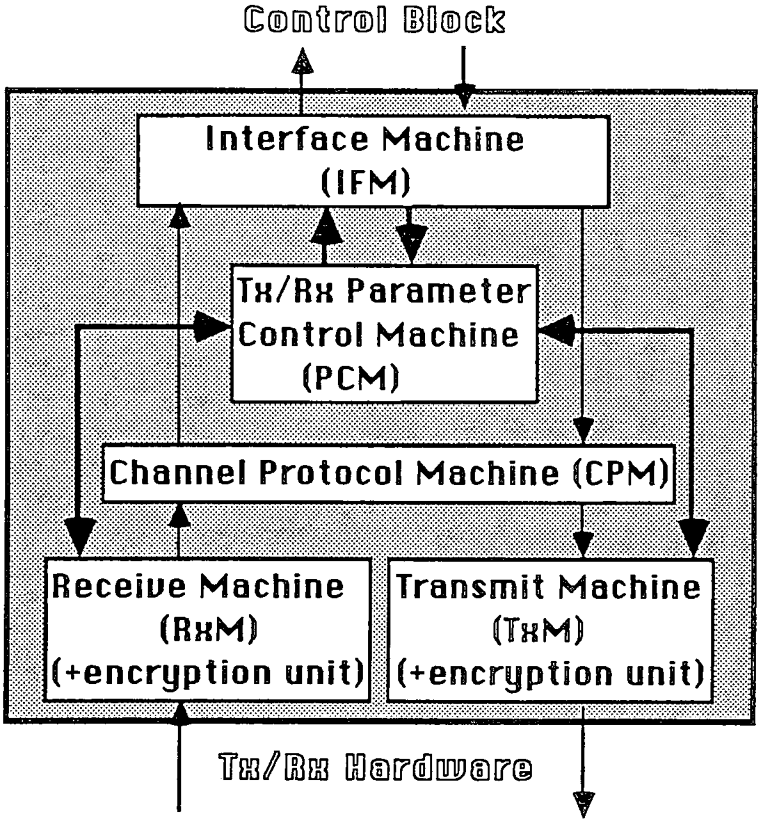

Control Block
Interface Machine (IFM)
Tx/Rx Parameter Control Machine (PCM)
Channel Protocol Machine (CPM)
Receive Machine (RxM) (+encryption unit)
Transmit Machine (TxM) (+encryption unit)
Tx/Rx Hardware

Fig 8.6 Logical Machines within the C-oriented Link Block

The IFM is used for the same purposes within this block as in the c-less case, but also has to deal with the routing of data to either the CB or its allocated I/O block (note that the primitive sequences passed between it and the CB are different in this case anyway). Therefore, whilst some software routines are the same, others are changed or added/deleted.

The TxM and RxM are both used for the same purposes as in the c-less case, but the frames that they assemble are different (potentially much larger with no address fields) and certain functions that they perform are selectable by the CB on channel setup. The frame structures of the data passed over the channel are illustrated in Fig 8.7 (error corrected) and Fig 8.8 (non-error corrected). The same frame delimiter sequences are used as in c-less channels, since the same hardware is used. For the case of error-corrected channels, the data is transmitted within frames of a pre-arranged length (between MCs). The maximum length of these frames is determined by the available memory in the block, used to store unacknowledged frames.

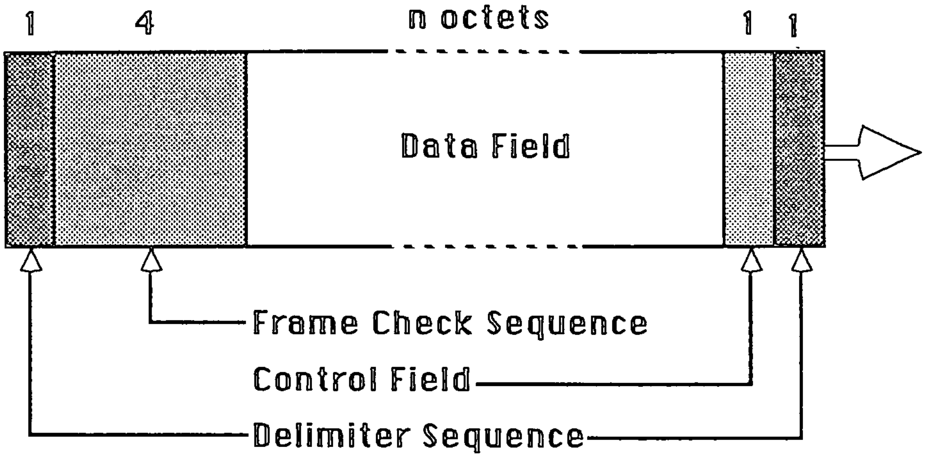


Fig 8.7 Generalised Frame passed over error-corrected channel.

Over channels not requiring error correction, the data is not encapsulated within frames since no FCS field is required. However, the delimiter sequence is used to herald the arrival of a control byte, used for the changing of channel sequence or flow control (either "stop" or "no limit" codes are sent). The abort and idle sequences are ignored in this case. Since the use of link encryption is user selectable, then the hardware/software module concerning this task within these machines can be by-passed. The control field passed over these connectionless channels between peer PCMs is illustrated in Fig 8.9.

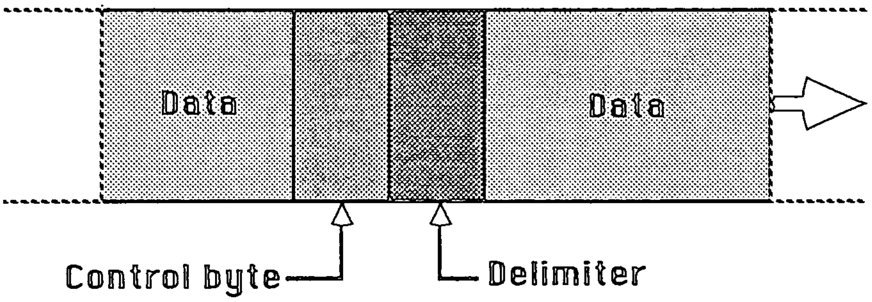


Fig 8.8 "Frames" passed over non-error corrected links.

The final machine within this block is the **Channel Protocol Machine** (CPM), which handles the protocols used over the channel. Since differing protocols are used for the differing channel types (eg full or half duplex), differing software will be loaded into the link block during its setup phase. The simple protocols used over the links are interpreted here, the frame used for this purpose being illustrated in Fig 8.10. The bits used for the transmitted frame's ID and the next received frame's expected ID are sufficient since only two frames can be sent without acknowledgement from the receiving LB (these 'counters" are both reset to zero on link initialisation or reset). This low number is due to the potentially large size of the frames, whilst the storage of only one frame was deemed innefficient since the time required to send the response would be wasted. It is envisaged that the response will be issued before the completion of the transmission of the subsequent frame, enabling back-to-back transmission of frames when required. After the reception of the "frame-accepted" byte, that memory used to store that frame (n) becomes available for the storage of another (n+2). If a "frame-rejected" control frame is received, or if the remote end is expecting a frame which the local end is not about to transmit, then the re-transmission of the appropriate frame is undertaken. (The frame-rejected byte can be used if the frame does not fulfill crieria other than incorrect FCS calculation). The flow control field is used in order to provide the required LLC service (ie tells receiving user how many more frames to pass to it). The codes used within this field are;

a) 111 = no frame limit
b) 000 = frame limit is zero (ie no data after this frame)
c) 001 to 110 = the number of frames after the current one

Since this field only allows specified limits up to six frames, if the limit specified within the primitive passed to the link block (ie L_DATA_FLOWCONTROL.req) is greater than this, the LB defers sending the control field until this value has fallen to six.

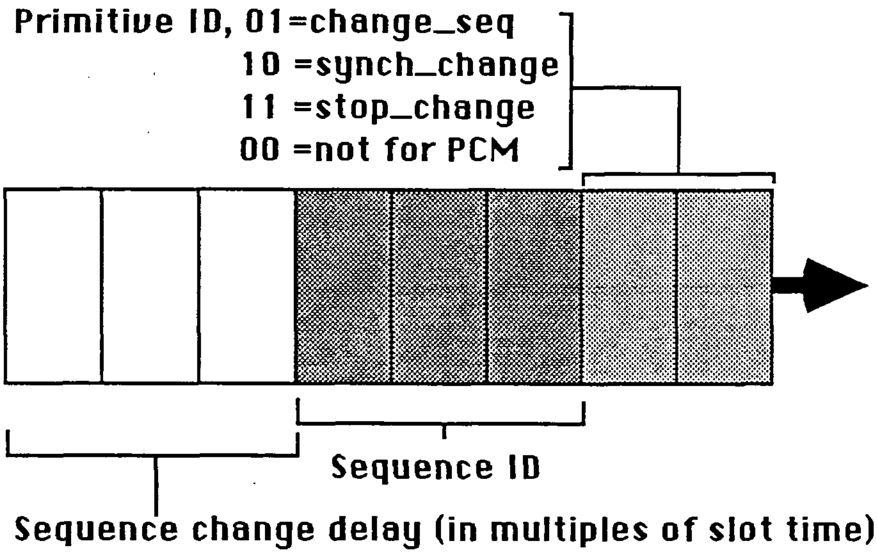

Primitive ID, 01=change_seq
10 =synch_change
11 =stop_change
00 =not for PCM
Sequence ID
Sequence change delay (in multiples of slot time)

Fig 8.9 Control octet passed between peer PCMs.

8.1.4 Input/Output Blocks

Since this block is required to be as simple, hence cheap, as possible, the few functions that it is required to perform do not justify grouping into multiple logical machines. Instead it is termed simply an **I/O Machine (IOM)** which is responsible for the following tasks. The major function of the block is the routing of messages between the network user and either the CB or an allocated link block. In order to do this, it has to differentiate between data and control frames during a c-o link (ie all data is passed via the CB on a c-less link) and so must include a small amount of interpretive software. The sending of the necessary signals for the correct transfer of the primitives over Interfaces 1,2 and 3 is the remaining function. It should be noted that no response at the primitive level is initiated at this block, this being accomplished by the IFMs within the CB and associated LB.

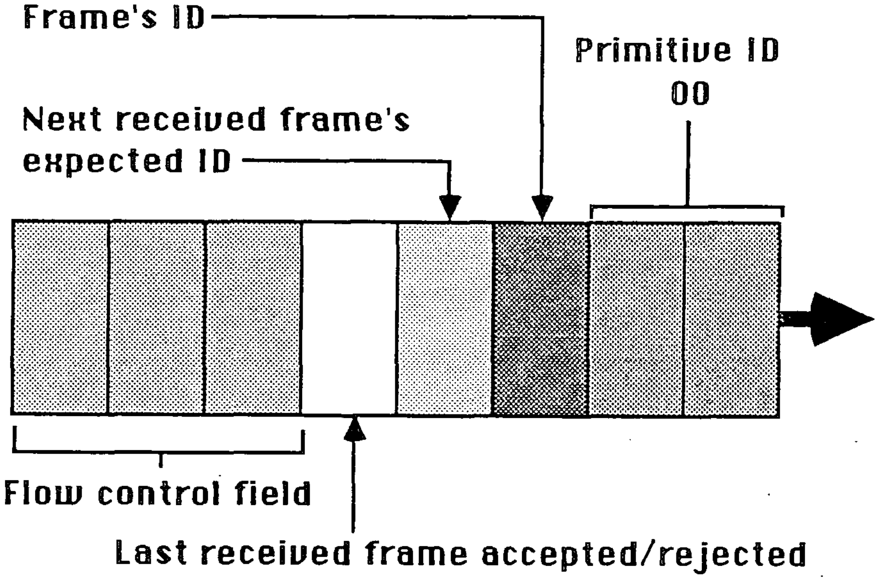

Frame's ID
Primitive ID
00
Next received frame's expected ID
Flow control field
Last received frame accepted/rejected

Fig 8.10 Control Field passed between peer CPMs

8.1.5 Control Block

This contains at least seven machines, of which only the IFM bears any resemblance to those previously described (see Fig 8.11). Additional machines could be added within this block, but are not considered within the text for reasons given later in this section. As just stated, the **Interface Machine (IFM)** performs a similar set of functions to the previously described machines of the same name. However, in this case, it must route data to the individual machines within the CB and route data back to the appropriate IO Blocks when necessary. As can be seen, it cannot route data to or from the machines containing sensitive or network specific information (ie SMM and PEM), in order to aid the properties of network security and transparency respectively. The **Intra-node Communications Machine (ICM)** performs a similar function to the IFM, but on data passing between the CB and the LBs used within the node.

The **Public-Key Encryption Machine (PEM)** is concerned with supplying the SMM with a public key for its own use, whilst also being responsible for the encryption/decryption of any data indicated by the SMM (which also provides the public-keys in the case of decryption). Since the calculation of a set of public and private keys requires much processing, this will be undertaken as a background task, so that a table of available sets is always available to the SMM, when a new set is required (more than one set should be available in case that particular code is in use elsewhere in the network, thus being rejected by the SMM). Precisely what key data is passed between these machines is dealt with in Appendix A.

The **Node Management Machine (NMM)** is resposible for the management of the node's resources (ie IOBs and LBs) and the recovery from local hardware/software failures. It acts as an interface between the SMMs logical view of channels and the node's physical representation of these. Therefore, it is responsible for the pairing of Link and IO blocks during c-o sessions and the mapping of this physical path to the logical connection considered by the SMM and COM. Therefore, changes in (or of) LBs due to hardware/software problems are masked from these latter machines. It is also responsible for supplying the SMM with data that it requires during channel setup (ie whether the node can support a further channel of a given priority) and also supplies data for updating the SMM's local resource tables when node users are added to, or removed from, the network. It also informs the SMM of any problems that have occured within the node that cannot be (or have not been) solved by local measures, thus requiring either network attention (eg duplicate address) or remote solution (eg channel failure).

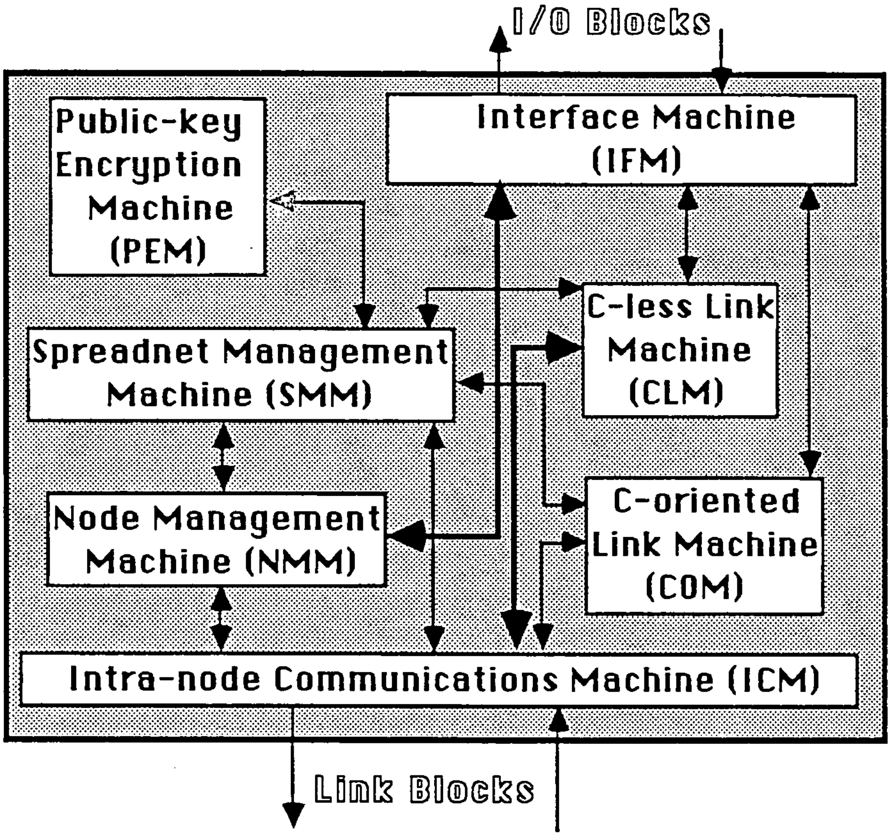

I/O Blocks
Public-key Encryption Machine (PEM)
Interface Machine (IFM)
C-less Link Machine (CLM)
Spreadnet Management Machine (SMM)
C-oriented Link Machine (COM)
Node Management Machine (NMM)
Intra-node Communications Machine (ICM)
Link Blocks

Fig 8.11 Logical Machines within the Control Block.

Since it contains all of the data to ascertain whether a connection from the local node is possible, it deals with such local requests before passing the request onto the SMM, issuing a connection refusal if necessary. Finally, in addition to organising the data flow between blocks, it is responsible for the correct mode setting of the LBs, supplying all of the data necessary (obtained from the SMM when required, such as the sequences to be used) during their periods in initialisation and setup modes. It is also responsible for alerting the LB to halt operations in run mode and to enter either of the other two modes in circumstances such as c-o link route changes, frame-length or threshold updates, or MC block re-allocation.

The **Connection-less Machine** (CLM) contains the SAP Component associated with connection-less data transfer between users and so is responsible for the provision of the LLC C-less service over Spreadnet (using the logical machines within the CB). The protocol state machine it contains is also used by the Station Component within the SMM, when the SAP field identifies it for routing to the SMM and not the IFM. Therefore, the machine may be in operation whilst the SAP Component within it is disabled to node users (this is always the case during node setup and could occur during operation depending on the priority given to such channels on a particular network). The frame used for the protocol between peer CLMs is illustrated in Fig 8.12. (note that the same frame is used between peer station components, where the SAP ID is 000 and the UI frame is not defined). If the XID-c frame is followed by a logical channel identification byte, the corresponding reply frame is followed by ~~n~~ 4 bytes giving the maximum allowed frame size (in bytes) over that channel (the channel ID byte precedes this field). The meaning of the Station Class field values is explained at the end of this section. The test frames may be followed by an information field as in 802.2.

Whilst the frames used between the peer machines are not the same as those in 802.2, their names and functions are the same and give rise to the same results to the user station. Therefore, the State Diagram and State Transition table describing the Type1 SAP in 802.2 also describes the protocol machine within the CLM. It should be noted that when the MC channel is the only c-less channel on the network, all of the data must pass over it at a lower priority than MC data. However, when the second allowed channel is formed, all of the user data is passed over this (whilst only some, if any, MC data is also passed over it).

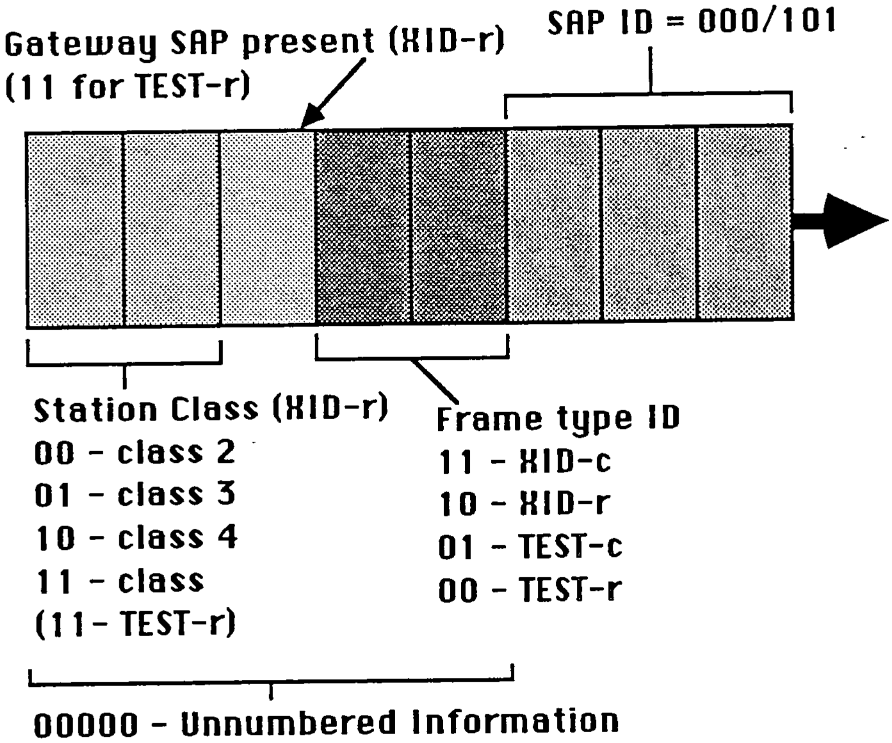


Fig 8.12 Control Field used between peer CLMs.

