ON THE FEASIBILITY OF USING STARLAN TO IMPLEMENT THE FASTBUS SERIAL NETWORK

By

Richard Cam

B. A. Sc. (Engineering Physics) University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

PHYSICS

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 1988

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of PHYSICS

The University of British Columbia
Vancouver, Canada

Date 28 OCTOBER 1988

Abstract

During the inception of FASTBUS, an autonomous data link, informally known as the FASTBUS Serial Network, was conceived as an auxiliary communications channel to be used in diagnostic applications for debugging FASTBUS systems. Since then, there have been several attempts at implementing this serial network, all of which have produced somewhat mixed results.

This thesis describes the latest attempt to implement a prototype FASTBUS serial network. By using a new LAN specification called StarLAN, some promising results have been obtained. These results show that StarLAN can be successfully adapted to implement the serial network without requiring major modifications. Furthermore, StarLAN seems to have the desirable characteristics of simplicity, low cost, an 'acceptable' data rate, and having multiple vendor support — advantages that previous implementation attempts had not possessed simultaneously. As such, StarLAN is the most promising solution to the serial network problem to have appeared so far.

This document will begin with a background discussion of FASTBUS, the FASTBUS Serial Network, and StarLAN. It will then discuss, in detail, how the prototype network was built, followed by a description and analysis of the performance measurements that were taken. It will also discuss considerations and options for a practical implementation of the serial network, based on experience from the work done with the prototype. Finally, a summary of the key results and an assessment of the StarLAN approach is given to conclude the thesis.

It is hoped that this document will be able to resolve many outstanding questions about a StarLAN-based serial network. Perhaps, FASTBUS users can now decide on

whether or not to use StarLAN to implement the FASTBUS Serial Network.

Table of Contents

List of Tables

List of Figures

Acknowledgement

The author gratefully acknowledges the contributions of the following individuals for their assistance in the course of the project that has culminated in this thesis. Of the staff at TRIUMF's microprocessor laboratory, Brian Evans and David Morris gave useful technical advice. The work area in the lab was borrowed from Robert Skegg. Layouts of all the printed circuit boards were done by Peter Bennett. Graham Waters was most helpful in showing how to access the laser printer through the network. PC's for the later performance measurements were generously provided by Prof. Glen Young and the Department of Harvesting and Wood Science at UBC. Consultation was sought on several occasions from Dr. H. Lee, formerly with the Department of Electrical Engineering at UBC, but who has since moved to Carleton University. Finally, I would like to thank my three supervisors, Bob Dobinson (now at CERN), Ken Dawson of TRIUMF, and Dr. Ed Auld of the Department of Physics at UBC, for guiding me safely through the past two years.

Chapter 1

Introduction

1.1 FASTBUS

FASTBUS is a data bus system used primarily in nuclear and particle physics experiments for high-speed data acquisition, processing, and control. It was developed from the late seventies through the early eighties in response to requirements from the high-energy physics community for a bus system with higher throughput in data acquisition and processing. Its development was prompted by the realization that existing data busses would be unable to cope with requirements imposed by future experiments. Improvements in accelerator technology as well as the increasing complexity of many experiments were leading to faster event rates and more measurements for individual events—factors which demanded higher data throughput than ever before in spite of the fact that most of the events measured in those experiments did not contain relevant data. FASTBUS was designed to alleviate this problem through two routes: first, by having higher data transmission rates (i.e., by 'brute force'), and second, by offering a multisegment architecture that allowed simultaneous data acquisition and processing (and therefore, real-time data *reduction*), thus reducing the load placed on the bus by the transmission of largely irrelevant data.

FASTBUS has been and still is used in many experiments at virtually all major nuclear and particle physics research facilities around the world. Although FASTBUS was designed for high-energy physics experiments, its capabilities do not limit it to those

applications. In general, it is feasible for any system that requires a bus with high data rates and multiprocessing capability. In 1984, FASTBUS was adopted as a formal ANSI and IEEE specification, the ANSI/IEEE 960 Standard.

1.1.1 Basic Operational Description of FASTBUS

FASTBUS [1] is a modular, segmented bus system: a typical application might have several 'Segments', each with its own set of devices. Each of these Segments operate independently but can also dynamically link with other Segments for inter-segment operations. A Segment has 32 multiplexed address/data lines plus additional lines for power distribution, control, timing, arbitration, and other functions. There are two types of Segments: One is a 'Crate Segment' where devices (i.e., electronic circuit cards) plug into connectors on a backplane in much the same way as peripheral cards plug onto the motherboard of a microcomputer. The other is called a 'Cable Segment' which, as the name suggests, links attached devices by multi-conductor cable. There are five types of devices which can attach onto a segment: 'Masters', 'Slaves', 'Processor Interfaces', 'Segment Interconnects', and 'Terminators'. A Master is a device which can acquire control of a segment (or other segments, in the case of inter-segment operations) and initiate operations. Slaves can only respond to instructions issued from Masters and cannot independently take control of the bus. Processor Interfaces link a Segment to a host processor (e.g., an external computer). Segment Interconnects link two or more Segments together temporarily so that inter-segment operations can be done. Segments are terminated at both ends by Terminators, which provide impedance-matching (to reduce reflections on signal lines). Crate segment Terminators are usually mounted on cards which contain electronic circuits to support functions related to timing, control, and addressing. Figure 1.1 shows the simple schematic diagrams of these two segments along with some attached devices.

[1] see [1], [2], and [3] for more details.

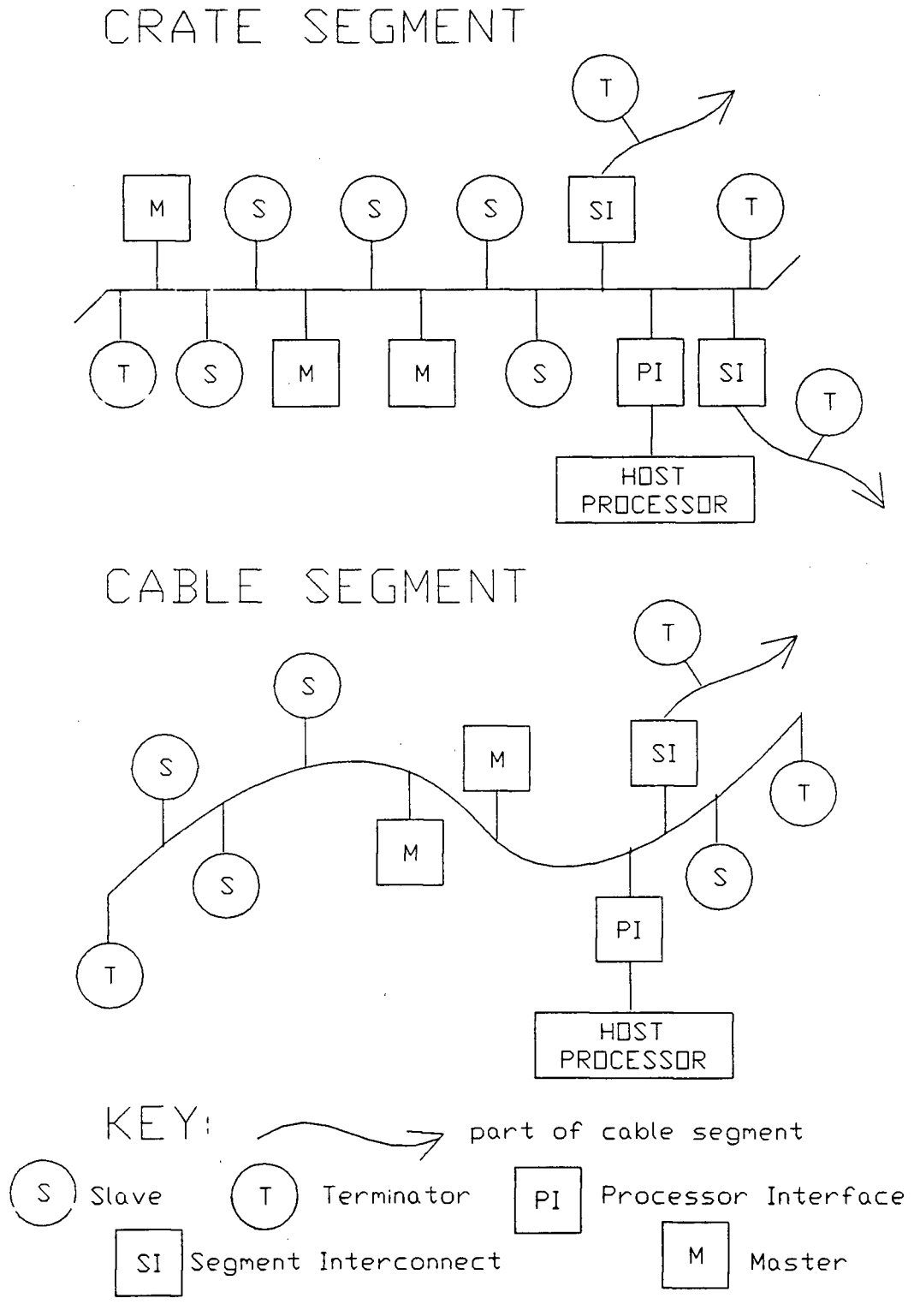


Figure 1.1: FASTBUS Segments

FASTBUS is different from many other busses in that more than one Master can be attached to the bus. This allows independent multiprocessing operations but complicates the system since a bus arbitration scheme is needed to decide which Master takes control of the Segment(s) in the event that more than one Master wants to use the bus at the same time. Furthermore, there is the possibility of 'starvation' among other Masters if one Master controls the bus for a disproportionate amount of time. Such a situation can occur because a bus Master may not be interruptible. (In FASTBUS, this problem can be avoided if all masters abide by the Assured Access protocol.) Nevertheless, the advantages of a multiple-master bus outweigh the added complexity by such a large margin that the current trend of high-performance busses is towards multiple-master systems (e.g., NuBus, used in the Apple MacIntosh II microcomputer).

Inter-segment communications are dynamically established by Segment Interconnects attached to the corresponding Segments. A Segment Interconnect physically couples a Backplane to a Cable Segment. These devices are responsible for handling inter-segment information traffic in a 'transparent' manner such that devices in different segments can be accessed as if they were intra-segment devices. They perform the dual tasks of isolating a segment from the rest of the system during intra-segment operations and connecting two or more segments together as if those were one contiguous segment when inter-segment operations are required. Route tables are used to determine the data path during inter-segment operations. The capability of inter-segment operations allows a multiple-master system to exploit its multiprocessing power not just within isolated segments (with their limited number of attached devices) but to the whole system. Furthermore, FASTBUS does not specify the topology for Segment interconnections. It does not specify which Segments should be linked or which should not. This gives the system designer maximum flexibility in adding inter-segment connections to suit the particular application and to maximize system throughput. For example, segments which often communicate with

each other should be grouped closely together so as to reduce inter-segment propagation delay. In addition, they should also have direct links to each other instead of indirect links (which pass over other Segments) so as not to disrupt intra-segment operations in those Segments. Figure 1.2 shows schematic diagrams of several interconnected Segments.

The number of Segments used in a FASTBUS application can run from as few as one to as many as several hundreds.

1.1.2 Motivation for the FASTBUS Serial Network

Regardless of the number of Segments, problems with a FASTBUS system can occur. Because of its complexity, it is usually not a trivial task to debug the bus system, especially if there are many inter-connected Segments and a large number of attached devices. Hence, as FASTBUS evolved, the FASTBUS Serial Network (FSN) was conceived to provide a system-wide communication network for diagnostic purposes. This network is autonomous from the rest of the bus system. Hence, it can remain effective even when the remainder of the system is not functional.

A typical FASTBUS diagnostic scenario might occur as follows: A faulty Segment Interconnect device is suspected to be causing problems with inter-segment operations. Data traffic between the affected Segments must be examined to determine if this is indeed the case. However, the FASTBUS system cannot be used to transfer data between those Segments because it is not operating properly—a broken FASTBUS system cannot be used to diagnose itself. Hence, the data must be sent either through an external facility or through an unaffected, internal, autonomous communication link.

Although the FASTBUS Serial Network was originally conceived for diagnostic purposes, its scope of application is not restricted to that area alone. The FSN can be used as the data channel in a 'backdoor initialization' scheme for starting up a FASTBUS system. For example, data for initializing route tables could be sent through the serial

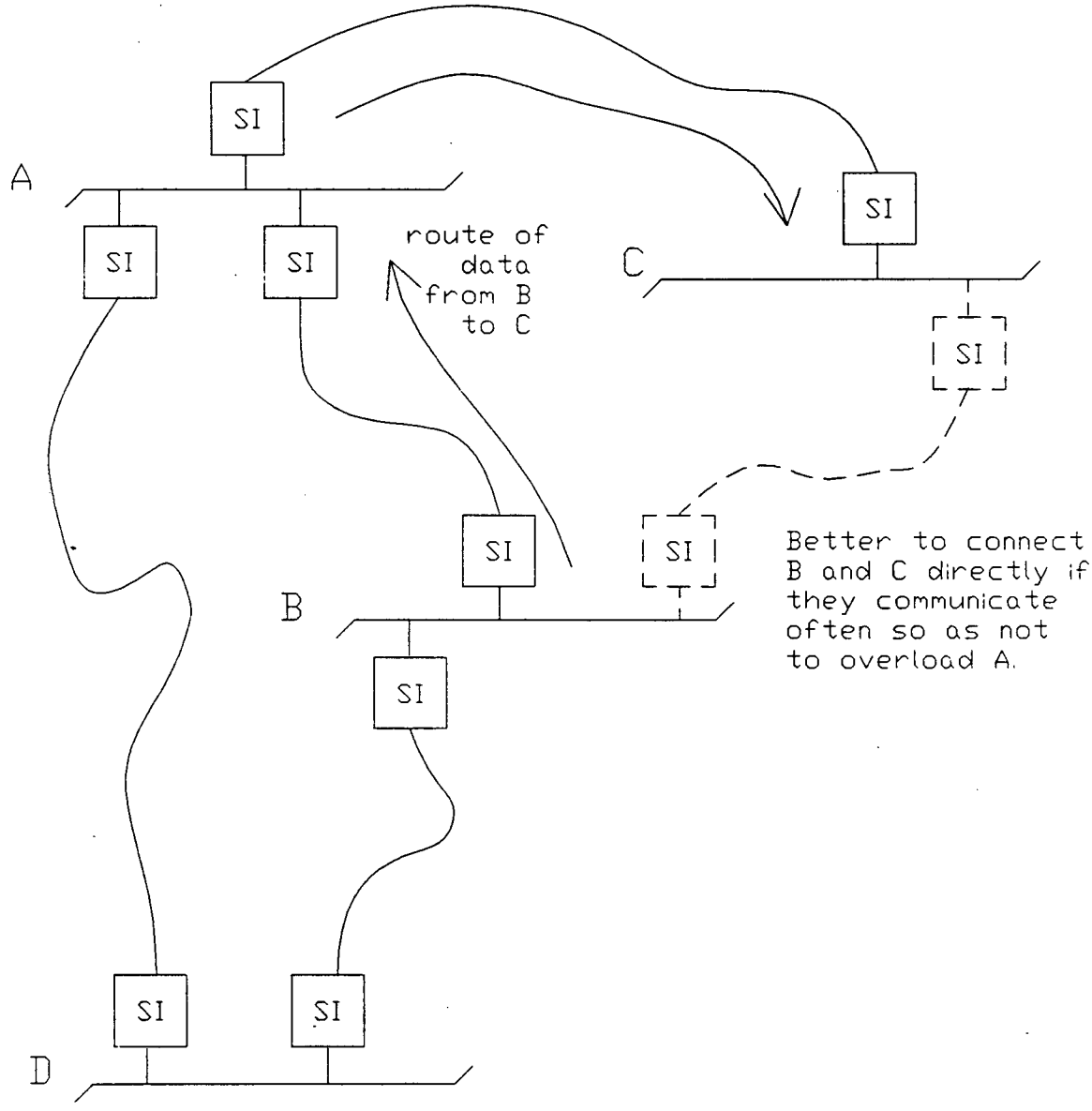


Figure 1.2: Interconnected FASTBUS Segments

network. Another application that could use the FSN is a system sentinel which monitors the FASTBUS system for abnormal operating temperature and power supply voltages. FASTBUS devices could periodically send data through the serial network to inform the sentinel of their operating conditions without placing an additional load on the inter-segment links. The FSN can also be used for remote switching of FASTBUS devices. A device can be controlled from a remote terminal through the serial network, which would be used to transmit programming data to the device and send back status information to the terminal. An operator would no longer have to open a crate and physically access the device to change its switches or read its status lights.

In spite of the wealth of applications which could use a separate data network, the FASTBUS Serial Network is, as of 1988, not yet operational. Formal specifications have not yet been given or agreed upon for either its implementation or applications. A brief history of the FSN (from [4]) is given in the next paragraphs to summarize past efforts and to put the current work into perspective.

1.1.3 Historical Background of the FASTBUS Serial Network

In mid-1979, a single line (SL) on the Crate backplane Segment was reserved for the FASTBUS Serial Network. This was done with the idea that some single-line, industry-standard network, such as Ethernet, could be used. However, there were problems with using only one line. For example, it was not practical to route the serial links in from one end of the Segment and out at the other end because each Segment would then contribute a large lumped capacitance to the line, resulting in impedance mismatches which, in turn, would cause enormous signal reflections (see figure 1.3). The line capacitance of the network would be very high since all transmitters and receivers (along with their load capacitances) were connected to the same line. A repeater could be used to link the external line to the Segment (figure 1.4). It would act essentially as a bridge, relaying

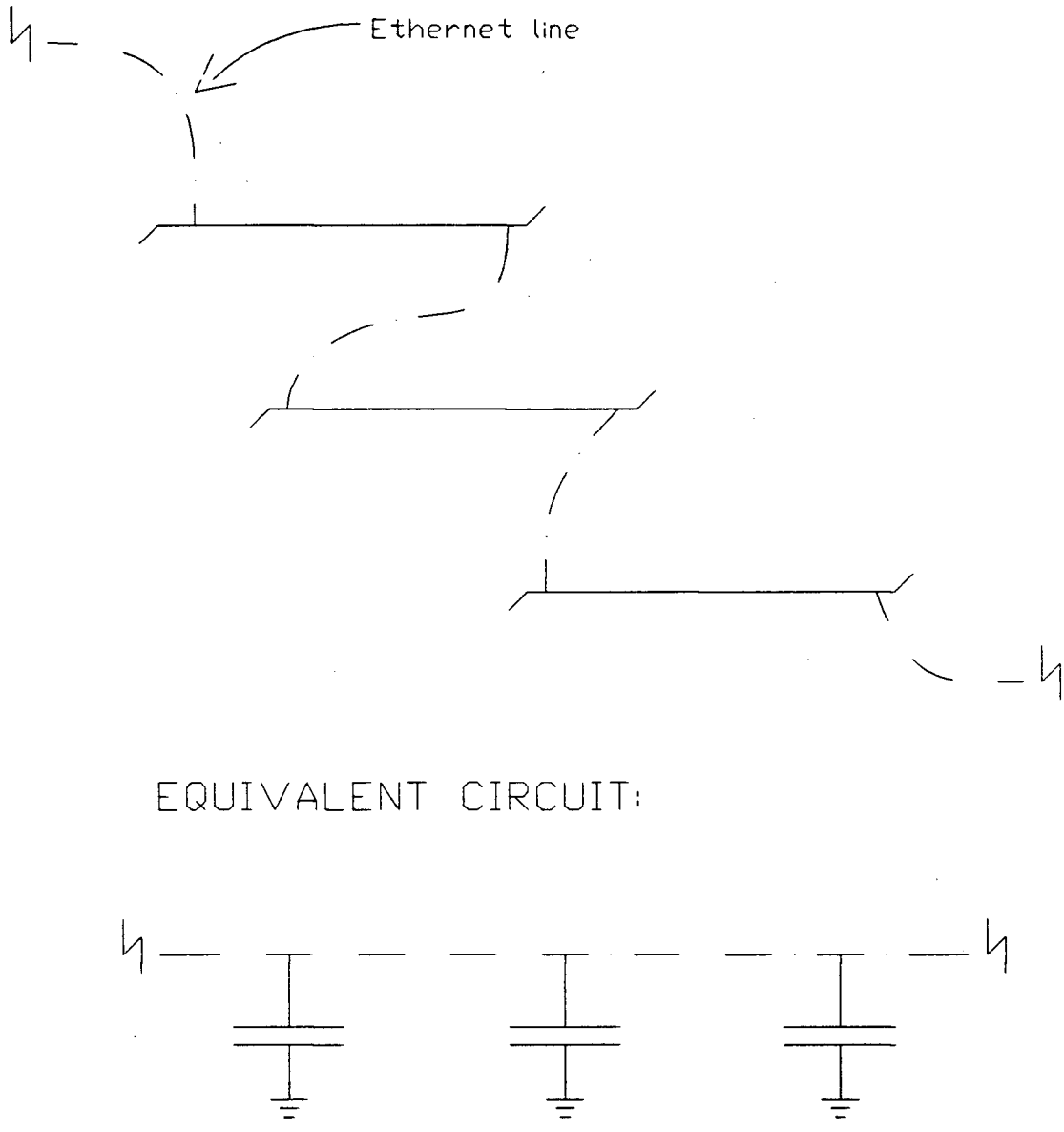


Figure 1.3: Single line direct routing scheme

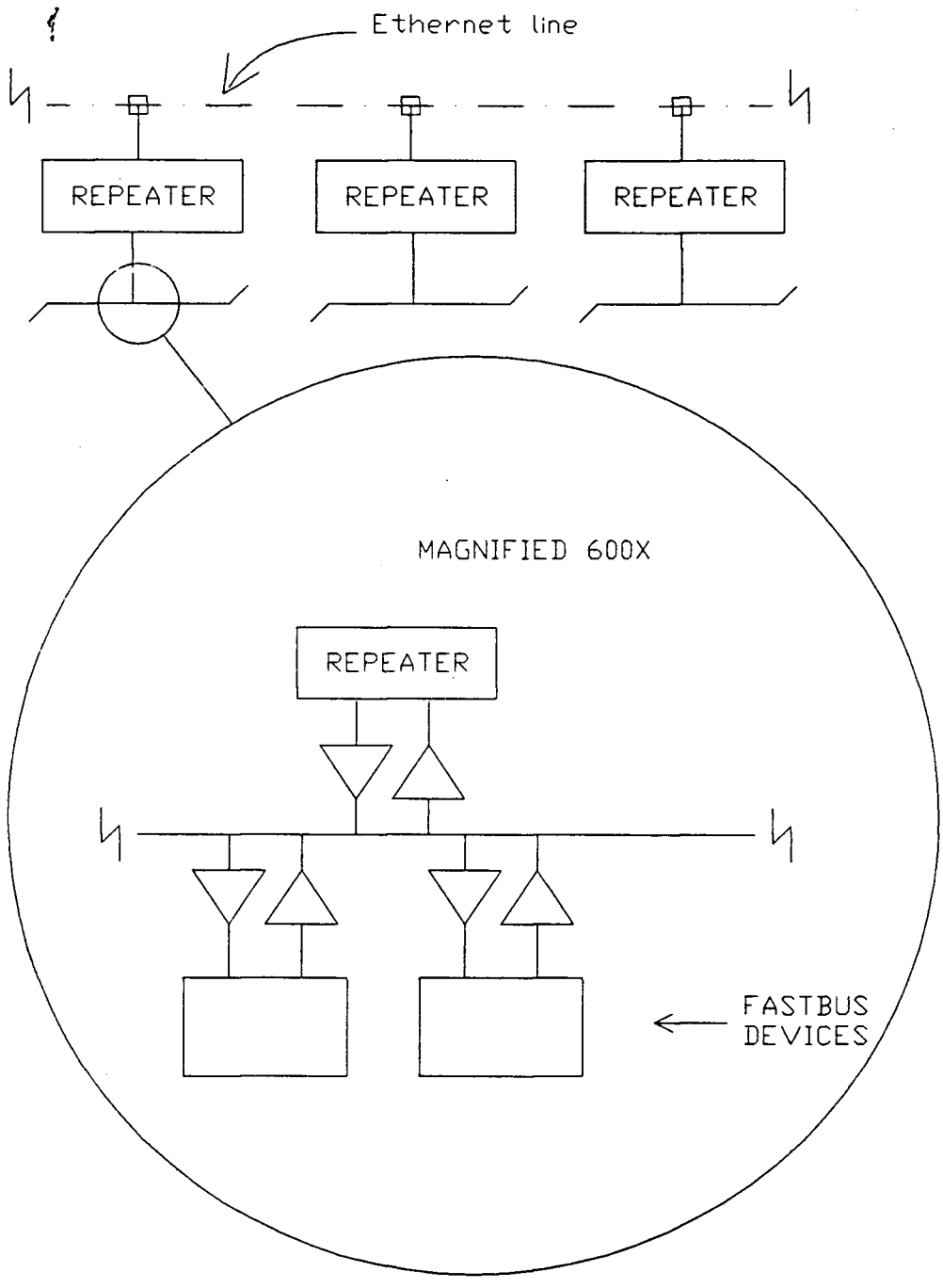


Figure 1.4: Single line repeater-based scheme

signals between the line and the Segment, while isolating the Segment's capacitive load. This was an expensive solution. Moreover, Ethernet, the candidate standard at that time, limited the number of such repeaters to only two per network. So, by October 1980, a second line (SLR) was added. (The current FASTBUS standard refers to these lines as 'Serial TX' and 'Serial RX'.) With this scheme (figure 1.5), one line would be used by all devices for transmitting while the other line would be used for receiving. A network 'bridge' would link data between the two lines and an external line. Overall capacitance load on the transmit and receive lines was also reduced since transmitters and receivers in each Segment were now connected to their respective lines instead of having to share one line.

At this point, Ethernet looked to be a very promising standard for the FASTBUS Serial Network. If integrated circuits that implemented Ethernet became available (as usually happens for popular industry standards), then it would be an easy task to use Ethernet for the FSN. However, Ethernet was also an evolving local area network (LAN) standard at that time and its specifications were not yet fixed. The data rate of Ethernet was initially set at 1 Mbps. This would have been ideal for the FSN since simple collision-detection circuitry could be used. However, that data rate was changed to 10 Mbps for higher LAN throughput, complicating not only the task of detecting collisions but also that of processing data faster. It was still possible, in principle, to use Ethernet, but it would require more expensive electronic circuits and, more significantly, more circuit board area. Hence, the idea of using Ethernet for the FASTBUS Serial Network was dropped and the search was on for suitable alternatives.

In October 1980, a research group at the Stanford Linear Accelerator Center (SLAC) began to develop a prototype serial communication network based on an SDLC (Serial Data Link Control) chip (the Zilog Z80-SIO). This system had a data rate of up to several hundred kbps and occupied only 13 cm^2 of circuit board area. The hardware was built in

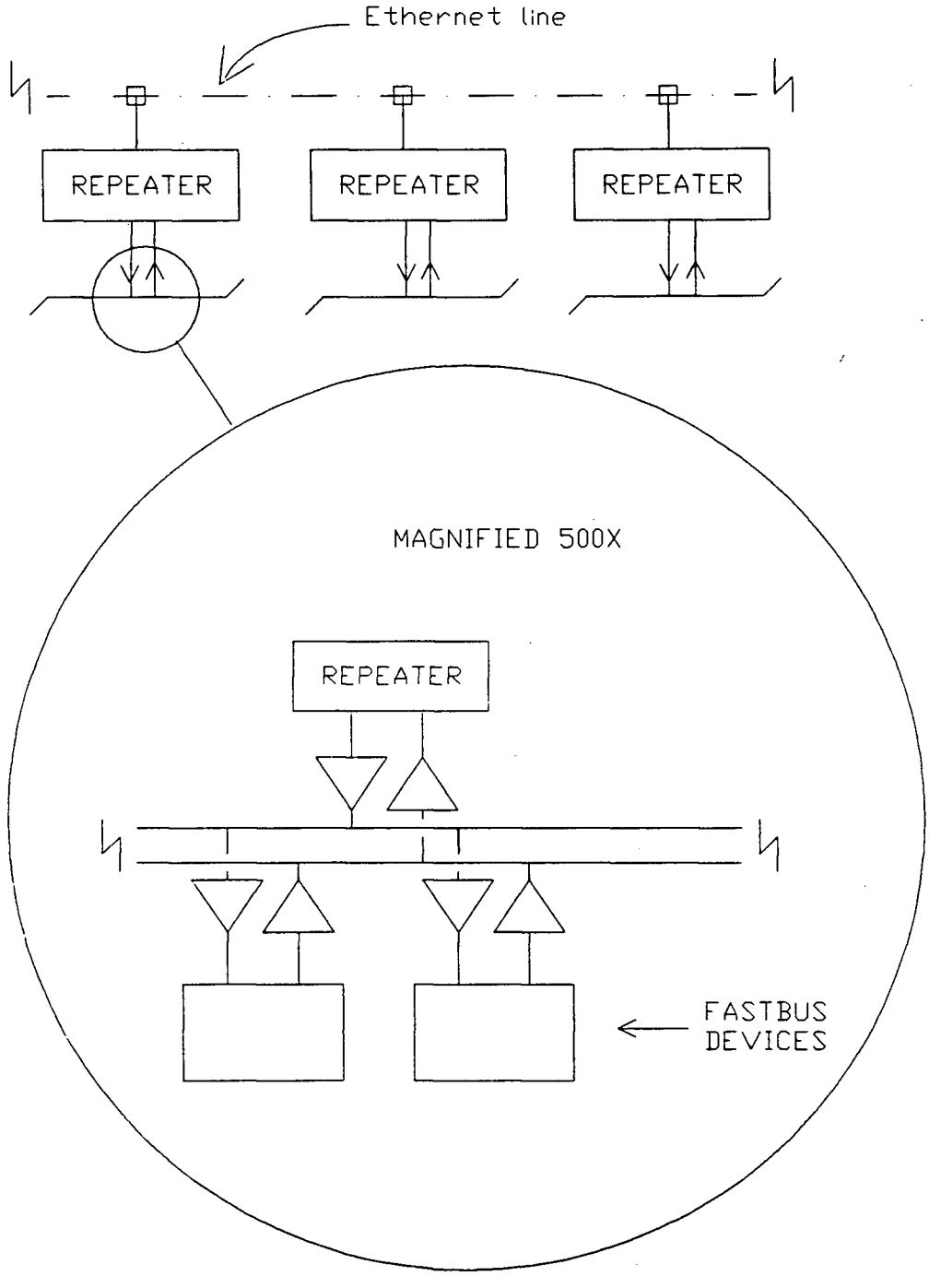


Figure 1.5: Two-line connection

1981 and some support software was developed shortly afterwards. However, the SLAC group was reluctant to implement the prototype because they were still hoping for an industry standard network to evolve that would fit their needs.