The **Connection-oriented Link Machine** (COM) contains the SAP Component associated with c-o data transfers, as well as part of the Connection Components of each of the links in progress. It is therefore responsible for the provision of the c-o LLC service to the network users (via the use of the other machines in the block). There is a larger departure from the 802.2 specification in this case, compared to that found in the c-less case, so the State Transition Diagrams and State Tables cannot be used. This large deviation is mainly due to the fact that once the link is set up, normal data transfer by-passes this machine.Since there is no LLC frame structure imposed on the data, all operations associated with these are not relevant during data transfer and so are not required (eg REJ and FRMR). Since the data will be encapsulated within the frames presented to the network, protocols within the users are deemed responsible for such measures. However, the FRMR frame is used (in a different way than within 802.2) to signal problems with messages passed over the MC. This control field is followed by the field causing the problem. This frame is used instead of timeouts, since the solution of problems during a connection is more time-critical than during setup, release etc.

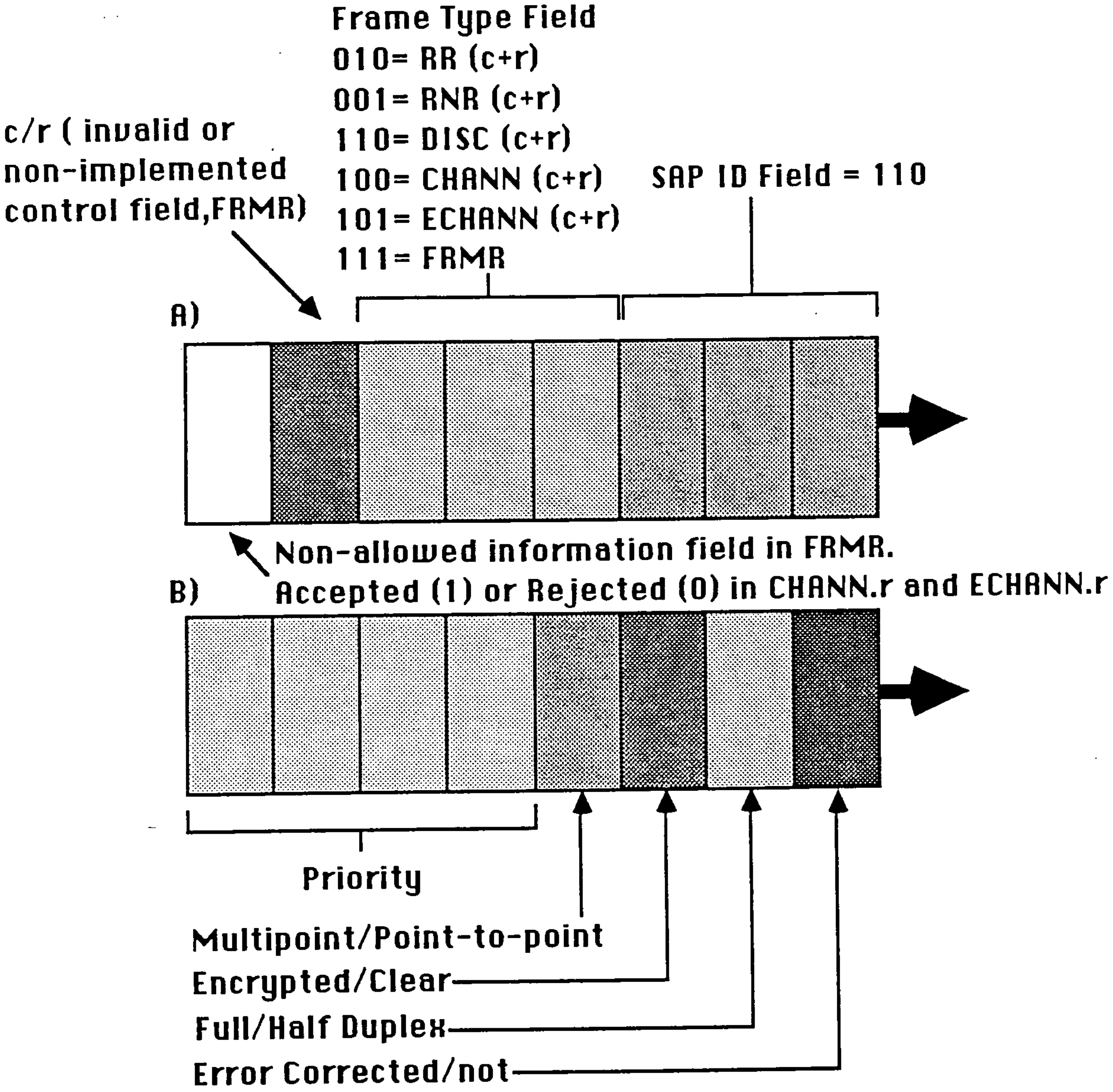


Fig 8.13 Control Field passed between peer COMs.

The frames used for the protocol between peer machines are illustrated in Fig 8.13. The figure shows the 5 types of frames passed between the peer machines. The Receiver Ready (RR) and Receiver Not Ready (RNR) frames provide the same information as in 802.2, as does the Disconnect (DISC) frame (within Spreadnet it is sent in conjunction with corresponding messages between SMMs which disable the channel). These control frames are proceeded by source and destination address fields in the usual way. However, the CHANN$\overset{c}{\text{x}}$ command frame is broadcast, since the initialising node does not know where the destination device is located. Address field expanded to broadcast (2 bytes), device ID (2 bytes), source node (1 byte) & logical channel (1 byte) ~~The device's ID is contained in the byte(s) following the control byte (a '1' in the first bit received indicates that the byte is the last one used)~~. The remaining byte, as illustrated in Fig 8.13.b, follows these and contains details concerning the type of channel required. The reply frame of this type is sent point-to-point, since both ends of the desired "connection" are now known. ~~No extra bytes follow the control byte in this case~~. Finally, both the command and response frames of the ECHANN type (which convey the encrypted channel parameters as discussed in Chapter 6) are followed by a number of information bytes. If the response frame is +ve, then the channel parameters are not returned, only the message source verification field. A _ve response frame contains all of the parameters required. The format of the information conveyed within these bytes is expanded upon within appendix A, along with an explanation of the meaning of the fields.

This machine co-operates with the SMM on channel setup and termination, since the latter has to provide many of the parameters required for the frames used to accomplish these tasks. The protocol frames are sent over the MC by default, but under high load conditions, the SMM can direct them onto the auxiliary channel, if all of the intended recipients of the message are connected to it. The machine contains a c-o protocol machine, with a record of the state of each of the c-o links in progress within the node.

The **Spreadnet Management Machine** (SMM) contains the functions related to the management of the parameters associated with the media access technique, so that such issues are hidden from the node user's equipment. The machine contains two SAPs, the Station Component and the Network Management Component. The former is as specified within the 802.2 document, including the optional duplicate address detection procedures. The frames used by this machine, in association with the CLM, have already been described.

The SMM is responsible for the calculation of the thresholds used in the receiver correlators, the selection of appropriate sequences for channels and the creation of a second c-less channel when necessary. It maintains up to date tables of the local users that it supports (ie their individual and group addresses, type and some verifiable priority value) as well as a table of all of the remote nodes with their public keys and class. It also contains a table of all of the usable sequences allocated to the network, their associated IDs and whether each is in use or not. It is responsible for the re-allocation of sequences to the relevant channels when the sequence length is shortened (ie the order of the "long" sequences in the table is preserved with the "shorter" sequences). It is also responsible for the encryption/decryption of relevant data, directing the PKE machine when appropriate. The control frame that this machine uses for its protocols is illustrated in Fig 8.14

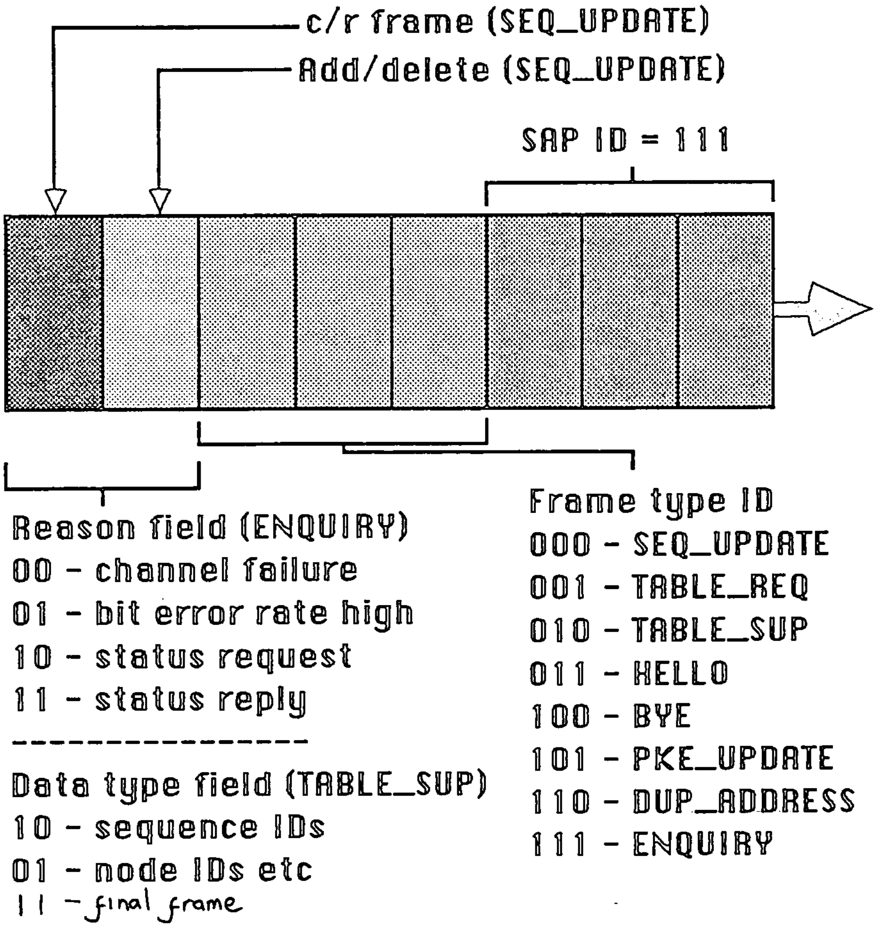


Fig 8.14 Control Field passed between peer SMMs .

The SEQ_UPDATE frame is followed by a sequence ID word(s), identifying the sequences to be added/deleted from sequence tables around the network. This frame is broadcast over the network. The TABLE_REQ frame is not followed by any auxiliary information and is transmitted to the node which is its predecessor within the MC's logical ring. The TABLE_SUP frame is followed by differing data fields depending on the data type bit within the control frame. If the bit is set, the following data consists of sequence IDs, one per octet pair, which are the sequences presently in use. If the bit is cleared, the data consists of sets of bytes containing a one-byte node ID field followed by a further class and PKE data field (specified in appendix A). The frame is directed to the node which requested the table (ie it is not broadcast). The HELLO frame, which is used to inform the SMMs around the rest of the network of the addition of a new node, is followed by the address fields ~~byte long node ID~~, then a data field containing the node service class and PKE information (as in TABLE_SUP frame). The BYE frame, which is used to signal the imminent removal of a node from the network, is followed by its address fields ~~byte long node ID~~. If any sequences have still to be released at this stage, then the sequence IDs (two octets each) follow the node ID. Both HELLO and BYE frames are broadcast messages.

The PKE_UPDATE frame is followed by a data field containing all of the relevant information required to specify the new public key to be associated with the node (see appendix A). The DUPLICATE_ADDRESS frame is followed by the node ID in question (one byte). The action taken on receipt of this information is unspecified at present, but is anticipated to be signalled to the user via the LLC management primitives (as yet undefined). Both of these frames are broadcast messages. The ENQUIRY frame is not followed by any further data if the "channel failure", or "bit error rate high" options are signalled. These merely draw attention to problems which are subsequently investigated by the receiving node. When the "status request" option is specified, followed by a channel ID byte, the receiving node replies with the "status reply" option set, followed by the same channel ID byte and then the channel parameter byte sent after the CHANN control frames related to that channel (this is conceptually similar to the XID frame, but is specifically used for channel fault recovery by the SMM).

8.1.6 Related Information

The meaning of the values describing the Class of Node, passed within the XID frame, describe the selection of logical machines within that node. It should be noted, that in addition to the four SAPs described in the text, three further SAPs and their logical machines have been identified for future systems. The first machine supports the, as yet unspecified, type 3 LLC service and is named the **Type Three Machine** (TTM). Similarly, the next machine of the trio supports the type 4 service, also yet to be specified. This machine is named the **Type Four Machine** (TFM). The remaining SAP is concerned with inter-network gateways. Nodes supporting a LEG gateway between multiple Spreadnet networks contain a **Gateway Machine** (GWM). As detailed in Chapter 3, other gateways to other forms of network are accomplished via DSGs, within the transport layer. Therefore the classes of nodes are as follows, with each being able to support a GWM, so giving 8 classes in all.

Class 2 - supports type 1 and type 2 services (as defined in 802.2).
Class 3 - supports type 1, type 2 and type 3 services.
Class 4 - supports type 1, type 2 and type 4 services.
Class a - supports all four types.

The addressing scheme used within Spreadnet results in the variation of the contents of the address fields within the Spreadnet frames during the "lifetime" of a communications link. The null address is placed within the destination field when the frame is to be broadcast (eg "Hello" frame and all c-less data frames), the field being two octets in length. When associated with a CHANN-c frame, the broadcast address is ~~modified to be 1 (ie lsb is one)~~ ZERO and the subsequent octet pair consists of the user device's individual or group ID (ie the destination field is increased in length). In both of these cases, the source address field is the node's address on the MC (one octet) followed by a null octet (ie not associated with a specific logical channel).

Directed frames contain the destination node's ID (one octet) followed by the number assigned to the logical channel associated with the connection (one octet) by the destination node (except in the case of CHANN-r frames when this has not been assigned, thus the null address is used). The former of these octets cannot contain the null value, thus avoiding possible confusion with broadcast addresses. This scheme was used since the number of nodes on a single LAN was not anticipated to rise over 255, nor was the number of logical links to each expected to rise above 256. In this case, the source address similarly contains the node and its own local logical channel ID.

Using the information given within this section, it is now possible to give a global view of the procedures involved in the changing of the sequence lengths used within the network. The node which wishes to alter the length of the sequence in use broadcasts a SEQ_UPDATE.c frame (with the 'add' bit set) followed by two null octets. This alerts the other nodes to the request and solicits a response from them in the usual way (see section 8.2.8 regarding the acknowledgement procedure). The actual procedures related to the sequence changes are begun by the broadcasting of a CHANGE_SS_SEQ.c frame over the MC. The changes are synchronised by the subsequent broadcasting of a SYNCH_CHANGE.c frame, which details a sufficiently large delay field to enable the changes of all of the other links to be completed (ie the units of delay are of a larger size when associated with the MC, as compared with that used for the individual links. The initiating node is also responsible for the change on the second broadcast channel if it is in use). After the delay has elapsed, the initiating node broadcasts a second SEQ_UPDATE.c frame, with the same attributes as the first, over the new MC sequence.

In the event that it does not receive acknowledgements from all of the nodes, then it knows that they will have been removed from the logical ring and so can remove them from its node table and cancel all links with it (a process which will eventually occur within all of the remaining nodes). When these "lost" nodes recover from the problem (using MC timeouts for not receiving the token) and find the new MC sequence, (ie they do not generate their own token in this case before testing all other MC sequences) they enter in the usual way, with all of the previous connections reset as a result (if not done already by the other nodes). Since the calculation and storing of the new sequence lengths occurs between the sending of the SS_SEQ.c frame and the associated acknowledgement, the links in use at the time are disrupted for the shortest possible time.

In the situation where the time taken to begin a new connection is critical, the lengthening of the code sequences can be speeded up in the following way. As previously stated , the sequences are changed to those holding the same position in a pre-assigned order of sequences in their families (ie each sequence is assigned an ID which indicates its assigned position in that family, see Fig.8.15). Therefore, it can be seen that at any given time a sequence of length N can be linked to a sequence in every family whose sequence length is greater than N. Therefore, multiple sequences could be loaded into the link blocks at any one time, saving recalculation time when sequence lengthening is required. Off-course, if the sequences relative position changes after sequence shortening occurs, this set of sequences must be entirely updated. Since the shortening of sequences increases the throughput of the links and does not restrict access to the network, the longer recalculation and loading time was considered acceptable.

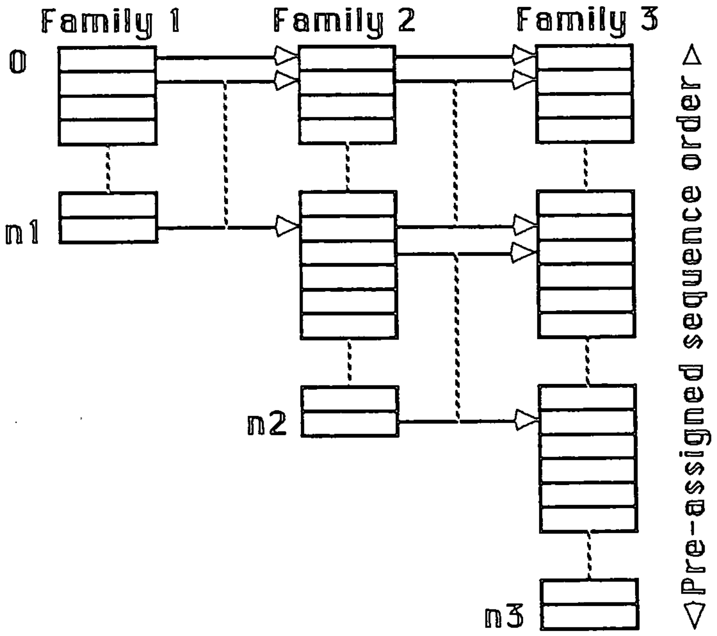


Fig 8.15 Correspondence of sequences between different families of codes.

8.2 Spreadnet Protocols

8.2.1 Method of Specification

Within this section, the protocols are specified in a manner similar to that used within IEEE 802.2. This consists of a state transition diagram, indicating the allowed transitions, accompanied by a state table which elaborates on the conditions for the transition and the actions which accompany them. In this way, the diagram is simplified and hence can be more readily understood. The tables used within this text are of a more informal nature than those presented within 802.2, since their primary aim here is to aid in the protocol's clear presentation.

This section deals primarily with the inter-node protocols used, only including intra-node signals when this is essential to the former's presentation. Actions or data originating or destined from/to parts of the node, other than that which is the primary unit under discussion at the time, are distinguished by being written in *italics* within the tables. Only the protocols designed specifically for Spreadnet are detailed here, the 802.4 protocols used for the Spreadnet ACM protocol are not included here, since they are well documented in the relevant IEEE text [see ref 3.16]. A more informal introduction to the network's protocols is presented within sections 6.3 and 6.4.

8.2.2 PCM protocol.

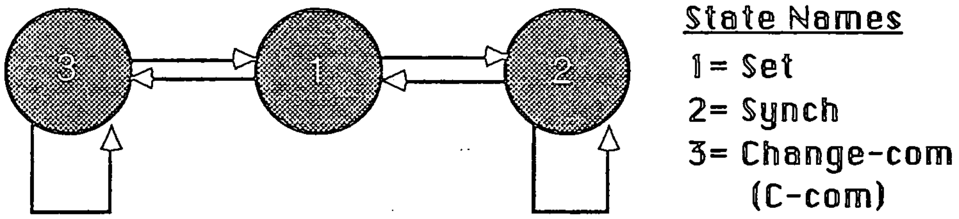


Fig 8.2.1 PCM protocol State Transition Diagram.

This protocol is used to synchronise the changing of the code sequence used over the channel in question. The "initiator" (see 7.2.4) sends the commands associated with this process, the remote node sending responses in the case of point-to-point, or nothing in the case of multipoint links. The state transition diagram associated with this protocol is illustrated by Fig 8.2.1, with the associated state table labelled as Table 8.1.

TABLE 8.1

Present State	Event	Actions	Next State
Set	receive CHANGE_SS_SEQ.c	Start timer for synch command, send ack, then halt data transmissions after completion of current frame (ie ack).	Synch
	SS_SELECT.r(initiate)	Transmit CHANGE_SS_SEQ.c and start timer for ack.	C_com
	SS_SELECT.r(not)	Start timer for change. If timeout *signal failure via SS_SELECT.i.*	Set
	Any other frame	No action.	Set
Synch	receive STOP_CHANGE.c	Stop and reset timer, send ack.	Set
	receive SYNCH_CHANGE.c	Restart timer with synch delay. *Send TXRX_SS_SEQ and _THRESH.* Data received before synch delay elapses is ignored. Send ack.	Set
	timeout for synch comm'	Enter in error log.	Set
	Any other frame	No action.	Synch
C_com	*SS_SELECT.r(initiate)*	Transmit STOP_CHANGE.c. If has now returned to original value then no more action Any other value, actions as in Set	Set
	receive CHANGE_SS_SEQ.r	Reset timer, send SYNCH_CH'GE.c with the final data frame and set timer to ack+synch delay. *Send TXRX_SS_SEQ and _THRESH.* If timeout, retry count is incremented if less than limit (and frame is re-sent), else signal failure via SS_SELECT.c. If ack received on new channel, signal success via the same primitive.	Set
	timeout for change ack	Retry if relevant count is less than max' value (inc' count), else *signal failure via SS_SELECT.i.*	Set
	Any other frame	No action.	C-com

It is assumed within this table, as in all subsequent ones, that any timers/counters activated during the processing are stopped and reset if their awaited target event occurs before they have run to completion. Also, the use of counters to record the number of retries made of any particular action are loaded with a generic "max" value. This will vary from instance to instance.

8.2.3 ACM protocol.

The protocol used here is as specified within the IEEE 802.4 standard, with the exception of the LB startup procedures described in 8.2.5. (ie the transition between the "Unpowered" and "Idle" states is replaced). Therefore the state table etc associated with the protocol is as presented within the 802.4 standard.

8.2.4 CPM protocol.

This protocol encompasses the different courses of action required for the different types of channels supportable between the LBs. The non-error-corrected channels do not store the frames between transmission and +ve acknowledgement, so the protocol is much simpler in this case. The state transition diagram is presented in Fig 8.2.2, the table within Table 8.2. Since frames received in error are discarded (ie if FCS is incorrect in the error corrected frames), the frame window size of 2 is used to detect out of sequence errors, thus indirectly signalling the frame error.

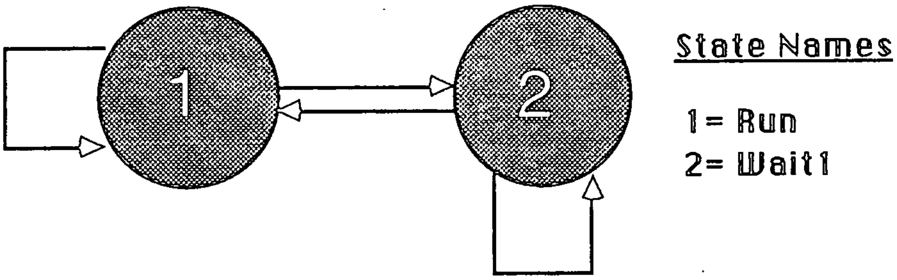


Fig 8.2.2 State transition diagram for CPM protocol.

TABLE 8.2

Present State	Event	Actions	Next State
Run	*LB_DATA_CONNECT.r (no error corr' or m'point)*	Transmit delimiter, then data until the end of primitive's data field (ie not stored in LB)	Run
	(error corrected frame)	Produce frame header, then t'mit delim', header, data and FCS in frame of max size, unless amount is smaller. Set ack timer after data frame is transmitted. (repeat if data still present on state re-entry and previous frame sent)	Wait1
	receive multipoint (or other non error corrected frames).	Pass data through to I/O block, send LB_DATA_CONNECT.i	Run
	All other frames.	No action.	Run
Wait1	receive frame_a. ack	Purge frame from LB memory, toggle received frame ID.	Run
	receive other frame.ack	If previous frame ack'd, then as for frame_a, else no action.	Run/ Wait1
	(data not all sent)	Send further frame if data still to be sent and <2 frames unack'd. Set timer for new frame, else no action undertaken	Wait1
	timeout for data frame_a (or any other's timeout)	Resend if retry count is less than max', else *signal failure via LB_DATA_CONNECT.c*. Log error.	Wait1/ Run

	receive frame_a. nack (or any other's nack)	(as for timeout).	Wait1
	All other frames.	No action.	Wait1
Run or Wait1	receive data frame	Process, send ack/nack and pass data on if ack, else dispose. If ack, toggle next rec' frame conter. (ie store data until all of frame has been received). If frame is frame_a, *LB_DATA_CONNECT.i.*	Present
	LB_DATA_FLOWCONTROL.c	Set field to appropriate value, if value >6, defer as stated in sec'8.3. Send after current frame if error corrected, or a.s.a.p if not. Start ack timer as for normal frame.	Present
	receive frame, flow control field<>111.	Set sent frame counter to value. *Send LB_DATA_FLOWCONTROL.i.* After alloted number of frames have been sent, halt transm'ns.	Present
	timeout for f-c frame ack	Resend if retry count<max, else *signal failure via confirm prim'.*	Present
	receive f-c frame ack	*Send LB_DATA_FLOWCONTROL.c* to signal success.	Present

8.2.5 Link Block Operating Modes.

For the link-block used for the MC within the node, the states entered are OFF, SETUP, INIT, IDLE respectively. Initially, the hardware is in the OFF state, but when powered up and tested by the NMM during node startup (when network specific software may also be loaded) it is moved to the SETUP state. Here it awaits its mode setting primitive (ie LB_SERVICE.r) along with those associated with the loading of the associated sequences (ie SS_SEQ.r). On entering the INIT mode, the LB searches for the sequence in use. On finding it, the LB enters the IDLE state and the 802.4 initialisation process can begin. For the auxilliary c-less channel, the INIT mode is not necessary, since the relevant data can be provided via the SMM, so it can be by-passed. During the initialization of a c-o channel the INIT phase is not required, and the final state is now the OFF state, described within section 8.2.4. If the LBs receive a new mode setting primitive during normal operation (or an LB_RESET.r (software purge)), then the block is returned to the SETUP state with all records of the previous channel purged from memory.

8.2.6 CLM protocol.

This protocol is as presented within the 802.2 specification, so that the table and transition diagram presented there are also applicable to Spreadnet.

8.2.7 COM protocol.

This protocol, which controls the setup, termination and managing of connection-oriented links, differs from that used within 802.2 because it does not control the actual transfer of the data. Also, since it is also concerned with the setting up of different types of channel, these states are unique to Spreadnet. The state table for this protocol is presented as Table 8.3, with the transition diagram being Fig 8.2.3.

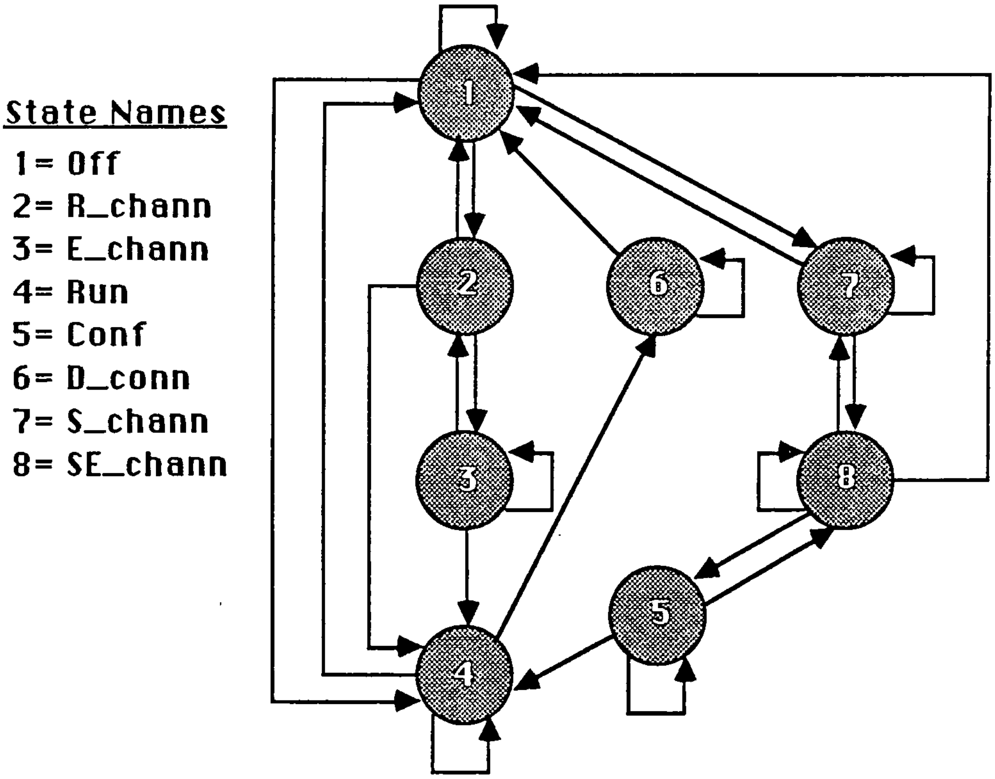


Fig 8.2.3 State TD for COM protocol.

TABLE 8.3

Present State	Event	Actions	Next State
Off	receive CHANN.c (point to point)	Form new conection entity within COM. Transmit +ve CHANN.r after consulting SMM. Start counter.	R_chann
		If link not possible, or not allowed, send -ve CHANN.r and inform SMM.	Off
	(multipoint)	Form new connection entity if SMM agrees, but send no reply. Start timer for seq' del' (SMM)	R_chann
		If SMM does not agree, ignore.	Off
	Channel request (SMM)	Send CHANN.c. Start timer if is point-to-point.	S_chann
		If multipoint, connection is assumed to be complete (ie *SEQ_UPDATE is sent from SMM*).	Running
	All other frames.	No action.	Off

R_chann	receive ECHANN.c	Process, send +ve/-ve ECHANN.r if point-to-point link was requested, else ignore.	E_chann
	Relevant sequences deleted from seq' table.	Multipoint channel confirmed.	Running
	Timeout for seq' del'	Resend CHANN.r if count<max, else reset and inform SMM.	R_chann /Off
	Timeout for ECHANN.r	Resend CHANN.r if count<max, else reset and inform SMM.	R_chann /Off
	All other frames.	No action.	R_chann
E_chann	*Relevant sequences deleted from seq' table.*	Point-to-p't channel confirmed Wait for first data from link initialising node's LB.	Running
	receive ECHANN.r	As in R_chann, since is a retry.	E_chann
	Timeout for seq' del'	As in R_chann	E_chann /Off
	Timeout for ECHANN.r	As in R_chann.	E_chann /Off
	All other frames	No action.	E_chann
S_chann	receive CHANN.r (+ve)	Produce ECHANN.c, start timer.	SE_chann
	receive CHANN.r (-ve)	Inform SMM.	Off
	Timeout for CHANN.r	Resend Chann.c if count<max, else reset and inform SMM	S_chann /Off
	All other frames	No action.	S_chann
SE_chann	receive ECHANN.r (+ve)	Inform SMM to send sequence update to confirm channel.	Conf
	receive ECHANN.r (-ve)	If returned sequences are suitable, then SMM sends seq' update to confirm channel.	Conf
		If not suitable, send ECHANN.c with a further sequence pair if count<max, else inform SMM.	SE_chann
	sequence(s) signalled to be in use elsewhere	Send ECHANN.c with new seq's to avoid sequence contentions.	SE_chann
	Timeout for ECHANN.r	Resend ECHANN.r if count<max, else inform SMM.	SE_Chann /Off
	All other frames.	No action.	SE_chann
Conf	*confirmation of update of relevant sequences.*	Channel negotiations complete.	Running
	sequence(s) signalled to be in use elsewhere.	As in SE_chann.	SE_chann
	Receive *command* assoc' with confirmed chann'.	Signal error to SMM.	Conf
Running	Receive RR.c or RNR.c (n/a to multipoint)	If LB is in Run mode and operating properly then send RR.r, else send RNR.r	Running
	receiver state enquiry (n/a to multipoint)	Send RR.c or RNR.c and await acks for use in *state reply*. Retry on timeout if count<max', else inform SMM of the prob'.	Running
	receive FRMR.r (n/a to multipoint)	Examine reason for rejection. If immediately solvable, then modify and re-send.	Running
		If not, tell SMM. If not solved by SMM, then send DISC.c.	D_conn

	receive a bad frame (n/a to multipoint)	Send FRMR.r and ignore frame.	Running
	receive DISC.c	Inform SMM, then send DISC.r if is point-t-p.	Off
	receive disc' command	Send DISC.c, starting a timer if is point-t-p, otherwise not.	D_conn /Off
	All other frames.	No action.	Running
D_conn	receive DISC.r.	Inform SMM.	Off
	receive RR.c or RNR.c	Send RNR.r	D_conn
	Timeout for DISC.r	Resend DISC.c if count<max, else proceed with disconnect'n	D_conn /Off
	All other frames.	No action.	D_conn

8.2.8 SMM protocol.

This protocol differs from the others mainly because it contains actions involving the broadcasting of data which must be acknowledged. These messages are related to the distributed Spreadnet management functions which are necessary for the correct management of the CDMA technique. The acknowledgement process for these messages (SEQ_UPDATE, HELLO, BYE, PKE_UPDATE) requires the local node issuing the command frame to wait for acknowledgements (always +ve, since messages received in error are discarded) from all of the nodes present on its node table. If all of these replies are not obtained, then the command is re-broadcast in the manner detailed within Table 8.4. Since the network configuration is anticipated to be of a relatively stable nature relative to commercial LANs, the bandwidth consumed during such transactions was not anticipated to be a problem. The state transition diagram for this protocol is illustrated by Fig 8.2.4.

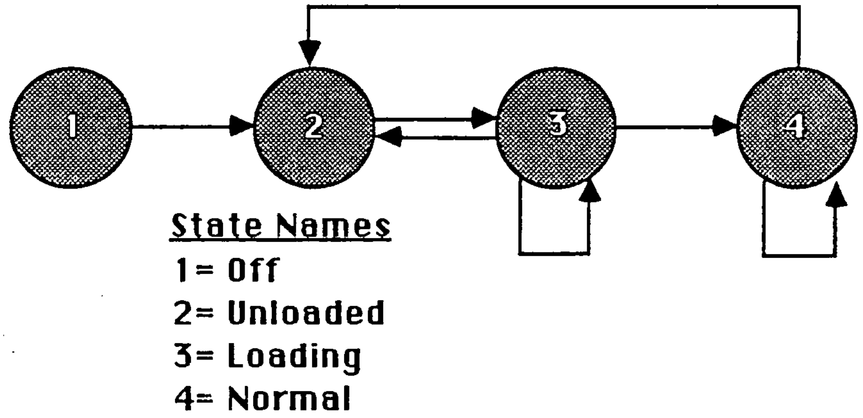


Fig 8.2.4 State transition diagram for SMM protocol.

TABLE 8.4

Present State	Event	Actions	Next State
Off	*Power on*	Start node initialisation	Unloaded
Unloaded	Node init' complete (other nodes present)	Send TABLE_REQ to node which is its predecessor in the logical ring and start timer for reception of the first TABLE_SUP.	Loading
	(only node present)	No action.	Normal
	Any other frame.	No action.	Normal
Loading	receive TABLE_SUP	Process data, send ack.	Loading
	receive last TABLE_SUP	Process data, check for dup_add if present, change own then send HELLO. Wait for acks.	Normal
	Timeout for TABLE_SUP	Retry if count<max else restart after node specific measures.	Loading
	Any other frame	No action	Loading
Normal	Receive DUP_ADDRESS	Send ack (other actions as yet unspecified, see 8.1.5)	Normal
	Receive ENQUIRY(prob)	Activate internal diagnostic routines. Send ack if problem considered solved, else log error (no ack).	Normal
	Receive ENQUIRY(stat)	Send ENQUIRY(stat) response after consulting tables etc.	Normal
	Receive details of link problem or require info	Send ENQUIRY(stat) or (prob), Start ack timer.	Normal
	Timeout for enquiry ack	Retry if counter<max, else terminate link.	Normal
	Receive ENQUIRY.ack	If (stat) then process, if (prob) and problem not solved, take further action (eg renegotiate the link). If solved, no action.	Normal
	Receive TABLE_REQ	Send TABLE_SUP frames, each time waiting for ack, with final frame wait for HELLO as ack.	Normal
	Timeout, TABLE_SUP.ack	Retry if count<max, else abort table send.	Normal
	Change public-key	Send PKE_UPDATE, await acks, resend if count<max if not all nodes in table send ack, else revert to previous code (ie send update for old sequence in the same way (Note. The new ack limit is given by the number that responded to the "failure".	Normal
	Receive PKE_UPDATE	Update tables, send ack.	Normal
	Receive SEQ_UPDATE	Add/delete sequences to/from table. Send ack. If is MC seq', or used on active link,ignore.	Normal
	Successful channel creation/termination.	Send SEQ_UPDATE, await acks. If not all nodes in table send ack, then resend if count<max, else accept and rely on LB	

	pre-conn' tests (sect' 8.1.3) to avoid possible contentions.	
Receive HELLO	Update tables, send ack.	Normal
Receive BYE	Update tables, send ack	Normal
Remove Node from logical ring	Send BYE, await acks. Resend if count<max, else proceed with removal.	Unloaded

Chapter Nine
Conclusion

Section 9.1 Conclusion

As detailed within Section 2.3, a requirement existed for a digital communications network to be used within the next generation of Naval Shipborne Command and Control Systems. After consideration of the LANs in existence, as well as those under consideration for international standardisation, it was decided that this need could best be met by a network specifically designed for the task. However, this decision was made with the proviso that the network could easily be interfaced to equipment which conformed to the ISO's OSI 7-layer model. This would enable Naval procurement of devices from a wide variety of manufacturers, who were anticipated to have taken up the use of the standards by the time that the network would be fitted to a Naval vessel. Furthermore, since the naval vessel was anticipated to have to undergo at least one major re-fit during its active service, addition and removal of equipment, as well as the possible modification of data rates, should not require large alterations to the cabling and/or network hardware and software.

Encoding the data sent over the network's cabling using pseudo- random binary sequences, in order to increase the bandwidth of the signal's power, was seen to offer properties that were desirable in the Naval network. These were especially the antijamming and concurrency properties of the channels, coupled with the random nature of the transmitted signals which makes useful wiretapping by an unwanted observer much more difficult. However, since no provision existed for the use of the technique within LANs in the communications standards, (hence no provision is made, in the interface to higher software layers, for parameters required for a system using this technique) the access mechanism was required to be transparent to the network's users. This resulted in the network being designed in the form of a collection of interconnected Network Interface Units (NIUs), which supported the LLC communications service, as defined by the IEEE 802 committee.

In order to manage all of the tasks associated with the access technique utilising the "Spread Spectrum" technique, Code Division Multiple Access (CDMA), these NIUs were required to contain more processing power than required in units supporting current techniques. The centralisation of the control of these tasks into network controller nodes was not seen to be desirable (although it would reduce the cost of the other nodes), as it would reduce the survivability of the network when a node is lost. Since the interconnection of relatively inexpensive devices was required over the network, the processing power within the NIU had to be available to multiple user devices. By decomposing the node into functional units and devising the network's architecture to support functionally distinct communications channels, the performance penalty imposed on each user due to NIU sharing was minimised.

The Spreadnet network architecture was designed in a modular way for a number of reasons. Firstly, as mentioned above, it allowed the sharing of expensive system management processing power amongst users. It also allowed the system implementor to design the system as a set of co-operating hardware modules, enabling the use of multiple redundant instances of each unit (reliability is a Naval requirement), easing the task of node repair (unit replacement) and allowing system upgrades to be limited to particular units when required (for economy in parts and development time). Modular design was also required because of the number of differing types of communications services required to be supported by the network. Since no single protocol could be used to give the optimum results for each of these types, a selection of these were made available to the network user. In order to efficiently (in terms of network bandwidth and channel performance) transport all of the signals passed between the network users, the tailoring of channels for the type of messages passed was undertaken.

It was noted that exchanges related to network management were predominantly transactional in nature and were transmitted to all of the connected nodes. This traffic would best be served by a connectionless data link service. However, since the Naval application required many dedicated long-term links which were always instantly available, a connection oriented data link service was also required. Therefore, a connectionless channel was made available to all of the nodes connected to the network (the Management Channel, with provision for an additional auxilliary channel of the same type), whilst connection oriented channels would be dynamically assigned to communicating parties. This grouping of the combined management functions for all of the supported channels was possible because of the relative stability in the Naval network's configuration. The loss of nodes during combat situations was seen to provide a possible busy reconfiguration period. In order to avoid the scenario where the MC was at its busiest when timeliness of message delivery was most important, the loss of data channels is handled by the communications channels as well. Discovery of node loss by another node does not result in the transmission of this discovery, thus reducing data bottleneck problems in the scenario. Finally, the system can be configured so that the number of message retries sent over the MC is minimised, again reducing the data traffic passed in the scenario.

The type of connectionless channel to be used for the MC was determined by the need for a determinate upper bound on delivery time of messages, the support of a priority system for the data passed, the use of a bus topology network and the ability to alter the data rate without the need for physical changes to the network topology. The access mechanism for the MC was taken from the IEEE 802.4 token-bus standard, with additional protocols handling Spreadnet specific tasks. This enabled the use of available software for the channel, reducing development time for any implementation.