The same group then tried to use the AppleTalk(tm) network. This network was based on a newer SDLC chip, the Zilog Z8530, and required only 1 in^2 of board area. It had a data rate of 230400 bps, which was slow enough for data processing without using a DMA (Direct Memory Access) controller (which would have been required for higher data rates, such as that of Ethernet). It was also compatible with the two-line serial network scheme and was already supported by commercial software for data transfer. This system was not implemented either, in spite of its promising characteristics, because the group wanted to delay standardization until the 'last possible moment' just in case a better system became available.

And, perhaps, a better system may have arrived. StarLAN, a new LAN standard, seems to have most, if not all of the features desired for the FASTBUS Serial Network. It has a data rate of 1 Mbps, which is not too slow, but slow enough to allow the use of simpler collision detection schemes and less expensive electronic circuitry. It still requires DMA support, but DMA controllers are less expensive now than they were before. Furthermore, CPUs with built-in DMA controllers are now available. But the most important aspect of StarLAN is that it is an industry standard, with wide multi-vendor support. StarLAN is similar in many respects to Ethernet (already a very popular and widely used LAN standard) and hence, can be easily connected to that network by a StarLAN-Ethernet 'bridge'. Many integrated circuit manufacturers have developed chips for StarLAN controllers, making it possible to build circuits occupying very little board space. Although StarLAN is more expensive than AppleTalk(tm), it has better prospects for long-term use and expansion. As of this time (late 1988), a better alternative to StarLAN has yet to come. StarLAN is a fairly young standard (it was standardized in

early 1987) and thus, is not expected to become 'obsolescent' at least for the next few years. Preliminary work on a StarLAN-based serial network has been done at TRIUMF (and perhaps also in other places). The contribution from TRIUMF is the subject of this thesis.

1.2 StarLAN

StarLAN, officially known as the IEEE 802.3 1BASE5 standard [2], is a new local area network (LAN) standard designed for office integration and automation. (Local Area Networks are short-distance data communication facilities connecting computers at distances of between a few meters to several kilometers apart, typically within a single or a localized cluster of buildings.) It was developed as a low-cost alternative to Ethernet (an earlier LAN standard) [3] for use in systems based on personal computers. These systems typically do not require the high performance of Ethernet, and consequently, cannot justify the higher connection costs associated with those networks. At that time (early 80's), existing LAN standards were designed primarily for centralized mainframe and minicomputer systems. A study done in 1985 [7] showed that only a small fraction of the estimated 10 million PCs being used were networked. This finding was especially significant in office environments, where PCs have become increasingly popular but function more efficiently when networked together with other resources in the office such as printers and mass storage media (data bases). These unconnected PCs represented a huge, untapped base of users waiting for a more affordable LAN. With the number of PCs expected to reach up to 20 million units by 1990, the development of cheaper LAN's became ever more urgent—and lucrative. StarLAN is one of the most recent entries in this class of LAN's.

StarLAN derives its lower costs from three characteristics. Firstly, it uses a relatively low data rate (1 Mbps), so that slower, and therefore cheaper, electronic circuits can be used. Secondly, it is designed to use, as the transmission medium, existing spare telephone cables in a building, eliminating additional costs of routing new cable. Telephone companies typically put more telephone wiring in a building than is initially required

[2] see [6]
[3] see [5]

because it is cheaper and easier to tap spare telephone wires than to route new cable through the building when more telephone lines are needed. Finally, StarLAN is an international standard with wide, multi-vendor support. Users can select StarLAN products from many manufacturers instead of being 'captive' customers, which usually happens with communication protocols developed by a single vendor (because they are not compatible with those of other vendors).

1.2.1 StarLAN in the Framework of the OSI Model and Other IEEE 802 Protocols.

A complicated series of operations is usually required when data are sent between computers. These operations comprise two basic sections: coding and routing. The coding section is concerned with how data from the source computer is presented or 'shown' to the destination computer. Source and destination computer systems may use different internal character collating sequences (e.g., ASCII and EBCDIC), in which case text messages will have to be translated so as to have the same appearance in the destination computer. Data containing confidential material may have to be encrypted for the purpose of privacy. The routing section is responsible for sending the coded data reliably and efficiently from source to destination. Error detection and/or correction codes are usually appended for use by the destination system to check for integrity of the message received. In a complicated network, the source data must be sent through intermediate points before reaching its final destination. Efficient and robust routing algorithms are required in such cases to select the (sub-)optimal path through the available intermediate points. The physical media for data communications are diverse and include, to mention a few, coaxial or fibre-optic cable, point-to-point radio-frequency links, and microwave satellite channels. Protocols, or sets of rules, are agreed upon between the sender and receiver for all aspects of data transfer such as the operations described above, as well as

for initiating and terminating transfers and recovering from situations where some part of a message is either lost or received with error(s).

The operations involved in moving data from one point to another are components in a communication system which help to implement its only goal: to send data *reliably* from the source to its destination. The sheer complexity involved in sending computer data (of which only the essential details were outlined above) requires a 'structured' scheme for all data-transfer operations. Several computer corporations, as well as national and international organizations have developed schemes, or 'architectures', for organizing these data-transfer operations. Of these, two will be discussed briefly as they are very closely related to StarLAN. The first is the OSI (Open System Interconnection) 7-Layer Reference Model specified by the International Standards Organization (ISO) and the second comprise the IEEE protocols, which are being developed by the IEEE 802 Standards Committee of the Institute of Electrical and Electronics Engineers (IEEE) in the United States.

The OSI Reference Model

OSI is a complete architecture for data communication. It is defined by a hierarchy of seven operational 'layers', each of which has its own set of functions, independent of the other layers. OSI does not specify the protocols for these layers, it only outlines the duties and responsibilities for those protocols. Figure 1.6 shows a schematic diagram of the seven layers. The application layer contains application-specific programs for the data communication requirement at hand. The presentation layer is responsible for functions such as character code conversion, text compression, and data encryption. The session layer sets up and maintains connections between applications. It is responsible for user authentication and for re-establishing interrupted or lost connections. The transport layer makes data transfer between applications as independent of the underlying network

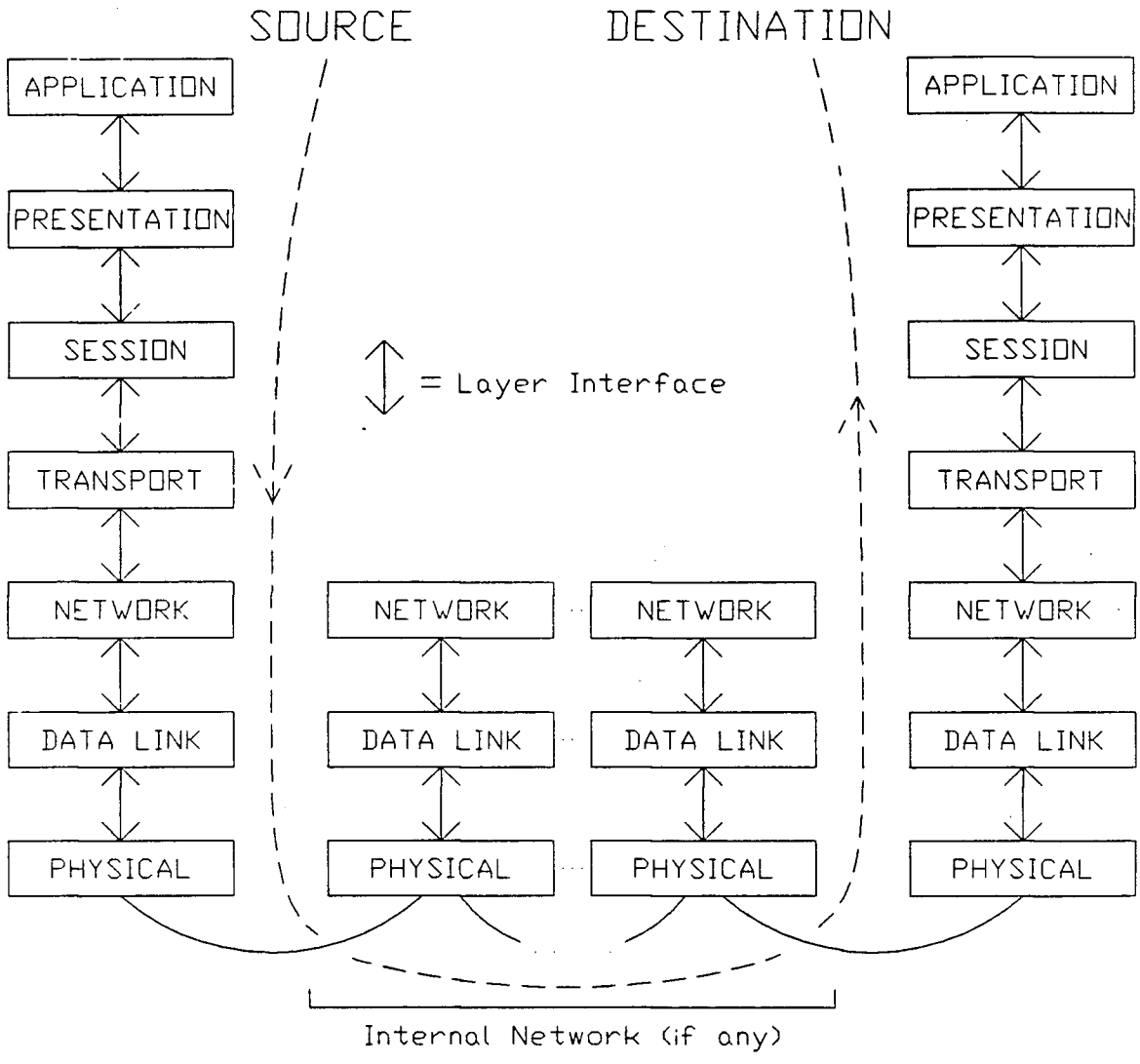


Figure 1.6: OSI 7-Layer Reference Model.

as possible. These four highest layers cover 'end-to-end' protocols because they concern source and destination applications only and are independent of the network used to physically connect them.

The lowest three layers specify network access protocols, whose functions concern only the network, independent of the particular application involved on the higher layers. The network layer provides services for setting up and maintaining the flow of messages in a network. It is responsible for directing message packets towards their destinations. Since source and destination users may not be connected by a direct physical link, the network layer is also responsible for routing and switching data messages through any intermediate junction in a network. The data link layer is responsible for message frame synchronization as well as error detection and correction. It is responsible for retransmission of message frames received in error. In general, it provides an apparently error-free connection between a transmitter and receiver. The physical layer specifies electrical and mechanical parameters for a direct physical link. These include the type of medium used (such as coaxial or fibre-optic cable), the signalling method (modulation technique, amplitude and period of electrical signals, etc.), as well as the connection scheme (e.g., number of connector pins, types of connectors or plugs). It is concerned with transmitting and receiving bit streams of message frames sent by the data link layer.

A communication network may not require all of the layers. Local area networks for example, typically do not use the transport, session, and presentation layers. Each layer must be 'open' to adjacent layers, that is, a protocol implemented at any one layer should be able to interface to *any* protocol of the adjacent layers. This means that a protocol at a layer should not have to know the implementation details of the other layers. If the implementation of one layer has to be changed for any reason, only the interfaces connecting that layer to adjacent layers will have to be changed, while protocols for the other layers remain unchanged. The OSI reference model provides an architecture for

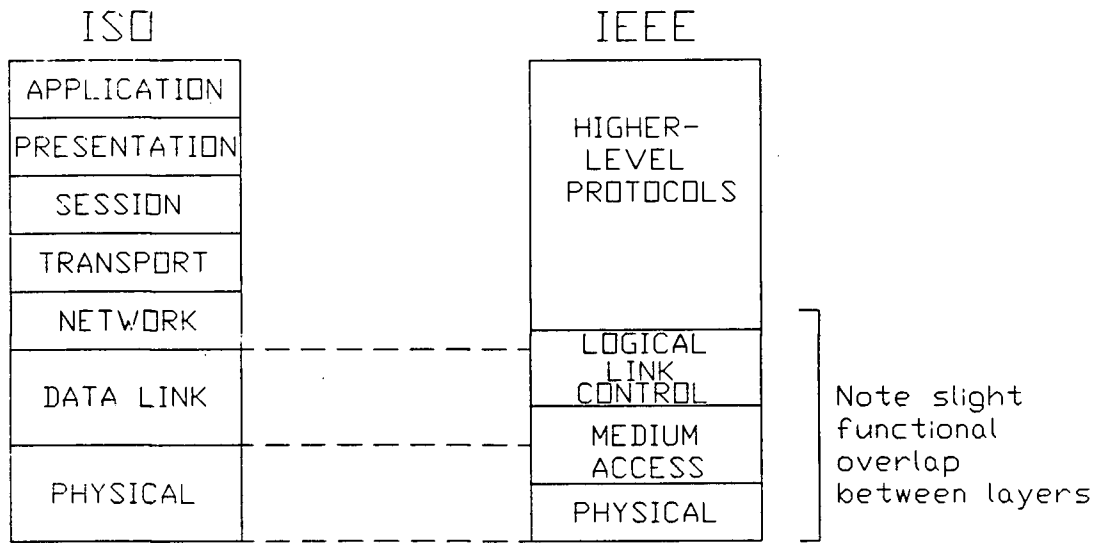


Figure 1.7: IEEE Local Area Network Architecture.

easing the task of developing and, just as important, maintaining a data communication system.

The IEEE 802 Protocols

The IEEE, through its 802 Standards Committee, has developed protocols for local area networks. These protocols are designed to resolve incompatibilities that could exist in LAN equipment supplied by different vendors. They do not conform exactly to the OSI model but are based on a closely related architecture shown in figure 1.7. As can be seen, the differences involve primarily the network access protocols.

The IEEE model essentially splits the OSI data link layer into two sublayers, one responsible for logical link control (LLC), the other for medium access control (MAC). The MAC layer implements the access method used in the network, which may be CSMA/CD, token ring, or token bus (to be discussed later). The LLC layer provides the remaining

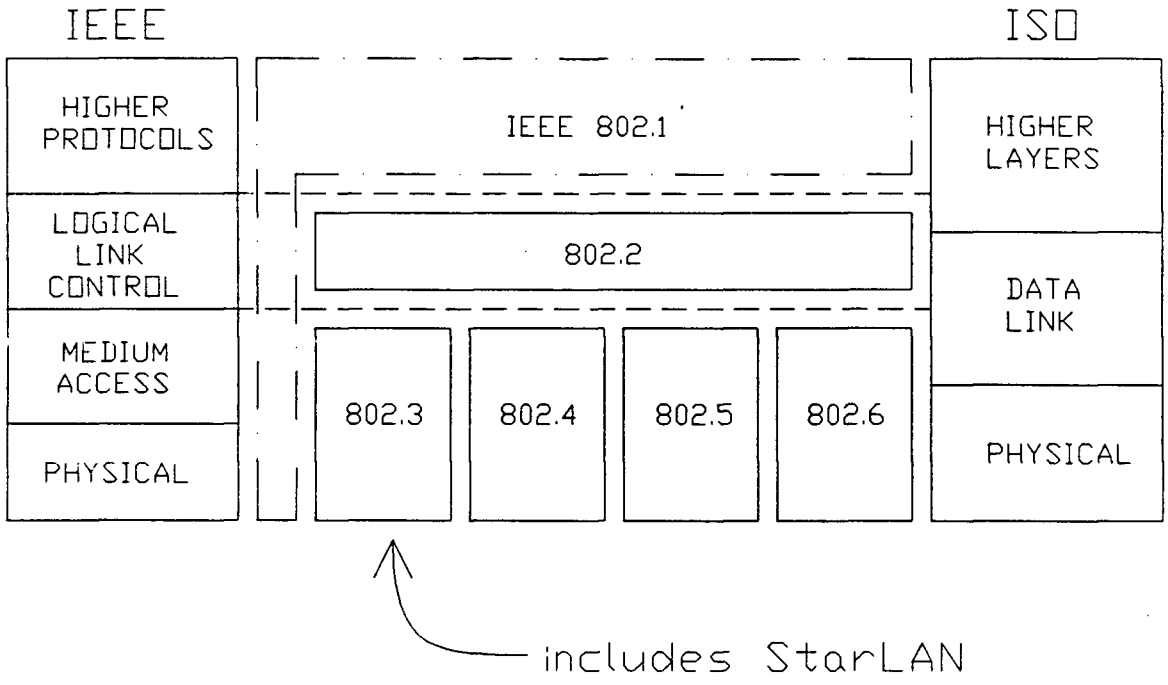


Figure 1.8: IEEE 802 Protocols

data link services, independent of the medium access technique used. IEEE 802 protocols conform to this modified architecture. They are enumerated below and a diagram showing where these protocols lie in the IEEE local network reference model is given in figure 1.8.

IEEE Standard 802.1 Describes the relationship between other 802 protocols and their relationship to the OSI reference model.

IEEE Standard 802.2 Common logical link control.

IEEE Standard 802.3 Group of protocols that use the CSMA/CD access method.

IEEE Standard 802.4 token-passing access method, bus topology.

IEEE Standard 802.5 token-passing access method, ring topology.

IEEE Standard 802.6 metropolitan area network.

StarLAN is a subset of the 802.3 group. It defines the implementation for a local area network at the physical and MAC layers. As such, it defines parameters for physical connections and the access method which, in the case for 802.3 protocols, is CSMA/CD.

1.2.2 Basic Operational Description of StarLAN

StarLAN, as the name suggests, has a star topology as shown in figure 1.9. The simplest form of the star topology is analogous to a bicycle wheel with a hub at the center, from which spokes radiate outwards. Message frames transmitted by connected nodes are sent to 'hubs' which in turn relay them to higher-level hubs in the network. Of course, there are a finite number of hub levels. The single hub at the highest level, at which there is no further connection to a topologically 'higher' level, is called the header hub. All other hubs are called intermediate hubs. When a message reaches the header hub, it is sent back down by that hub to be propagated to the rest of the network by lower-level hubs.

There are essentially two other network topologies for interconnecting nodes in a network. These are the bus and ring topologies, shown in figure 1.10. Two IEEE standards, 802.4 and 802.5, use bus and ring connection schemes respectively. Ethernet, another IEEE 802.3 protocol, also uses the bus topology. This is one significant difference between Ethernet and StarLAN.

StarLAN does not specify which transmission medium should be used to connect hubs and nodes in the network. It only requires any node-node or hub-node link to have a maximum propagation delay of 4 bit times and no more than 6.5 dB attenuation in the signal frequency spectrum between 500 kHz and 1 MHz. There is no limit to the number of connections at each hub layer but a maximum of only five hub layers are allowed in the network. Data bits are transmitted at a rate of 1 Mbps using Manchester encoding

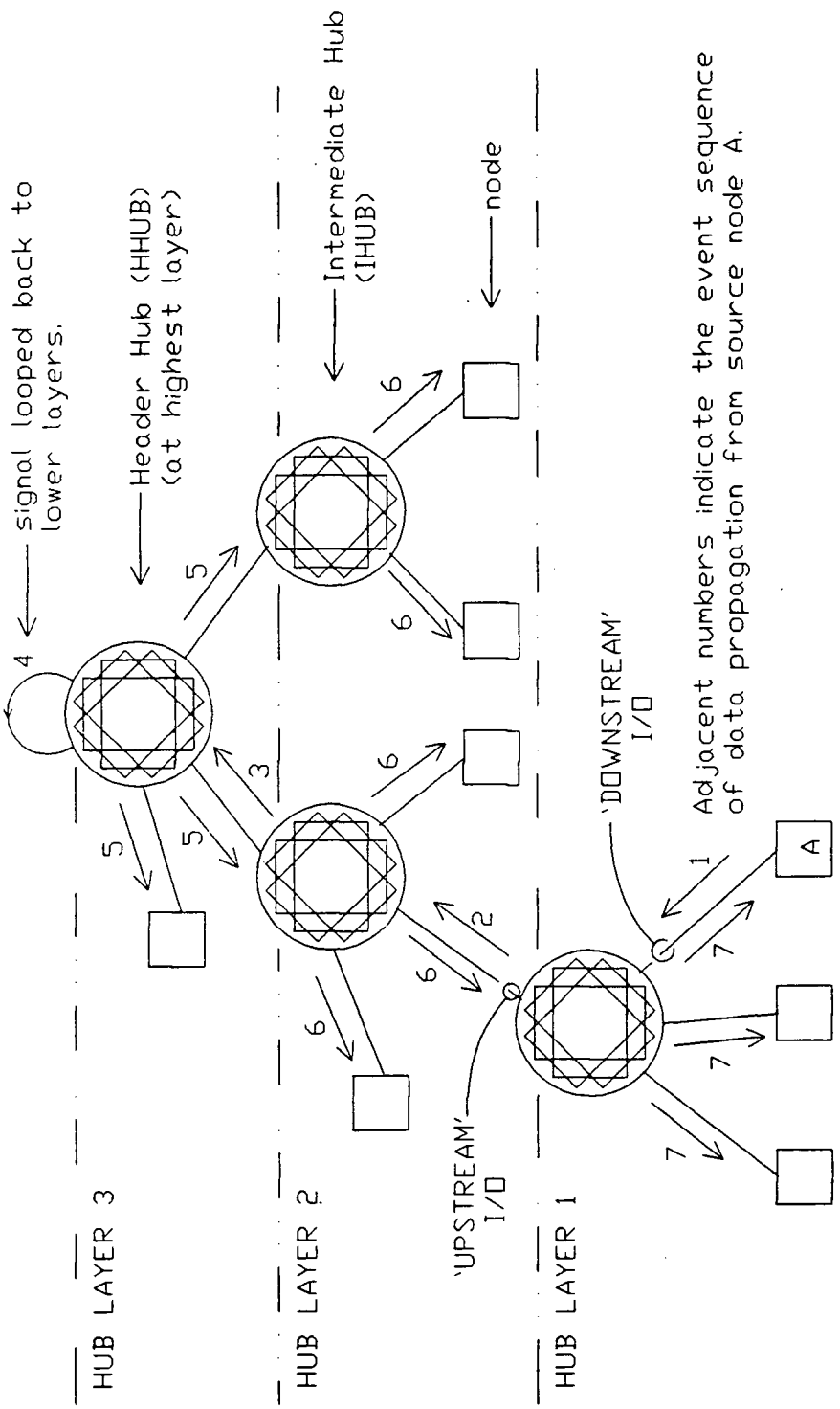


Figure 1.9: Star Topology

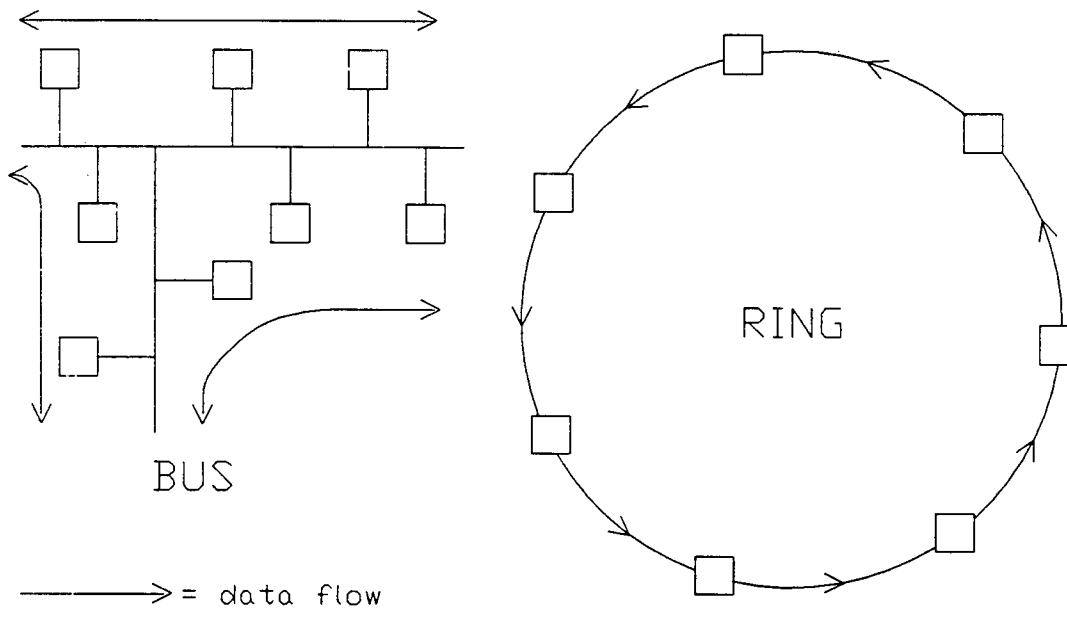


Figure 1.10: Bus and Ring Topologies

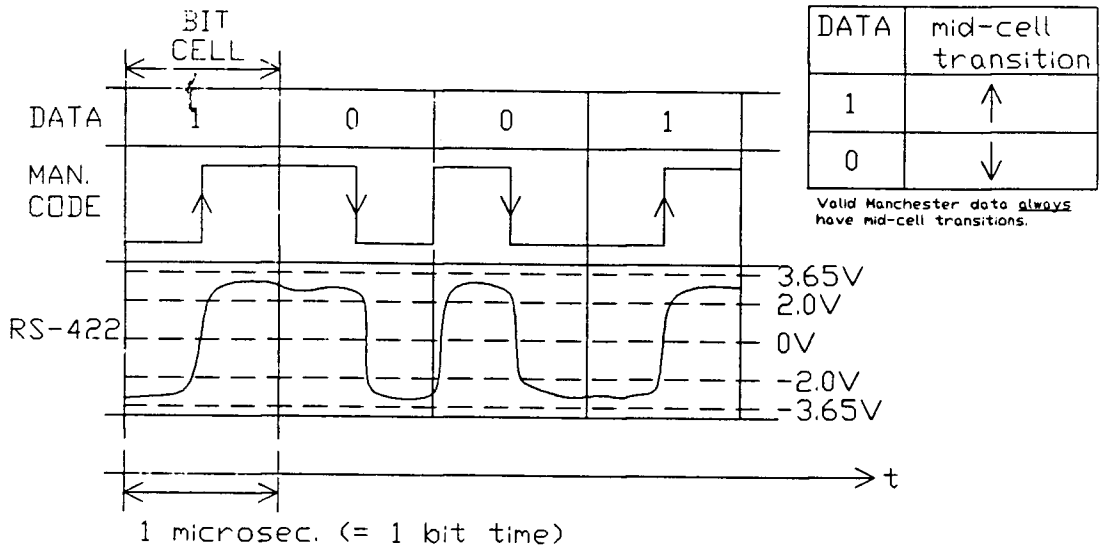


Figure 1.11: Manchester Encoding and RS-422 Voltage Levels.

and RS-422 voltage levels as shown in figure 1.11.

Since no signal multiplexing scheme is used, only one channel is available for all users connected to the network. It has to be shared and this means that only one user or node can be in the process of transmitting a message frame at any given time. Two or more simultaneous transmissions will result in a 'collision' where the messages involved become superimposed and consequently garbled. In StarLAN (and Ethernet as well), all nodes employ an access method called CSMA/CD (Carrier Sense Multiple Access with Collision Detection) when sending message frames. This method is described below.