The provision of link privacy, an important feature within a military network, was provided using a combination of public and private key techniques. The method had the added advantage that it acted as a means of verifying the source of the private key, so that such messages could not be imitated or tampered with by another malicious user. The use of the DES algorithm was suggested because it is available in silicon and has been shown to be breakable only at the expense of very considerable processing time (ie very unlikely to be present on the naval network). The system allows the updating of all of the key sequences used within the network. New private keys are used on channel setup each time (each can be used by more than one channel concurrently, as long as this is purely by chance and hence cannot be calculated by a codebreaker). The need for updating the public key associated with each node was seen as a means of increasing security, but these changes were anticipated to be infrequent, so that the MC bandwidth consumed was not considered to be noteworthy.

Since the users of the network were not envisaged to be experienced computer staff, all of the procedures involved with the initialisation, alteration and maintenance of the Spreadnet network were automated, so that the network would recover from a power failure (ie nodes would re-arrange themselves into a logical ring, calculate sequence lengths and thresholds). Nodes could also re-enable pre-defined links if the Control Block contains an appropriate startup procedure, useful for links such as those between radar sensors and their associated displays. As well as making the Spread Spectrum parameters transparent to the user, measures were taken so that the user could not obtain them. This was in order to increase network security, by not indicating which sequences, public-keys etc are in use at any particular time. This was accomplished by designing fixed data routes between logical machines within the Control Block, allowing protection mechanisms to be built into particular machines on the allowed routes.

Apart from the specification of the Spreadnet network architecture, with explanations of the effects that particular requirements had on it, the work presented within this thesis deals with the following issues. Since the network was required to conform to the OSI model for communications systems, Spreadnet was defined in a similar way to that used by International standards bodies. Therefore, the services provided by one software layer to another were described in a form which is a level of abstraction higher than a description of an implementation. This resulted in the signals being passed across these interfaces to be described in the form of primitives. In this way, conformance to the OSI (or IEEE 802) specifications could easily be seen. Additionally, flexibility in the methods of implementing the Spreadnet specification was allowed, an important consideration due to the rapid development of electronic hardware, and the envisaged time scale for the use of such networks by the Navy. This abstract definition of an interface was used to describe all of the interfaces defined to exist, both externally to the network and within its own "block' architecture. The only exception to this was the description of Network Management signals passed from the users to the Node (and vice-versa). This exclusion was unavoidable because such messages are currently being defined by the standards bodies, so that compliance with these is necessary for overall compliance. However, until the details are finalised, the most that could be done was the indication of the interfaces over which such information would be passed, as well as the frame structures containing instances of such information passed over the network.

In contrast to this abstraction, the protocols used over the network were rigorously defined, as were the frame structures by which individual instances of the protocol machines interact. All of the frames defined are unique to Spreadnet, since the media access technique ruled out the use of methods in use today. The specification also differs from those presented by the ISO and the IEEE, in that Spreadnet uses multiple protocols operating in parallel in order to provide a link-level service. This differs from the usual single protocol used at this level, and reflects the emphasis within spreadnet of parallel processing. In order to clarify the purposes of each of the protocols, the 'logical machines' between which they are passed were described, in terms of the functions that they are required to supply to the node as a whole. Further 'machines' which were concerned with the management of the node itself were also described in this way, although the frames passed between them were not defined, due the abstract form of the internal description of the node.

In order to clarify how the nodes operate, in relation to attached local users as well as remote nodes, informal descriptions of the functioning of the network were presented. Such a section was included to allow the sections concerned with the more formalised specifications to be sufficiently terse to be useful as reference works.

Another section of the work was concerned with the development of a testbed system. This was required in order to obtain parameters relating to the physical layer functions of the network. These, in turn, were to be used within simulations of the network to obtain further parameters required before full scale system implementation was undertaken. The portion of the testbed system development presented is concerned with the Tx/Rx control units. Details of the hardware and software used by these units are given, along with details of their intended use. Although the incomplete development of the Tx/Rx hardware (undertaken elsewhere, see ref 4.15) precluded the production of useful test results, the software was extensively tested for correctness. Routines were produced which could be used in a future implementation of the Spreadnet system.

In summary, this work has resulted in the description of a decentralised data communications network architecture suitable for use within a Naval shipborne command and control system. This architecture enables the use of the required properties provided by the Spread Spectrum access technique, whilst remaining compatible with network-user products conforming to International Data Communications standards (ie the relevant parts of the OSI model). The architecture is specified in a manner which does not preclude the use of new and emerging computer technologies in any implementation. In addition, a testbed system was designed and built to study the properties of the access technique, which had not previously been used in such a manner.

9.2 Future Work

The work presented within this volume provides a framework for the implementation of a Spread Spectrum LAN. Within this, the presentation of the protocols to be used by such a network were presented in a number of seperate sections. Whilst the correctness of the individual protocols were informally verified by techniques such as 'walk-throughs' (where a set of initial conditions and subsequent actions are envisaged and the protocol's progress noted), the fact that network operation is dependant on the correct joint action of multiple protocols operating in parallel means that unforseen problems could exist. The number of possible overall network states is sufficiently large to make informal methods impractical, so that formal verification is required. A number of techniques exist, each being able to verify a number of properties associated with communications protocols, but none have yet been presented which can verify *all* of them.

Due to the large number of states possible for each node, the automation of any technique chosen was seen to be very desirable. The technique normally associated with the "Extended Finite State Machine" approach to protocol specification (of which the method used within the text is an example) is "Deductive Inference". However, since this technique relies to some degree on human intuition, its automation was not seen to be easily accomplished. After consideration of a number of the techniques presented within the technical press, it seems that the most suitable approach to the task would be via the simulation of the network. This simulation would draw on the results of the one constructed elsewhere to obtain physical layer parameters (including bit error rates), so that the channels themselves would not have to be modelled. In order to keep the complexity of the simulator within reasonable bounds, the following approach was planned. A series of seperate simulations for each of the functional blocks would be undertaken and the various parameters regarding their performance obtained. Each of these simulations would also verify the individual (or group) protocols associated with the block. In this way, a 'black box' representation of the block can be formulated, for use in the simulation of the entire node or even an entire network. A further stage in this process could involve the simulation of the individual logical machines within each of the blocks.

Such a simulation could also yield the best methods for communications between logical machines, as well as the performance of differing algorithms used for node resource allocation etc. Finally, the utilization of each of the channels could be studied in this manner, for varying mixes of user and channel types. Particularly, the point at which the auxilliary c-less channel is required to carry MC data would be studied with interest, since the overheads involved in its commissioning require that they are performed in advance of the situation. A preliminary study of such work was undertaken, using the SIMULA language and the DEMOS simulation package based on it. It was seen that such a language would be well suited to supplying the information required from such a suite of simulation studies.

In addition to work associated with protocol verification, the presentation of the Spreadnet Protocols using a rigorous specification technique was seen to be essential for future implementors. Such work would not seek to be as 'user friendly' as that presented within Chapter 8, so would resolve any possible ambiguities. Whilst the 'Extended Finite Machine' approach was considered the clearest technique for describing all of the protocols, they could also be specified completely graphically using petri-nets (although their resultant complexity may counteract the clarity advantage of using a wholly graphical technique). From this, work involved with the actual implementation of a network can be undertaken. Nodes which timeshare a single powerful processor within each block, or even within each node, are seen to be possible in addition to the case where each 'logical machine' is controlled by its own processor. The provision of multiple redundant Control Blocks and Network Highways (as in the case of the ASH Naval network) could also be investigated at this time.

Work associated with the physical layer parameters, which could affect the parts of the system studied within the text, concerns the use of sequences of varying length (or of varying type) at the same time. As far as is known, such studies have only been made on two concurrent sequences which are in 'chip' synchronism. This is due to the extemely large amount of processing time required to obtain the necessary results for each of the large number of sequence combinations possible. Such studies could enable the allocation of shorter sequences to channels requiring greater throughput, with longer sequences allocated to channels specifying higher degrees of reliability. This would also mean that global sequence length changes would not be necessary.

Further work is required on the specification of network gateways, both to other Spreadnet networks and to existing LANs and WANs. Such networks should follow the rough guidelines presented within Chapter 3 of this work. Also, as the standards for the new classes of LLC and the Management protocols are published, their incorporation into Spreadnet should follow.

Appendix A

Data Encryption Measures

a- Secret-Key Cryptography

As detailed within Sections 6.3.4 and 6.4.4, the "cipher feedback" (CFB) mode of operation was seen to be the most suitable choice of the options available via the DES encryption algorithm. This mode of operation transforms the DES from a block cipher (in its basic ECB or electronic code book mode) to a self-synchronizing stream cipher, which operates on cleartext strings of 1 to 64 bits in length (8 bits in our case, since the protocols are byte oriented). This is a form of "ciphertext autokey" (CTACK) cipher where the transmitted ciphertext is used as input for key-stream generation. The secret key is 56 bits in length and is used for encrypting the key-stream in this case. This results in a cipher with inter-bit dependance, but which is self-synchronizing (ie resumes correct operation following transmission errors, after reception of 64 unaffected ciphertext bits in this case).

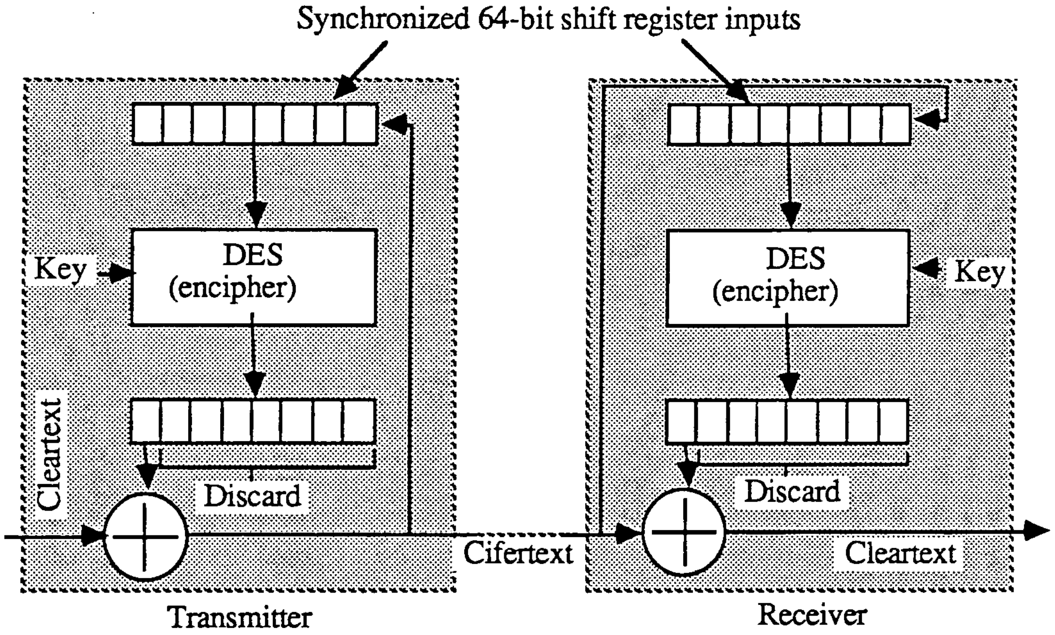


Fig a.1 CFB mode of use of DES encryption algorithm.

The inputted cleartext is combined, by modulo-2 addition, with a matching number of key-stream (output) bits generated by the DES block cipher (any extra DES output bits are discarded). The transmitted ciphertext is fed into 64-bit shift registers that form the input to the basic DES at both ends of the connection, making the key stream a function of all transmitted ciphertext. Fig a.1 displays this configuration of the DES for use with 8-bit bytes. In this mode, as with all CTACK ciphers, cryptographic synchrony is acheived only if the sender and receiver use the same key and both input shift registers contain the same bit pattern. For the latter condition to be satisfied, an "initial fill" for the shift register must be agreed upon between the communicating parties. This could be a network-wide pre-arranged value, or be individual to each communications link. In the case of Spreadnet, it was decided to pass this 8-bit value between the communicating parties during the initialization of the link. This was due to the extra protection afforded at a minimal additional processing cost. Therefore, the parameters that are required to be passed between the communicating parties on link setup consist of a 56-bit encryption key and an 8-bit initial fill, a total of 8 bytes of information.

b- Public-key Cryptography

The only known public-key cryptosystem which offers the property that the encryption (private) and decryption (public) keys can be used for both encryption and decryption of the data stream (ie the encryption key can also decrypt data encrypted with the decryption key). This is necessary for the encryption and validation scheme described in Section 6.4.4. The algorithm used is as follows;

Consider a node, A, with a cleartext message, a ciphertext message C, its public encryption transformation E and its private decryption transformation D.

$E(M) = M^e \bmod n$ ------ encryption transformation

$D(C) = C^d \bmod n$ ------ decryption transformation

Within these equations, n is the product of two large secret prime numbers (p and q) calculated by the node, with e and d being mutual multiplicative inverses modulo the product of (p-1)(q-1). In other words, if (p-1)(q-1)=x, then (ed)/x=1. In this way, d cannot be obtained from the public e without knowing the factors p and q, which are increasingly difficult to calculate as their lengths increase. It can therefore be seen that the information that must be given to a remote node, in order for it to encrypt data, consists of e and n.

c- Encryption related Information transfer over Spreadnet

Within the ECHANN type of frame, passed between peer COMs, there is a field containing the necessary data relating to the private-key encryption of data over the link. As mentioned in section a of this appendix, this amounts to 8 octets of information. In addition to this, the destination node's ID is added to the beginning of this field (one octet), so that when it is finally decrypted using the transmitting node's public-key, correct decryption can be confirmed (thus validating the information's source). A further four octets are required to specify the two sequences to be used (two octets each). The first sequence ID corresponds to the sequence to be used*from* the node issuing the ECHANN.c frame, the second is the sequence to be used in the opposite direction. The negative response frame will contain a different pair of IDs if the original selection is not acceptable for any reason (the positive response frame contains only the destination node's ID, for frame validation). Therefore the field is 13 octets in length as cleartext, but of a variable ciphertext length, depending on its contents and the values of the encryption keys used. Therefore, if the field decrypts to be of a length greater than 13 octets, the message could be ignored or the first 13 octets could be retained whilst all else is discarded. Since this data is sent back to the originating node in the form of a response frame (the node ID is different off-course), the second course of action is taken within Spreadnet. If the data sent back to the originating node is different to that which it originally sent, then the process is repeated (with a pre-arranged limit before the protocol is reset).

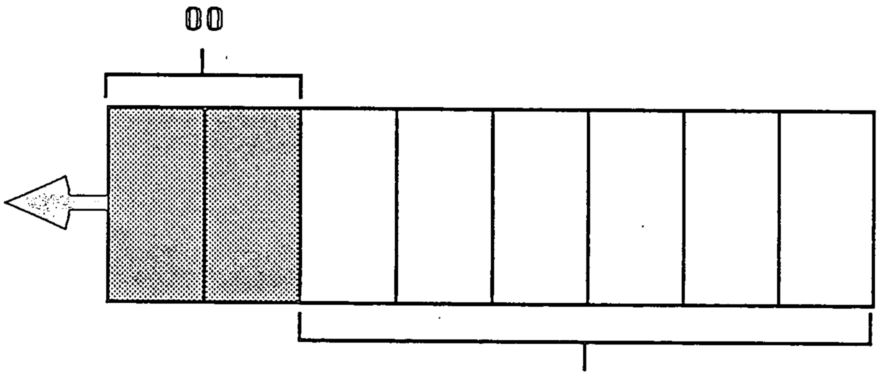
As stated in the previous section, the values of e and n have to be passed to specify a node's public-key. The PKE_UPDATE frame contains this information. Since the length of this field varies with the length of the prime numbers used to generate them (the greater the length, the harder it is to factorise their product, n), this field will be implementation and/or network specific (depending on the degree of link privacy required within a given system). However, the order of their transmission will be fixed as e followed by n. The TABLE_SUP and HELLO frames also contain this field, but in these cases e is preceded by a 3-bit class identification field (as within XID-r).

Appendix B

IEEE 802.4 Frame Control Fields

Within section 8.1, the use of the undefined frame control field was detailed. For completeness, the meaning of the other frame types will be given here. Since the access control machine of this specification is unchanged for use within Spreadnet, these frames and their usage are as detailed within the standard.

The first of these frame types is concerned with the control of the MAC technique in use (see Fig b.1). The response windows specified after the names of some of the frames are periods of time during which the frame sender listens for reponses, as well as defining when particular nodes are allowed to respond. Each response window is defined to be of the same duration as the slot time for the network.



000000--Claim_token
000001--Solicit_successor_1 (has 1 response window)
000010--Solicit_successor_2 (has 2 response windows)
000011--Who_follows (has 3 response windows)
000100--Resolve_contention (has 4 response windows)
001000--Token
001100--Set_successor

Fig b.1 MAC control field.

The second of these frame types (see fig b.2) contains data to be passed to the layer above the MAC (ie normally LLC data). The control field contains a priority field, the use of which is detailed within the standard.

The third and final of the pre-defined control frame types is concerned with the passing of data related to the management of the MAC technique (see Fig b.3). The data to be passed within these frames has not yet been defined within the standard, so therefore is not defined here.

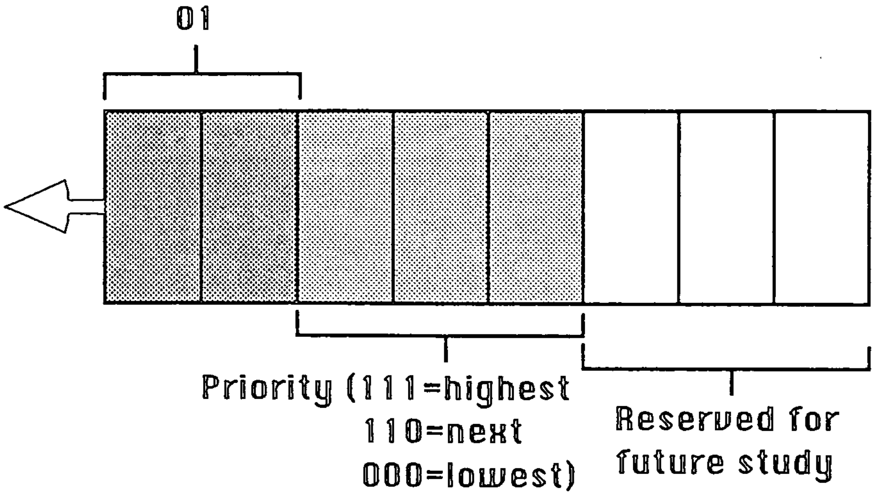


Fig b.2 LLC data frame.

The final frame type, which was undefined within the standard, is used within Spreadnet for the transfer of information related to the changing of the code sequences in use over that communications channel (se Fig 8.5).

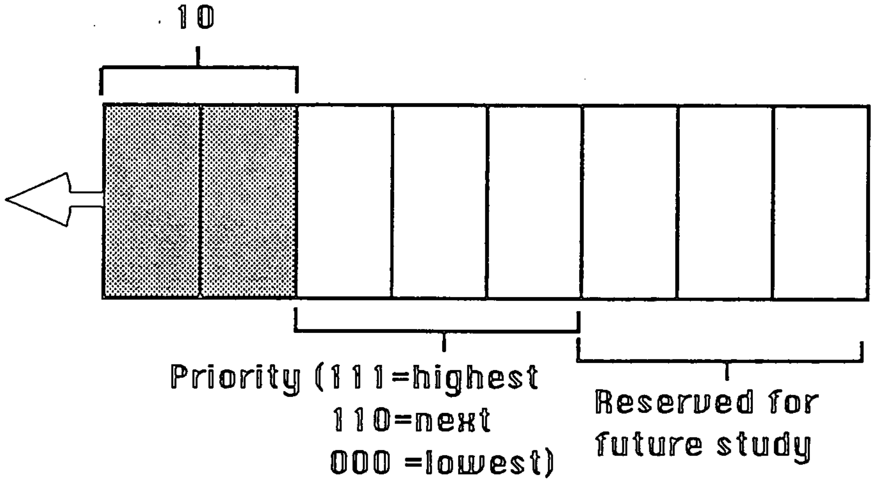


Fig b.3 Management data frame.