When a node has a message to send, it first checks its receive line to make sure that no other node is transmitting. If no one else is using the network, the node begins transmission. Otherwise, it waits for the other node to complete transmission first. That was the 'carrier sense' part of the access method. The 'collision detection' part now follows. Because there is a finite propagation delay (due to hub processing and cable transmission delays) for a signal from one node to reach all the others, the remaining

nodes in the network will, for a brief interval, not know that a transmission has already started. If one or more of those nodes begin transmission on the erroneous assumption that no one else is transmitting, then a collision will occur. When a transmitting node detects a collision (by monitoring the signal on its receive line), it halts its current transmission and sends a brief jam pattern to enforce the collision (i.e., increase the level of collision) so as to ensure its detection by other nodes involved with the colliding transmissions. After the jam pattern has been transmitted, each node 'backs off' by ceasing transmission for a random length of time before retransmitting the entire message. This reduces the probability of having two or more nodes with the same back-off period, which will result in another collision when those nodes begin retransmission. When the backoff period expires, the node then waits until the channel is free before retransmitting its message frame. A collision can occur again during any retransmission attempt. If that happens, the backoff process described earlier is repeated again.

CSMA/CD belongs to a class of access methods which rely on channel contention or 'channel grabbing' to send messages. Another way for accessing a common channel avoids contention altogether by providing a transmission sequence for users connected to the network. This technique, called token passing, involves the passing of a 'token' (a special message frame giving the node receiving it the sole right to transmit) around the network (see figure 1.12). A node possessing the token uses the network during its allocated time for any messages to be transmitted. When its time is up, the node passes the token onto the next node in the ring. In that way, only one node transmits at a time, avoiding collisions. As mentioned before, this is the access method used in two related IEEE protocols, 802.4 and 802.5.

A StarLAN message frame is composed of several fields as shown in figure 1.13. Message transmission always starts with the preamble which is used for synchronization. The end of the preamble is marked by a start-of-frame delimiter (SFD). Source and

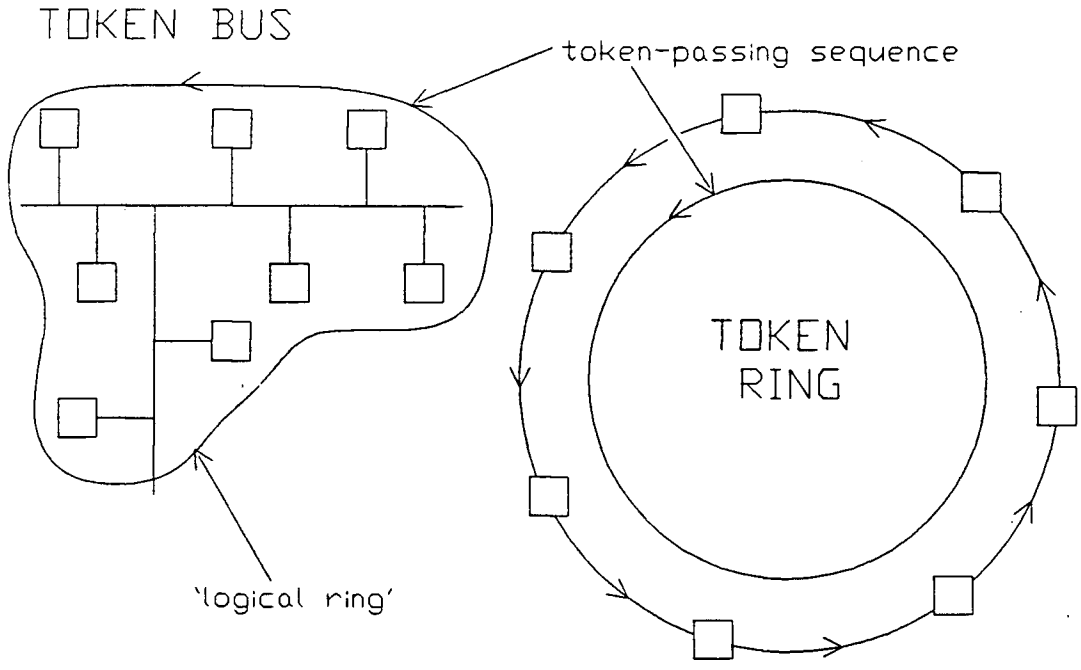


Figure 1.12: Token-passing Access Method.

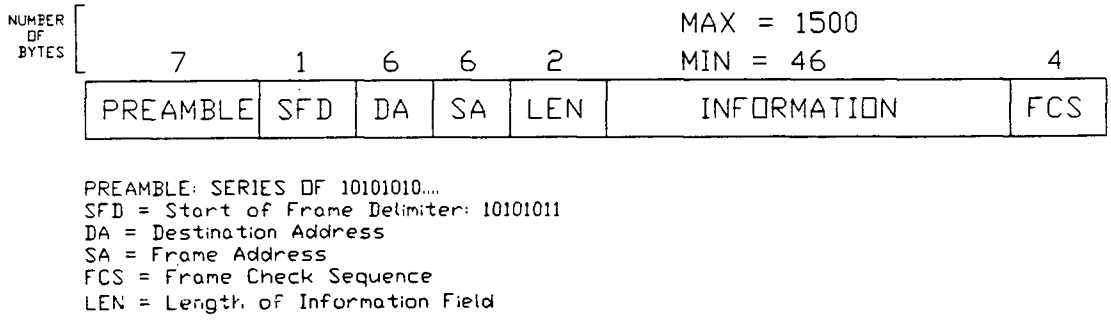


Figure 1.13: StarLAN framing format

destination address fields then follow the SFD. These identify the node from which the message originated and the node to which the message is intended respectively. All nodes check the destination address field of an incoming frame to determine whether or not they should receive the remainder of the frame, which consists of the information field followed by the frame check sequence (FCS). The information field contains data used for the particular communication application while the FCS is a cyclic redundancy check (CRC) code which is calculated by an algorithm that uses the bits of the address and information fields. When a node receives a frame, the CRC is recalculated and compared with the FCS, which was calculated when the message was originally transmitted. If the two do not match, then an error exists in the frame (which may have been caused by, among others, noise in the tranmission medium). Not all errors can be detected by a CRC. The effectiveness of a CRC (and FCS) depends on the algorithm as well as the amount and distribution of bit errors in a message. 100% error detection cannot be guaranteed for all circumstances by any CRC algorithm.

StarLAN specifies a minimum interframe spacing (IFS) of 96 bit times (96 $\mu sec.$). This is the minimum time between adjacent frames sent in the network. Its purpose is to give processors in receiving nodes enough time to 'recover' between receptions of back-to-back frames. Received frames are usually not processed immediately on reception, but are temporarily stored in buffers. The interframe spacing gives receivers some time to do short tasks after receiving a frame such as switching receive buffers so that each received frame is stored in its own buffer area. A node waiting for another one to complete transmission has to wait an additional IFS period before transmitting, to ensure that all frames are spaced apart by at least this minimum separation time. When a StarLAN node detects the absence of carrier (when no other node is transmitting), it waits for an additional IFS period, after which it transmits regardless of whether or not the channel has become busy (by another transmitting node) after that period.

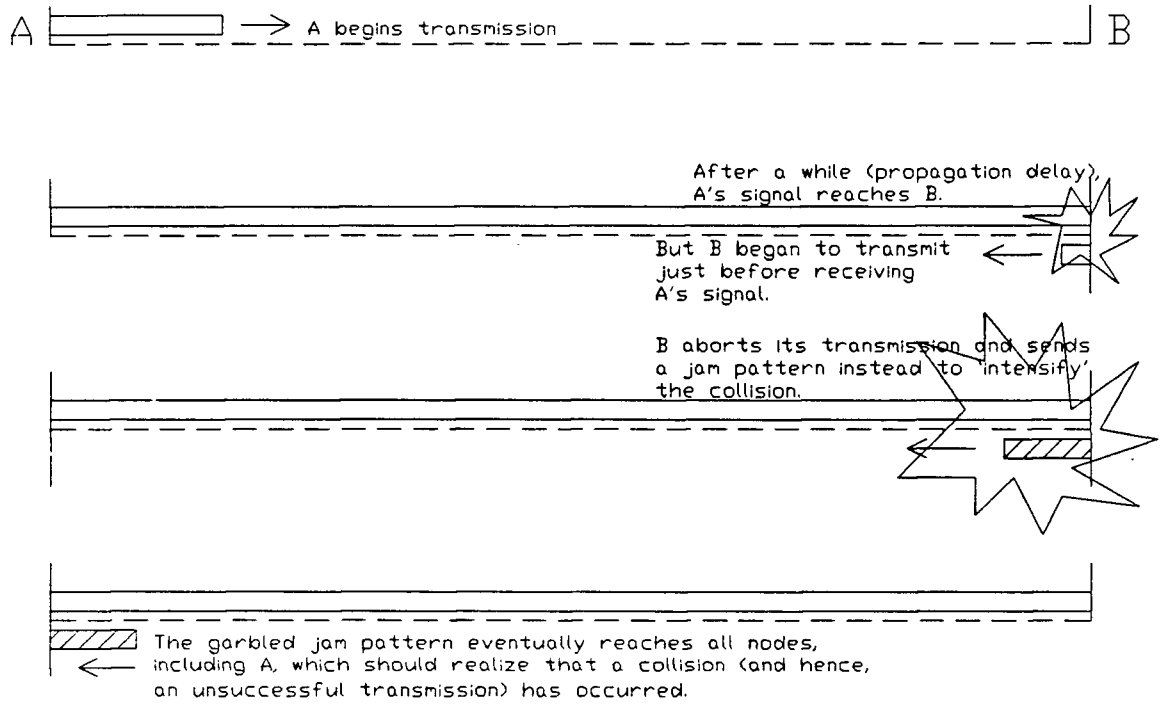


Figure 1.14: Worst-Case Scenario for Collision Detection.

StarLAN specifies several parameters related to collision detection and recovery. Minimum jam time (set at 32 bit times), is the minimum length of time that a node transmits a jam pattern when a collision is detected. Slot time (set at 512 bit times), determines the backoff times when a collision occurs as well as the minimum message frame length. The slot time is computed as the jam time plus the longest time taken for a signal to reach the farthest node and return back (the 'round-trip' propagation delay). It is the worst-case time for a collision to be detected. Figure 1.14 shows how this conclusion is arrived at. A node must still be in the process of transmitting a message to detect a collision. If it completes its transmission without detecting a collision and a collision actually occurred, the frame is not retransmitted and is lost. For this reason, message

frames sent in a CSMA/CD environment must be longer than the slot time. A colliding node backs off an integral number of slot times so as to avoid overlapping into any collision interval that might occur with nodes retransmitting (and still colliding) at an earlier time. This number is determined in StarLAN by what is called a 'truncated binary exponential backoff' algorithm. The number of slot times to back off after the kth retransmission attempt (of the same frame) is randomly selected from the range of $[0, 2^n)$, where n is the smaller of k and 10. Hence, the range of slot times from which a colliding node randomly selects its backoff time is doubled each time a backoff period fails to resolve the initial collision. This doubling stops after the 10th retransmission attempt to simplify the algorithm's implementation. Clearly, the probability of having a collision on retransmission decreases as the range for selecting the backoff period increases. StarLAN limits the number of retransmission attempts (stemming from the original collision) to 15. In other words, a node has up to 16 chances to successfully transmit a frame. If that limit is reached, the node ceases further retransmission and informs the higher layer protocol of the unsuccessful frame transmission, otherwise, it indicates that the frame was successfully transmitted.

In a StarLAN network, collision detection is performed first by the hubs. When a hub senses that more than one of its 'downstream' input lines is transmitting simultaneously it begins to transmit a collision presence signal (CPS). This is nothing more than a bit stream with Manchester code violations (see figure 1.15). Other hubs higher up in the network will receive this signal and transmit their CPS upstream. The header hub then relays this signal back down the network so that all nodes receive the CPS and are thus informed that a collision has occurred. Collision enforcement and backoff is then performed by the nodes which transmitted the colliding frames. Hubs stop transmitting their CPS when transmission ceases on all of their downstream inputs.

Hubs are also responsible for jabber inhibition— a hub should shut off its link to

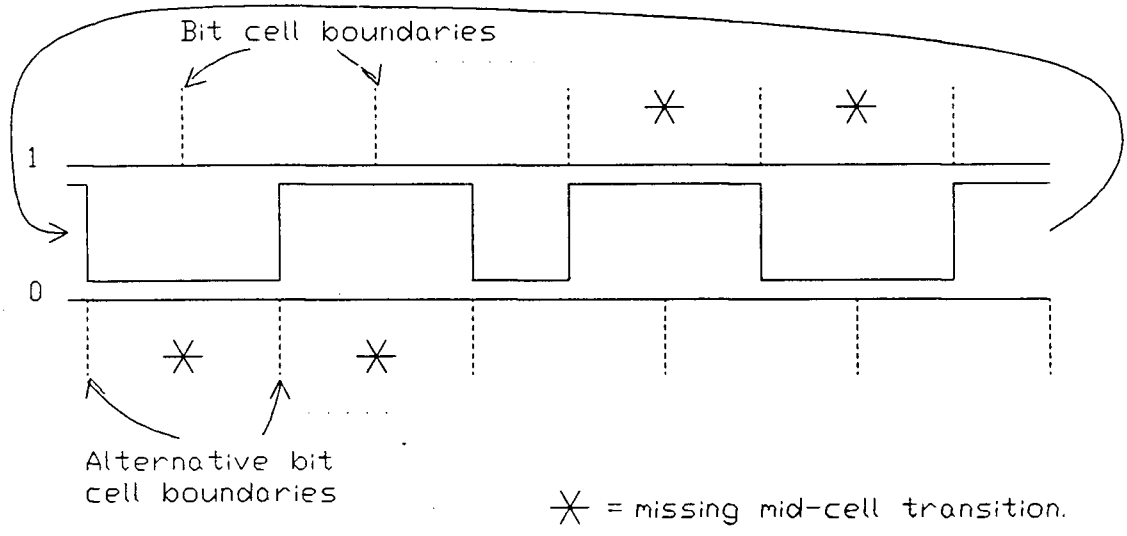


Figure 1.15: Collision Presence Signal

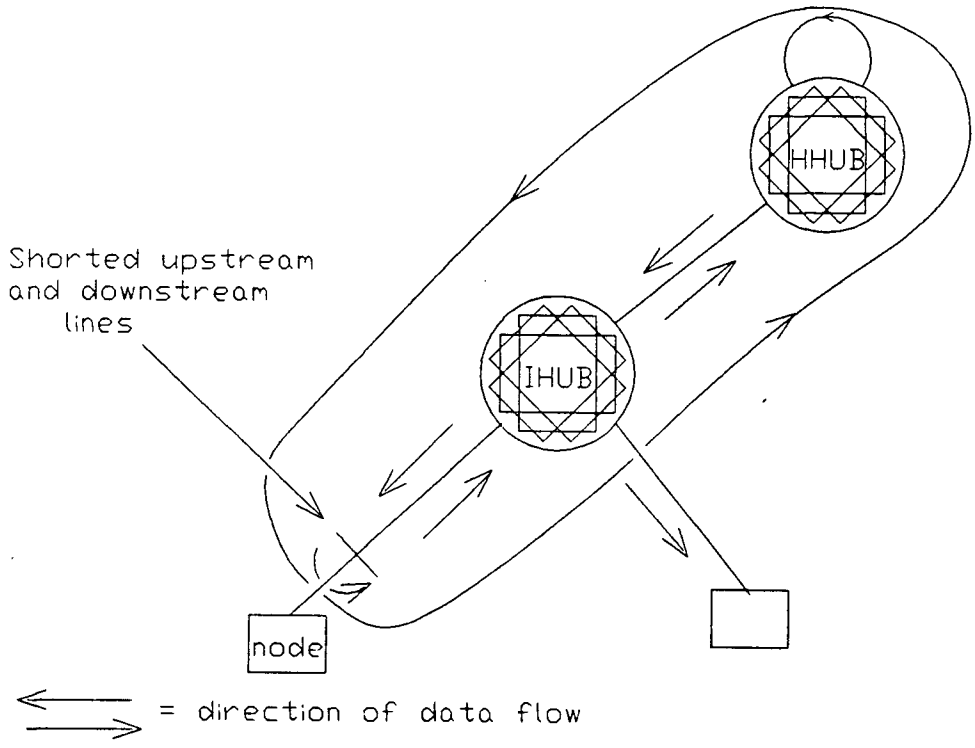


Figure 1.16: Infinite Loop in a StarLAN Network.

any downstream input that transmits for an inordinately long time. This prevents long transmissions from tying up the network. Such a situation can occur if an upstream and downstream line are accidentally connected together. When that happens, the downstream signals will be coupled upstream and be transmitted through the network in an infinite loop (see figure 1.16). When one of a hub's downstream inputs is transmitting much longer than it should, the hub responds by sending a CPS (forcing a collision back-off sequence) to get all nodes to cease transmission momentarily. If its offending input does not cease transmission, then that input is logically disconnected from the network

(by simply being ignored by the hub). An inhibited input will be readmitted to the network only when the hub receives a transmission from that input while no signal is being sent downstream from the higher hub layers—this will never occur if the upstream and downstream lines are short-circuited together as discussed previously.

Figure 1.17 gives a schematic diagram of a 'typical' StarLAN network in a hypothetical office. Users can up/download files and application programs from remote data bases and computer systems. Electronic mail can be sent between users while output devices such as printers can be shared. The wiring details for a sample office are shown in figure 1.18. Note how the star topology blends neatly with the structure of the telephone wiring system. StarLAN was designed with this environment in mind.

1.2.3 StarLAN and the FASTBUS Serial Network

As mentioned earlier, StarLAN also seems to be a very promising candidate for implementing the FASTBUS Serial Network. One more reason for why this is so is the star topology, which allows FASTBUS crate segments to be easily connected to a StarLAN network. Figure 1.19 shows a schematic diagram of a StarLAN-based serial network on a hypothetical FASTBUS system. Masters, Slaves, and Segment Interconnects attach to the serial network as StarLAN nodes. The SERIAL TX line is used by the nodes for transmission while the RX line is used for reception. A StarLAN hub, mounted on a card, is plugged at a slot in the back of each FASTBUS crate segment. These hubs are connected to higher-level hubs as in an orthodox StarLAN network. So, it can be seen that the serial network is just like an ordinary StarLAN network except for the lowest hub level (at each FASTBUS crate), where the nodes and hubs' downlink portions are connected in a bus topology, using different voltage levels (ECL as opposed to RS-422 for orthodox StarLAN). Crate segments can be easily connected to or disconnected from the network by adding or removing their own link to the higher-level hubs. Although

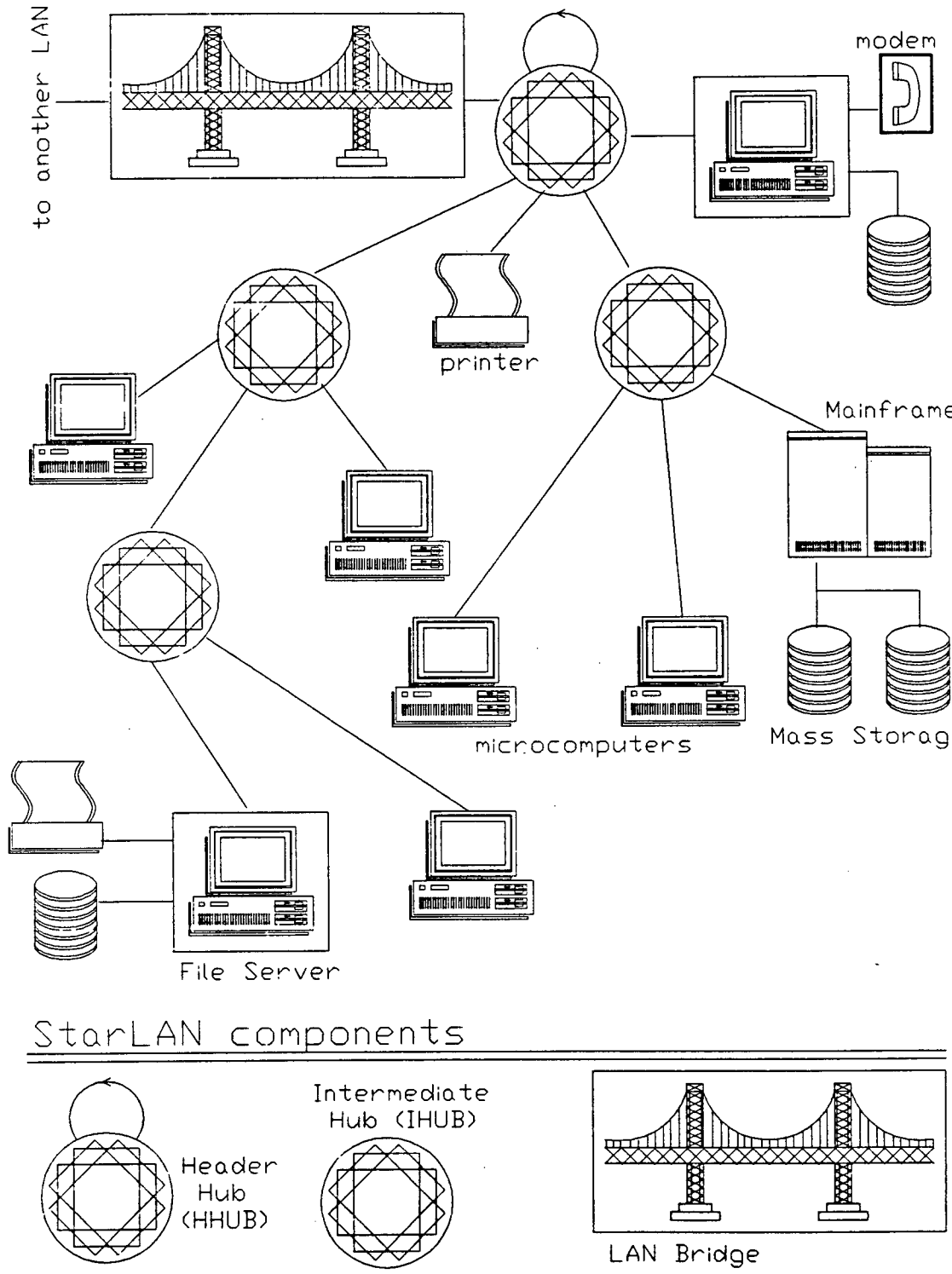


Figure 1.17: StarLAN in an integrated office

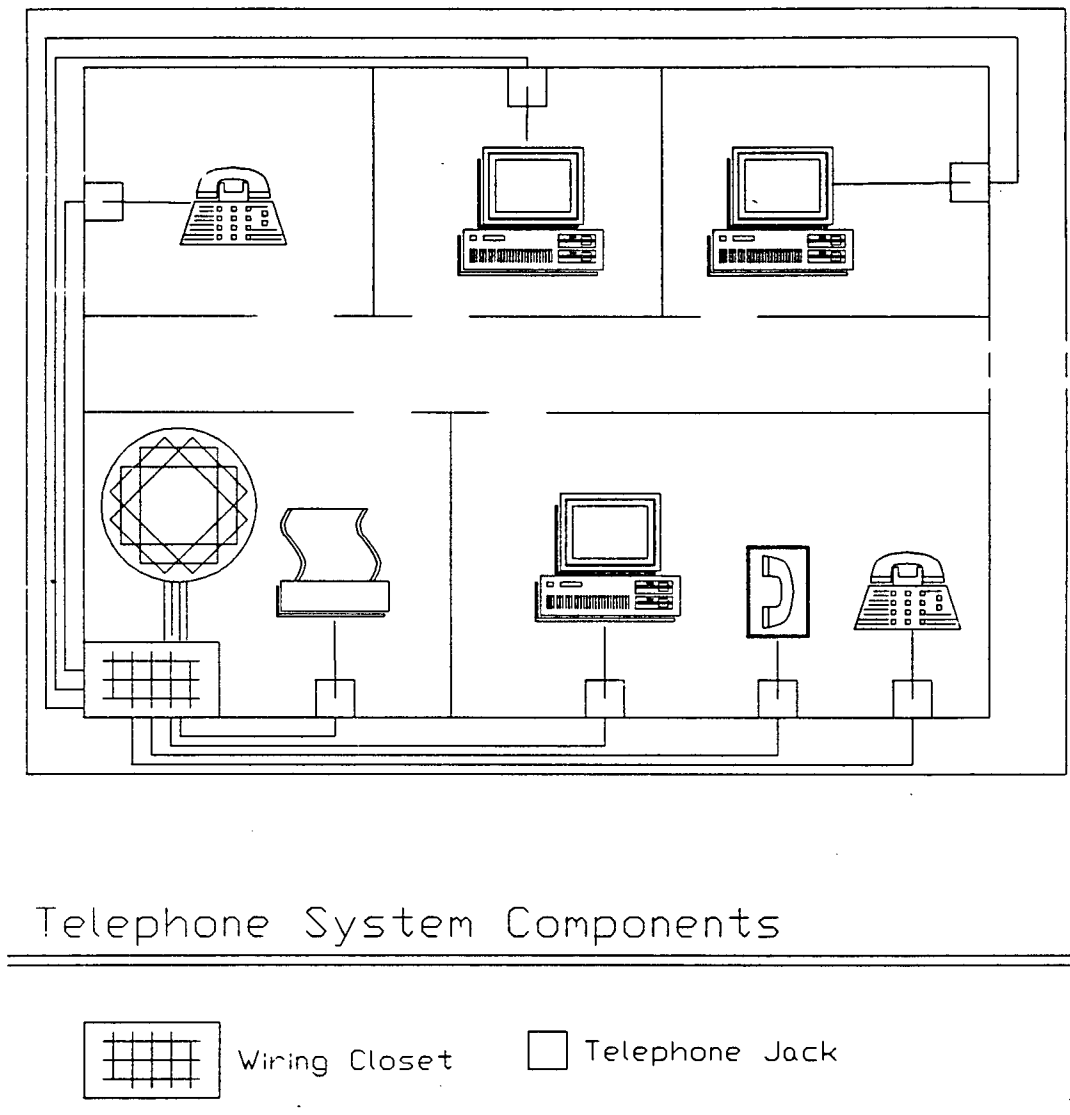


Figure 1.18: StarLAN network using intra-building telephone cables

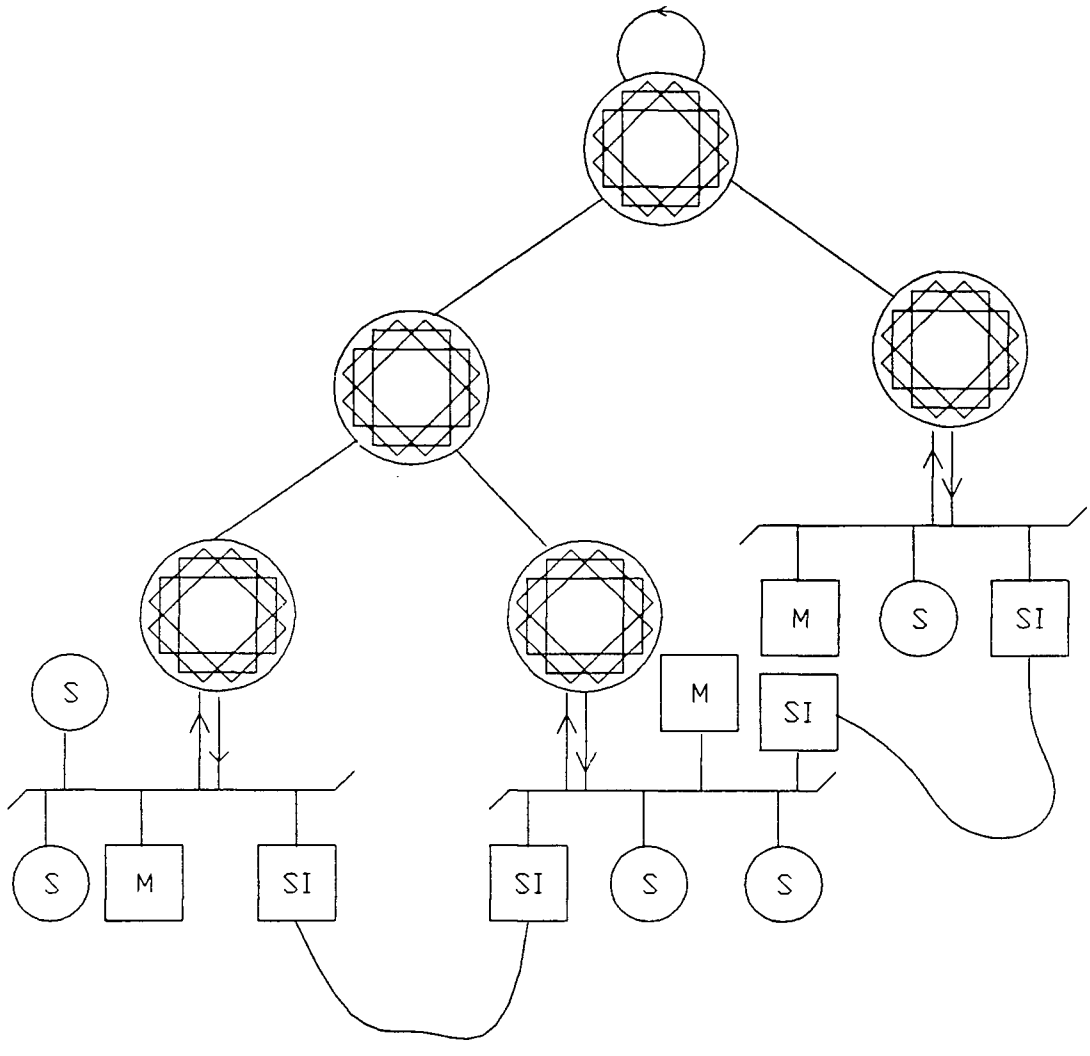


Figure 1.19: StarLAN-based serial network on FASTBUS

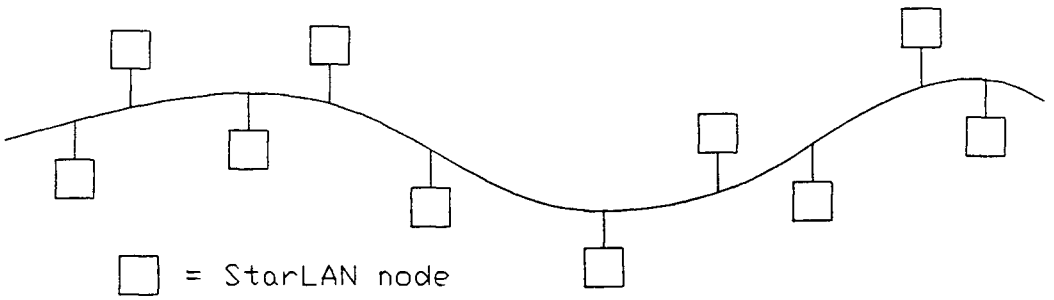


Figure 1.20: Bussed StarLAN network

FASTBUS segments can be connected in an arbitrary fashion, data paths during inter-segment operations are determined by route tables which prevent crosspoints and loops from occurring (they do not occur in StarLAN because of jabber inhibition as well as the star connection topology). Hence, operations in the StarLAN-based serial network and inter-segment operations in a FASTBUS system function with equivalent topologies.