Appendix C
Transmitter Controller Software

sting of TRAN2 at 15:59:17 on JUL 4, 1985 for CCid=APK9

```
      ORG 800;
      /*****************************************************************/
      /*****************************************************************/
      /*********** SPREADNET TRANSMITTER - PHASE 1 NETWORK ***************/
      /*****************************************************************/
      /*****************************************************************/

           LENGTH    :EQU  $80;    /* MAXIMUM CODE LENGTH IS 127+1 BYTES */
           TABLE     :EQU  $2000;  /* BASE ADDRESS OF SELECTED CODES TABLE */
           ATABLE    :EQU  $2200;  /* BASE ADDR'OF SELECTED CODES ADDR'TABLE*/
           NDATA     :EQU  $6000;  /* START OF DATA TABLE FOR TRIAL */
           CHAR      :EQU  $7FFE;  /* ADDRESS OF CHAR */
           IDSTACK   :EQU  $2500;  /* ADDRESS OF STACK FOR SELECTED IDs */
           TAP       :EQU  $3000;  /* STACK FOR STORING BITS TO BE TAPPED */
           MAXTAP    :EQU  $7;     /* MAX' NUMBER OF FEEDBACK TAPS ALLOWED */
           TOP       :EQU  $6;     /* MAXIMUM VALUE FOR TAP LOCATION */

           CODE_A_POOL :EQU $4000; /* BASE ADDRESS OF CODE ADDRESS POOL */
           CODE_D-POOL :EQU $5000; /* BASE ADDRESS OF CODE DATA POOL */

      /*****************************************************************/
      /****************** INIT *****************************************/
      /*PURPOSE             INITIALIZATION OF PIAS,ACIAS, ****************/
      /*                    GLOBAL ADDRESSES AND TX FAIL INTERRUPT ********/
      /*INITIAL CONDITIONS  NONE ****************************************/
      /*REGISTER USAGE      A0,*****************************************/
      /*****************************************************************/


             PIA1      :EQU        $3FF41 ; /* KDM PIA1 BASE ADDRESS */
             PIA2      :EQU        $3FF40 ; /* KDM PIA2 BASE ADDRESS */

             PIADDA    :EQU        $0;      /* OFFSETS FROM THE BASE ADDRESS */
             PIADA     :EQU        $0;
             PIACA     :EQU        $4;
             PIADDB    :EQU        $2;
             PIADB     :EQU        $2;
             PIACB     :EQU        $6;

             IMSKO     :EQU        $2000;  /* LEVEL 0 INTERRUPT MASK */

             A1_CNTL   :EQU        $38; /* CONFIG'CONTROL REG', PORT A PIA1 */
             B1_CNTL   :EQU        $39;
             A2_CNTL   :EQU        $10; /* PORT A ON PIA2 */
             B2_CNTL   :EQU        $28;

             ACIA    :   EQU       $3FF01; /* TERM ACIA BASE ADDRESS */
             ACIACR  :   EQU       $0;     /* ACIA CONTROL REG  */
             ACIASR  :   EQU       $0;     /* ACIA  STATUS REG  */
             ACIADR  :   EQU       $2;     /* ACIA DATA REG   */
             AMODE   :   EQU       $45;    /* ACIA OPERATING MODE */
             MRESET  :   EQU       $03;    /* MASTER RESET */
             TDRE    :   EQU       $1;     /* ACIA TRANSMIT DATA REG' EMPTY */
             CR      :   EQU       $0D;    /* ASCII CARRIAGE RETURN */
             JMP     MENU;

       /*****************************************************/
       /* SUBROUTINE      PIAS ******************************/
```

```
/* FUNCTION        PIAS SETUP ***********************/
/* REG USAGE       A0 *******************************/
/***************************************************/
PIAS : MOVEM.L  A0,-(A7);
       MOVEA.L  PIA1,A0;          /* LOAD BASE ADDRESS OF PIA1 TO A0 */
       CLR.B    PIACA(A0);                 /* SELECT DDRA */
       MOVE.B   #$FF,PIADDA(A0);           /* ALL DATA LINES ARE OUTPUTS */
       MOVE.B   #A1_CNTL, PIACA(A0);       /* SET UP CONTROL REGISTER */
       BSET.B   #2, PIACA(A0);             /* ADDRESS DATA REGISTER */


       CLR.B    PIACB(A0);                 /* SELECT DDRB  */
       MOVE.B   #$FF,PIADDB(A0);           /* ALL DATA LINES ARE OUTPUTS */
       MOVE.B   #B1_CNTL, PIACB(A0);       /* SET UP CONTROL REGISTER */
       BSET.B   #2, PIACB(A0);             /* ADDRESS DATA REGISTER */
       BSET.B   #7, PIADB(A0);             /* SET CHIP SELECT BIT */

       MOVEA.L  PIA2,A0;                   /* NOW FOR PIA2 */

       CLR.B    PIACA(A0);
       MOVE.B   #$FF,PIADDA(A0);
       MOVE.B   #A2_CNTL, PIACA(A0);
       BSET.B   #2, PIACA(A0);

       CLR.B    PIACB(A0);
       MOVE.B   #$FF,PIADDB(A0);
       MOVE.B   #B2_CNTL, PIACB(A0);
       BSET.B   #2, PIACB(A0);

       LEA      TXFAIL,A0;        /* ROUTINE ADDRESS TO A0 */
       MOVE.L   A0,@$7C;          /* SEND TO VECTOR TABLE, LEVEL 7INT'*/
       MOVEM.L  (A7)+,A0;
       RTS;

TXFAIL: LEA      TXFAIL1,A1;      /* TX FAIL MESSAGE STRING ADDRESS */
LOOPT : BSR.S    OUTCH;           /* SEND CHARACTER */
        CMPI.B   #CR,(A1)+;       /* CR ? */
        BNE      LOOPT;           /* NO, GET NEXT CHAR */
        BRA       TXFAIL;         /* YES, DO AGAIN (TEMP)<<----- */



/************************************************************/
/*************** OUTPUT A CHARACTER  TO THE TERMINAL *********/
/*************** A0 CONTAINS THE ACIA BASE ADDRESS ***********/
/************************************************************/

OUTCH : BTST.B   #TDRE, ACIASR(A0);    /* READY TO SEND */
        BEQ.S    OUTCH ;               /* NO, CONTINUE CHECKING */
        MOVE.B   (A1), ACIADR(A0);     /* SEND CHARACTER */
        RTS;


/************************************************************/
/*************** INPUT A CHARACTER FROM THE TERMINAL *********/
/*************** A0 CONTAINS THE ACIA BASE ADDRESS ***********/
/************************************************************/
```

```
INCH : MOVE.B   ACIASR(A0),D0;     /* READ ACIA STATUS REG */
       LSR.B    #1,D0;             /* TEST FOR RECEIVE REG EMPTY */
       BCC.S    INCH ;             /* NO, CONTINUE CHECKING */
       MOVE.B   ACIADR(A0),D6;     /*MASK OFF PARITY BIT */
       AND.B    #$7F,D6;

ECHO : BTST.B   #TDRE, ACIASR(A0); /* ECHO CHAR INPUT */
       BEQ.S    ECHO;
       MOVE.B   D6,ACIADR(A0) ;
       RTS;

/******************* OUTPUT STRINGS ********************/

STRING :     DC $0A0D; DC 'ENTER ID OF CODE TO BE LOADED INTO';
DC $0A0D ;             DC 'TRANSMITTER, INTEGER THEN CR ';
DC $0A0D ;             DC '*********************************';
DC $0A0D2E;
STRING2    :  DC $0A0D; DC 'CODE ID > ' ; DC $58 ;
STRING3    :  DC $0A ; DC 'CODES ALL ENTERED ' ; DC $0A0D ;
TXFAIL1    :  DC $0A ; DC 'TX FAILED ' ; DC $0A0D ;
LOADED     :  DC $0A ; DC 'CODES LOADED' ; DC $0A0D ;
CPPROMPT1  :  DC $0A0A0D ; DC 'ENTER POOL ID '; DC $3E ;
ENTTAP     :  DC $0A0A; DC 'ENTER BIT NUMBERS TO BE TAPPED, ' ;
DC $0A0D   ;  DC 'INTEGER THEN CR ';
DC $0A0D   ;  DC 'ENTER IN ASCENDING ORDER          ' ; DC $0A0D;
DC 'VALUES MUST BE WITHIN THE RANGE 0 TO MAXTAP (6 NOW) ' ; DC $0A0D ;
DC '"." EXITS FROM TAP ENTERING PROGRAMME ' ; DC $0A0D ;
DC '*************************************' ; DC $0A0A0D2D;
TAPQ       :   DC $0A0D ; DC 'TAP BIT NUMBER' ; DC $3E ;
NXTCODEQ :     DC $0A ; DC '"." FOR NEXT CODE,. ELSE QUIT'   ; DC $0A0D ;
TAPERR     :   DC $0A    ; DC 'YOU ENTERED NO TAPS=INVALID ' ; DC $0A0D ;
YEUCH      :   DC $0A    ; DC 'TAP VALUE TOO LARGE, RETRY'   ; DC $0A0D ;
DLOAD1     :   DC $0A0A0A0D ;
DC '*********************************************'; DC $0A0D;
DC '***** TRANSMITTING DATA ***********************'; DC $0A0D;
DC '*********************************************'; DC $0A0D2E;
WHICODE : DC $0A0D ;
DC '*********************************************'; DC $0A0D;
DC '***** SELECT CODE NUMBER (1 TO 8 ) *************'; DC $0A0D;
DC '***** FOR THE CODE TO BE USED ON THE DATA ******'; DC $0A0D;
DC '*********************************************'; DC $0A0A0A0D;
DC 'SELECTION ' ; DC $3E ;
MHELLO : DC $0A0D ; DC 'HELLO ' ; DC $0A0D2E;
MTX: DC $0A0D;
DC '*********************************************'; DC $0A0D;
DC '********* SPREADNET TRANSMITER SOFTWARE ********'; DC $0A0D;
DC '********* PHASE ONE NETWORK ********************'; DC $0A0D;
DC '*********************************************'; DC $002E;
SMENU: DC $0A0D;
DC '*********************************************'; DC $0A0D;
DC '********* SS-LAN TRANSMITTER-PHASE 1 ***********'; DC $0A0D;
DC '*********************************************'; DC $0A0D;
DC '**                                           **'; DC $0A0D;
DC '**   SELECT MODE OF OPERATION (ENTER NUMBER)   **'; DC $0A0D;
DC '**                                           **'; DC $0A0D;
DC '**   1- GENERATE M-SEQUENCES AND LOAD 8        **'; DC $0A0D;
DC '**   2- SEND DATA                              **'; DC $0A0D;
DC '**   3- RESELECT SEQUENCE TO BE USED OVER LINK **'; DC $0A0D;
DC '**   4- XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX **'; DC $0A0D;
DC '**   5- XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX **'; DC $0A0D;
DC '**                                          **'; DC $0A0D;
DC '*********************************************'; DC $0A0A0D;
DC 'ENTER SELECTION '; DC $3E;
TRYAGN: DC $0A0D;
DC 'YOU ENTERED A NON-VALID SELECTION, RETRY  '; DC $0A0A0D2E;


/****************************************************/
/*ASCII TO DECIMAL CONVERSION OF ONE BYTE *************/
/*D6 CONTAINS INPUT AND RESULT-NON DIGITS GIVE ERROR **/
/************ALSO USES D0 AND D1***********************/


ASC2D : MOVEM.L   D0/D1,-(A7);
        MOVEQ     #-1,D1;          /* SET ERROR FLAG */
        MOVE.B    D6,D0;           /* GET CHARACTER */
        SUB.B     #'0',D0;         /* IS CHAR BELOW ASCII 0 ? */
        BCS.S     FIN;             /* YES, THEN NOT A DIGIT */
        CMP.B     #9,D0;           /* CHAR ABOVE ASCII 9 ? */
        BHI.S     FIN;             /* YES THEN NOT A DIGIT */
        EXG       D0,D1;           /* VALID CHAR TO D1 */
FIN :   MOVE.B    D1,D6;           /* SAVE DIGIT OR ERROR FLAG */
/************  TEST FOR ERROR FLAG LATER <<<----------- */
        MOVEM.L   (A7)+,D0/D1;
        RTS;

/*******************************************************/
/**********RUN-CHANGE TX TO RUN MODE ********************/
/*******************************************************/

RUN : MOVEM.L  A0,-(A7);
      MOVEA.L  PIA1,A0;
      BCLR.B   #3, PIACB(A0);  /* CB2 IS NOW LOW, SO RUN MODE */
      MOVEM.L  (A7)+,A0;
      RTS;

/*******************************************************/
/*********SETUP-CHANGE TX TO SETUP MODE *****************/
/*******************************************************/

SETUP : MOVEM.L  A0,-(A7);
        MOVEA.L  PIA1,A0;
        BSET.B   #3, PIACB(A0);  /* CB2 IS NOW HIGH, SETUP MODE */
        MOVEM.L  (A7)+,A0;
        RTS;


/**************************************************************/
/*SUBROUTINE    DATALOAD *************************************/
/*PURPOSE       REPEATED LOADING OF DATA TABLE TO PIA2 **********/
/*INITIAL CONDITIONS  GLOBAL VARIABLE DATA, CONTAINING START ******/
/*                    ADDRESS OF DATA TABLE ***********************/
/*REGISTER USAGE      D0/D3 AND A0/A2  ****************************/
/**************************************************************/
```

```
DATALOAD :

           MOVEM.L    D0/D3-A0/A2,-(A7);     /* SAVE REGISTERS USED */
           LEA       @SEND2,A0;          /* ADDRESS OF EXCEPTION VECTOR */
           MOVE.L     A0,@$70;           /* SEND TO VECTOR TABLE ,LEVEL 4 */

           MOVEA.L   NDATA,A0;           /* POINTS TO START OF LIST */
           MOVE.W     (A0)+,D1;          /* D1 CONTAINS LIST LENGTH */
           MOVE.W     #0,D0;
           CMP.W      D1,D0;             /* IS LIST NON-0 IN LENGTH? */
           BEQ.S      END;               /* NO, THEN GO TO END */

           MOVEA.L    PIA2,A1;           /* A1 CONTAINS PIA2 BASE */
           MOVE.B    #$1C,PIACA(A1);     /* ENABLES INTERRUPT FROM  PIA2 */

           MOVE.W     #$0,D3;          /* DUMMY DATA FOR TX STARTUP STEP */
           MOVEP.W    D3,PIADA(A1);    /* WRITE OP SENDS IT TO TX */


           MOVE.B     PIADA(A1),D3;    /* DUMMY READ CLEARS BIT 7 */
READY  :   STOP       #IMSKO;          /* ENABLES ALL INT'S AND WAITS */

           SUBQ.W     #1,D1;           /* LIST LENGTH TO GO, AFTER INT' */
           CMP.W      D0,D1;           /* D1=0? */
           BNE.S      READY;           /* NO, THEN CONTINUE ALONG TABLE */


RESTART: MOVEA.L    NDATA,A0;          /* RESET COUNTER TO TABLE START */
         MOVE.W      (A0)+,D1;
          BRA        READY;

SEND2  : MOVE.W      (A0)+,D2;         /* DATA TO REGISTER */
         MOVEP.W     D2,PIADA(A1);     /* DATA TO PIA2       */
         TST.B       PIADA(A1);        /* DUMMY READ CLEARS BIT 7 */
         RTE;

END:     MOVEM.L     (A7)+, D0/D3-A0/A2;
         RTS;


/***********************************************************/
/* SUBROUTINE      CODELOAD ******************************/
/* PURPOSE         SEND SELECTED CODES TO TX ***********/
/* I.CONDS         ID NUMBERS ARE ENTERED ONTO A STACK*/
/*                 ID(8) ON TOP,A5 IS POINTER *********/
/*                 "LENGTH","TABLE","POOL" G.VARIABLES*/
/* REG USAGE       A0/A2-DO/D7 ALSO A5 ****************/
/***********************************************************/

CODELOAD :
           MOVEM.L    D0/D7-A0/A4,-(A7);   /* SAVE REGISTERS USED */
           MOVE.L     A6,-(A7);
           CLR.L      D0;
           MOVE.B     #0,D5;     /* FOR COMPARISON */
           MOVEA.L    #TABLE,A1;   /* INITIALIZE TABLE POINTER */
           MOVEA.L    #ATABLE,A6;
```

```
/* NOTE THAT THE FIRST ELEMENT IS STORED AFTER ONE DECREMENT */

           MOVEA.L    CODE_D_POOL,A0; /* BASE OF DATA POOL */
           MOVEA.L    CODE_A_POOL,A3; /* BASE OF ADDRESS POOL */
           MOVE.B     #7,D0;        /* INIT COUNTER TO 8 */
AGAIN:     MOVE       #LENGTH,D1;   /* MAX CODELENGTH TO D1 */
           CLR.L      D2;
           MOVE.B     (A5)+,D2;     /* MOVE ID TO D2       */
           MULU       D2,D1;        /* LENGTH*ID NUMBER    */
           MOVE.L     D1,D4;        /* COPY OFFSET */
           ADD.L      A0,D1;        /* CODE ADDRESS IN A0 */
           MOVE.L     D1,-(A1);     /* DATA ADDRESS ON STACK */
           ADD.L      A3,D4;
           MOVE.L     D4,-(A6);     /* ADDR ADDRESS ON STACK */
           CMP.B      D0,D5;
           DBEQ       D0,AGAIN;     /* AFTER ALL 8 DONE, PROCEED */


/* CODE ADDRESSES ARE NOW ON STACK, CODE 1 ON TOP,A1 POINTER */
/* THE DATA TO BE SENT TO TX MUST NOW BE ASSEMBLED IN D1 */

DONE:      CLR.L      D4;           /* CLEAR REG TO BE USED FOR ADDRESS */
           CLR.L      D0;           /* CODE NUMBER COUNTER TO 0 */
           MOVE.L     #LENGTH,D5;   /* CODE LENGTH TO D5 */
           CLR.L      D3;

BIGLOOP:   CLR.L      D7;           /* CLEAR BYTE COUNTER */
           MOVEA.L    (A1)+,A2;     /* DATA START ADDRESS TO A2 */
           MOVEA.L    (A6)+,A4;     /* ADD  START ADDRESS TO A4 */
           MOVE.L     #1,D7;        /* INIT BYTE COUNTER */
           CLR.L      D1;           /* CLEAR REG FOR DATA ASSEMBLY */
CDLOOP:    MOVE.B     (A2)+,D1;     /* DATA BYTE IN D1 */