1.2.4 Variations to Standard StarLAN

StarLAN without Hubs

Spurred by the drive to reduce network connection costs even further, some designers, principally at Advanced Micro Devices (AMD), took the step of designing a variant of StarLAN which eliminated hubs entirely ([18], [19]). This variation, known as 'bussed' or 'hubless' StarLAN, eliminates hubs, the most expensive component in a standard StarLAN network. In this scheme, all nodes are connected together in a bus topology as shown in figure 1.20. Figure 1.21 shows two possible ways for connecting the nodes

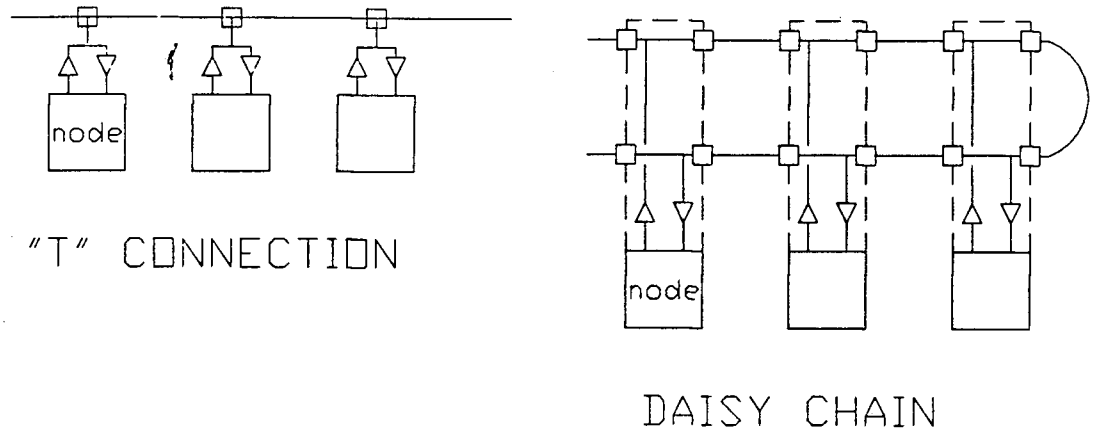


Figure 1.21: Node connection schemes for bussed StarLAN

together. In the T connection, the cable (typically coaxial cable) is tapped by a connector which penetrates the insulation and outer shield to make contact with the inner conductor (which carries the signal). In the daisy-chain configuration, adjacent nodes are connected by pieces of cable. Each node splits the cable terminations into two, one for the node itself, the other for connection by another length of cable to the next node in the chain. As of this time, the daisy chain configuration seems to be the more feasible mode of connection. Two observations can be made immediately about a bussed StarLAN network. First, intra-building telephone cable can no longer be used because of the bus topology. This means that additional wiring has to be used, increasing network connection costs to some extent. Second, node connection and disconnection is now more complicated, especially for the daisy chain scheme favored by AMD. Furthermore, bussed StarLAN is not yet an official IEEE standard. Detailed specifications are still being developed for approval by the IEEE 802 Standards Committee. At any rate, bussed StarLAN may have some useful features in the context of the FASTBUS Serial Network. Two possible connection

schemes are shown in figure 1.22. Figure 1.22a shows a completely bussed network while figure 1.22b shows a hybrid scheme using both standard and bussed StarLAN. Finally, it must be emphasized that, unlike the connection scheme illustrated in figure 1.19 (using standard StarLAN), the bus schemes have not yet been prototyped for feasibility studies and performance evaluation.

10 Mbps StarLAN

In the latest twist to the StarLAN saga, a high-speed, 10 Mbps version of standard StarLAN is being developed [8]. This version was conceived in response to consumer demand for a local area network that was faster than StarLAN but could still use intra-building telephone cable. Apparently, many LAN users (or budding LAN users) prefer the data rate set by Ethernet at 10 Mbps and are unwilling to use a cheaper 1 Mbps LAN even though the lower data rate is perfectly suitable for their applications. The 10BaseT Study Group of the IEEE 802.3 Standards Committee is in charge of the development for this new LAN standard and has discussed specifications to be adopted. The Electronic Industries Association (EIA) is expected to publish the maximum running length of unshielded twisted-pair wire for 10 Mbps StarLAN soon.

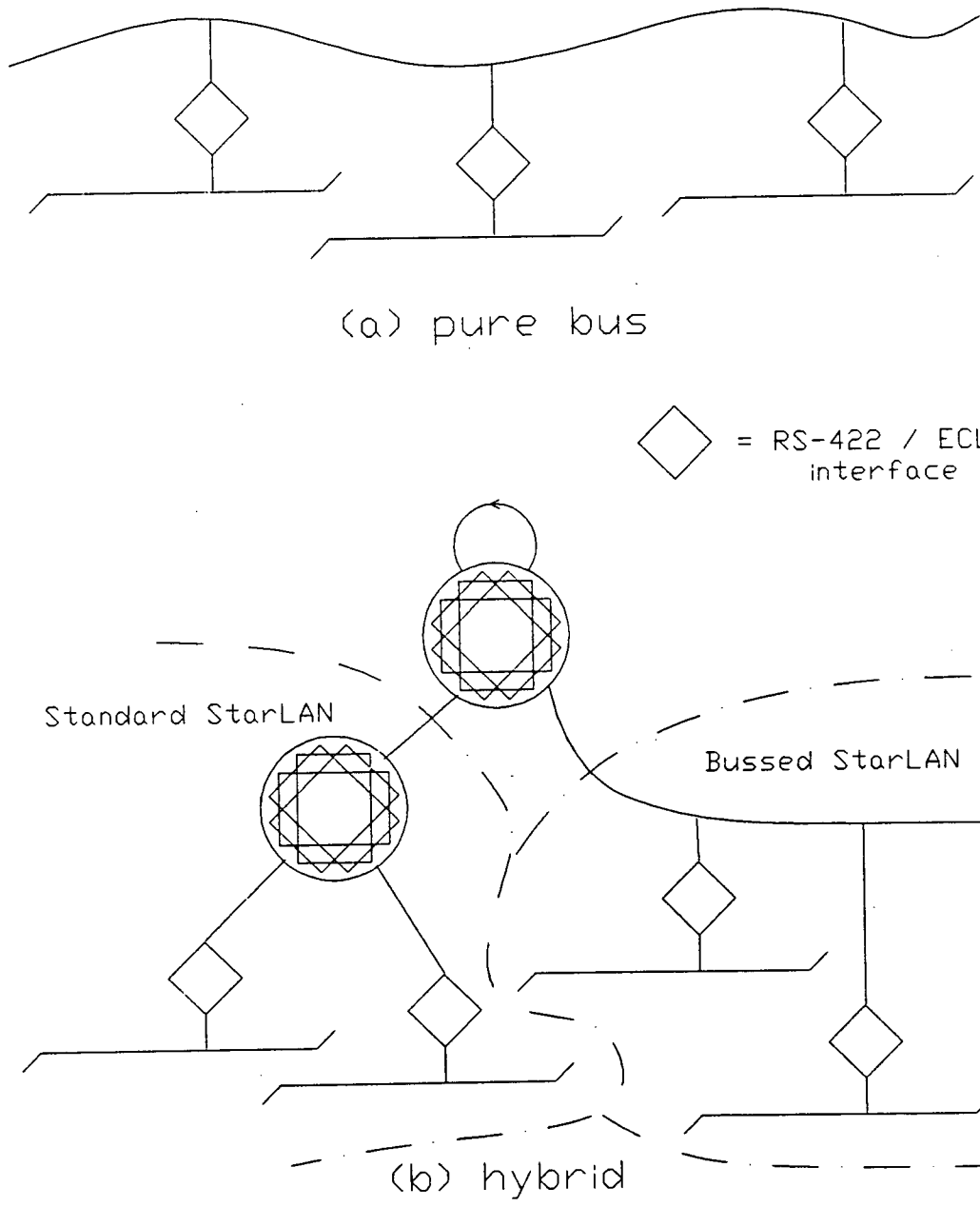


Figure 1.22: Bussed StarLAN in the FASTBUS Serial Network.

Chapter 2

Building the Prototype Serial Network

2.1 A Simple StarLAN Network

At the start of the research (early 1987), StarLAN was still an emerging local area network (LAN) standard. Nobody in the laboratory staff had any working knowledge of StarLAN in both hardware and software. Hence, two StarLAN design kits were procured from Intel to expedite the task of building a working network. These kits contained a total of four printed circuit boards designed to plug into IBM PC expansion slots. The kits also included some components (such as the Intel 82588 LAN controller chip), software for driving the boards, as well as documentation that proved to be helpful for newcomers to StarLAN ([9], [7], [10]). Other parts, including wiring and a StarLAN hub, were obtained separately. A PAL (Programmable Array of Logic gates) chip was also required for each board. This chip's programming instructions were given in the Abel PAL programming language. However, only one PAL 'compiler', called PALASM2 (from Monolithic Memories), was available at the laboratory. Hence, the programming instructions had to be translated to PALASM2 code before the chips could be programmed. A short simulation program was also written to verify the translated code. A listing of the programming code and simulation results is given in appendix A.1.

Experience in electronics hardware development had shown, time and time again, that hardware bugs could be very difficult to solve (much more so than software bugs). Hence, a very cautious approach was taken which, though somewhat tedious, greatly reduced

the chances of unpleasant surprises and frustration when everything was soldered and plugged in. The Intel circuit design was thoroughly checked so that the circuit's operation was understood in detail. The schematic was also cross-checked with the printed circuit layout. All minor board components were tested before being soldered or plugged in. Intermediate hardware tests were done at various stages of assembly. Nothing (well, almost nothing) was left to chance. All of these precautionary steps were also taken during the construction of other electronic circuits that followed. The boards were carefully assembled and then tested with three IBM XT compatible computers using the software supplied with the kits (see [9] for more details). No problems, major or minor, were encountered. A diagram of the network connections for the tests is shown in figure 2.23a. Three sets of twisted-pair telephone cable, each 25 feet long, were used to connect the Intel boards from their telephone jacks to the telephone jacks of a Retix HB-6 StarLAN hub. [1] This hub could connect up to six nodes together. It could also be switched to function as either an intermediate or a head hub.

With this rudimentary StarLAN network, message frames could be sent from one PC to another. The Intel software allowed the user to load a message in hexadecimal or ASCII format and to set the message length. Another useful feature was a 'repeat transmit' mode which was used to continuously send messages. However, the transmission rate could not be altered. StarLAN parameters, such as source and destination addresses, could also be set by the user. A 'transmission session' could be initiated by starting transmissions in repeat-transmit mode and terminated at any time by keyboard entry. Statistics from transmission and reception of frames were updated in real time on the terminal screen. Detailed collision data (i.e., the distribution of the number of retransmission attempts to resolve a collision) could be displayed at the end of a transmission session. To check the system for reliable operation, each PC was set to transmit

[1] see [11] for more details.

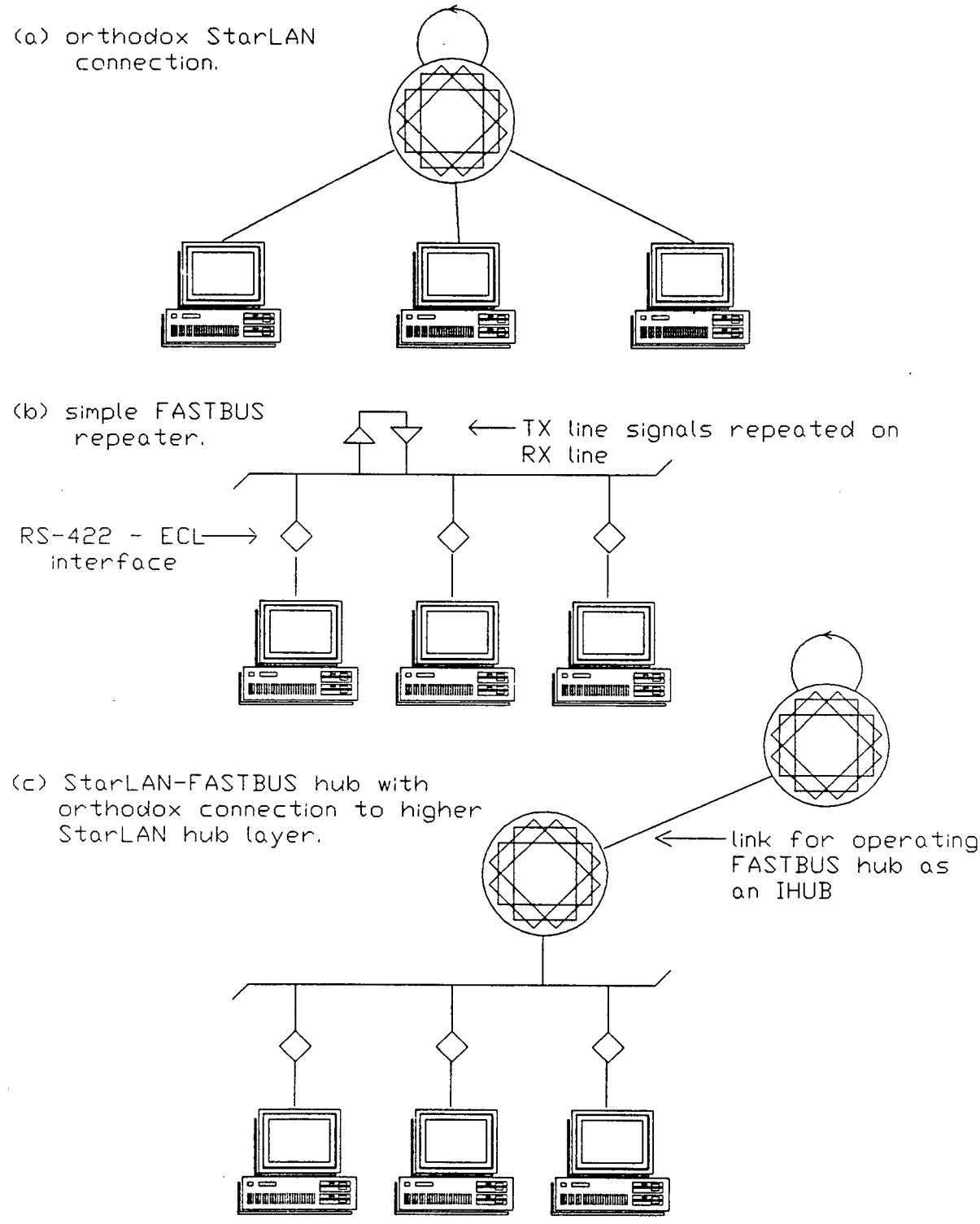


Figure 2.23: StarLAN Network Configurations

to another PC in a circular fashion. Three transmission sessions were done, one each for information field lengths of 48, 512, and 1500 bytes. The 48-byte length corresponds to the shortest field length that can be transmitted without producing a 'short frame' error in the program. Although the shortest field length allowed by StarLAN is 46 bytes, a small bug in the LAN controller chips restricted the shortest length to 48 bytes. The 1500-byte length corresponds to the longest field length for StarLAN frames while a 512-byte length was chosen to be the 'intermediate length' frame. A transmission session was started by programming each PC, one by one, to transmit repeatedly. After some time had elapsed, say 5 minutes, the PC transmissions were stopped in the same sequence as they were started. The statistics displayed on each terminal were first checked for unusually large transmit or receive errors. Only a few errors were observed, so they were assumed to be caused by random processes in the course of the transmission session. Next, the number of 'good' transmissions from each PC was compared to the number of 'good' receptions at the corresponding destination PC. The good receptions were either equal to the good transmissions or were slightly less by only a very small amount. These implied that the number of lost frames were miniscule in comparison to the total number of transmissions that were reported as being 'good'. Hence, it was concluded from the tests that reliable communications were established in the network.

2.2 Using StarLAN in a FASTBUS Environment

After the completion of this rudimentary StarLAN network, an interface system that would enable StarLAN signals to be routed through a FASTBUS backplane was developed after verification tests on the basic StarLAN network were completed. The interface system consisted of two major parts: the first was a StarLAN-FASTBUS transceiver (essentially an RS-422-ECL signal translator; the FASTBUS backplane uses ECL logic

whereas StarLAN uses RS-422 voltage levels) and the second was a StarLAN hub designed to work on the FASTBUS backplane.

A Transceiver Between StarLAN and FASTBUS

A schematic diagram of the StarLAN-FASTBUS transceiver is shown in figure 2.24. This circuit is a transceiver between standard StarLAN and the FASTBUS backplane that provides a transparent interface between three-state RS-422 StarLAN signals and two-state ECL voltage levels. Figures 2.23b and 2.23c shows how The transceiver is used to connect the Intel StarLAN boards (constructed earlier) to a FASTBUS backplane. Standard StarLAN signals are fed into the transceiver, converted to ECL voltage levels, and sent through a line in the backplane. Signals received from the corresponding backplane line are converted back to StarLAN-specification RS-422 signals and relayed to the StarLAN board.

A not-too-small part of the transceiver's design was borrowed from a section of the Intel StarLAN boards. The task of incorporating some of Intel's design was not too difficult after some reverse-engineering was done on the board's schematic diagram. A TTL-to-ECL translator chip was used to drive the backplane while an ECL-to-TTL translator received the backplane signals. Special ECL bus driver and receiver chips were not used because other FASTBUS circuits developed in the laboratory were known to work without the use of these chips. A description of how the transceiver works is given in appendix A.2.

The transceiver is more than just a voltage level translator. ECL logic has two states: high (1) and low (0), and the low logic level is used for the quiescent state when nothing is being transmitted. On the other hand, StarLAN RS-422 signalling for twisted-pair telephone cable has three states: differential high (1) (when the voltage difference between the reference wire to its pair is positive), differential low (0) (when the difference

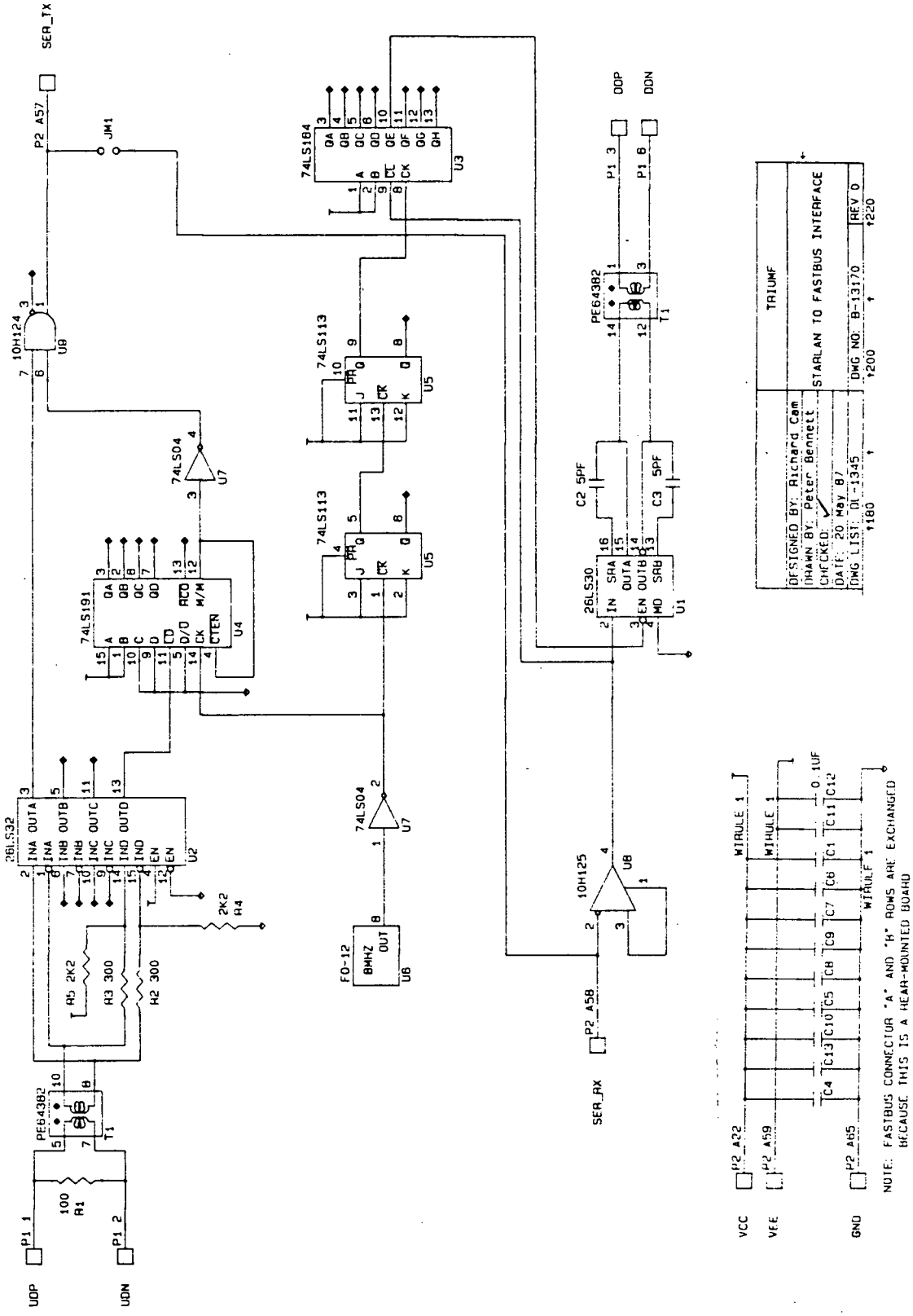


Figure 2.24: Schematic diagram of the StarLAN-FASTBUS tranceiver.

is negative), and inactive (when the line is released to its quiescent state by shutting off the RS-422 driver's output). StarLAN transmissions begin with a differential low from the inactive state and voltage swings occur as defined by the Manchester-encoded bit stream, whose end is marked by a differential high lasting approximately two microseconds long, after which the line is released back to its inactive state. Hence, it takes more than just a simple voltage translation to map the three StarLAN RS-422 signal states to the corresponding ECL logic levels and vice versa.

Four boards were made, one for each Intel StarLAN board. A simple backplane repeater was constructed as well. This repeater would simply relay the logic levels at the transmit line (B57) to the receive line (B58). Thus, the backplane could be tested as a bus topology network without a hub. The repeater consisted of an ECL-to-TTL translator which was cascaded to a TTL-to-ECL translator. The input of the first chip was connected to the transmit line while the output of the second chip was connected to the receive line. Those particular chips were chosen simply because they happened to be immediately available (spare chips for the transceiver boards).

The connection diagram for this simple bussed network is shown in figure 2.23b. Three XT compatible microcomputers were used, as was done before when the StarLAN boards themselves were tested. The transceiver boards and repeater were plugged into the rear section of an unused FASTBUS crate. Three sets of 25-foot telephone cable were used to connect the telephone jacks of the transceivers to similar jacks on the StarLAN boards. Again, the tests of the simple StarLAN network were repeated in this network. No anomalies were observed and, as with the previous network, it was concluded that reliable communications were achieved.

A StarLAN Hub for the FASTBUS Backplane

The development of the StarLAN-FASTBUS hub (or 'crate hub') was complicated because it was not at all certain that one could obtain, in a reasonable time, an integrated circuit which would implement the functions of a StarLAN hub. This chip, the Western Digital WD83C510 [2], was just about to be introduced to the market at that time (May-June 1987). It was not possible to obtain this chip from local sources or even directly from Western Digital. No other semiconductor company was known to be closer to mass-production of a StarLAN hub IC than Western Digital. An attempt was made to obtain this chip indirectly through an associate at the University of Illinois Center for Supercomputing. Nevertheless, the possibility of obtaining even one sample chip was not assured. So, in the mean time, a crate hub that did not use the WD83C510 was designed.

In general, standard StarLAN hubs cannot be coupled directly to a bussed line. This is because a collision in StarLAN is defined as the condition where more than one node is transmitting at the same time. Since collision detection is a function of the hub and only one node is assumed to be connected to each 'downstream' input channel, a hub would only have to watch out for simultaneous transmissions from any of its 'downstream' channels in order to detect collisions. In the case of a FASTBUS backplane, the transmit line can be connected to several nodes. If the signals on this line were simply connected to a standard StarLAN hub by, say, the transceiver circuit described previously, the StarLAN hub would 'see' only one channel and erroneously assume that only one node was connected to that channel even though that would not generally be the case. It is possible that a collision involving any of the nodes in a FASTBUS backplane could slip through without being detected by the hub. That of course depends on the particular

[2] details given in [12]

hub's design. The only way to 'rigorously' catch these bus collisions as well as any anomalous signals in the backplane is to detect them by a separate circuit that watches for missing mid-cell transitions and pulses of improper width. The output of such a circuit could then be connected to another hub input. It would transmit an arbitrary StarLAN signal to that input upon detecting a collision. The hub would then sense at least two inputs transmitting simultaneously (one from the backplane line, the other from the auxilliary collision detection circuit) and flag the collision. This scheme would require two hub inputs per FASTBUS crate, a somewhat clumsy, but nevertheless feasible idea. Later on, it became clear that a rigorous collision detection function was a major design component of a crate hub anyway and the decision was made to design a complete self-contained crate hub for the FASTBUS backplane rather than implement the separate collision detector – external hub scheme as described above. The effort required to go all the way and design a crate hub was justified by the hub's elegance. The crate hub would be connected to all the nodes of the crate in a bus topology and itself would connect with a higher-layer hub as part of a star topology network.

The first version of the hub (that did not use a StarLAN hub chip) was a fully synchronous digital circuit whose block diagram is shown in figures 2.25, 2.26, and 2.27. An initial attempt was made to design the entire circuit heuristically, but it soon proved to be not only an awkward but ultimately, an impossible method given the complexity involved. So, the circuit was designed with a combination of formal and heuristic digital design techniques. One technique would be used when the other seemed to be more complicated. This circuit never got past the design phase because a prototype sample of the WD83C510 arrived, thanks to the efforts of Bob Downing from the University of Illinois. The hub controller chip probably saved at least a month of work to get a functioning hub, even though this meant starting over to design a new circuit based on the WD83C510. Figure 2.28 shows a schematic diagram of the resulting circuit, showing

a dramatic simplification of this implementation over the design that did not use the a hub chip. This was due, in part, to the design of the WD83C510 itself, which 'rigorously' detected collisions in each of its inputs, eliminating the need for a robust external collision detector (the 83C510 will flag a collision if an input receives pulses of improper widths or with missing mid-cell transitions, as well as if more than one input is receiving data). Another chip used in the circuit that is worth mentioning here is the SN75061 from Texas Instruments. This chip is a TTL–RS-422 transceiver designed especially for StarLAN applications. It provides the functions that would have previously required two separate chips (e.g., Am26LS30 and Am26LS32, which were used by the transceiver circuit when the TI chip was not yet obtained). Appendix A.3 explains how the hub circuit works. The circuit was built (very carefully, since only one WD83C510 was available and it was unlikely that another one could be quickly obtained) and tested. Figure 2.23c shows a diagram of the system used to test the circuit. As was done during previous tests, three XT compatibles with Intel StarLAN boards were used. Twenty-five–foot lengths of telephone cable coupled the StarLAN boards to their corresponding StarLAN-FASTBUS tranceivers, which were plugged at the back of a FASTBUS crate. The hub circuit was also plugged into the back of the FASTBUS crate to connect its 'downstream' channel to the transceivers. Its 'upstream' channel was connected to the Retix HB-6 StarLAN hub. This system allowed one to test the hub circuit for proper operation either as an intermediate or a header hub (by leaving the 'upstream' link connected or disconnecting it respectively). It also allowed the hub-detection logic of the WD83C510 to be tested.