           MOVE..B    (A4)+,D4;     /* ADDRESS BYTE IN D4 */
           ASL        #8,D1;        /* SHIFT DATA TO REG TOP */
           MOVE.W     #13,D2;       /* PUT SHIFT COUNT INTO D2 */
           MOVE.W     D0,D3;        /* CODE NUMBER TO D3 */
           ASL        D2,D3;        /* SHIFT CODE NUMBER */
           ADD.W      D3,D1;        /* ADD CODE NUMBER TO D1 */

           ADD.B      D4,D1;             /* ADDRESS TO D1 */
           BSET.B     #7,D1;             /* SET CHIP SELECT BIT,PIA1-PORTB*/
           MOVEA.L    PIA1,A0;
           MOVEP.W    D1,PIADA(A0);  /* DATA TO PIA1 */

           BCLR.B     #7,PIADB(A0);  /* CLEAR CHIP SELECT */
           BCHG       #3,PIACA(A0);  /* CA2 HIGH TO LOW */
           BCHG       #3,PIACA(A0);  /* CA2 LOW TO HIGH */
           BSET.B     #7,PIADB(A0);  /* SET CHIP SELECT */

           CMP.W      D7,D5;             /* COUNT=LENGTH */
           BEQ.S      NEXTCODE;          /* YES, THEN GET NEXT CODE */
           ADDQ.W     #1,D7;             /* NO, THEN INCREMENT BYTE COUNT */
           BRA        CDLOOP;            /* GET NEXT DATA BYTE */

 NEXTCODE : CMP.W     #7,D0;             /* ALL CODES DONE ? */
            BEQ       LEAVE;             /* YES, THEN LEAVE ROUTINE */
            ADDQ.W    #1,D0;             /* NO, THEN INCREMENT CODE COUNT */
```

```
                BRA     BIGLOOP;        /* GET NEXT CODE */

LEAVE: MOVE.L   (A7)+,A6;
       MOVEM.L  (A7)+,D0/D7-A0/A4;   /* RESTORE USED REGISTERS */
       RTS;




/**********************************************************************/
/********************** MAIN ****************************************/
/********* Enters sequences into pool and loads a selection ****/
/********* into the Tx hardware **********************************/
/**********************************************************************/

MAIN : MOVEM.L  D0/D7-A0/A6,-(A7);
       LEA      @ACIA, A0;          /* SET UP ACIA TO TERMINAL */
       MOVE.B   #MRESET, ACIACR(A0);
       MOVE.B   #AMODE,  ACIACR(A0);
       LEA      MTX,A1;
AMTX:  BSR      OUTCH;
       CMPI.B   #$2E,(A1)+;
       BNE      AMTX;

       JSR      @CDAG;          /* GENERATE ANY NEW CODES */
       JSR      @SETUP;         /* TRANSMITTER IN SETUP MODE */

       CLR.L    D2;             /* RESET ID COUNTER */
       LEA      STRING,A1;      /* INSTRUCTION TO TERMINAL */
LOOP : BSR      OUTCH;
       CMPI.B   #$2E,(A1)+;
       BNE      LOOP;

         MOVE.W   #0,D2;          /* INIT CODE COUNTER */
         MOVEA.L  IDSTACK,A5;     /* BASE OF STACK TO A5 */
NEW    : LEA      STRING2,A1;     /* PROMPT TO TERMINAL */
LOOP2  : BSR      OUTCH;
         CMPI.B   #$58,(A1)+;
         BNE      LOOP2;

LOOP3  : BSR      INCH;           /* INPUT CODE ID */

         CMPI.B  #CR,D6;         /* CR ? */
         BEQ.S   GOTCR;          /* YES, THEN NEXTCODE */
         BSR     ASC2D;          /* ASCII TO DEC CONVERSION */
         MOVE.B  D6,-(A5);       /* ID ON STACK */
         BRA     LOOP3;
GOTCR  : CMP     #7,D2;          /* ALL 8 CODES READ? */
         BEQ.S   NEXT;           /* YES, THEN CONTINUE */
         ADDQ.W  #1,D2;          /* NO, THEN INCREMENT COUNT */
         BRA     NEW;            /* GET NEXT ID */
NEXT   : LEA     STRING3,A1;     /* MESSAGE TO TERMINAL */
LOOP4  : BSR     OUTCH;
         CMPI.B  #CR,(A1)+;
         BNE     LOOP4;
         JSR     @PIAS;
         JSR     @CODELOAD;
```

```
          LEA      LOADED,A1;      /* CODES LOADED MESSAGE */
 LOOP5  : BSR      OUTCH;
          CMPI.B   #CR,(A1)+;
          BNE      LOOP5;

          MOVEM.L (A7)+,D0/D7-A0/A6;
          RTS;


 /***********************************************/
 /* SUBROUTINE   CODE ADDRESS AND DATA GENERATOR **/
 /* FUNCTION     DO ABOVE AND STORE IT ************/
 /* INIT CONDS   NEEDS "CODE-D-POOL","CODE-A-POOL"*/
 /*              "MAXTAP","TAPS","LENGTH"**********/
 /* REG USAGE    D0/D7-A0/A6                     */
 /***********************************************/

 CDAG   : MOVEM.L   D0/D7-A0/A6,-(A7); /* SAVE USED REGS */

 CDAG2   : LEA    @ACIA,A0;
           MOVE.B #MRESET, ACIACR(A0);
           MOVE.B #AMODE, ACIACR(A0);
           CLR.L    D6 ;
 PROMPT  : LEA    CPPROMPT1,A1;  /* CODE POOL PROMPT */
 ALOOP   : BSR    OUTCH;
           CMPI.B #$3E,(A1)+;    /* LOOK FOR > */
           BNE     ALOOP;
 BLOOP  :  BSR     INCH;         /* INPUT POOL NUMBER */
           CMPI.B #CR,D6;        /* CR? */
           BEQ.S   GCR2   ;      /* YES, THEN CONTINUE */
           BSR     ASC2D;
           CLR.L   D2;
           MOVE.B D6,D2;         /* SEND ID TO D2 */
           BRA     BLOOP;

 GCR2    : CLR.L    D1;
           MOVE.W   #LENGTH, D1;   /* MAX CODE LENGTH TO D1 */
           MOVEA.L #CODE_D_POOL, A6;
           MOVEA.L #CODE_A_POOL, A2;

           MULU     D2,D1;       /* CALCULATE OFFSET */
           ADDA.L   D1,A6;       /* A6 = START FOR NEWCODE D TABLE */
           ADDA.L   D1,A2;       /* A2 = START FOR NEWCODE A TABLE */

  /* NOW TO ENTER TAP NUMBERS */

           LEA      ENTTAP,A1;  /* ENTER TAP INSTRUCTION */

  TAPLOOP1 : BSR     OUTCH;
             CMPI.B #$2D, (A1)+;
             BNE     TAPLOOP1;
             CLR.L  D3;          /* CLEAR TAP COUNT REGISTER */
             MOVEA.L #TAP,A3;    /* BASE OF STACK FOR TAP NUMBERS */
             CLR.L  D5;
             MOVE.B MAXTAP,D5;   /* MAX NUMBER OF TAPS ALLOWED */
  LTAPQ    : LEA     TAPQ,A1;    /* TAP PROMPT */
```

```
TAPL2      : BSR     OUTCH;
             CMPI.B  #$3E,(A1)+;   /* LOOK FOR > */
             BNE     TAPL2   ;

TAP3       : BSR     INCH;
             CMPI.B  #$2E,D6;      /* LOOK FOR "." WHICH IS FINISH */
             BEQ.S   EXIT;         /* YES, THEN EXIT */
             CMPI.B  #CR,D6;       /* CR? */
             BEQ.S   YUP;          /* YES, THEN CONTINUE */
             BSR     ASC2D;        /* CONVERT INPUT */
             CMPI.B  #TOP,D6;      /* VALUE > 6 ? */
             BGT     OVTOP;        /* YES, SEND ERROR MESSAGE */
             MOVE.B  D6,D4;        /* CONTAINS MAX TAP VAL ON EXIT */

             MOVE.B  D6,-(A3);     /* INPUT TO STACK */
             BRA     TAP3;
YUP  :       ADDQ.B  #1,D3;        /* INC TAP COUNTER */
             CMP.B   D3,D5;        /* MAXTAP REACHED */
             BEQ.S   YUP2;         /* YES, THEN CONTINUE */
             BRA     LTAPQ;        /* SEND NEXT PROMPT */

EXIT  :      CMPI.B  #$0,D3;     /* NO TAPS ENTERED ? */
             BNE     YUP2;       /* NO,THEN CONTINUE */
             JMP     ERROR1;     /* YES, THEN DO NEXT CODE OR EXIT */
/* TAP NUMBERS ARE NOW ENTERED ONTO STACK,A3 IS POINTER */


YUP2  :      ADDQ.B #1,D4;       /* ADDRESS REG LENGTH IN D4 */
             CLR.L   D7;
             ADDQ.B #1,D7;
             ASL.B   D4,D7;      /* 2**REGLNGTH=CODELENGTH+1 ) */
             CLR.L   D1;         /* CLEAR MASK REGISTER */
             MOVE.B D7,D1;
             SUBQ.B #1,D1;       /* REGISTER MASK IN D1= C'LENGTH */

/* D1 IS NOW THE CODE ADDRESS REGISTER MASK AND D7=CODELENGTH */
/* READY TO CALCULATE THE CODE NOW */


CODECALC : CLR.L   D5;      /* CLEAR DATA REGISTER */
           CLR.L   D4;      /* CLEAR BYTE COUNTER */
           CLR.L   D6;      /* CLEAR ADDRESS REGISTER */

/********************** SEND SAFETY BYTE **********************/
          BSET.B   #1,D5; /* SAFETY DATA, XOR BIT IS SET */
          JSR      SEND;

/*********** FIRST BYTE ONWARDS  =  CALCULATED ****************/
FIRST:     LEA      @XOR,A5;      /* ADDRESS OF XOR "REG" */
           CLR.L    (A5);
           CLR.L    D0;           /* INITIALISE TAP COUNT */
           ADDQ.B #1,D4;          /* INCREMENT BYTE COUNT */
           CMPI.B #1,D4;          /* FIRST BYTE? */
           BEQ.S    ADD8;         /* YES, THEN SET BIT 3 */
           CMP.B    D1,D4;        /* LAST BYTE? */
           BEQ.S    ADD3;         /* YES, THEN SET BIT 2 */
           BRA      GOGO;         /* SAVE THE INFORMATION */
ADD8:      ADDQ.B #8,D5;          /* BIT 3 SET */
```

```
            MOVE.B #1,D6;        /* INITIALISE GENERATOR */
            BRA    GOGO;
ADD3:       ADDQ.B #4,D5;        /* BIT 2 SET */
            BSR    SEND;
            BRA    FILL;         /* LEAVE GENERATOR */
GOGO:       BSR SEND;

            MOVEA.L #TAP,A3;     /* RESET TAP POINTER */
TAPLOOP4:   MOVE.B   -(A3),D2;   /* BITNO TO BE TESTED IN D6 */

            ADDQ.B    #1, D0;    /* INC TAP COUNT */
            BTST.B    D2,D6;     /* TAP THE BIT */
            BEQ.S     TESTAP ;   /* IF 0, ALL TAPS DONE ? */
            ADDI.B   #$1,(A5);   /* TO BE USED FOR XOR RESULT */
TESTAP :    CMP.B     D0,D3;     /* ALL TAPS DONE ? */
            BNE       TAPLOOP4;  /* NO, THEN DO NEXT TAP */
ZDATA  :     ASL      #1,D6;     /* SHIFT GENERATOR */
             BTST     #0,(A5) ;  /* TEST XOR RESULT */
             BNE      ADDO;
             JMP      XORDONE;

ADDO     :   CLR.L    D5;        /* CLEAR DATA REGISTER */
             BSET.B   #1,D5;     /* SET XOR BIT */
             BSET.B   #0,D6;     /* ONE, THEN ADD THIS TO GEN */
             BRA      DATAFIND;
XORDONE  :   CLR.L    D5;
             AND.B    D1,D6;     /* MASK OFF UNWANTED BITS */


DATAFIND:    BTST.B  D2,D6;           /* TEST UPPER BIT = DATA */
             BEQ.S   FIRST;           /* IF ZERO, NEW WORD IS DONE */
             BSET.B  #0,D5;           /* DATA IS A 1 */

             BRA FIRST;

/* FILL REMAINDER OF ALLOCATED RAM SPACE WITH RESET DATABYTE */

FILL :      CMP.B  #$7F,D1;    /* IS SEQUENCE OF MAXIMUM LENGTH */
            BEQ.S   NONEED;    /* YES, THEN ALL FILLED */
            ADDQ.B  #1,D1;
            MOVE.B  D1,D6;     /* FIRST EMPTY ADDRESS */
            MOVE.B  #$10,D5;    /* "RESET" BYTE,A2 AND A3 ARE SET */
            BSR     SEND;
            BRA     FILL;

NONEED :    LEA     NXTCODEQ,A1; /* AGAIN OR QUIT PROMPT */
PLOOP  :    BSR     OUTCH;
            CMPI.B #CR,(A1)+;
            BNE     PLOOP;
QLOOP  :    BSR     INCH;
            CMPI.B #$2E, D6   ;  /* "." FOR NEXT CODE */
            BNE     RET;         /* YES, DO NEXT SEQUENCE */
            JMP     CDAG2 ;
RET    :    MOVEM.L  (A7)+,D0/D7-A0/A6;
            RTS;

/*************** SEND ****************************/
SEND : MOVE.B  D5,(A6)+;       /* SEND NEW DATA TO DATASTACK */
```

```
            MOVE.B  D6   ,(A2)+;      /* SEND NEW ADDRESS TO ASTACK */
            RTS;

    /**************** ERROR1 *************************/
    ERROR1  : LEA     TAPERR,A1;
    RLOOP   : BSR     OUTCH;
              CMPI.B #CR,(A1)+;
              BNE     RLOOP;
              JMP     NONEED;  /* NEW CODE OR QUIT PROMPT */

    /*************** OVTOP ***************************/
    OVTOP   : LEA     YEUCH, A1;
    OVLOP   : BSR     OUTCH;
              CMPI.B #CR, (A1)+;
              BNE     OVLOP;
              JMP     LTAPQ;  /* ENTER NEW TAP VALUE */


    /**********************************************/
    /** SUBOUTINE  WHICH--SELECTS CURRENT CODE *******/
    /**                 FROM THE EIGHT AT THE TX ***/
    /** USES D4-D6 AND A0-A1 *************************/
    /**********************************************/

    WHICH :
              MOVEM.L   D4/D6-A0/A1,-(A7);

              LEA         WHICODE,A1;          /* MESSAGE STRING */
    AWHICH:   BSR         OUTCH;
              CMPI.B     #$3E,(A1)+;
              BNE         AWHICH;

    BWHICH :  BSR         INCH;                /* INPUT THE NUMBER */
              CMPI.B      #CR,D6;
              BEQ.S       WHICR;
              BSR         ASC2D;               /* CONVERT TO DECIMAL */
              MOVE.B      D6,D4;               /* D4 CONTAINS THE DATA */
              BRA         BWHICH;

    WHICR :   MOVE.W      #13,D5;              /* SHIFT COUNT */
              ASL         D5,D4;               /* DATA IS SHIFTED */
              MOVEA.L     PIA1,A0;
              MOVEP.W     D4,PIADA(A0);        /* SEND INFO TO TX */

              MOVEM.L   (A7)+, D4/D6-A0/A1;
              RTS;


    /**********************************************************/
    /*********SUBROUTINE HELLO - PRINTS HELLO TO TERMINAL **********/
    /*********** FOR DEBUGGING ************************************/
    /**********************************************************/

    HELLO : MOVEM.L  A0/A1,-(A7);
            LEA      @ACIA,A0;
            MOVE.B   #MRESET,ACIACR(A0);
            MOVE.B   #AMODE,ACIACR(A0);
```

```
           LEA      MHELLO,A1;
   HIYA :  BSR      OUTCH;
           CMPI.B   #$2E,(A1)+;
           BNE      HIYA;

           MOVEM.L  (A7)+,A0/A1;
           RTS;

 /****************************************************************/
 /************************ MENU ***********************************/
 /******** Displays a choice of functions to  be completed by tx */
 /****************************************************************/

 MENU : MOVEM.L  D5/D6-A0/A1,-(A7);
          LEA      @PIA2,A2;               /* SETUP PIA2 */
          CLR.B    PIACA(A2);
          CLR.B    PIADDA(A2);             /* ALL LINES ARE OUTPUTS */
          MOVE.B   #A2_CNTL,PIACA(A2);
          BSET.B   #2, PIACA(A2);
          CLR.B    PIACB(A2);
          CLR.B    PIADDB(A2);
          MOVE.B   #B2_CNTL,PIACB(A2);
          BSET.B   #2, PIACB(A2);
 DMENU:   LEA      @ACIA,A0;
          MOVE.B   #MRESET,ACIACR(A0);
          MOVE.B   #AMODE,ACIACR(A0);

          LEA      SMENU,A1;
 LMENU:   BSR      OUTCH;
          CMPI.B   #$3E,(A1)+;
          BNE      LMENU;

 WMENU:   BSR      INCH;
          CMPI.B   #CR,D6;
          BEQ.S    BMENU;
          BSR      ASC2D;
          MOVE.B   D6,D5;       /* DATA TO D5 */
          BRA      WMENU;

 BMENU:   CMPI.B   #1,D5;       /* CHOICE 1 ? */
          BEQ      XMAIN;
          CMPI.B   #2,D5;       /* CHOICE 2 ? */
          BEQ      XLDDATA;
          CMPI.B   #3,D5;       /* CHOICE 3 ? */
          BEQ      XCHOICE;
 XMAIN:   BSR.L    MAIN;
          BRA      DMENU;
 XLDDATA: BSR.L    LDDATA;
          BRA      DMENU;
 XCHOICE: BSR.S    CHOICE;
          BRA      DMENU;

 /****************************************************************/
 /**************** CHOICE *****************************************/
 /******* Enables re-selection of the code sequence used over ***/
 /******* the communications link *********************************/
 /****************************************************************/
```

```
CHOICE : JSR @SETUP;
         JSR @PIAS;
         JSR @WHICH;
         JSR @RUN;
         RTS;

/*****************************************************************/
/**** LOAD DATA sends the test data to the tx hardware *********/
/*****************************************************************/
LDDATA: MOVE.L  A0,-(A7);
        LEA     DLOAD1,A1;
DLOADLP: BSR    OUTCH;
        CMPI.B  #$2E, (A1)+;
        BNE     DLOADLP;
        JSR     @DATALOAD;
        RTS;


XOR   : DS.B      2;
   ORG NDATA;  /* TO MAKE LOADING SEPERATE DATA FILE EASIER */
DATA  : DS.W    100;
```

Appendix D
Receiver Controller Software

```
      ORG @$800;
    /*********************************************************************/
    /*********************************************************************/
    /*********** SPREADNET RECEIVER SOFTWARE - PHASE 1 NETWORK ***********/
    /*********************************************************************/
    /*********************************************************************/

      LENGTH        :EQU $80;      /* MAX' CODE LENGTH IS 127+1 BYTES */
      TABLE         :EQU $2000;    /* BASE ADDR'OF SELECTED CODES TABLE */
      ATABLE        :EQU $2200;    /* BASE ADDR'OF SEL' CODES ADDR' TABLE */
      CHAR          :EQU $7FFFE;   /* ADDRESS OF CHAR */
      IDSTACK       :EQU $2500;    /* ADDRESS OF STACK FOR SELECTED IDs */
      TAP           :EQU $3000;    /* STACK FOR STORING BITS TO BE TAPPED */
      MAXTAP        :EQU $7;       /* MAX' NUM' OF FEEDBACK TAPS ALLOWED */
      TOP           :EQU $6;       /* MAX' VALUE FOR TAP LOCATION */
      CODE_A_POOL   :EQU $4000 ;   /* BASE ADDR' OF CODE ADDR' POOL */
      CODE_D_POOL   :EQU $5000 ;   /* BASE ADDR' OF CODE DATA  POOL */
      WORD          :EQU $1FFE0;   /* TOPOF FILE STORAGE STACK */
      ERRSTCK       :EQU $1E000;   /* TOP OF ERROR STACK */

    /*******************************************************************/
    /****************** INIT ******************************************/
    /*PURPOSE               INITIALIZATION OF PIAS,ACIAS, ****************/
    /*                      GLOBAL ADDRESSES AND TX FAIL INTERRUPT ********/
    /*INITIAL CONDITIONS NONE *****************************************/
    /*REGISTER USAGE        A0,****************************************/
    /*******************************************************************/


           PIA1       :EQU        $3FF41 ; /* PROFI PIA ADDRESS */
           PIA2       :EQU        $3FF40 ;

           PIADDA     :EQU        $0;        /* OFFSETS FROM THE BASE ADDRESS */
           PIADA      :EQU        $0;
           PIACA      :EQU        $4;
           PIADDB     :EQU        $2;
           PIADB      :EQU        $2;
           PIACB      :EQU        $6;

           IMSKO      :EQU        $2000;  /* LEVEL 0 INTERRUPT MASK */

           A1_CNTL    :EQU        $38;    /* CONTROL REGISTER VALUE, PORT A ON PIA
           B1_CNTL    :EQU        $39;
           A2_CNTL    :EQU        $30;
           B2_CNTL    :EQU        $0;

           ACIA :      EQU        $3FF01; /* TERM ACIA BASE ADDRESS */
           ACIACR :    EQU        $0;     /* ACIA CONTROL REG  */
           ACIASR :    EQU        $0;     /* ACIA  STATUS REG  */
           ACIADR :    EQU        $2;     /* ACIA DATA REG   */
           AMODE  :    EQU        $45;    /* ACIA OPERATING MODE */
           MRESET :    EQU        $03;    /* MASTER RESET */
           TDRE   :    EQU        $1;     /* ACIA TRANSMIT DATA REG' EMPTY */
           CR     :    EQU        $0D;    /* ASCII CARRIAGE RETURN */
           BEGIN  :    EQU        $0040;  /* BEGINNING OF FILE MARKER */
           ENDF   :    EQU        $0025;  /* END OF FILE MARKER */
           GOOD   :    EQU        $4444;  /* CORRECT CHARACTER IN WERR TEST */
           NUM    :    EQU        $1000;  /* NUM' OF WERR'S BEFORE MESSAGE */
```

```
        WDCYC  :  EQU      $007E;  /* NUMBER OF WORDS INPUT PER TEST */
        TSTCYC :  EQU      $0004;  /* NUMBER OF TESTS TO BE MADE */

         JMP      MENU;

   /**************************************************/
   /* SUBROUTINE     PIAS ******************************/
   /* FUNCTION       PIAS SETUP ************************/
   /* REG USAGE      A0 ********************************/
   /**************************************************/

   PIAS : MOVEM.L  A0,-(A7);
          MOVEA.L  PIA1,A0;         /* LOAD BASE ADDRESS OF PIA1 TO A0 */
          CLR.B    PIACA(A0);           /* SELECT DDRA */
          MOVE.B   #$FF,PIADDA(A0);     /* ALL DATA LINES ARE OUTPUTS */
          MOVE.B   #A1_CNTL, PIACA(A0); /* SET UP CONTROL REGISTER */
          BSET.B   #2, PIACA(A0);       /* ADDRESS DATA REGISTER */

          CLR.B    PIACB(A0);           /* SELECT DDRB  */
          MOVE.B   #$FF,PIADDB(A0);     /* ALL DATA LINES ARE OUTPUTS */
          MOVE.B   #B1_CNTL, PIACB(A0); /* SET UP CONTROL REGISTER */
          BSET.B   #2, PIACB(A0);       /* ADDRESS DATA REGISTER */
          BSET.B   #7, PIADB(A0);       /* SET CHIP SELECT BIT */

          MOVEA.L  PIA2,A0;    /* NOW FOR PIA2 */

          CLR.B    PIACA(A0);
          CLR.B    PIADDA(A0);
          MOVE.B   #A2_CNTL, PIACA(A0);
          BSET.B   #2, PIACA(A0);
          CLR.B    PIACB(A0);
          CLR.B    PIADDB(A0);         /* ALL DATA LINES ARE INPUTS */
          MOVE.B   #B2_CNTL, PIACB(A0);
          BSET.B   #2, PIACB(A0);

          LEA      TXFAIL,A0;      /* ROUTINE ADDRESS TO A0 */
          MOVE.L   A0,@$7C;        /* SEND TO VECTOR TABLE, LEVEL 7INT' /
          MOVEM.L  (A7)+,A0;
          RTS;

   TXFAIL: LEA      TXFAIL1,A1;    /* TX FAIL MESSAGE STRING ADDRESS */
   LOOPT : BSR.S    OUTCH;         /* SEND CHARACTER */
           CMPI.B   #CR,(A1)+;     /* CR ? */
           BNE      LOOPT;         /* NO, GET NEXT CHAR */
           BRA       TXFAIL;       /* YES, DO AGAIN (TEMP)<<----- */

   /************************************************************/
   /*************** OUTPUT A CHARACTER  TO THE TERMINAL *********/
   /*************** A0 CONTAINS THE ACIA BASE ADDRESS ***********/
   /************************************************************/

   OUTCH : BTST.B   #TDRE, ACIASR(A0);  /* READY TO SEND */
           BEQ.S    OUTCH ;             /* NO, CONTINUE CHECKING */
           MOVE.B   (A1), ACIADR(A0);   /* SEND CHARACTER */
           RTS;

   /************************************************************/
   /*************** INPUT A CHARACTER FROM THE TERMINAL *********/
```

```
/*************** A0 CONTAINS THE ACIA BASE ADDRESS ***********/
/**************************************************************/

INCH : MOVE.B   ACIASR(A0),D0; /* READ ACIA STATUS REG */
       LSR.B    #1,D0;         /* TEST FOR RECEIVE REG EMPTY */
       BCC.S    INCH ;         /* NO, CONTINUE CHECKING */
       MOVE.B   ACIADR(A0),D6; /* MASK OFF PARITY BIT */
       AND.B    #$7F,D6;

ECHO : BTST.B   #TDRE, ACIASR(A0); /* ECHO CHAR INPUT */
       BEQ.S    ECHO;
       MOVE.B   D6,ACIADR(A0) ;
       RTS;

/******************* OUTPUT STRINGS ********************/

STRING :   DC $0A0D; DC 'ENTER ID OF CODE TO BE LOADED INTO TX,';
DC $0A0D ;             DC '*********INTEGER + CR*****************';
DC $0A0D2E;
STRING2 :  DC $0A0D; DC 'CODE ID > ' ; DC $58 ;
STRING3 :    DC $0A ; DC 'CODES ALL ENTERED ' ; DC $0A0D ;
TXFAIL1 :   DC $0A ; DC 'RX FAILED ' ; DC $0A0D ;
LOADED :      DC $0A ; DC 'CODES LOADED' ; DC $0A0D ;
CPPROMPT1 :   DC $0A0A0D ; DC 'ENTER POOL ID '; DC $3E ;
ENTTAP :  DC $0A0A; DC 'ENTER BIT NUMBERS TO BE TAPPED, ' ;
DC $0A0D ;  DC 'INTEGER THEN CR ';
DC $0A0D ;  DC 'ENTER IN ASCENDING ORDER       ' ; DC $0A0D;
DC 'VALUES MUST BE WITHIN THE RANGE 0 TO MAXTAP (6 NOW) ' ; DC $0A0D ;
DC '"." EXITS FROM TAP ENTERING PROGRAMME ' ; DC $0A0D ;
DC '**************************************' ; DC $0A0A0D2D;
TAPQ      :    DC $0A0D ; DC 'TAP BIT NUMBER' ; DC $3E ;
NXTCODEQ :    DC $0A ; DC '"." FOR NEXT CODE, ELSE QUIT'   ; DC $0A0D ;
TAPERR    :    DC $0A   ; DC 'YOU ENTERED NO TAPS=INVALID ' ; DC $0A0D ;
YEUCH     :    DC $0A;    DC 'TAP VALUE TOO LARGE, RETRY'   ; DC $0A0D ;
WHICODE  : DC $0A0D ;
DC '**********************************************'; DC $0A0D;
DC '***** SELECT CODE NUMBER (1 TO 8 ) *************'; DC $0A0D;
DC '***** FOR THE CODE TO BE USED ON THE DATA ******'; DC $0A0D;
DC '**********************************************'; DC $0A0A0A0D;
DC 'SELECTION ' ; DC $3E ;
MHELLO : DC $0A0D ; DC 'HELLO ' ; DC $0A0D2E;
SMENU: DC $0A0D;
DC '**********************************************' ; DC $0A0D;
DC '************ SS-LAN RECEIVER-PHASE 1 ***********' ; DC $0A0D;
DS '**********************************************' ; DC $0A0D;
DC '**                                           **' ; DC $0A0D;
DC '**    SELECT MODE OF OPERATION (ENTER NUMBER) **' ; DC $0A0D;
DC '**                                           **' ; DC $0A0D;
DC '**  1- GENERATE AND LOAD P-N SEQUENCES        **' ; DC $0A0D;
DC '**  2- RECEIVE AND ENTER FILE INTO MEMORY     **' ; DC $0A0D;
DC '**  3- TEST WORD ERROR RATE                   **' ; DC $0A0D;
DC '**  4- RESELECT CODE TO BE TESTED FROM 8 IN RX**' ; DC $0A0D;
DC '**  5- WORD ERROR RATE TEST (VERSION 2)       **' ; DC $0A0D;
DC '**                                           **' ; DC $0A0D;
DC '**********************************************' ; DC $0A0A0D;
DC 'ENTER SELECTION ' ; DC $3E;
TRYAGN: DC $0A0D;
DC 'YOU ENTERED A NON-VALID SELECTION, RETRY    ' ; DC $0A0A0D2E;
```

```
ERRORS: DC $0A0D;
DC '.. AND ANOTHER "NUM" ERRORS.' ; DC $0A0A0D2D;
MFILET: DC $0A0D;
DC '<<<<< AWAITING BEGINNING OF FILE WORD, "BEGIN" >>>>>>>' ; DC $0A0D0A2D
MBFILET: DC $0A0D;
DC '<<<<<<<< FILE TRANSFER IN PROGRESS >>>>>>>>>>>>>>>>>>>' ; DC $0A0A0D2D
ALLDONE: DC $0A0D;
DC '<<<<<<<< FILE TRANSFER COMPLETE >>>>>>>>>>>>>>>>>>>>>>' ; DC $0A0A0D2D
MWERR: DC $0A0D;
DC '<<<<<<<< WORD ERROR RATE TEST IN PROGRESS >>>>>>>>>>>>' ; DC $0A0A0D2D
MESSA: DC $0A0D;
DC 'NUMBER OF WORDS RECEIVED IN TEST =  '; DC $3E3E;
MESSOUT: DC $0A0D;
DC 'NUMBER OF ERRORS RECEIVED IN TEST=  '; DC $3E3E;


/**************************************************************/
/*********ASCII TO DECIMAL CONVERSION OF ONE BYTE *************/
/*********D6 CONTAINS INPUT AND RESULT-NON DIGITS GIVE ERROR **/
/********************ALSO USES D0 AND D1***********************/

ASC2D : MOVEM.L   D0/D1,-(A7);
        MOVEQ     #-1,D1;          /* SET ERROR FLAG */
        MOVE.B    D6,D0;           /* GET CHARACTER */
        SUB.B     #'0',D0;         /* IS CHAR BELOW ASCII 0 ? */
        BCS.S     FIN;             /* YES, THEN NOT A DIGIT */
        CMP.B     #9,D0;           /* CHAR ABOVE ASCII 9 ? */
        BHI.S     FIN;             /* YES THEN NOT A DIGIT */
        EXG       D0,D1;           /* VALID CHAR TO D1 */
FIN :   MOVE.B    D1,D6;           /* SAVE DIGIT OR ERROR FLAG */
/************  TEST FOR ERROR FLAG LATER <<<----------- */
        MOVEM.L   (A7)+,D0/D1;
        RTS;

/*********************************************************/
/**********RUN-CHANGE TX TO RUN MODE ********************/
/*********************************************************/

RUN : MOVEM.L  A0,-(A7);
      MOVEA.L  PIA1,A0;
      BCLR.B   #3, PIACB(A0);   /* CB2 IS NOW LOW, SO RUN MODE */
      MOVEM.L  (A7)+,A0;
      RTS;

/*********************************************************/
/*********SETUP-CHANGE TX TO SETUP MODE *****************/
/*********************************************************/

SETUP : MOVEM.L  A0,-(A7);
        MOVEA.L  PIA1,A0;
        BSET.B   #3, PIACB(A0);   /* CB2 IS NOW HIGH, SETUP MODE */
        MOVEM.L  (A7)+,A0;
        RTS;


/*****************************************************/
/* SUBROUTINE    CODELOAD ***************************/
/* PURPOSE       SEND SELECTED CODES TO TX **********/
```