Briefly, the hub-detection logic is used by the WD83C510 to sense if it is connected to a hub at a higher level in a StarLAN network. The hub-detection logic works as follows. Suppose the chip is functioning as an intermediate hub. If, after a short waiting period, no signal is received from its upstream channel, it assumes that it is not connected to a higher hub level and automatically proceeds to function as a header hub (if there was a higher

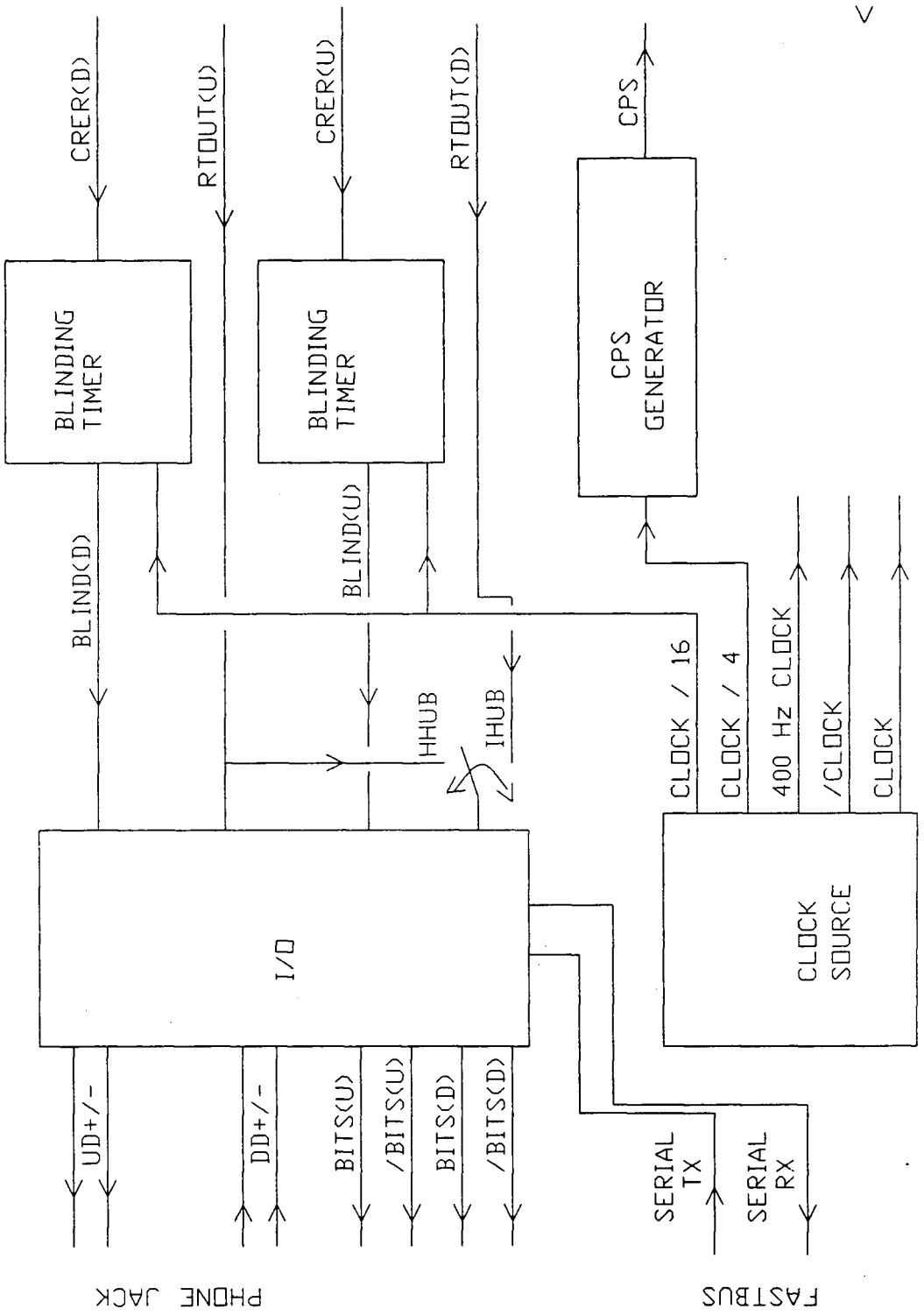


Figure 2.25: Block Diagram of Hub Circuit: I/O Section

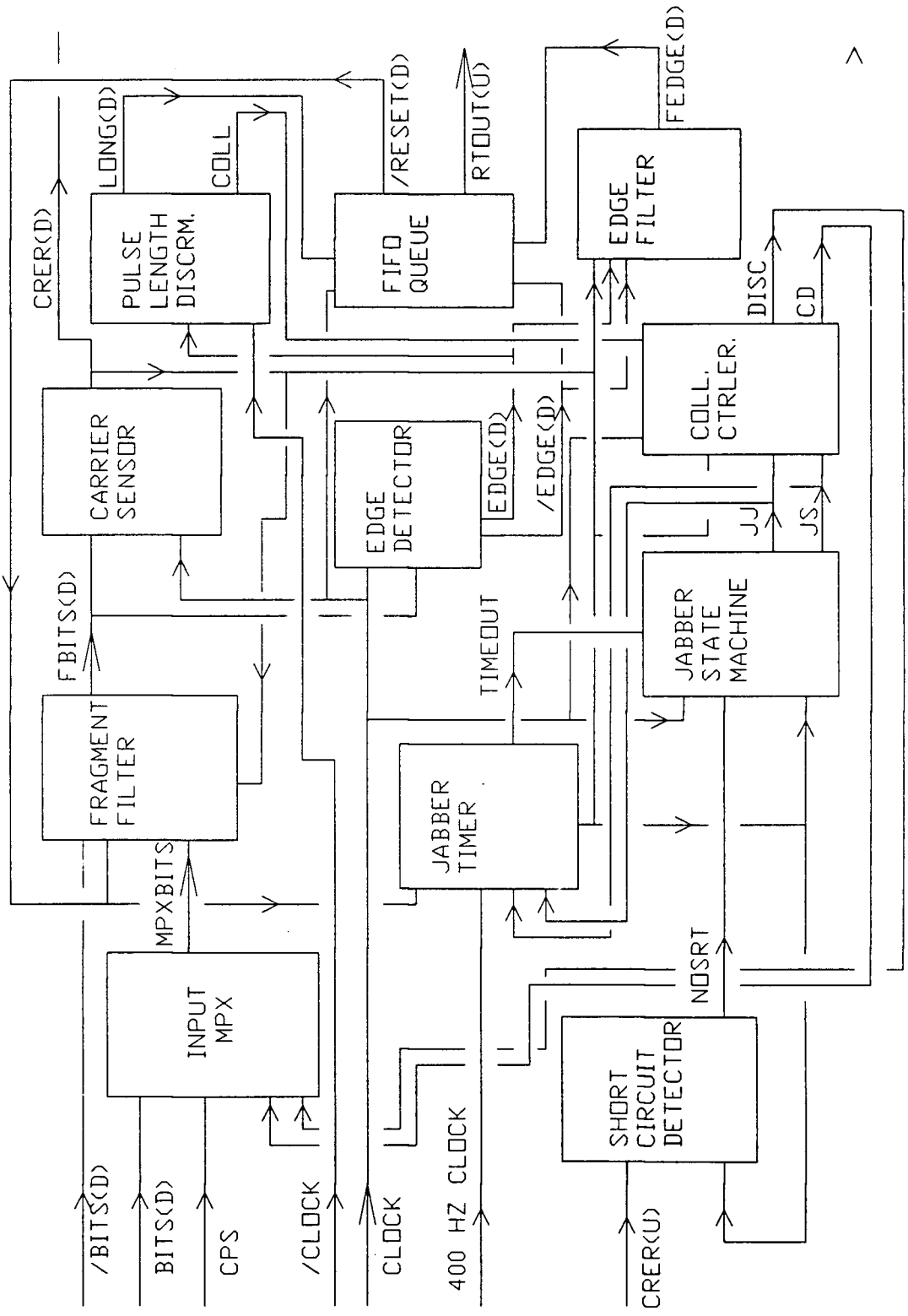


Figure 2.26: Block Diagram of Hub Circuit: Downstream Section

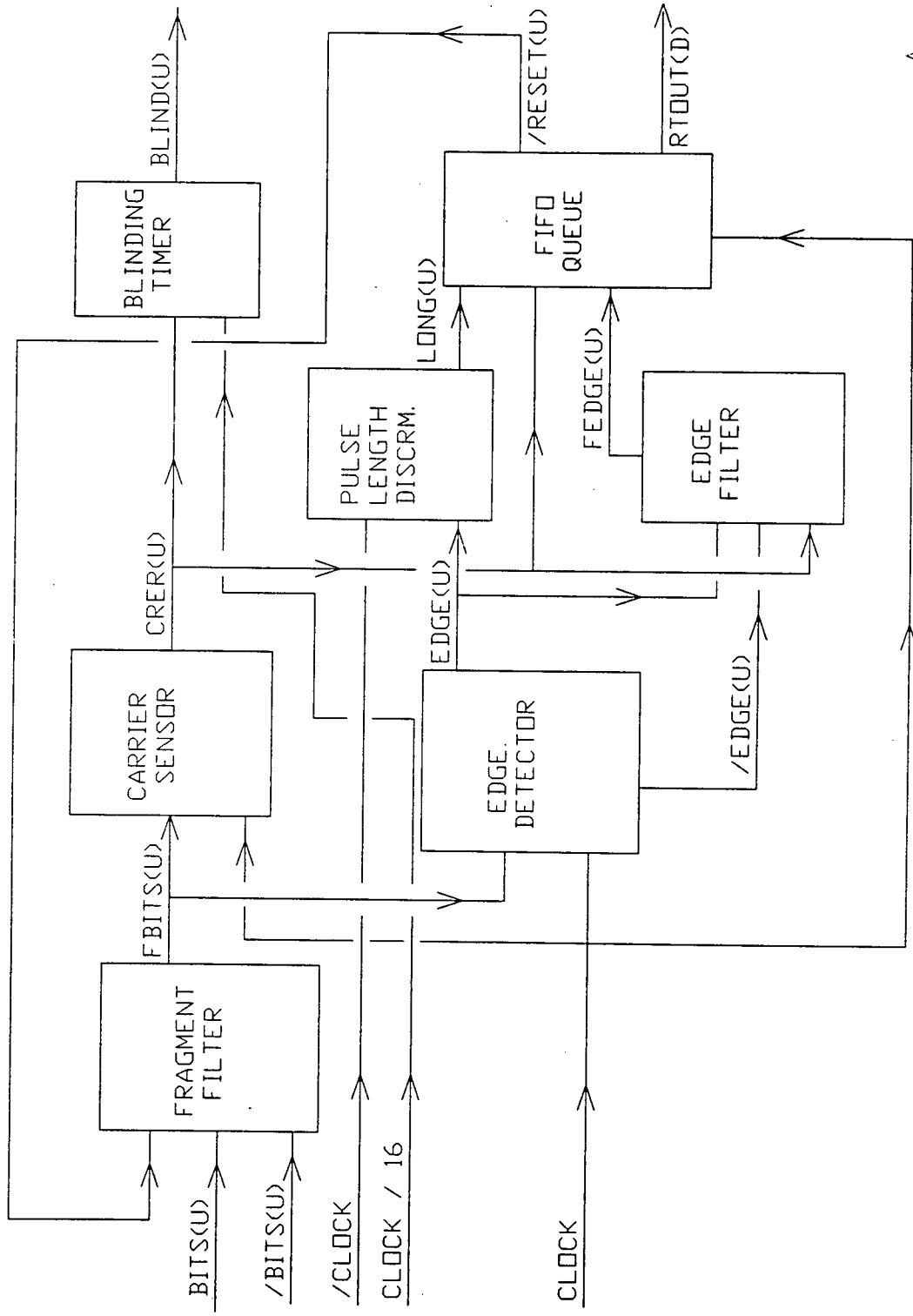


Figure 2.27: Block Diagram of Hub Circuit: Upstream Section

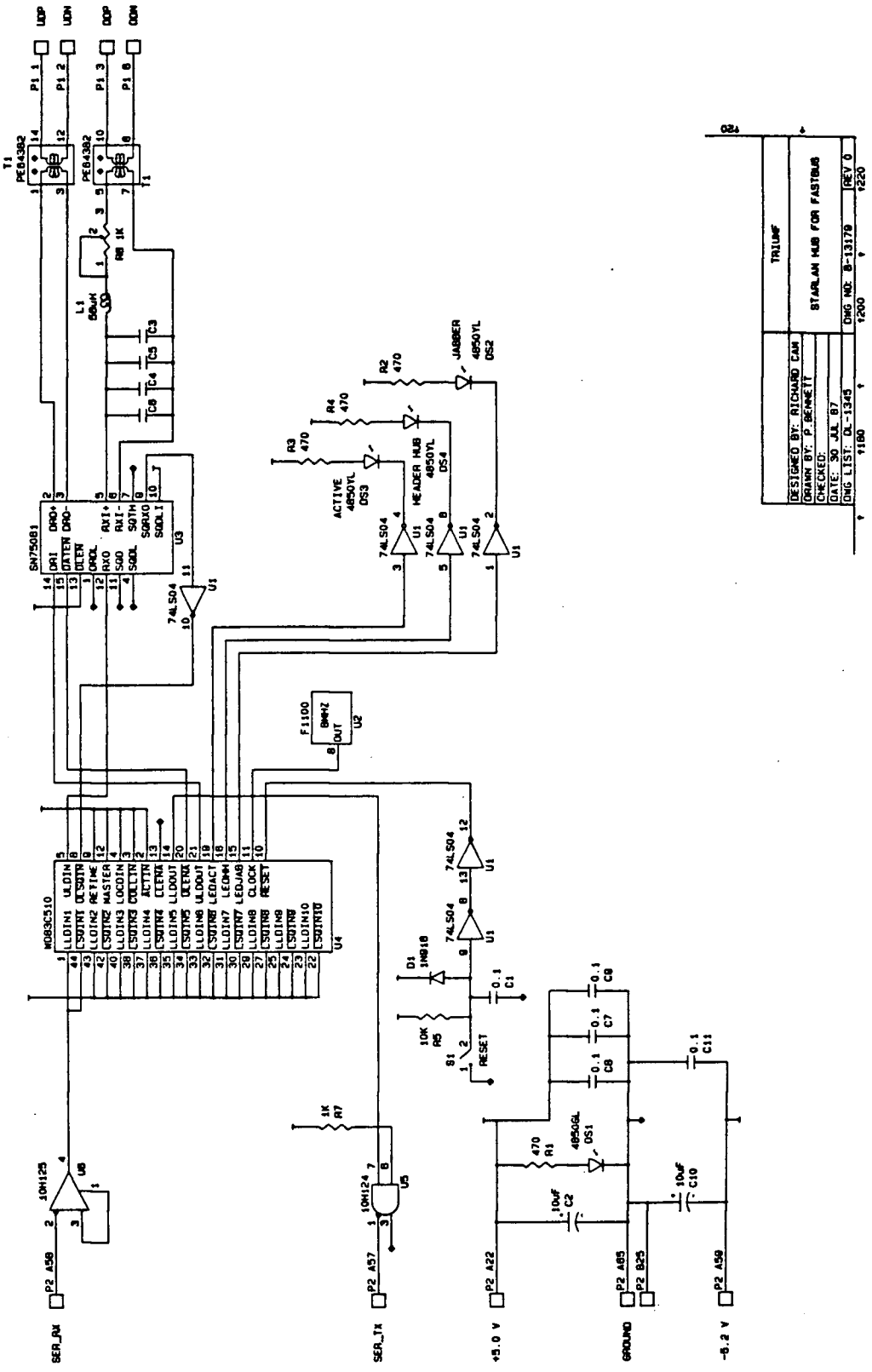


Figure 2.28: Schematic Diagram of Hub Circuit with WD83C510 Controller

hub level, the higher hub would have either repeated the lower hub's transmission or, in the event of a collision, sent a collision presence signal (CPS)). When the chip operates as a header hub, all downstream signals are immediately looped back. Now suppose that signals are received on the upstream channel of the hub chip in header hub mode. The hub-detection logic assumes that these signals must have come from a higher-level hub and the hub chip switches its operation from a header hub to an intermediate hub. A frame will be lost if it is transmitted during an intermediate-to-header hub transition because the hub chip will loop back only the remainder of the frame downstream when it switches to a header hub. This usually results in a CRC error. Two or more frames may be involved in a collision during a header-to-intermediate hub transition if a frame is received on the upstream channel while the hub is looping back a downstream frame.

As before, the Intel program was used to drive the system and collect performance statistics. Fortunately, everything worked perfectly right away without any problems. The on-chip hub-detection logic was tested by alternately connecting and disconnecting the link to the Retix StarLAN hub. The hub chip was observed to switch properly without causing a major disruption. The tests showed that data could be transferred reliably when the circuit was operating in either intermediate or header-hub mode. For the first time, data communications were successfully established in a prototype of the envisioned two-line StarLAN-based serial network.

Chapter 3

Measurements

3.1 Electrical Characteristics of StarLAN Signals in the FASTBUS Backplane

StarLAN describes a protocol for data communications at the media access control (MAC) sublayer and the physical layer. The MAC sublayer specifications are fixed but the physical layer has some flexibility. In particular, StarLAN does not specify the medium of propagation for data transmission between a node and a hub (or between two hubs). It only requires the propagation delay of a node-hub/hub-hub link to be no greater than 4 bit times and that the signal (pulse) jitter should not exceed 62.5 ns per link (pulse jitter is defined as the amount of time by which a pulse edge deviates from when it is expected to occur). Furthermore, up to five hub levels are allowed, with no restriction on the number of nodes per hub level.

In an ordinary StarLAN network, RS-422 signals are sent through unused intra-building twisted-pair telephone lines in a star topology. On the other hand, ECL signals must be used to transmit data on the serial lines in the FASTBUS backplane, which connect the nodes within the crate in a bus topology. In departing from standard StarLAN, it is important to stay within any specified timing parameters at the physical layer so that reliable performance will not be compromised. To this end, measurements were taken to compare parameters and characteristics of the StarLAN-based serial network to those required in an orthodox StarLAN network. Measurements of propagation delay and

pulse jitter were made for different capacitive loading configurations in the backplane.

ECL signals have very small voltage swings (of the order of 1V difference between logic states) and short transition times (typically, 1ns). Because of these characteristics, they are used in high-speed applications but are more difficult to work with than slower logic signals such as those from CMOS or TTL. In general, electrical connections for ECL signals have to be treated as transmission lines because the propagation delays of such connections are often much longer than ECL transition times. To keep this thesis from getting any longer than it already is, a detailed discussion of ECL characteristics and transmission lines will not be given here. Instead, the reader is referred to journal articles and books listed in the bibliography ([13], [14], [15], [16], [17]). Nevertheless, in order to relate the effect of capacitive load in the backplane to electrical parameters such as propagation delay and pulse jitter, it is necessary to introduce some of the results from transmission line theory.

Two transmission line parameters of interest are the propagation delay and the characteristic impedance of the line. For an unloaded line, the propagation delay, t_{pd}, and the characteristic impedance, Z_o, are constants at high frequencies (relevant to fast signals like ECL, which have significant high-frequency components) and are given as follows.

$$t_{pd} = \sqrt{LC} \quad (sec./m.)$$

$$Z_o = \sqrt{L/C} \quad (ohms)$$

L and C are the inductance and capacitance per unit length respectively. For a loaded backplane bus, the corresponding parameters, t_{pdL} and Z_{oL}, are given by the following relations.

$$t_{pdL} = t_{pd}\sqrt{1 + C_L/C} \quad (sec./m.)$$

$$Z_{oL} = \frac{Z_o}{\sqrt{1 + C_L/C}} \quad (ohms)$$

C_L is the additional load capacitance (from attached devices) per unit length. It can be seen from the equations above that the effect of added capacitance is to increase the propagation delay and to reduce the characteristic impedance of the line. Since the characteristic impedance changes with the backplane load, it is not possible to terminate a backplane line with a perfect impedance match for all loading conditions. When an impedance mismatch exists at any point in the line, the incident signal is reflected back, the extent of which depends on the amount of mismatch. Small amounts of reflections can influence the characteristics of signal transitions and hence, contribute to pulse jitter. If reflections become large enough, it may even be impossible to transmit reliably at all. It is because of the reasons given above that one has to consider the effect of loading conditions on the characteristics of a backplane.

Capacitors were used to approximate the loading effects of ECL devices connected to the backplane lines. The maximum signal line capacitive load for a FASTBUS device is 12 pF. Since FASTBUS cards can be plugged into both the front and back of a FASTBUS crate, the maximum load per slot position is 24 pF. As a first attempt, the backplane's serial lines were loaded with 33 pF capacitors per slot position to simulate an overloaded backplane. A diagram of one capacitive load is shown in figure 3.29 (which shows a 15 pF capacitor, the load that was used later on). Twenty-six of these capacitor loads were made, one for each slot position in the FASTBUS crate. The leads of the capacitors were terminated with gold-coated AMP 87000 series contacts which were crimped on by an AMP crimp tool (model number 90340-1).

Two termination cards were built, one for each end of the backplane. Each card consisted of a FASTBUS card edge connector and a 20-turn trimmer resistor, both of which were mounted on a piece of phenolic plugboard. The trimmer resistors were adjusted to 56 ± 0.1 ohms.

A PC was connected to a transceiver and was set to transmit continuously to generate

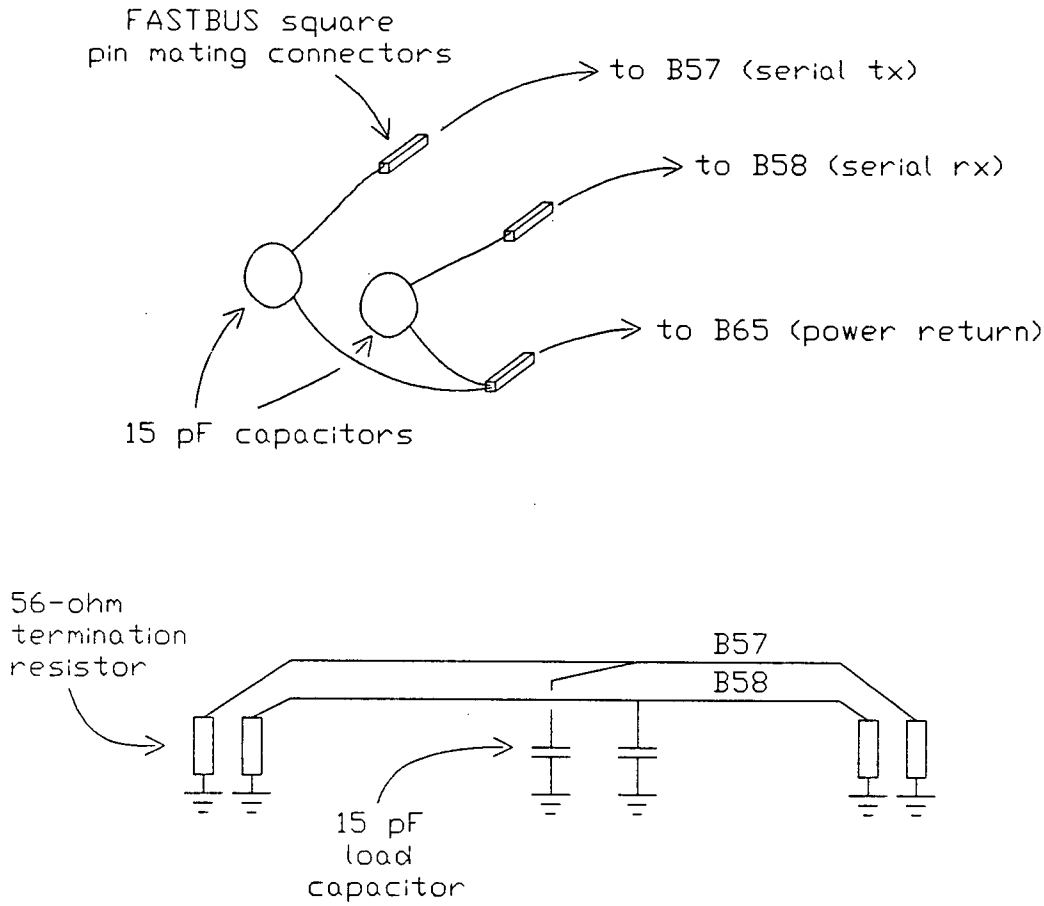


Figure 3.29: Sample capacitor load for the FASTBUS backplane.

StarLAN signals in the backplane. Waveforms of the signals were observed by attaching an oscilloscope probe to a short stub that could be inserted into slot pins at the backplane. The oscilloscope system was based on a Tektronix 7904 mainframe which used a 7B92A dual time base and two 7A18 amplifiers, which had a bandwidth of 75 MHz. Standard 10 Mohm 13pF oscilloscope probes were used. There was some concern as to whether or not the system would be capable of tracking the fast ECL signal swings. However, the oscilloscope was observed to adequately track 1V/ns square waves (simulating ECL pulses) from a signal generator. Hence, the existing setup was used. Observations showed that there was an enormous amount of reflection when ECL signals were sent through the serial line. These reflections were causing oscillations in the ECL signals, resulting in spurious state transitions. A less 'stressful', but more realistic capacitive load was then evaluated instead.

A new set of 26 15pF metal film capacitor loads were made (see figure 3.29). Each capacitor was checked to be within $\pm 10\%$ of 15 pF with a capacitance meter. Two-line capacitor loads were made although only one line had to be loaded for the measurements because these loads would be required for communication tests later on. A board with an ECL-to-TTL translator was also built to convert ECL signals at the receiver end to TTL. Signals were measured at the input to the TTL-ECL translator at the transmitter (signal source) and at the ECL-TTL translator output at the receiver. These measurement points were chosen because they were buffered from the backplane and, as a result, showed cleaner transitions, leading to more consistent measurements. Figure 3.30 shows the reference transition points for the signal source and the receiver.

Clearly, it was impractical to consider all possible backplane loading configurations. Hence, only a 'canonical' subset was selected for the measurements of propagation delay and pulse jitter. Diagrams of these test configurations are shown in figures 3.31 and 3.32. Capacitor loads were placed in each slot position at the loaded sections. Measurements

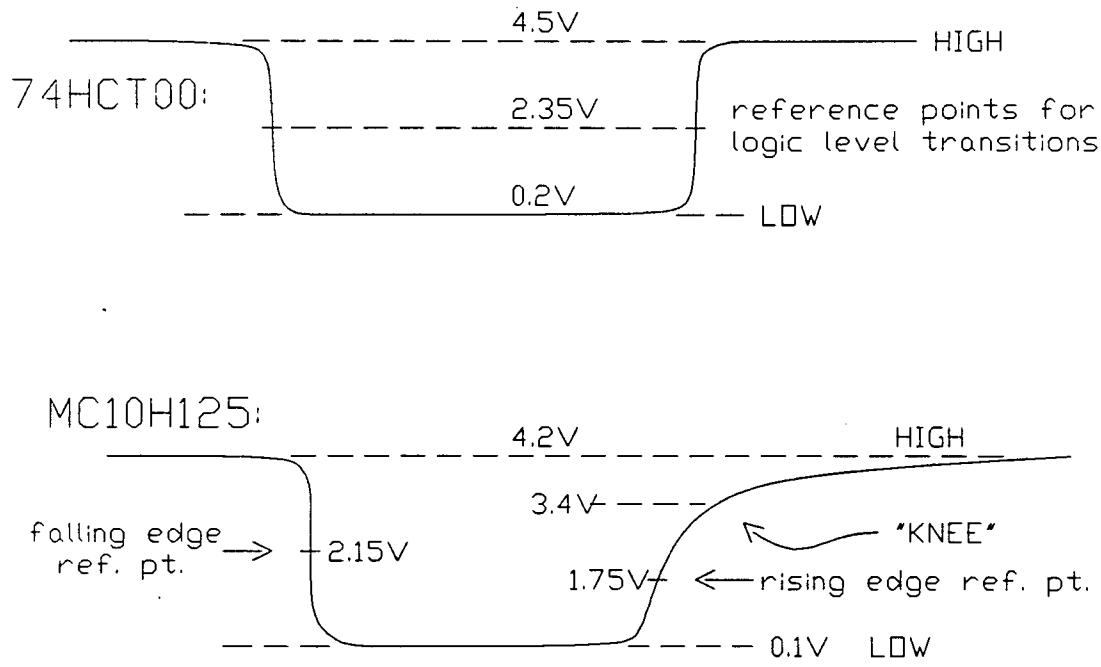


Figure 3.30: Reference points for edge transitions.

of pulse jitter and propagation delay were made for each configuration and are tabulated in appendix B. The amount of pulse jitter was obtained by taking the absolute value of the difference in propagation delay between the rising and falling edge of the source pulse to the corresponding pulse edges that were triggered at the receiver. Photographs of oscilloscope traces were taken. They are also shown in appendix B. Each photograph shows three traces, one on the top half, and two on the bottom. The top trace shows the ECL signal as seen at the receiver. The bottom traces show two TTL edge transitions. The first edge corresponds to the source at the input to the TTL-ECL translator. The second edge that occurs afterwards is from the receiver at the output of the ECL-TTL translator.

Most of the traces appeared to be well-behaved, as a sample in figure 3.33 shows. However, there was one exception (as shown in figure 3.34) where there is a bumpy transition in the output of the ECL-TTL translator (bottom trace). This is a little strange because the ECL signal on the top trace is not oscillating and thus, should not trigger the glitch at the translator output. At any rate, an LS TTL logic gate was connected to the translator output in that configuration and was not observed to be affected by the glitch. The largest amount of pulse jitter was measured to be 6 ns and this occurred in the fully loaded backplane when the source and receiver were both near the midpoint of the line (reference number 18). The longest propagation delay was measured to be 16.2 nsec. and this occurred, as expected, when the source and receiver were placed at opposite ends of a fully loaded backplane (reference number 21). These measured values for the worst-case pulse jitter and propagation delay are well within the StarLAN limits of 62.5 ns and 4 microseconds respectively.

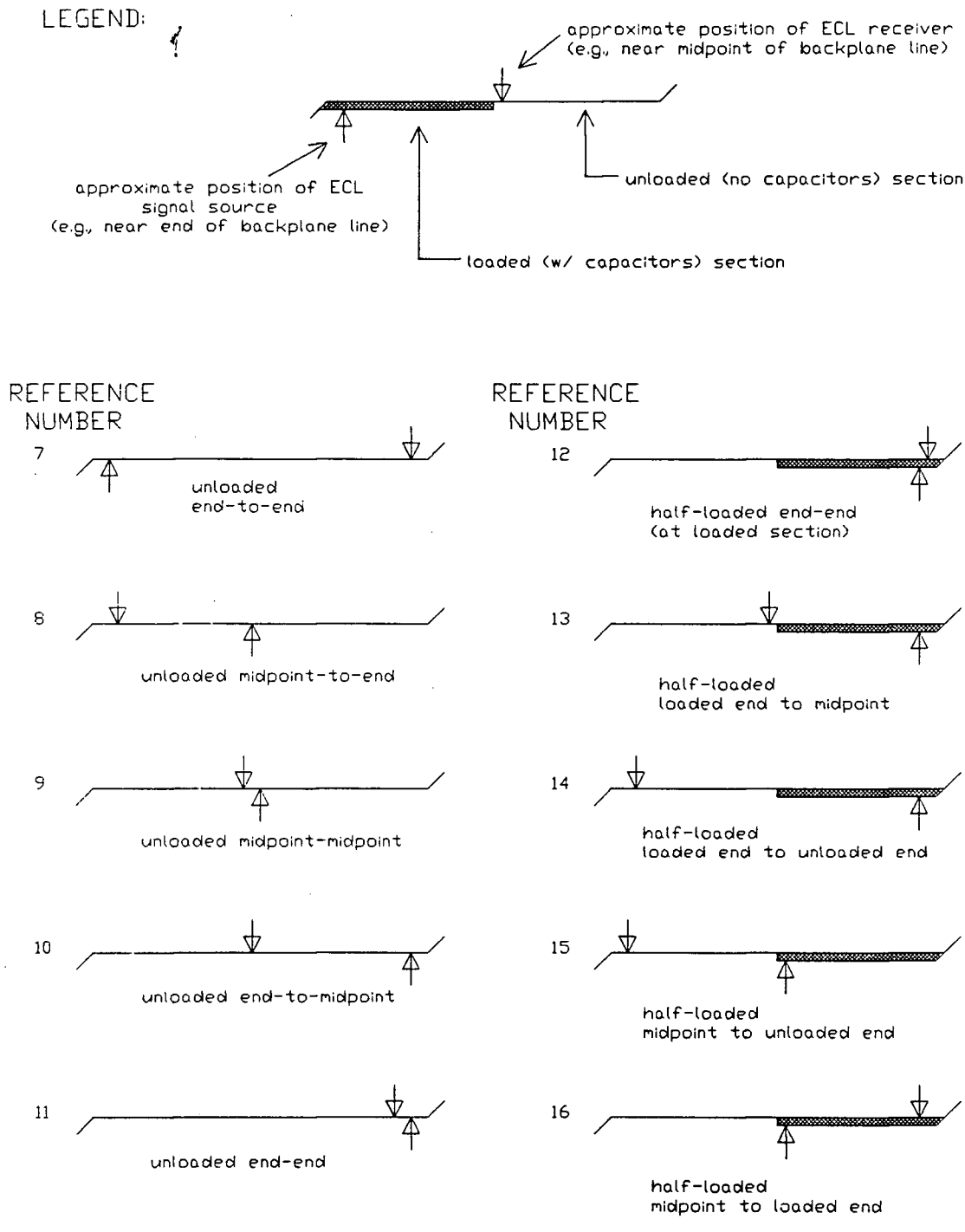


Figure 3.31: Schematic diagram of electrical measurements.

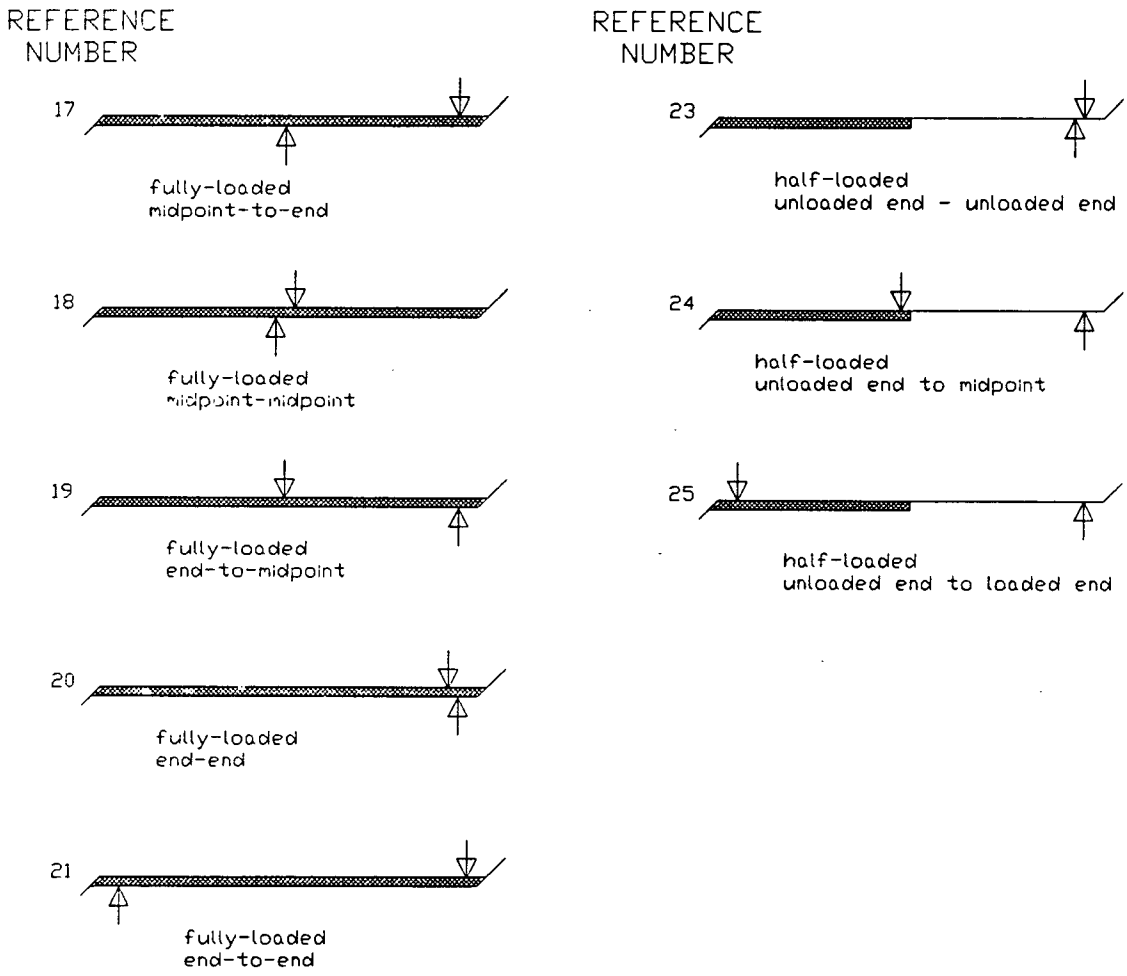


Figure 3.32: Schematic diagram of electrical measurements (continued).

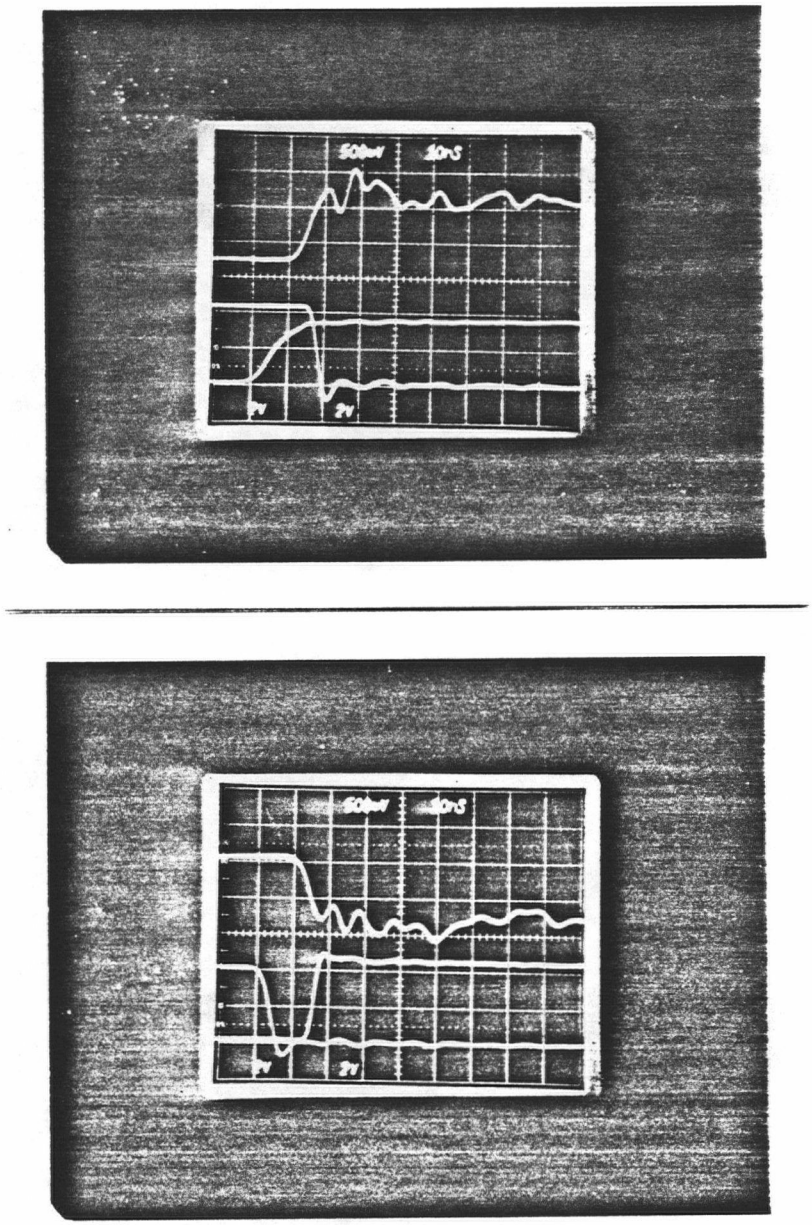


Figure 3.33: Typical oscilloscope traces (ref. 25).

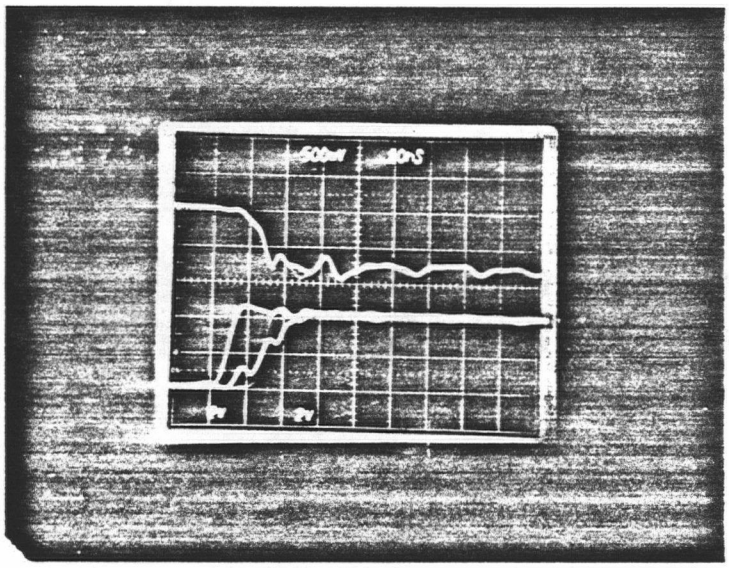


Figure 3.34: Oscilloscope trace of bumpy edge transition (ref. 17).

3.2 Network Performance Measurements

The fully loaded prototype serial network was expected to behave just like an orthodox StarLAN network after the pulse jitter and propagation delay were found to be very small. Nevertheless, there was still a lingering trace of doubt in that expectation because the nodes in the crate were connected in a bus topology whereas standard StarLAN nodes are connected in a star topology. Perhaps a difference in the connection topologies could lead to different network behaviour. Hence, tests were made to compare the performance characteristics of the serial network to that of standard StarLAN.

The equipment setups for the performance measurements were identical to those used when the serial network was built and tested. This time, however, the backplane was fully loaded with 15 pF capacitive loads in every slot position. The four configurations for these measurements are shown in figure 2.23. As was the case with the earlier tests,

the Intel program was used in all three computers to drive the network. A better program than the one supplied by Intel would have been more helpful since the frame transmission rate could not be controlled through the program and all of the data had to be manually copied from the terminal screen from each computer. To make matters more difficult, the numbers in the screen were shown in hexadecimal format which had to be converted to decimal base later. The program was used anyway to obtain some preliminary results.

As before, three different information field lengths were used: a 'short' length 48 bytes long, a 'medium' length of 512 bytes, and a 'long' 1500-byte length. The PCs were programmed to transmit to an address that was different from any of their source addresses. This would allow the PCs to transmit as fast as they could without receiving frames. Statistics from the data transmissions were recorded at the end of each transmission session (using the software in 'repeat transmissions' mode). The software was an interactive program that responded to input from the keyboard. Hence, it was not possible to 'trigger' all three computers to start and stop simultaneously with the Intel program. Instead, the following procedure was adopted. One computer was selected to start transmissions first. The second computer would start its transmissions 10 seconds after the first one and the third 10 seconds after the second. The transmissions were then stopped five minutes after the first computer was started, beginning with the first, followed by the second and the third computers 10 seconds apart. The duration of the transmission sessions was chosen to be long enough such that the 20-second starting and ending sequences would not comprise a considerable portion of the transmission session.

The results of these transmissions are tabulated in appendix C. It can be seen that the data from all four configurations do not differ very much. They all tend to behave similarly for the frame lengths chosen. The differences in the data between different configurations are due in part to statistical fluctuations and differences in overall propagation delay and hub characteristics. From the data, one can conclude that the serial

network behaves just like a standard StarLAN network. In other words, the subnetwork of nodes in a FASTBUS crate 'look' like star-topology nodes to hubs at higher levels in the network.

While the performance data showed encouraging results, there was a peculiar pattern in the distribution of the retransmission attempts. The distribution of the number of retransmissions in a collision resolution interval should taper off to zero as the number of retransmissions required to resolve an initial collision increases. Such a distribution is expected because the range of the random backoff periods is doubled for each of the first ten unsuccessful retransmission attempts. This means that the probability of two or more stations choosing the same backoff period becomes smaller with more retransmission attempts. However, the data showed a gap in the distribution from about nine retransmissions to the maximum of 15. It seemed that if a collision was not resolved within a short number of retransmissions, then it would not be resolved at all. This peculiarity could be due to the pseudorandom nature of the way the backoff period is computed by the LAN controllers. Perhaps, two or more LAN controllers may happen to compute identical backoff periods in some collision intervals. One question that was not addressed by these measurements was the amount of variability in the data if the transmission session was repeated over and over again. A few transmission sessions were repeated and the resulting data were found to change only slightly. Nevertheless, the general statistical variability of the data was not yet known. The performance data would be more meaningful if some measure of their variability could be given as well. Hence, an extension of the preliminary performance measurements was planned. This new set of measurements would span the entire range of frame lengths allowed by StarLAN in small increments. The measurements for each frame length would be repeated so that the variance in the data could be estimated.

The deficiencies of the Intel program (as a tool for data collection) gradually became

more obvious as more performance measurements were made. Each transmission session involved the monotonous task of manually starting and stopping the transmissions, followed by copying the data off the screen and converting them from hexadecimal to decimal base. Clearly, it would be impractical to use the Intel program to collect data from a large number of transmission sessions, as would be the case for the new set of measurements. The initial test transmissions and performance measurements had shown that data could be transmitted reliably over both the orthodox StarLAN and the serial networks. It would seem ironic if further performance measurements did not involve an automated scheme which would use the network itself to collect data from all the PCs.

So, an automated network driver program was conceived to replace the Intel program for the next set of performance measurements. There would be two versions of the driver program, a 'master' program which would control the 'slave' programs. The two versions would work as follows. The network master program sends initialization data through the network to each slave prior to a transmission session (the master also initializes itself). The initialization data sets the length of the frame, its destination address, the duration of the transmission session, and a delay parameter to control the transmission rate. A 'start transmitting' signal is then broadcasted by the master to begin the transmission session. Each slave pauses for a short time before starting so that the master can prepare itself. During the transmission session, data are collected by the master and every slave. Slaves do not respond to commands that the master may give until after the end of the transmission session. All transmissions cease after the elapsed time has exceeded the programmed duration. However, frames will continue to be received for a short time afterwards to allow for a margin of error in the local timing of each PC. After a pause at the end of the session, the master prompts each slave, one by one, for the data collected during the session. Each slave responds to the prompt by sending the data over to the master via the network. The master then stores the data in a disk file, which completes

the data collection for a transmission session. By programming the master to set up transmission sessions in sequence, the task of obtaining performance data can be left to the networked computers, thus saving a poor graduate student from what would be repetitive, boring, and tedious (as well as humbling!) work.

The first attempt to write the driver program was a dismal failure. It was written in Pascal and was compiled with the Microsoft Pascal (version 3.2(2)) compiler running under DOS 3.2. Pascal was chosen because the author had a lot of experience with writing programs in this language. The Microsoft compiler was selected because it had been used before without any problem (although none of the earlier programs compiled under it were anywhere as long as the driver program). These choices for the language and compiler thus seemed sensible initially as program development could proceed without delays from having to learn another language (such as C) or to use another compiler. So, the program was expected to be completed quickly without too much trouble. However, events that were to come later on soon extinguished all traces of the initial optimism. The basic problem with the program was that it would crash and render the operating system inoperable when an interrupt from the Intel board occurred. The exact cause of the problem has never been determined. Of course, the possibility of a subtle bug in the source code cannot be ruled out but it is also possible that the compiler was not generating the correct code for handling interrupts. The compiler had either been stalling or generating spurious compiler-failure messages from time to time when the driver program was compiled. This annoying problem was eventually fixed by *reducing* the compiler's stack size, a solution that came only by chance after attempts to solve the problem by increasing the stack size were unsuccessful. None of the compiler problems occurred when short programs were compiled. The problems seemed to occur only if the compiler was 'stressed' by long programs (the driver program was about 1500 lines long). At any rate, the compiler's behaviour only fueled doubts of its overall reliability. Even

if compilation was successful, the executable code could not be 'trusted' to have been generated correctly. This lack of trust made the task of debugging the program very difficult. One could not determine which part of the program's problems were due to the compiler (if any) and which were caused by programming errors. The interrupt nature of the problem did not make debugging any easier because of its random nature. The exact behaviour of any program crash could not be repeated. After about three months of fruitless work, the effort was abandoned and the decision was made to switch to a different programming language (C) and compiler (Borland Turbo C v1.5).

Rather than translate the Pascal program into C (and possibly copying any bugs over as well), a new driver program was written from scratch. After some time was spent to debug and enhance the code, a working driver program was completed AT LAST. One 'slave' version and four 'master' versions of the driver program were developed. Each version of the master program collected data for a particular network configuration. Data analysis programs were then used to reduce the data down to a few meaningful parameters. Input data files containing (x,y) coordinates of data points were then made from the reduced data and were transferred from the PC system to MTS at UBC. Plots of these files were made by using Cuechart and Tellagraf (plotting programs). For each network configuration, transmission data were collected from each PC for information field lengths ranging from 50 to 1500 bytes long in 50-byte increments. Ten 5-minute transmission sessions were run for each field length. This allowed means and standard deviations to be computed for a sample size of ten. All PCs were programmed to transmit as much as possible (without any delay loop to reduce their transmission rate) during the transmission sessions. An orthodox StarLAN topology was used for all configurations. The earlier performance data (from the Intel program) had shown that there was no fundamental difference between the behaviour of the serial network and an ordinary StarLAN network. Therefore, it was felt that the new data could be safely generalized

to the case of the serial network as well. Appendix E contains a flowchart of the entire driver program.

Figure 3.35 shows a plot of the average throughput for a single-PC network (throughputs for a 2- and 3-PC network (discussed later) are also shown for comparison). Throughput, often abbreviated as S, is defined as the number of successful transmissions per unit time, normalized to the bit rate, which in the case of StarLAN, is 1 Mbps. It is always less than 1. An estimate of the software overhead time used by the program to prepare a frame for transmission was calculated (see appendix D) and translated to the field length of a frame which would take a LAN controller the same amount of time to transmit. The calculated values for overhead time varied for different frame lengths, but not by much. The mean overhead time was computed to be 5632 microseconds, which is the time required to transmit a frame whose information field length is 678 bytes long. The translated longest and shortest overhead times were 682 and 674 bytes respectively. The significance of the overhead time will be discussed later.

The throughput for a PC running the driver program was calculated to be about one-sixth (for short frames) to about seven-tenths (for the longest frame) of the throughput when the Intel program was used. This showed that the Intel program was far more efficient (shorter overhead time) than the driver program. Therefore, results obtained with the driver programs were not expected to reproduce the performance data collected with the Intel program. At any rate, no attempt was made to compare the data because the amount of variance in the earlier data was not known. Furthermore, the old data covered only three field lengths while the new data covered a comprehensive range of field lengths.

Figure 3.36 shows a plot of the average throughput for a 2-PC system ('PC3' and 'PC4' are just labels for the two PCs; note also that they have the same curve and cannot be distinguished separately in the graph). The individual throughputs level off

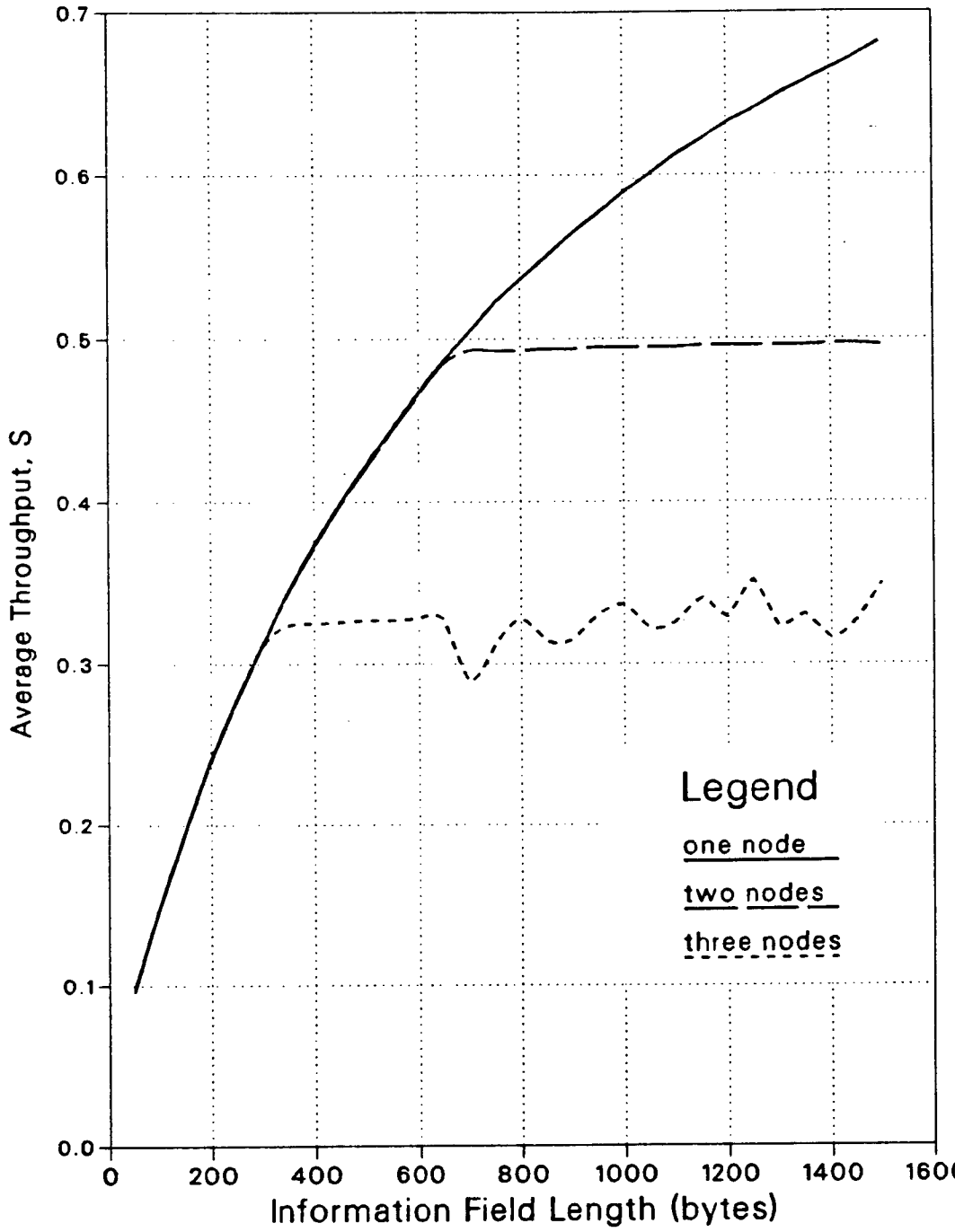


Figure 3.35: Average throughput for a single-PC network.

at approximately 700 bytes when the total throughput approaches the channel bit rate. Figure 3.37 shows a plot of the corresponding offered load. Offered load, or G, is defined here as the total number of attempts to access the channel (successful transmissions, collisions, and deferred transmissions) per *unit* time, normalized to the bit rate. It can be greater (even much greater) than 1. At about 650-700 bytes, it increases sharply due to a sudden increase in the number of deferred transmissions, as shown in figure 3.38, which plots the number of deferrals as a fraction of the offered load. These transitions occur at just about the equivalent software overhead time of 678 bytes. Before the transition point, the frame transmission time is longer than the time it takes to prepare a frame for transmission. The frame transmitted by one PC is completed before the other PC has prepared the next frame for transmission. Hence, the two PCs more or less take turns when transmitting and deferrals are relatively rare. (PC4 defers more often than PC3 because it probably runs somewhat faster, and therefore, can prepare a frame for transmission sooner than PC3. Every now and then, it 'catches up' with PC3 and is consequently forced to defer transmission until PC3 finishes.) Beyond the transition point, when the frame transmission time becomes longer than the overhead time, one PC will still be transmitting while the other has already prepared another frame. The other PC then has to defer transmission until the transmitting PC has finished. Both PCs still take turns transmitting but must now defer transmission at every attempt to transmit. Collisions rarely occurred in the two-PC system. The amount of fluctuation in throughput and offered load was very small and will not be shown.