sting of REC1 at 15:45:49 on JUL 4, 1985 for CCid=APK9

```
   /* I.CONDS          ID NUMBERS ARE ENTERED ONTO A STACK*/
   /*                  ID(8) ON TOP,A5 IS POINTER *********/
   /*                  "LENGTH","TABLE","POOL" G.VARIABLES*/
   /* REG USAGE        A0/A2-DO/D7 ALSO A5 ****************/
   /***************************************************/

   CODELOAD :
            MOVEM.L   D0/D7-A0/A4,-(A7);  /* SAVE REGISTERS USED */
            MOVE.L    A6,-(A7);
            CLR.L     D0;
            MOVE.B    #0,D5;    /* FOR COMPARISON */
            MOVEA.L   #TABLE,A1;  /* INITIALIZE TABLE POINTER */
            MOVEA.L   #ATABLE,A6; /* INITIALIZE TABLE ADDRESS POINTER */

   /* NOTE THAT THE FIRST ELEMENT IS STORED AFTER ONE DECREMENT */

            MOVEA.L   CODE_D_POOL,A0; /* BASE OF DATA POOL */
            MOVEA.L   CODE_A_POOL,A3; /* BASE OF ADDRESS POOL */
            MOVE.B    #7,D0;          /* INIT COUNTER TO 8 */
   AGAIN:   MOVE      #LENGTH,D1;     /* MAX CODELENGTH TO D1 */
            CLR.L     D2;
            MOVE.B    (A5)+,D2;       /* MOVE ID TO D2     */
            MULU      D2,D1;          /* LENGTH*ID NUMBER  */
            MOVE.L    D1,D4;          /* COPY OFFSET */
            ADD.L     A0,D1;          /* CODE ADDRESS IN A0 */
            MOVE.L    D1,-(A1);       /* DATA ADDRESS ON STACK */
            ADD.L     A3,D4;
            MOVE.L    D4,-(A6);       /* ADDR ADDRESS ON STACK */
            CMP.B     D0,D5;
            DBEQ      D0,AGAIN;       /* AFTER ALL 8 DONE, PROCEED */

   /* CODE ADDRESSES ARE NOW ON STACK, CODE 1 ON TOP,A1 POINTER */
   /* THE DATA TO BE SENT TO TX MUST NOW BE ASSEMBLED IN D1 */

   DONE:    CLR.L     D4;         /* CLEAR REG TO BE USED FOR ADDRESS */
            CLR.L     D0;         /* CODE NUMBER COUNTER TO 0 */
            MOVE.L    #LENGTH,D5; /* CODE LENGTH TO D5 */
            CLR.L     D3;

   BIGLOOP: CLR.L     D7;         /* CLEAR BYTE COUNTER */
            MOVEA.L   (A1)+,A2;   /* DATA START ADDRESS TO A2 */
            MOVEA.L   (A6)+,A4;   /* ADD  START ADDRESS TO A4 */
            MOVE.L    #1,D7;      /* INIT BYTE COUNTER */
            CLR.L     D1;         /* CLEAR REG FOR DATA ASSEMBLY */
   CDLOOP:  MOVE.B    (A2)+,D1;   /* DATA BYTE IN D1 */
            MOVE.B    (A4)+,D4;   /* ADDRESS BYTE IN D4 */
            ASL       #8,D1;      /* SHIFT DATA TO REG TOP */
            MOVE.W    #13,D2;     /* PUT SHIFT COUNT INTO D2 */
            MOVE.W    D0,D3;      /* CODE NUMBER TO D3 */
            ASL       D2,D3;      /* SHIFT CODE NUMBER */
            ADD.W     D3,D1;      /* ADD CODE NUMBER TO D1 */

            ADD.B     D4,D1;          /* ADDRESS TO D1 */
            BSET.B    #7,D1;          /* SET CHIP SELECT BIT,PIA1-PORTB*/
            MOVEA.L   PIA1,A0;
            MOVEP.W   D1,PIADA(A0); /* DATA TO PIA1 */

            BCLR.B    #7,PIADB(A0); /* CLEAR CHIP SELECT */
            BCHG      #3,PIACA(A0); /* CA2 HIGH TO LOW */
            BCHG      #3,PIACA(A0); /* CA2 LOW TO HIGH */
            BSET.B    #7,PIADB(A0); /* SET CHIP SELECT */

            CMP.W     D7,D5;        /* COUNT=LENGTH */
            BEQ.S     NEXTCODE;     /* YES, THEN GET NEXT CODE */
            ADDQ.W    #1,D7;        /* NO, THEN INCREMENT BYTE COUNT */
            BRA       CDLOOP;       /* GET NEXT DATA BYTE */

    NEXTCODE : CMP.W  #7,D0;        /* ALL CODES DONE ? */
              BEQ     LEAVE;        /* YES, THEN LEAVE ROUTINE */
              ADDQ.W  #1,D0;        /* NO, THEN INCREMENT CODE COUNT */
              BRA     BIGLOOP;      /* GET NEXT CODE */

    LEAVE: MOVE.L    (A7)+,A6;
           MOVEM.L   (A7)+,D0/D7-A0/A4;  /* RESTORE USED REGISTERS */
           RTS;


    /**********************************************************************/
    /************************* MAIN ***********************************/
    /******** Enters sequences into pool and loads a selection **********/
    /******** into the Rx hardware                              **********/
    /**********************************************************************/

    MAIN:  MOVEM.L   D0/D7-A0/A6,-(A7);
           LEA       @ACIA, A0;
           MOVE.B    #MRESET, ACIACR(A0);
           MOVE.B    #AMODE,  ACIACR(A0);

           JSR       @CDAG;         /* GENERATE ANY NEW CODES */
           JSR       @SETUP;        /* TRANSMITTER IN SETUP MODE */

           CLR.L     D2;            /* RESET ID COUNTER */
           LEA       STRING,A1;     /* INSTRUCTION TO TERMINAL */
    LOOP : BSR       OUTCH;
           CMPI.B   #$2E,(A1)+;
           BNE       LOOP;

           MOVE.W    #0,D2;         /* INIT CODE COUNTER */
           MOVEA.L   IDSTACK,A5;    /* BASE OF STACK TO A5 */
    NEW :  LEA       STRING2,A1;    /* PROMPT TO TERMINAL */
    LOOP2 : BSR      OUTCH;
           CMPI.B    #$58,(A1)+;
           BNE       LOOP2;

    LOOP3 : BSR     INCH;           /* INPUT CODE ID */

            CMPI.B  #CR,D6;         /* CR ? */
            BEQ.S   GOTCR;          /* YES, THEN NEXTCODE */
            BSR     ASC2D;          /* ASCII TO DEC CONVERSION */
           MOVE.B   D6,-(A5);       /* ID ON STACK */
            BRA     LOOP3;
    GOTCR : CMP     #7,D2;          /* ALL 8 CODES READ? */
            BEQ     NEXT ;          /* YES, THEN CONTINUE */
            ADDQ.W  #1,D2;          /* NO, THEN INCREMENT COUNT */
            BRA     NEW;            /* GET NEXT ID */
```

```
NEXT  :  LEA      STRING3,A1;    /* MESSAGE TO TERMINAL */
LOOP4 :  BSR      OUTCH;
         CMPI.B   #CR,(A1)+;
         BNE      LOOP4;
         JSR     @PIAS;
         JSR     @CODELOAD;
         LEA      LOADED,A1;

         MOVEM.L (A7)+,D0/D7-A0/A6;
         RTS;


/********************************************/
/* SUBROUTINE   CODE ADDRESS AND DATA GENERATOR **/
/* FUNCTION     DO ABOVE AND STORE IT ************/
/* INIT CONDS   NEEDS "CODE-D-POOL","CODE-A-POOL"*/
/*              "MAXTAP","TAPS","LENGTH" *********/
/* REG USAGE    D0/D7-A0/A6 *********************/
/********************************************/

CDAG2   : LEA    @ACIA,A0;
          MOVE.B #MRESET, ACIACR(A0);
          MOVE.B #AMODE, ACIACR(A0);
          CLR.L   D6 ;
PROMPT  : LEA    CPPROMPT1,A1;   /* CODE POOL PROMPT */
ALOOP   : BSR    OUTCH;
          CMPI.B #$3E,(A1)+;    /* LOOK FOR > */
          BNE     ALOOP;
BLOOP   : BSR     INCH;          /* INPUT POOL NUMBER */
          CMPI.B #CR,D6;         /* CR? */
          BEQ.S   GCR2   ;       /* YES, THEN CONTINUE */
          BSR     ASC2D;
          CLR.L   D2;
          MOVE.B D6,D2;          /* SEND ID TO D2 */
          BRA     BLOOP;

GCR2    : CLR.L   D1;
          MOVE.W #LENGTH, D1;   /* MAX CODE LENGTH TO D1 */
          MOVEA.L #CODE_D_POOL, A6;
          MOVEA.L #CODE_A_POOL, A2;

          MULU     D2,D1;       /* CALCULATE OFFSET */
          ADDA.L   D1,A6;       /* A6 = START FOR NEWCODE A TABLE */
          ADDA.L   D1,A2;       /* A2 = START FOR NEWCODE D TABLE */

/* NOW TO ENTER TAP NUMBERS */

             LEA    ENTTAP,A1;  /* ENTER TAP INSTRUCTION */
TAPLOOP1  : BSR    OUTCH;
             CMPI.B #$2D, (A1)+;
             BNE     TAPLOOP1;
             CLR.L  D3;         /* CLEAR TAP COUNT REGISTER */
             MOVEA.L #TAP,A3;   /* BASE OF STACK FOR TAP NUMBERS */
             CLR.L  D5;
             MOVE.B MAXTAP,D5; /* MAX NUMBER OF TAPS ALLOWED */
LTAPQ     : LEA     TAPQ,A1;    /* TAP PROMPT */
TAPL2     : BSR     OUTCH;
             CMPI.B #$3E,(A1)+;  /* LOOK FOR > */
```

```
            BNE     TAPL2    ;
TAP3      : BSR     INCH;
            CMPI.B  #$2E,D6;      /* LOOK FOR "." WHICH IS FINISH */
            BEQ.S   EXIT;         /* YES, THEN EXIT */
            CMPI.B  #CR,D6;       /* CR? */
            BEQ.S   YUP;          /* YES, THEN CONTINUE */
            BSR     ASC2D;        /* CONVERT INPUT */
            MOVE.B  D6,D4;        /* CONTAINS MAX TAP VAL ON EXIT */

            MOVE.B  D6,-(A3);     /* INPUT TO STACK */
            BRA     TAP3;
YUP :       ADDQ.B  #1,D3;        /* INC TAP COUNTER */
            CMP.B   D3,D5;        /* MAXTAP REACHED */
            BEQ.S   YUP2;         /* YES, THEN CONTINUE */
            BRA     LTAPQ;        /* SEND NEXT PROMPT */
EXIT :      CMPI.B  #$0,D3;       /* NO TAPS ENTERED ? */
            BNE     YUP2;         /* NO,THEN CONTINUE */
            JMP     ERROR1;       /* YES, THEN DO NEXT CODE OR EXIT */

/* TAP NUMBERS ARE NOW ENTERED ONTO STACK,A3 IS POINTER */

YUP2 :      ADDQ.B #1,D4;       /* ADDRESS REG LENGTH */
            CLR.L   D7;
            ADDQ.B #1,D7;
            ASL.B   D4,D7;      /* 2**REGLNGTH=CODELENGTH +1   */
            CLR.L   D1;        /* CLEAR MASK REGISTER */
            MOVE.B D7,D1;
            SUBQ.B #1,D1;       /* REG' MASK IN D1 = CODE LENGTH */

/* D1 IS NOW THE CODE ADDRESS REGISTER MASK AND D7=CODELENGTH */
/* READY TO CALCULATE THE CODE NOW */


CODECALC : CLR.L   D5;     /* CLEAR DATA REGISTER */
           CLR.L   D4;     /* CLEAR BYTE COUNTER */
           CLR.L   D6;     /* CLEAR ADDRESS REGISTER */

/********** SEND SAFETY BYTE **********************************/
          BSET.B  #1,D5;     /* SAFETY DATA, XOR BIT IS SET */
          JSR     SEND;

/* FIRST BYTE ONWARDS  =  CALCULATED */

FIRST :   LEA      @XOR,A5; /* ADDRESS OF XOR "REG" */
          CLR.L    (A5);
          CLR.L    D0;      /* INIT TAP COUNT */
          ADDQ.B    #1,D4;  /* INC BYTE COUNT */
          CMPI.B    #1,D4;  /* FIRST BYTE? */
          BEQ.S     ADD8;   /* YES, THEN SET BIT 3 */
          CMP.B     D1,D4;  /* LAST BYTE? */
          BEQ.S     ADD3;   /* YES, THEN SET BIT 2 */
          BRA       GOGO;   /* SAVE THE INFORMATION */
ADD8 :    ADDQ.B    #8,D5;  /* BIT 3 SET */
          MOVE.B    #1,D6;  /* INITIALISE GENERATOR */
          BRA       GOGO;
ADD3 :    ADDQ.B    #4,D5;  /* BIT 2 SET */
          BSR       SEND;
          BRA       FILL;   /* LEAVE GENERATOR */
```

```
GOGO :      BSR       SEND;

             MOVEA.L #TAP,A3;   /* RESET TAP POINTER */
TAPLOOP4 : MOVE.B  -(A3),D2; /* BIT NO TO BE TESTED IN D6 */

             ADDQ.B  #1,D0;     /* INC' TAP COUNT */
             BTST.B  D2,D6;     /* TAP THE BIT */
             BEQ.S   TESTAP;    /* IF 0, ALL TAPS DONE? */
             ADDI.B  #$1,(A5); /* TO BE USED FOR XOR RESULT */
TESTAP :     CMP.B   D0,D3;     /* ALL TAPS DONE? */
             BNE     TAPLOOP4; /* NO, THEN DO NEXT ONE */
ZDATA   :    ASL     #1,D6;     /* SHIFT GENERATOR */
             BTST    #0,(A5);   /* TEST XOR RESULT */
             BNE     ADD0;      /* XOR IS A ONE */
             JMP     XORDONE;   /* XOR IS A ZERO */

ADD0     :   CLR.L   D5;        /* CLEAR DATA REGISTER */
             BSET.B  #1,D5;     /* SET XOR BIT */
             BSET.B  #0,D6;     /* ADD XOR TO GENERATOR */
             BRA     DATAFIND;
XORDONE :    CLR.L   D5;        /* CLEAR DATA REGISTER */
             AND.B   D1,D6;     /* MASK OFF UNWANTED BITS */

DATAFIND : BTST.B  D2,D6;       /* TEST UPPER BIT = DATA */
             BEQ.S   FIRST;     /* IF 0, NEW WORD IS DONE */
             BSET.B  #0,D5;     /* DATA IS A 1 */
             BRA     FIRST;

/* FILL REMAINDER OF ALLOCATED RAM SPACE WITH RESET DATABYTE */

FILL :      CMP.B   #$7F,D1;    /* IS SEQUENCE OF MAXIMUM LENGTH */
            BEQ.S    NONEED;    /* YES, THEN ALL FILLED */
            ADDQ.B   #1,D1;
            MOVE.B   D1,D6;     /* FIRST EMPTY ADDRESS */
            MOVE.B   #$7,D5;    /* "RESET" DATA BYTE */
            BSR      SEND;
            BRA      FILL;

NONEED :    LEA     NXTCODEQ,A1; /* AGAIN OR QUIT PROMPT */
PLOOP   :   BSR     OUTCH;
            CMPI.B #CR,(A1)+;
            BNE     PLOOP;
QLOOP   :   BSR     INCH;
            CMPI.B #$2E, D6   ;  /* "." FOR NEXT CODE */
            BNE     RET;         /* YES, DO NEXT SEQUENCE */
            JMP     CDAG2  ;
RET    :    MOVEM.L  (A7)+,D0/D7-A0/A6;
            RTS;

/*************** SEND ****************************/
SEND : MOVE.B   D5,(A6)+;      /* SEND NEW DATA TO DATASTACK */
       MOVE.B   D6,(A2)+;      /* SEND NEW ADDRESS TO ASTACK */
       RTS;

/*************** ERROR1 **************************/
ERROR1  : LEA     TAPERR,A1;
RLOOP   : BSR     OUTCH;
          CMPI.B #CR,(A1)+;
```

```
            BNE      RLOOP;
            JMP      NONEED;


    /**************************************************/
    /** SUBOUTINE   WHICH--SELECTS CURRENT CODE *******/
    /**                    FROM THE EIGHT AT THE TX ***/
    /** USES D4-D6 AND A0-A1 **************************/
    /**************************************************/

    WHICH :
            MOVEM.L      D4/D6-A0/A1,-(A7);

            LEA          WHICODE,A1;           /* MESSAGE STRING */
    AWHICH: BSR          OUTCH;
            CMPI.B       #$3E,(A1)+;
            BNE          AWHICH;

    BWHICH : BSR         INCH;                /* INPUT THE NUMBER */
            CMPI.B       #CR,D6;
            BEQ.S        WHICR;
            BSR          ASC2D;               /* CONVERT TO DECIMAL */
            MOVE.B       D6,D4;               /* D4 CONTAINS THE DATA */
            BRA          BWHICH;

    WHICR : MOVE.W       #13,D5;              /* SHIFT COUNT */
            ASL          D5,D4;               /* DATA IS SHIFTED */
            MOVEA.L      PIA1,A0;
            MOVEP.W      D4,PIADA(A0);        /* SEND INFO TO TX */

    MOVEM.L   (A7)+, D4/D6-A0/A1;
            RTS;


    /************************************************************/
    /*********SUBROUTINE HELLO - PRINTS HELLO TO TERMINAL **********/
    /*********** FOR DEBUGGING ***************************************/
    /************************************************************/

    HELLO : MOVEM.L  A0/A1,-(A7);
            LEA      @ACIA,A0;
            MOVE.B   #MRESET,ACIACR(A0);
            MOVE.B   #AMODE,ACIACR(A0);

            LEA      MHELLO,A1;
    HIYA :  BSR      OUTCH;
            CMPI.B   #$2E,(A1)+;
            BNE      HIYA;

            MOVEM.L  (A7)+,A0/A1;
            RTS;


    /*************************************************************/
    /************************* MENU ********************************/
    /**** Displays a menu of choices of functions to be completed ***/
    /**** by the receiver ********************************************/
    /*************************************************************/
```

```
MENU:   MOVEM.L    D5/D6-A0/A1, -(A7);
        LEA        @PIA2,A2;
        CLR.B      PIACA(A2);
        CLR.B      PIADDA(A2);                  /* ALL LINES ARE INPUTS */
        MOVE.B     #A2_CNTL, PIACA(A2);
        BSET.B     #2, PIACA(A2);
        CLR.B      PIACB(A2);
        CLR.B      PIADDB(A2);
        MOVE.B     #B2_CNTL, PIACB(A2);
        BSET.B     #2, PIACB(A2);
DMENU:  LEA        @ACIA,A0;
        MOVE.B     #MRESET,ACIACR(A0);
        MOVE.B     #AMODE,ACIACR(A0);

        LEA        SMENU,A1;
LMENU:  BSR         OUTCH;
        CMPI.B      #$3E,(A1)+;                 /* LOOK FOR > TO END OUTPUT */
        BNE         LMENU;

WMENU:  BSR         INCH;
        CMPI.B      #CR,D6;
        BEQ.S       BMENU;
        BSR         ASC2D;
        MOVE.B      D6,D5;                      /* DATA TO D5 */
        BRA         WMENU;

BMENU:  CMPI.B      #1,D5;                      /* CHOICE 1,LOAD CODES */
        BEQ        CMAIN;
        CMPI.B      #2,D5;                      /* CHOICE 2,TRANSFER FILE */
        BEQ        NFILET;
        CMPI.B      #3,D5;                      /* CHOICE 3,CHECK ERROR RATE */
        BEQ        CWERR;
        CMPI.B      #4,D5;
        BEQ        ZCHOICE;
        CMPI.B      #5,D5;
        BEQ        CWERR2;

        LEA         TRYAGN ,A1;
TRYMR:  BSR         OUTCH;
        CMPI.B      #$2E,(A1)+;            /* LOOK FOR END OF MESSAGE,"." */
        BNE         TRYMR;
        BRA        DMENU;                  /* BACK TO MENU */
CMAIN:  BSR.L       MAIN;
        BRA         DMENU;
CWERR:  BSR.L       WERR;
        BRA         DMENU;
ZCHOICE:  BSR       CHOICE;
          BRA       DMENU;
NFILET:   BSR       FILET;
          BRA       DMENU;
CWERR2:   BSR       WERR2;
          BRA       DMENU;


/*******************************************************************/
/***** CHOICE- To reselect the code to be sent over the channel *****/
/***** the choice to be made from the 8 codes in the hardware. ******/
```

```
/*********************************************************************/

 COICE: JSR   @SETUP;         /* WE ARE IN SETUP MODE AGAIN */
        JSR   @PIAS;          /* SETUP THE PIAS */
        JSR   @WHICH;         /* RE-SELECT CODE */
        JSR   @RUN;           /* RUN MODE AGAIN */
        RTS;

/*********************************************************************/
/********************** FILE TRANSFER ********************************/
/**** Loads words into a stack pointed to by A6 after receiving the **/
/* start word (BEGIN). Finishes when receivesEOF (ENDF). Top of file */
/* stack is given by WORD ********************************************/
/*********************************************************************/

FILET : MOVEM.L     D5-A1/A6, -(A7);
        LEA         @MFILET,A1;             /* AWAITING MESSAGE */
ZFILET: BSR         OUTCH;
        CMPI.B      #$2D, (A1)+;
        BNE         ZFILET;

        LEA        @WORD,A6;                /* TOP OF FILE STACK */
        MOVEA.L     PIA2,A2;                /* SELECT PIA2 */
FILBEG: BSR         PIAIN;
        CMPI.W      #BEGIN,D5;              /* BEGINNING OF FILE ? */
        BNE.S       FILBEG;                 /* NO, THEN GET NEXT WORD */
        LEA         @MBFILET,A1;            /* IN PROGRESS MESSAGE */
YFILET: BSR         OUTCH;
        CMPI.B      #$2D, (A1)+;
        BNE         YFILET;

AFILET: BSR         PIAIN;              /* YES, THEN BEGIN FILE INPUT */
        CMPI.W      #ENDF,D5;           /* EOF ? */
        BEQ.S       BFILET;             /* YES, THEN FINISH */
        MOVE.W      D5, -(A6);          /* NO, THEN ENTER DATA ON STACK */
        BRA         AFILET;             /* GET NEXT WORD */

 FILET: LEA         @ALLDONE,A1;        /* TRANSFER COMPLETE MESSAGE */
 FILET: BSR         OUTCH;
        CMPI.B      #$2D,(A1)+;
        BNE         PFILET;

        MOVEM.L     (A7)+, D5-A0/A6;
        JMP         DMENU;
        RTS;

/*********************************************************************/
/************ PIAIN **************************************************/
/*Inputs a word from a PIA after receiving a signal on CB2(new data)*/
/* After reading word, sends pulse on CA2 requesting more data ******/
/* Data words are storedin D5 ***************************************/
/*********************************************************************/

PIAIN:  BTST.B  #6,PIACB(A2);       /* IS BIT 6 SET? */
        BEQ.S   PIAIN;              /* NO, THEN TEST  AGAIN */
        MOVE.B  PIADA(A2),D5;       /* YES, THEN DATA WORD TO D5 */
        LSL.W   #8,D5;
        CLR.W   D4;
```

```
          MOVE.B  PIADB(A2),D4;
          ADD.W   D4,D5;

          BSET.B  #3, PIACA(A2);      /* CA2 GOES HIGH */
          BCLR.B  #3, PIACA(A2);      /* CA2 GOES LOW */


          RTS;

    /****************************************************************/
    /************ WORD ERROR RATE TEST ****************************/
    /** Reads in words from a PIA, logging those which do not *****/
    /** correspond to GOOD and then printing a message after ******/
    /** NUM such errors are detected ****************************/
    /****************************************************************/

    WERR : MOVEM.L   D0/D5-A0/A2, -(A7);
           LEA       MWERR,A1;                /* WELCOME MESSAGE */
    XWERR: BSR       OUTCH;
           CMPI.B    #$2D, (A1)+;
           BNE.S     XWERR;
    WERCLR: LEA      ERRORS,A1;

          CLR.L      D1;                     /* CLEAR ERROR COUNT */
    AWERR: BSR       PIAIN;
          CMPI.W     #GOOD,D5;               /* CORRECT WORD */
          BEQ.S      AWERR;                  /* YES, THEN GET NEXT ONE */

          ADDQ.W     #1,D1;                  /* INC ERROR COUNT */
          CMPI.W     #NUM,D1;                /* NUM ERRORS FOUND */
          BNE.S      AWERR;                  /* NO, THEN GET NEXT WORD */

    BWERR: BSR       OUTCH;
          CMPI.B     #$2D, (A1)+;            /* LOOK FOR "-" TERMINATOR */
          BNE        BWERR;
          BSR        OUTD7;
          BRA        WERCLR;                 /* ALL PRINTED, GET NEXT WORD */

          MOVEM.L  (A7)+,D0/D5-A0/A2;        /* NEVER REACHED */
          RTS;


    /*********************************************************************/
    /************OUTD7-Outputs contents of D7 to the terminal ***********/
    /*********************************************************************/

    OUTD7: MOVEM.L    D0-A1,-(A7);
           LEA        DAT7OUT, A1;
           MOVE.L     D7,(A1);               /* CONTENTS OF D7 TO DAT7OUT */
           CLR.B      D0;                    /* CLEAR BYTE COUNTER */
    HEYY:  BSR        OUTCH;
           ADDQ.B     #1,D0;                 /* INCREMENT BYTE COUNT */
           CMPI.B     #$0D,(A1)+; /* INELEGANT METHOD OF INCREMENTING A1 */
           BEQ.S      HEYY;                  /* NEVER OCCURS */
           CMPI.B     #4,D0;                 /* AL REG' DONE? */
           BNE        HEYY;                  /* NO, THEN GET NEXT */
           MOVEM.L    (A7)+,D0-A1;
```

```
        RTS;

   /*******************************************************************/
   /********************** WERR2 *************************************/
   /** This is the second version of the error rate test program, which*/
   /** stores data on a stack which is printed out on completion of the*/
   /** required number of tests ***************************************/
   /*******************************************************************/

WERR2 : MOVEM.L  D0/D7-A1/A5,-(A7);
        LEA      @ERRSTACK,A5;        /* TOP OF ERROR COUNT STACK */
XXXX :  LEA      @PIA2,A2;
        LEA      MWERR,A1;            /* WELCOME MESSAGE */
XWERR2: BSR      OUTCH;
        CMPI.B   #$2D,(A1)+;
        BNE.S    XWERR2;
        CLR.L    D3;                  /* CLEAR TEST COUNT */
CWERR:  CLR.L    D1;                  /* CLEAR WORD COUNT */
        CLR.L    D2;                  /* CLEAR ERROR COUNT */
AWERR2: BSR      PIAIN;
        CMPI.W   #GOOD,D5;            /* CORRECT WORD ? */
        BEQ.S    CORRWD;              /* YES, THEN CONT' WITH TEST */
        ADDQ.L   #1,D2;               /* NO, THEN INC' ERROR COUNT */
CORRWD: ADDQ.L   #1,D1;               /* INCREMENT WORD COUNT  */
        CMPI.L   #WDCYC,D1;           /* WORD LIMIT REACHED */
        BNE.S    AWERR2;              /* NO, GET NEXT WORD */
        MOVE.L   D2,-(A5);            /* STORE ERROR COUNT */
        ADDQ.L   #1,D3;               /* INC' ERROR COUNT */
        CMPI.L   #TSTCYC,D3;          /* MAX' NUM' OF TESTS MADE? */
        BNE.S    CLWERR;              /* BACK TO START OF NEXT TEST */
        LEA      @ERRSTCK,A5;         /* TOP OF ERROR STACK AGAIN */
        LEA      MESSA,A1;            /* NUM' WORDS RECEIVED = ? */
TTTT:   BSR      OUTCH;
        CMI.B    #$3E,(A1)+;
        BNE.S    TTTT;

        MOVE.L   WDCYC,D7;
        BSR      WRITEO;  /* OUTPUT NUM' OF WORDS RECEIVED IN TEST */
        BSR      PRNTHX;
        CLR.L    D3;             /* CLEAR TEST COUNT */
ERRSC:  LEA      MESSOUT,A1;     /* NUM' ERORS RECEIVED = ? */
ERRSB:  BSR      OUTCH;
        CMPI.B   #$3E,(A1)+;
        BNE.S    ERRSB;
        MOVE.L   -(A5),D7; /* OUTPUT NUM' OF ERR'S RECEIVED IN TEST */
        BSR      WRITEO;
        BSR      PRNTHX;
        ADDQ.L   #1,D3;          /* INCREMENT TEST COUNT */
        CMPI.L   TSTCYC,D3;      /* ALL TESTS DONE? */
        BNE.S    ERRSC;          /* NO, DO NEXT TEST */
        MOVEM.L  (A7)+,D0/D7-A1/A5;
        RTS;

   /*******************************************************************/
   /****** WRITEO - This will output a bufferfull of hex data to the ***/
   /****** output device in the form of decimal characters. Can handle */
   /****** words up to 16-bits long in present configuration ***********/
   /*******************************************************************/
```

```
WRITEO : MOVEM.L  D3/D7-A3/A5,-(A7);
         LEA      OPBUFF+1,A3;   /* TOP OF BUFFER AFTER LENGTH */
         CLR.W    D4;            /* CLEAR BYTE COUNT */
         MOVE.W   D7,D2;         /* DATA TO D2 */
         TST.W    D2;
         BPL.S    POSNO;
         MOVE.B   #$2D,(A3)+;    /* DASH INTO BUFFER */
         ADDQ.B   #1,D4;         /* INCREMENT BYTE COUNT */
         NEG.W    D2;
POSNO:   MOVE.L   $10000,D3;
WRITO:   CLR.B    D0;
WRIT1:   ADDQ.B   #$1,D0;
         SUB.W    D3,D2;
         BPL.S    WRIT1;
         SUBQ.B   #$1,D0;
         ADD.W    D3,D2;
         DIVU     #$A,D3;
         TST.B    D4;
         BEQ.S    WRIT3;
OUTWR:   ADD.B    #$30,D0;
         MOVE.B   D0,(A3)+;      /* MOST SIG' DATA CHARACTER TO BUFF' */
         ADDQ.W   #1,D4;         /* INCREMENT BYTE COUNT */
WRIT3:   CMP.W    #$1,D3;
         BNE.S    WRITO;
         ADD.B    #$30,D2;
         MOVE.B   D2,(A3)+;     /* LEAST SIG' DATA CHARACTER TO BUFF' */
         ADDQ.B   #1,D4;        /* INCREMENT BYTE COUNT */
         MOVE.B   D4,@OPBUFF;   /* LENGTH OF BUFFER TO BUFFER TOP */
         MOVEM.L  (A7)+,D3/D7-A3/A5;
         RTS;


/*******************************************************************/
/************* PRNTHX - Prints the text string produced by Writeo ***/
/************* to the output device (ACIA) *************************/
/*******************************************************************/

PRNTX:   /* SPECIFICALLY FOR AFTER WRITEO, SO NO REG'S SAVED ON STACK */
        LEA       OPBUFF,A6;
        LEA       @ACIA,A0;
        CLR.W     D2;            /* CLEAR LENGTH COUNTER */
        MOVE.B    (A6)+,D2;      /* MESSAGE LENGTH TO D2 */
        SUBQ.B    #$1,D2;        /* DECREMENT LENGTH FOR DBRA */
PTO:    MOVE.B    (A6)+,D0;      /* DATA TO D0 */
        BSR       BOUTCH;
        DBRA      D2,PTO;        /* IF NOT ALL DONE,GET NEXT BYTE */
        RTS;

/*******************************************************************/
/********** BOUTCH - new version of outch using D0 *****************/
/**********          Needs acia address in A0 **********************/
/*******************************************************************/

BOUTCH: BTST.B  #1, (A0);      /* READY TO SEND */
        BEQ.S   BOUTCH;        /* NO, CONTINUE CHECKING */
        MOVE.B  D0,2(A0);      /* SEND CHARACTER */
        RTS;
```

```
     XOR     : DS.B   2;
     DAT7OUT : DS.B   4;
     OPBUFF  : DS.B  10;
```