Figure 3.39 shows the average throughput for a 3-PC network where frames are not received by any PC (the destination addresses of the frames are different from their source addresses). 'PC1', 'PC3' and 'PC4' are labels to identify the three PCs. Throughput saturation occurs at approximately 350 bytes but fluctuations in the average throughput do not occur until after 650-700 bytes. A plot of standard deviation in throughput

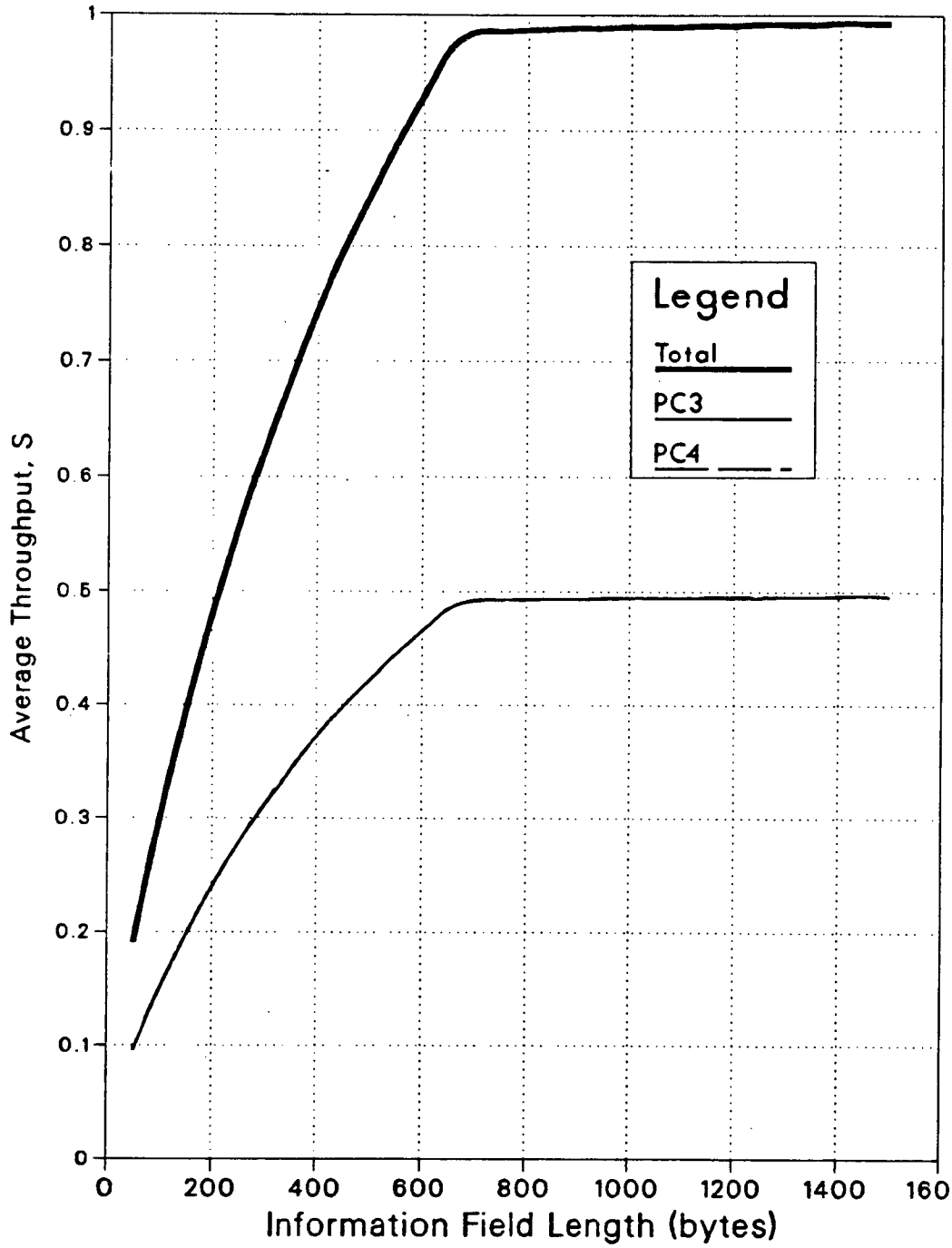


Figure 3.36: Average throughput for a 2-PC network.

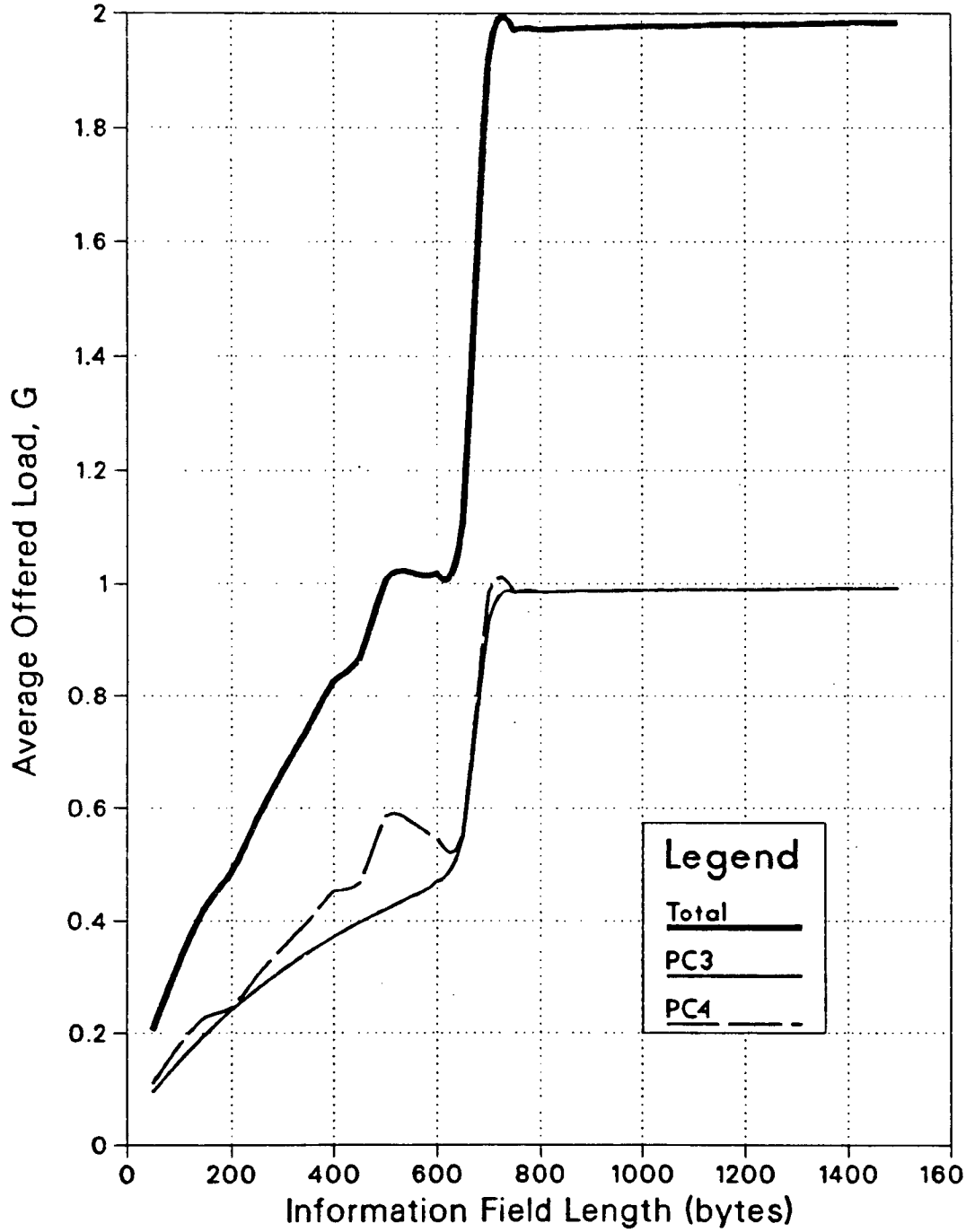


Figure 3.37: Average offered load for a 2-PC network.

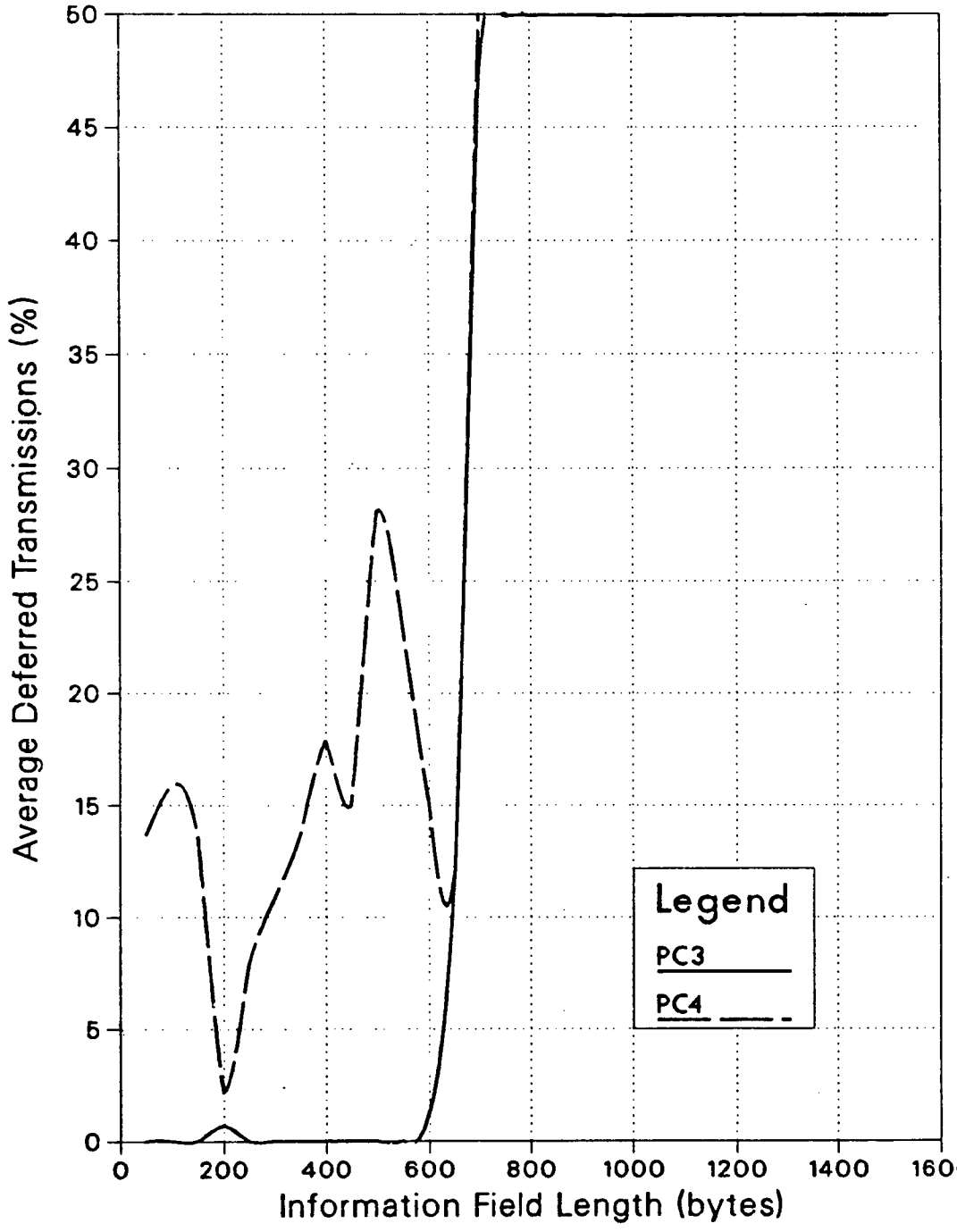


Figure 3.38: Number of deferrals expressed as a fraction of offered load.

expressed as a fraction of the average throughput is shown in figure 3.40. Note the sudden increase in the fractional deviation at about 650-700 bytes. By contrast, the fractional deviation of the total throughput remains close to zero. Similar behaviour is observed in the fractional deviation and the average of the offered load, as can be seen in figures 3.41 and 3.42. Plots of the number of collisions, deferred transmissions, and exhausted retransmission attempts, all expressed as a fraction of the offered load, are shown in figures 3.43, 3.44, and 3.45. At the point of throughput saturation (350 bytes), all PCs begin to defer at about every other transmission attempt. Note the large number of deferrals by PC1 before the 350-byte point. This indicates that PC1 is probably running faster than the other PCs (as explained earlier in the 2-PC case for figure 3.38). Collisions are rare until about 700 bytes, where there is a sharp increase in the number of collisions and exhausted retransmission attempts (which indicate unresolved collision intervals). As with the 2-PC configuration, the transition point occurs when the frame transmission time becomes as long as the overhead time. For field lengths equal to or longer than the equivalent overhead time of about 678 bytes, two PCs will have prepared their frames for transmission while the third PC has not yet finished transmitting. This ensures that channel contention between two PCs will occur once the third PC has finished. The chaotic situation that results from the contention leads to more unpredictable behaviour as is shown in the plots of fractional deviations.

The last configuration to be discussed is a 3-PC network where one PC transmits to the other PC in a circular manner so that each PC is the receiver of another PC's transmissions. Since the PCs are now receiving as well as transmitting frames, the effective overhead time for preparing a frame for transmission is somewhat longer. (Received frames have to be processed before a transmit command can be sent to the LAN controller.) Hence, the transition point is expected to shift to a longer field length. Figures 3.46 and 3.47 show the average throughput and offered load for different field lengths.

Note the dip in both graphs at 800 bytes and the increased fluctuation of the individual PC curves which begin at about the same field length. The cause of the dip in both graphs is seen in the average number of deferrals on figure 3.48, which shows a notch (indicating a sudden drop in the fraction of deferrals by all three PCs) at about 800 bytes. Associated with this notch is a sharp increase in the number of collisions which shows up as a spike in figure 3.49. This shows that the frequent collisions obviously had a detrimental effect on not only the throughput of the individual PCs, but also the total throughput of the network as well. The drop in deferrals (figure 3.48) is a peculiarity that probably arises from the transmission sequence of the PCs and the nature of their transmissions (to transmit as often as possible whenever there are no received frames to process). A plot of the number of lost frames (good transmissions which were not received at all), expressed as a fraction of the total number of frames received is shown in figure 3.50. A sharp increase in the number of lost frames occurs at 800 bytes for one PC while slight increases occur for the two other PCs at 850 bytes. A satisfactory explanation has not been found for this observation. The number of buffer overflows (frames which could not be received because the frame buffers were full; the driver programs have 16 receive buffers) is shown in figure 3.51. Note the sudden increase which begins at around 800-850 bytes. The increase is probably due to the increased frequency of bad received frames, which take a longer time to process (to identify the type of error) than good frames. This indicates that the transmission rate for any PC is generally faster than the processing rate of bad received frames. To round out the remaining graphs, plots for the number of exhausted retransmission attempts, and fractional deviations of throughput and offered load are shown in figures 3.52, 3.53, and 3.54. All three parameters increase sharply at around 800 bytes (although the fractional deviation of the offered load is also high between 50 and 450 bytes). By extending the results from the configurations

discussed earlier, it was concluded that the effective overhead time increased to approximately 800-850 information field bytes since this was the region in the graphs where the onset of drastic changes in network behaviour and adverse performance characteristics were observed. The increased overhead time is expected in view of the fact that each PC has to process received frames. If a frame is received, the currently running task is interrupted and another buffer is allocated for the next incoming frame. Hence, the time to prepare a frame for transmission is lengthened if frames are received during the preparation interval for a transmission.

The results of these measurements have implications that should be considered when a small number of nodes are continually transmitting frames at a high rate in a network where the propagation delay is short. A reduction in communication reliability as well as throughput can occur if the frame transmission and overhead times are roughly equal. Sharp changes in network behaviour occur when the frame transmission time is equal to or longer than the overhead time. In general, fluctuations of throughput and offered load increase sharply beyond a critical transition point. Before the critical point, channel access can be characterized as essentially free from contention. Beyond the critical point, there is a marked increase in the amount of contention for channel use, which influences the variability of many transmission characteristics. Network performance may be optimized to some extent if test transmissions are evaluated to determine the transmission and reception behaviour for the application at hand. The results of these performance measurements also show that one can indirectly estimate the software overhead time for frame transmission by locating the critical transition point from transmission data of a 2- or 3-node network (as described earlier).

One interesting question about the performance data obtained here is how they compare to theoretical results and measurements from other network configurations (such

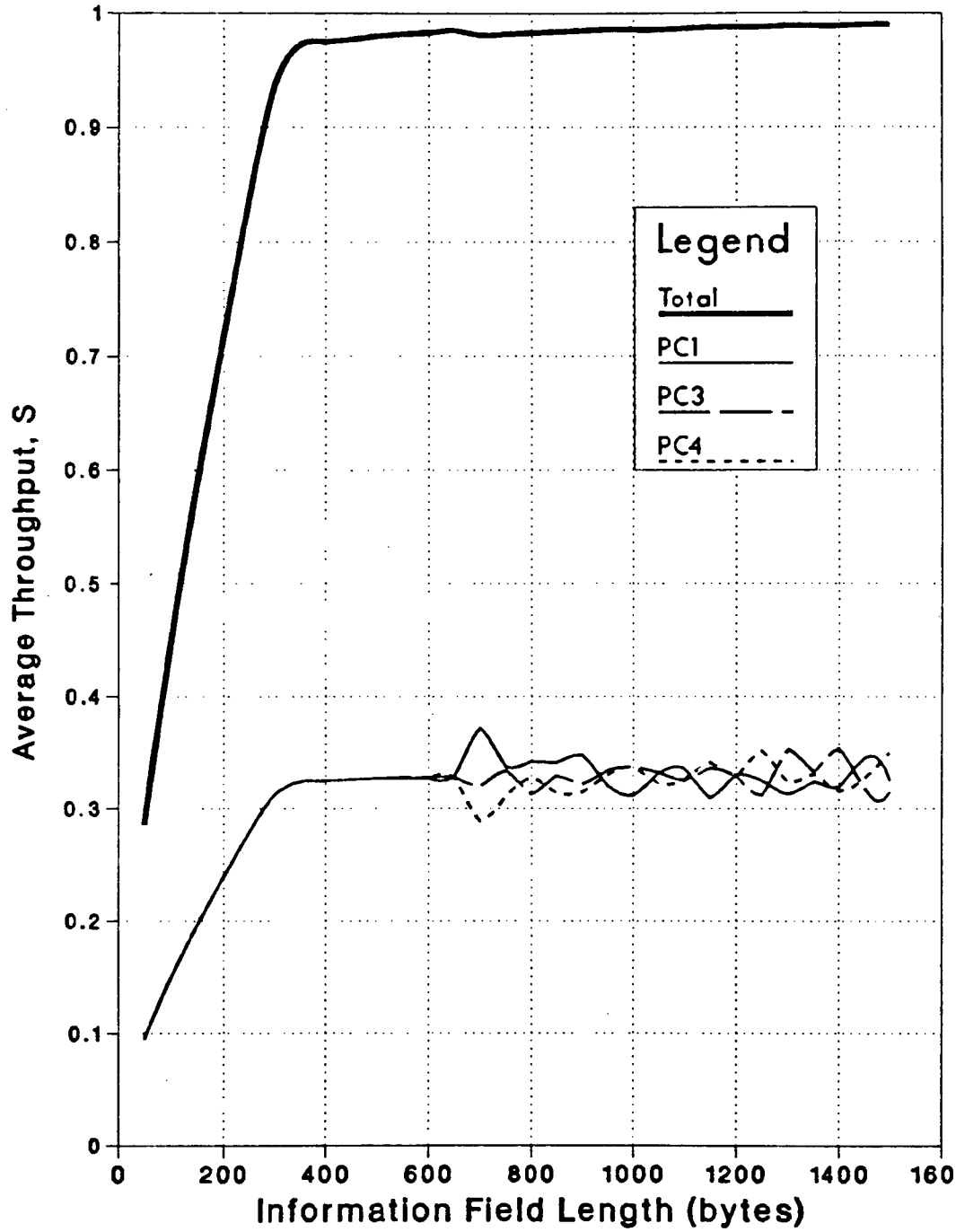


Figure 3.39: Average throughput for a 3-PC network without frame receptions.

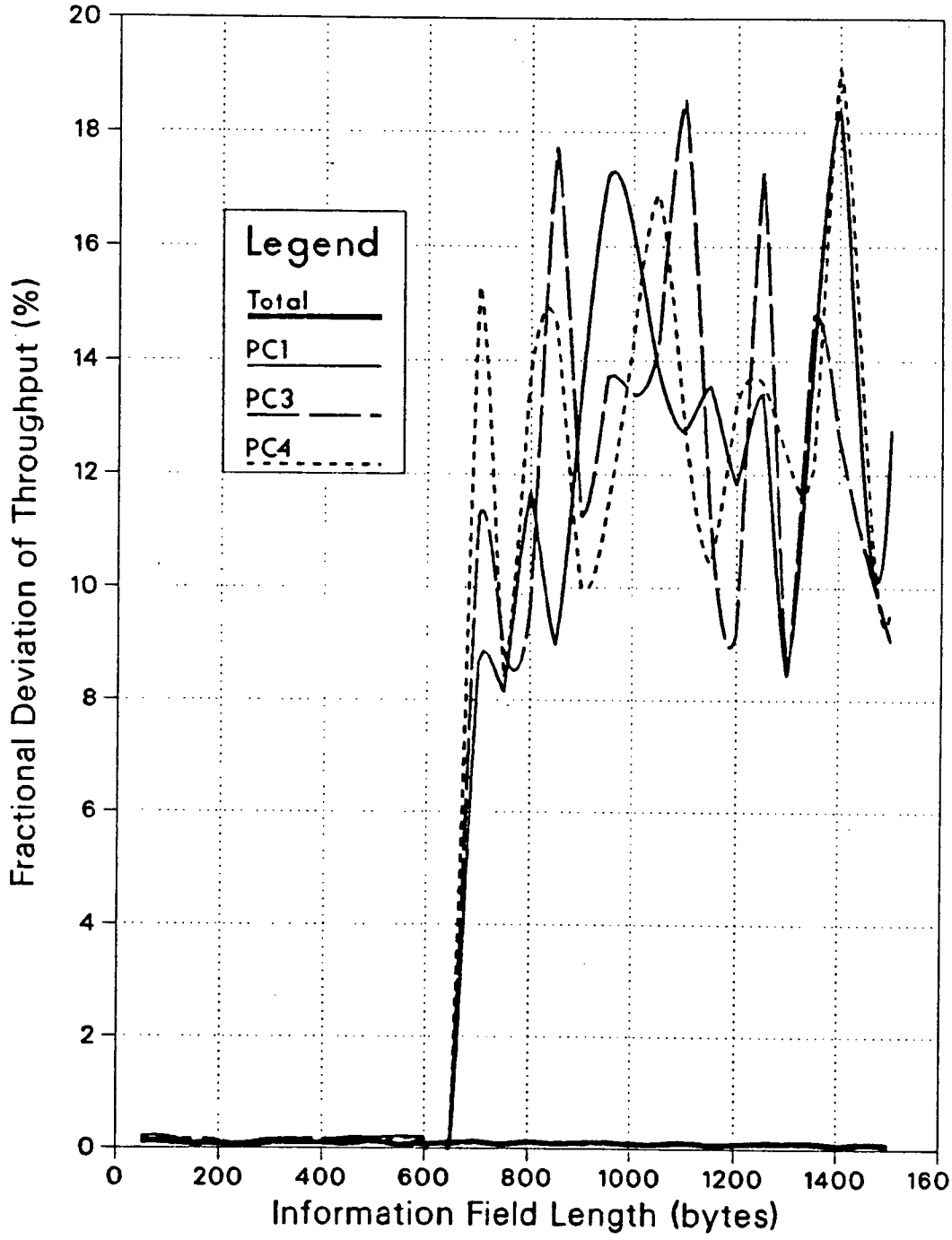


Figure 3.40: Fractional standard deviation of throughput.

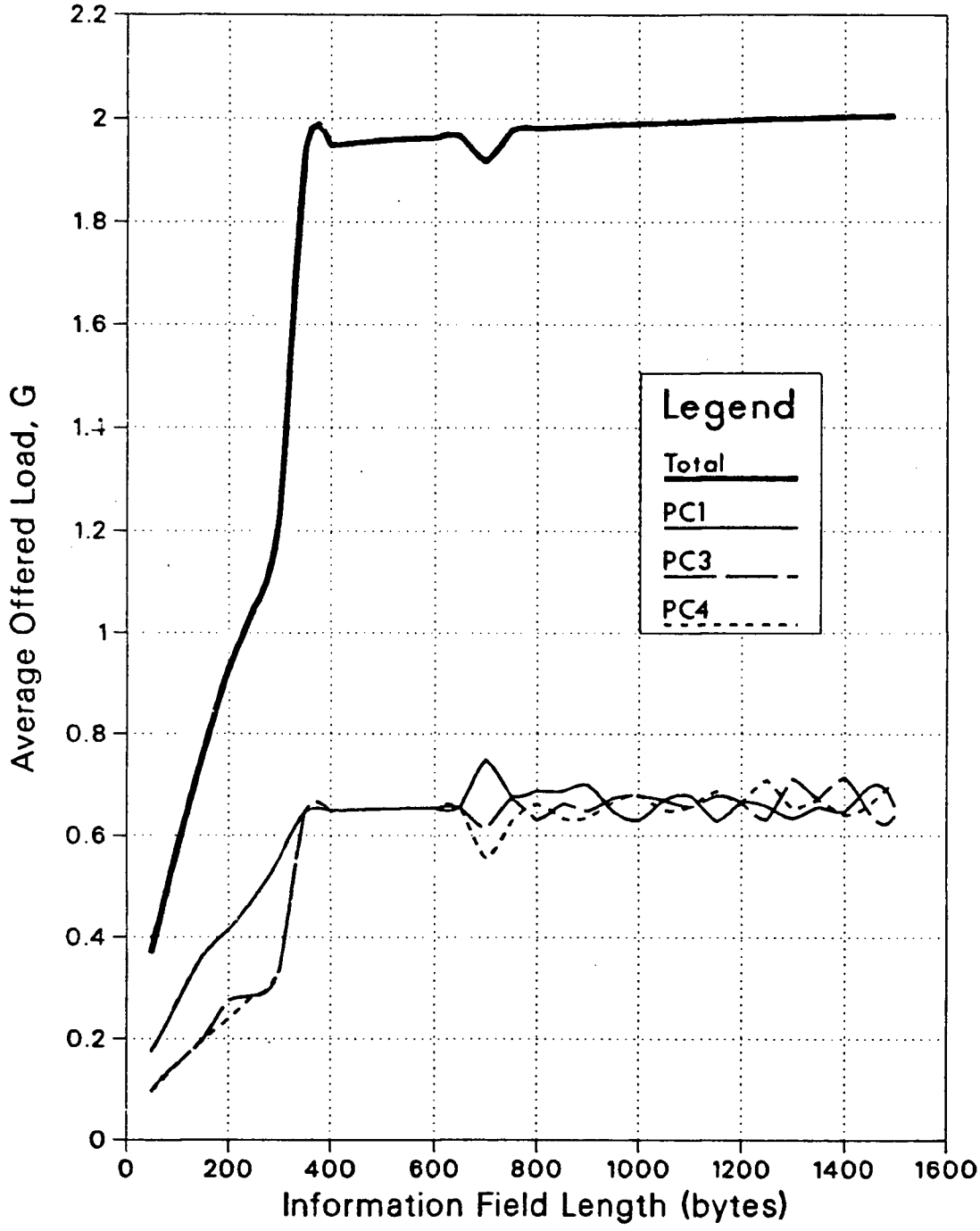


Figure 3.41: Average offered load for a 3-PC network without frame receptions.

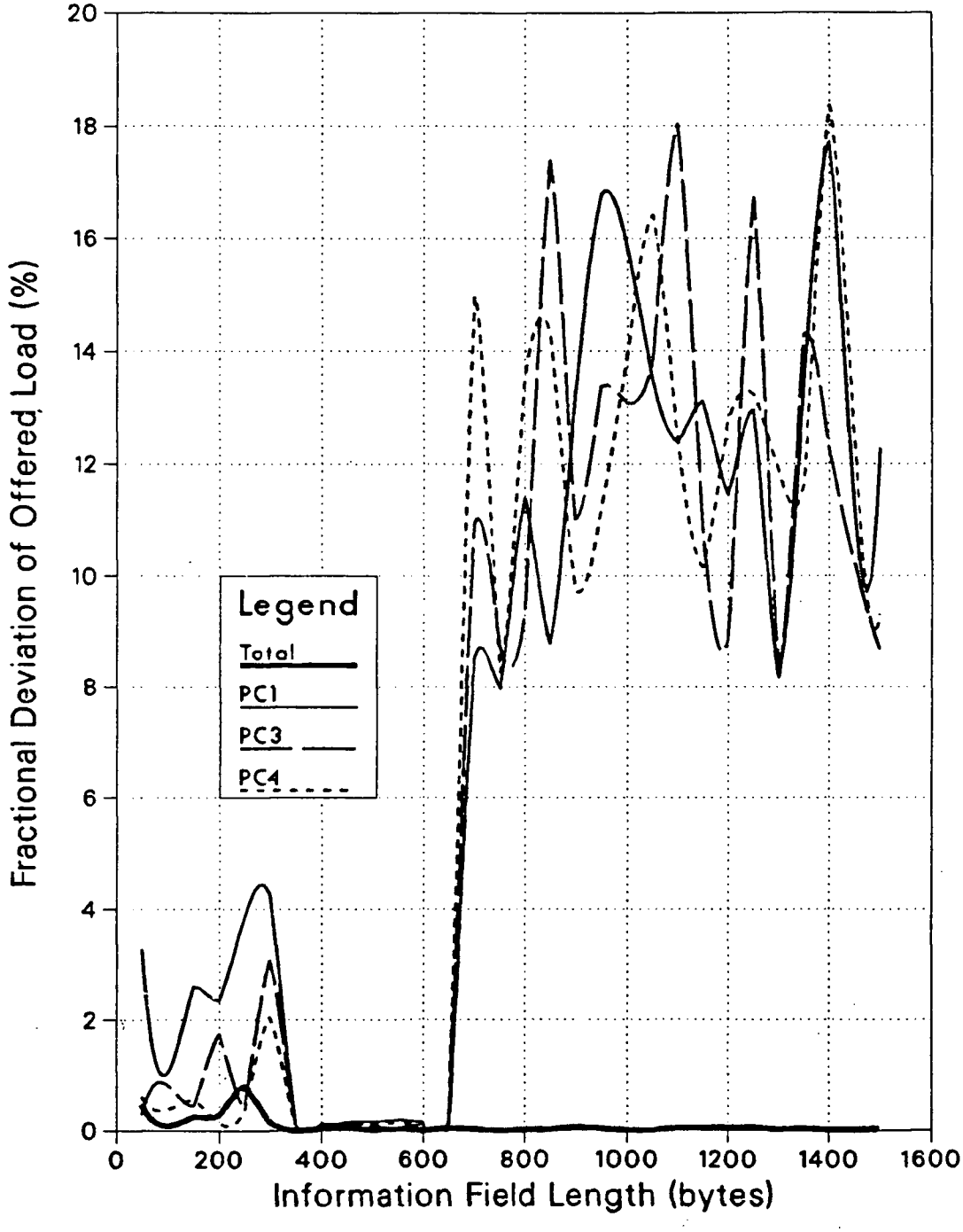


Figure 3.42: Fractional standard deviation of offered load.

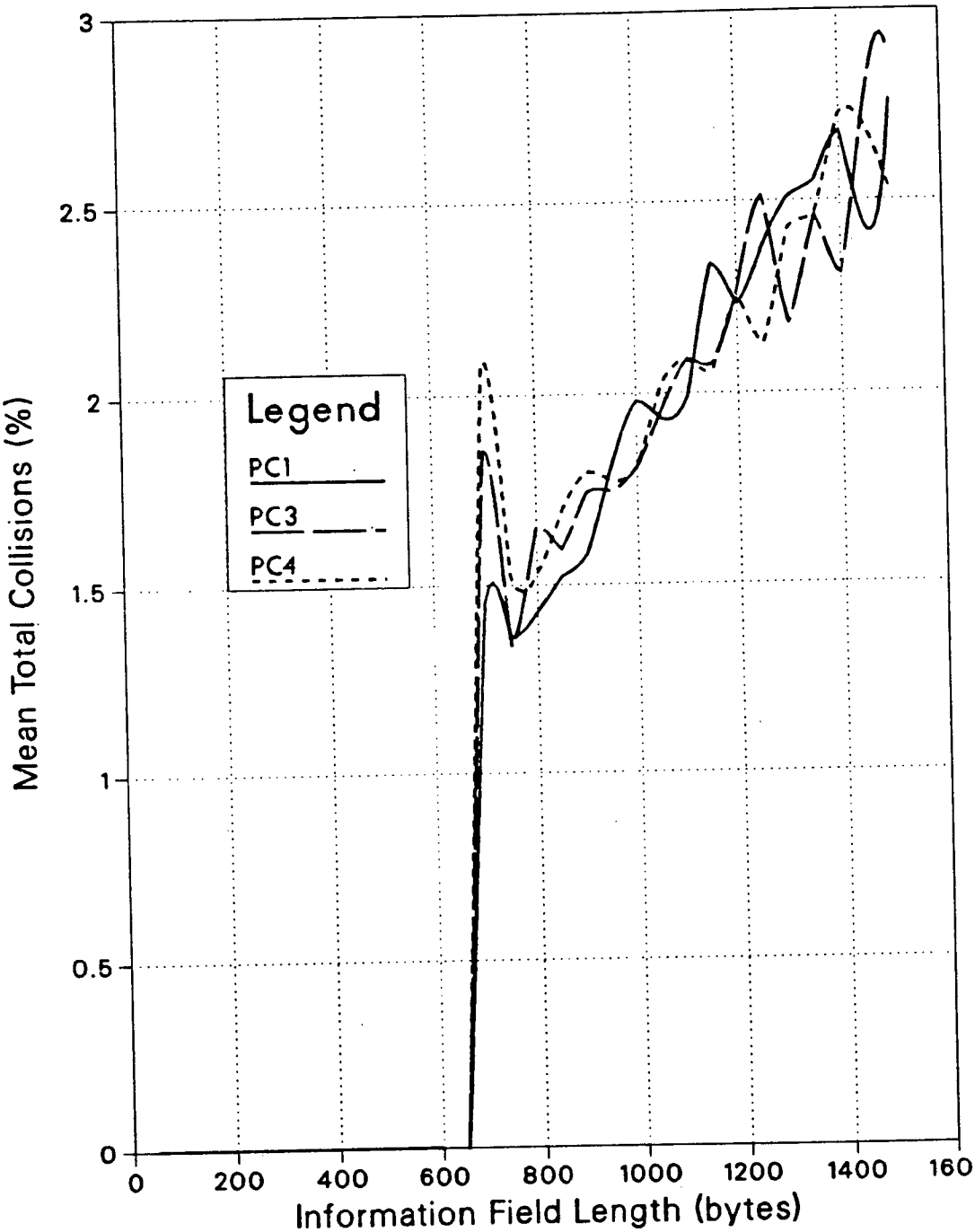


Figure 3.43: Number of collisions expressed as a fraction of offered load.

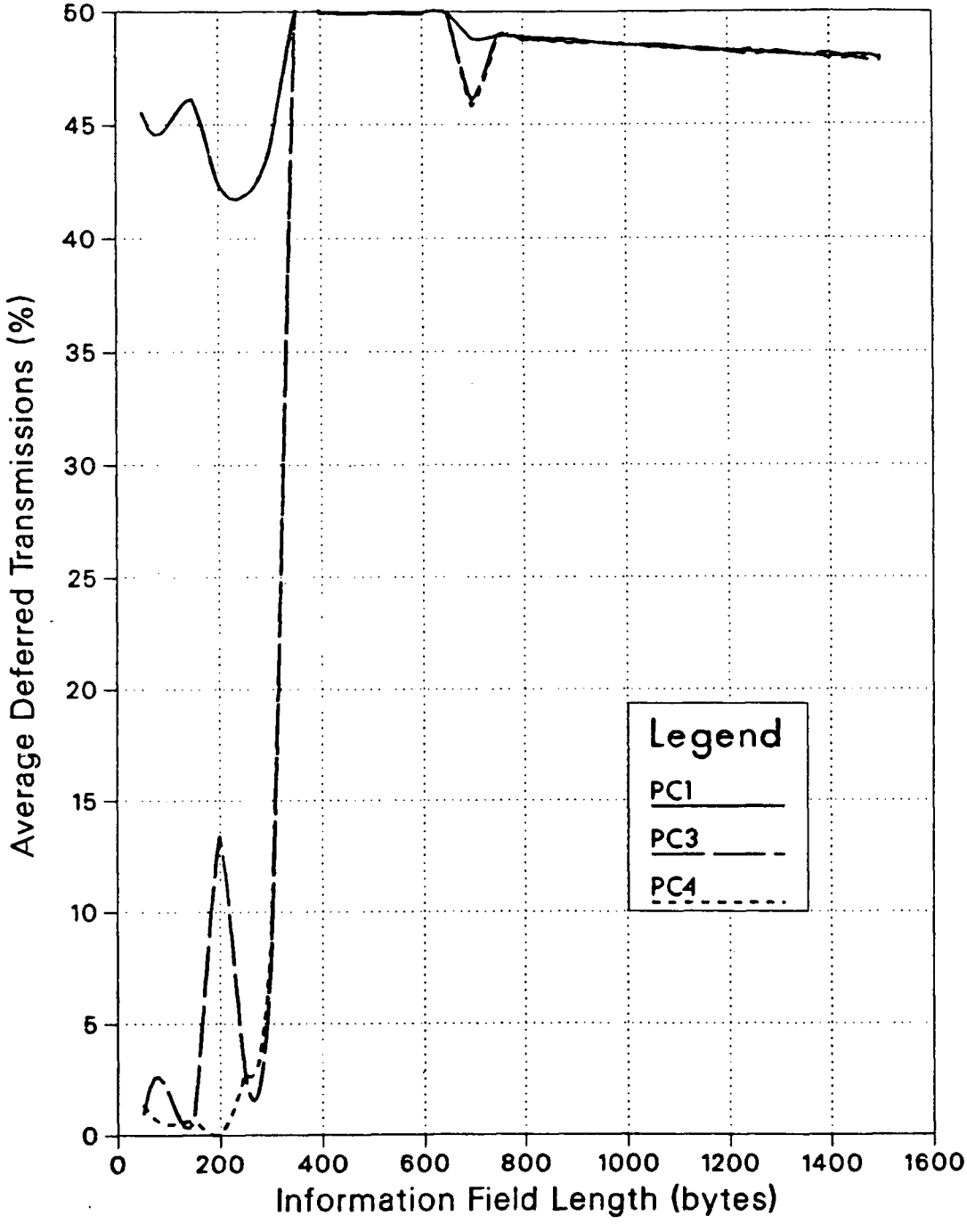


Figure 3.44: Number of deferrals expressed as a fraction of offered load.

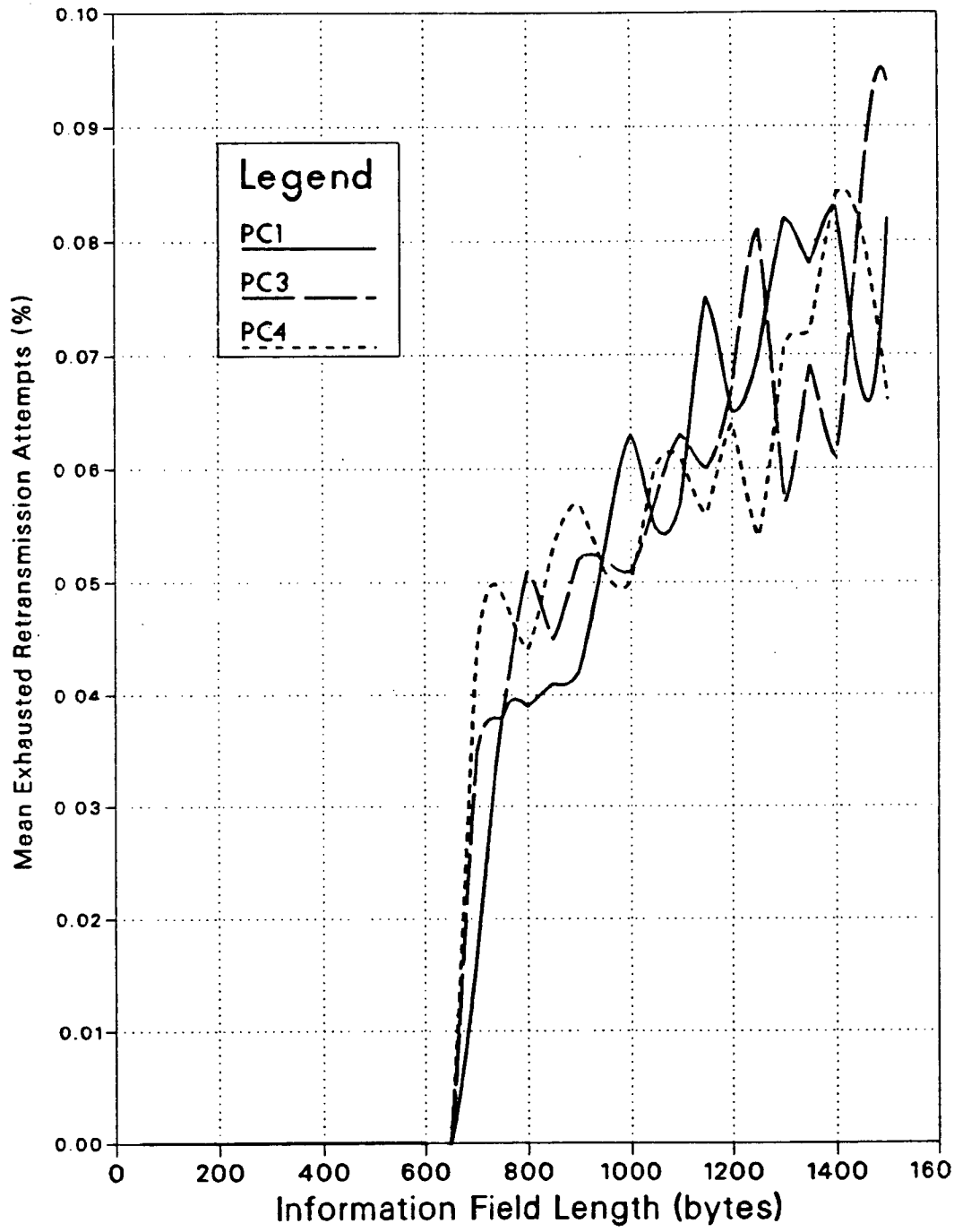


Figure 3.45: Number of unresolved collision intervals as a fraction of offered load

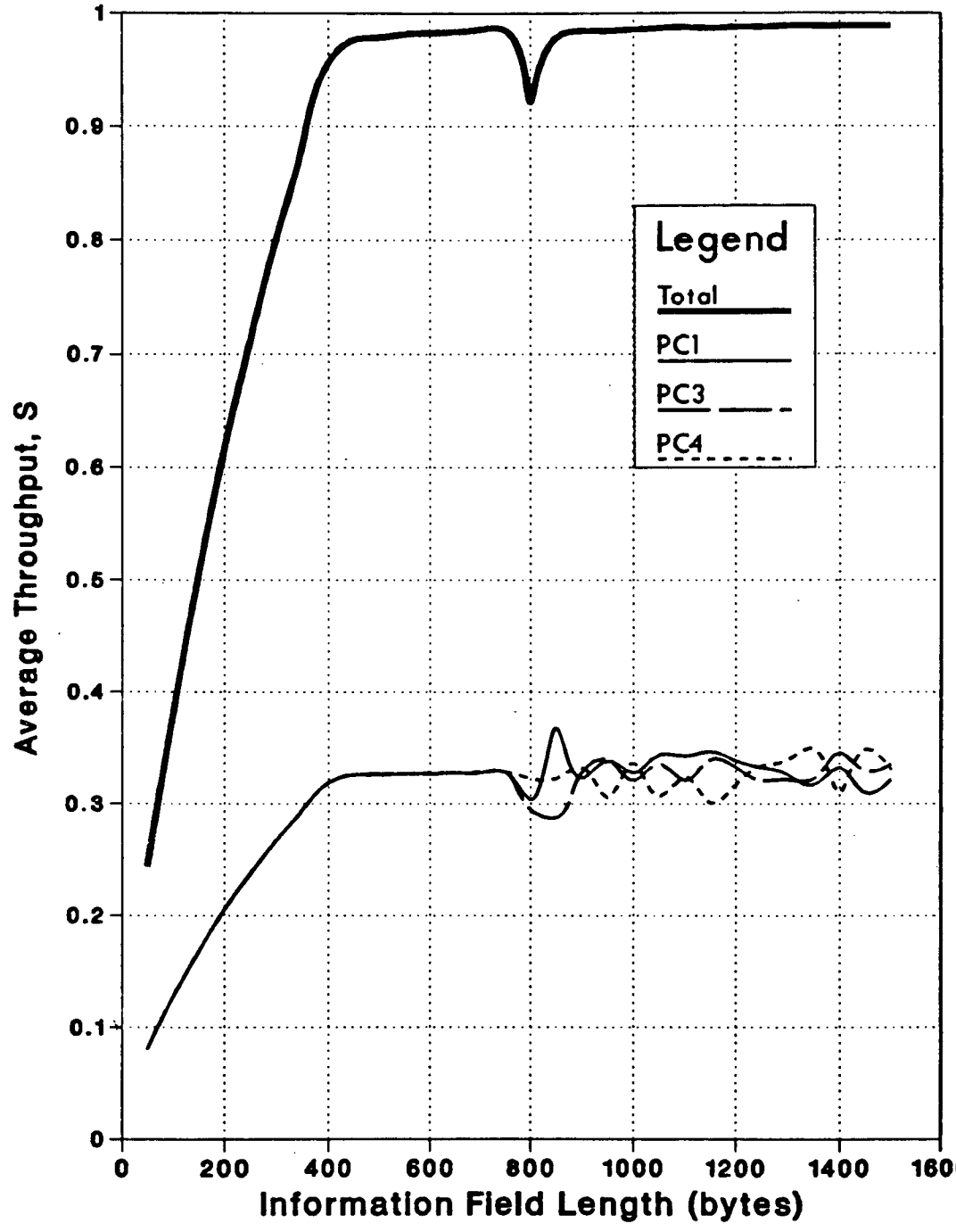


Figure 3.46: Average throughput for a 3-PC network with frame receptions.

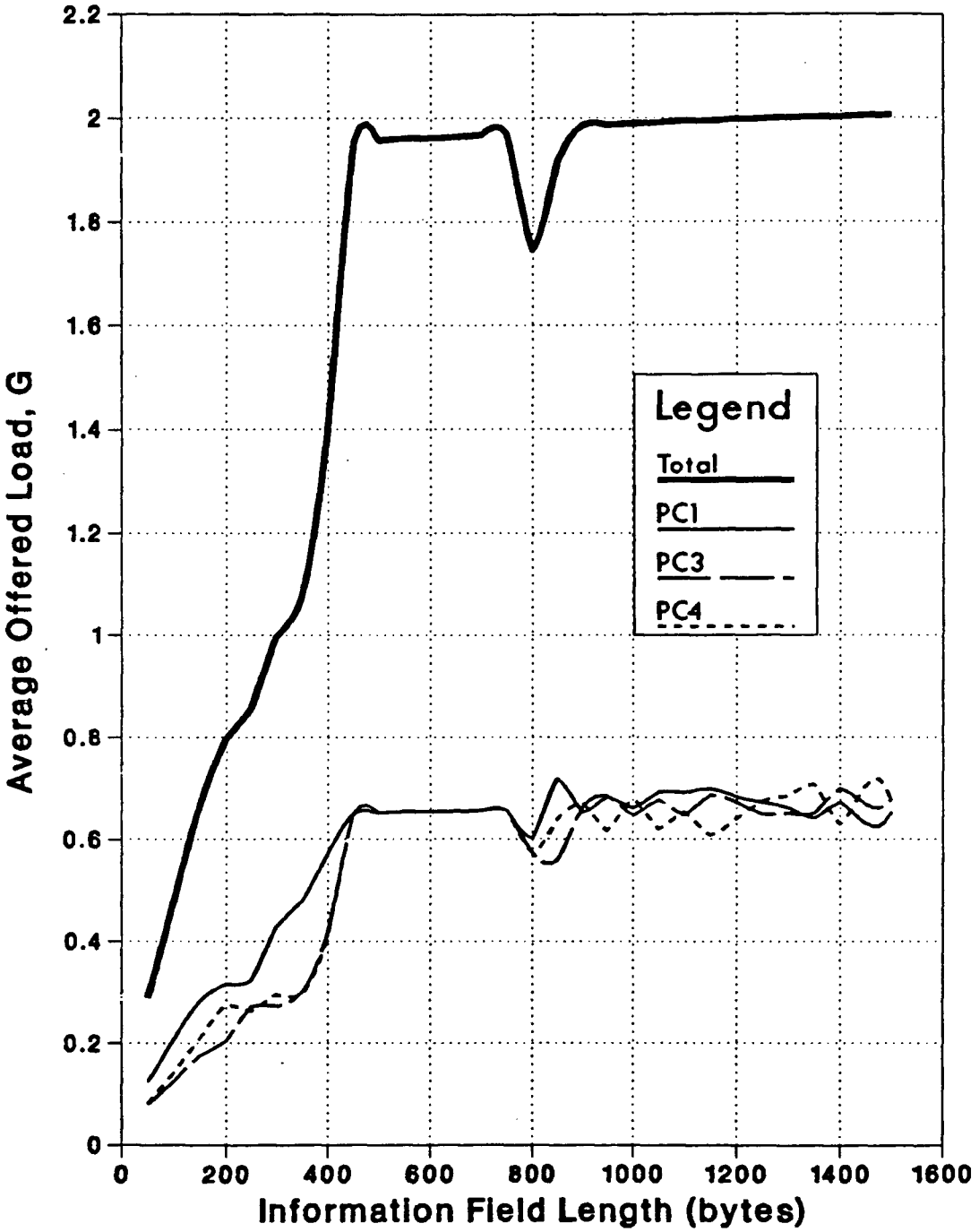


Figure 3.47: Average offered load for a 3-PC network with frame receptions.

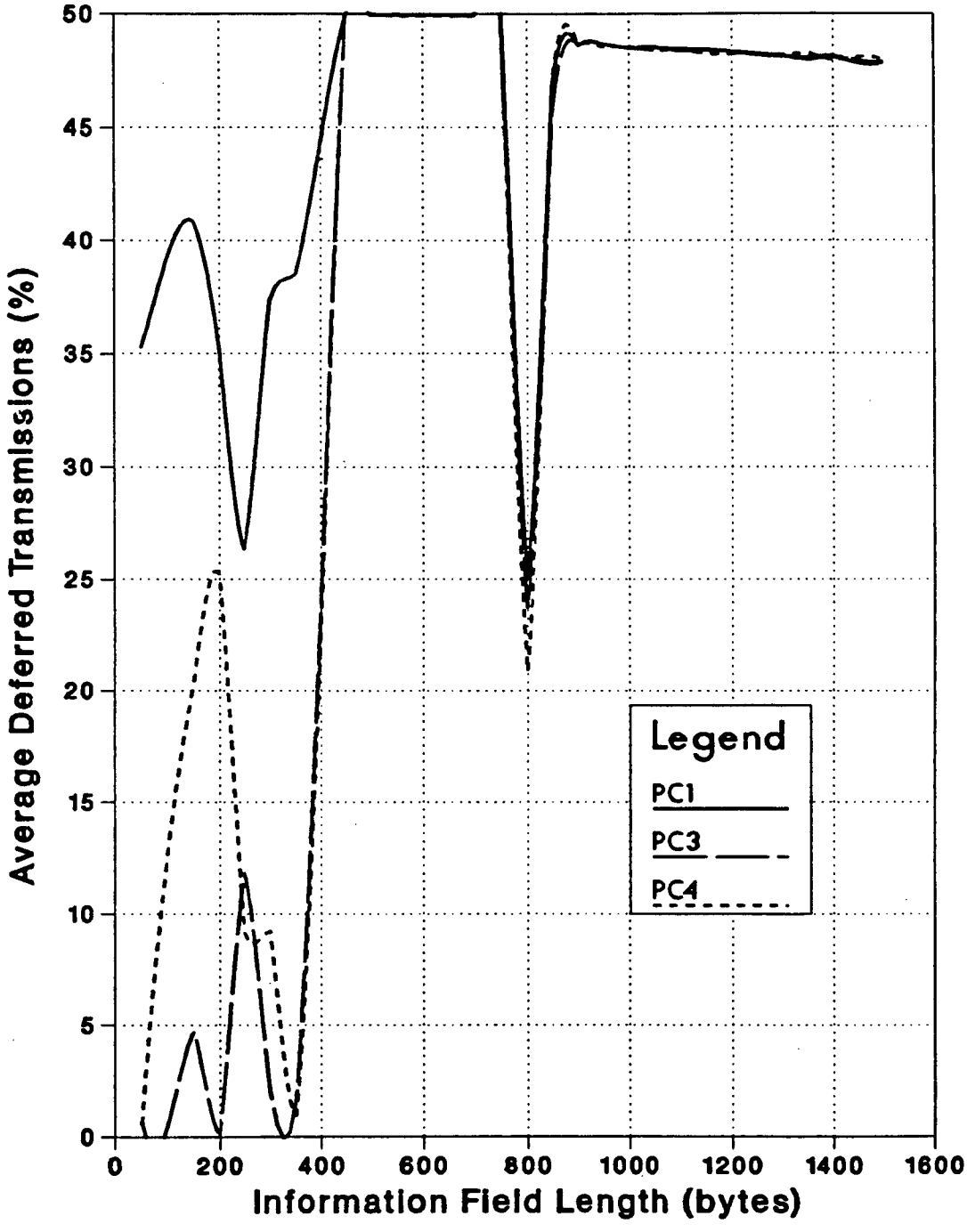


Figure 3.48: Deferred transmissions expressed as a fraction of the offered load.

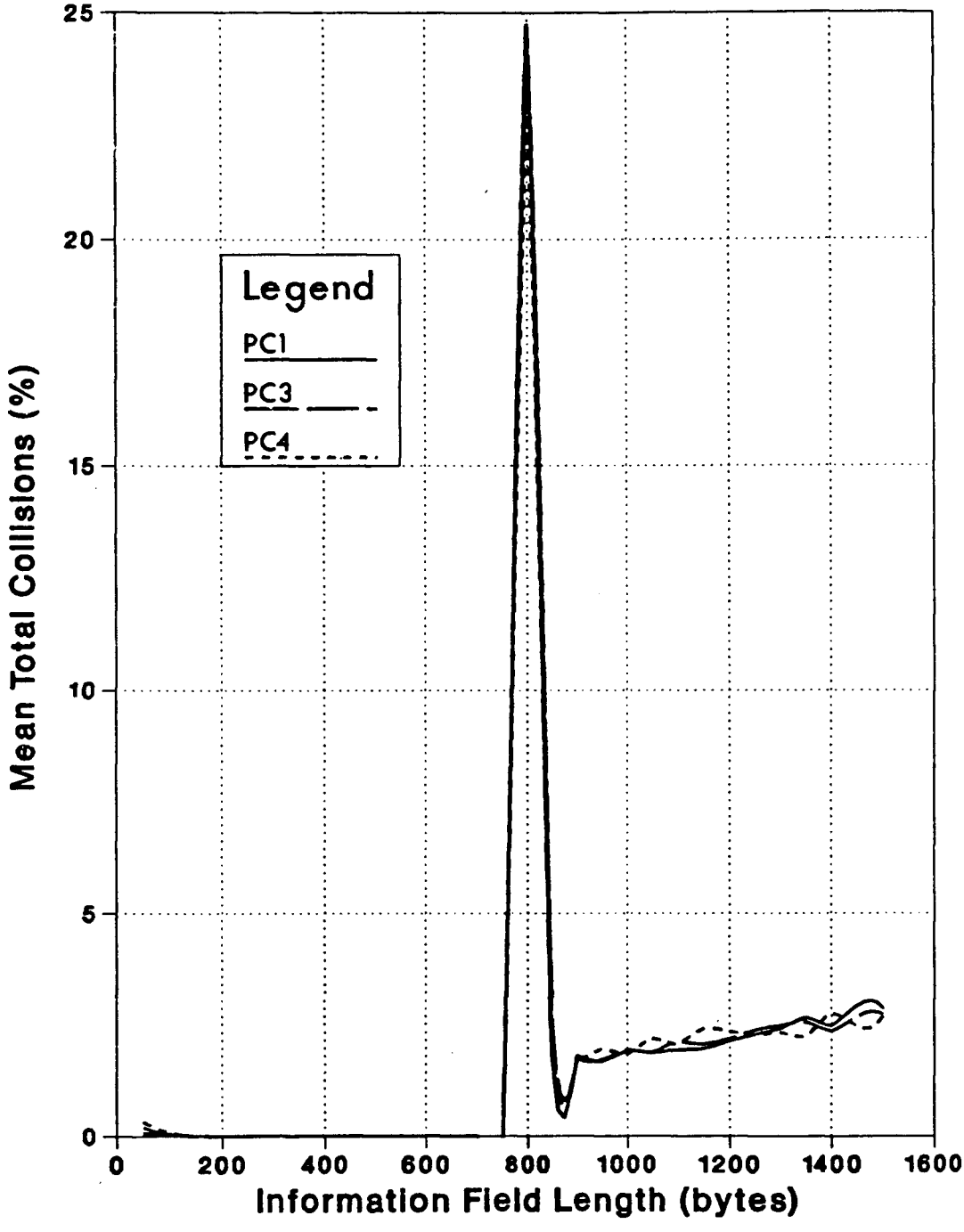


Figure 3.49: Collisions expressed as a fraction of the offered load.

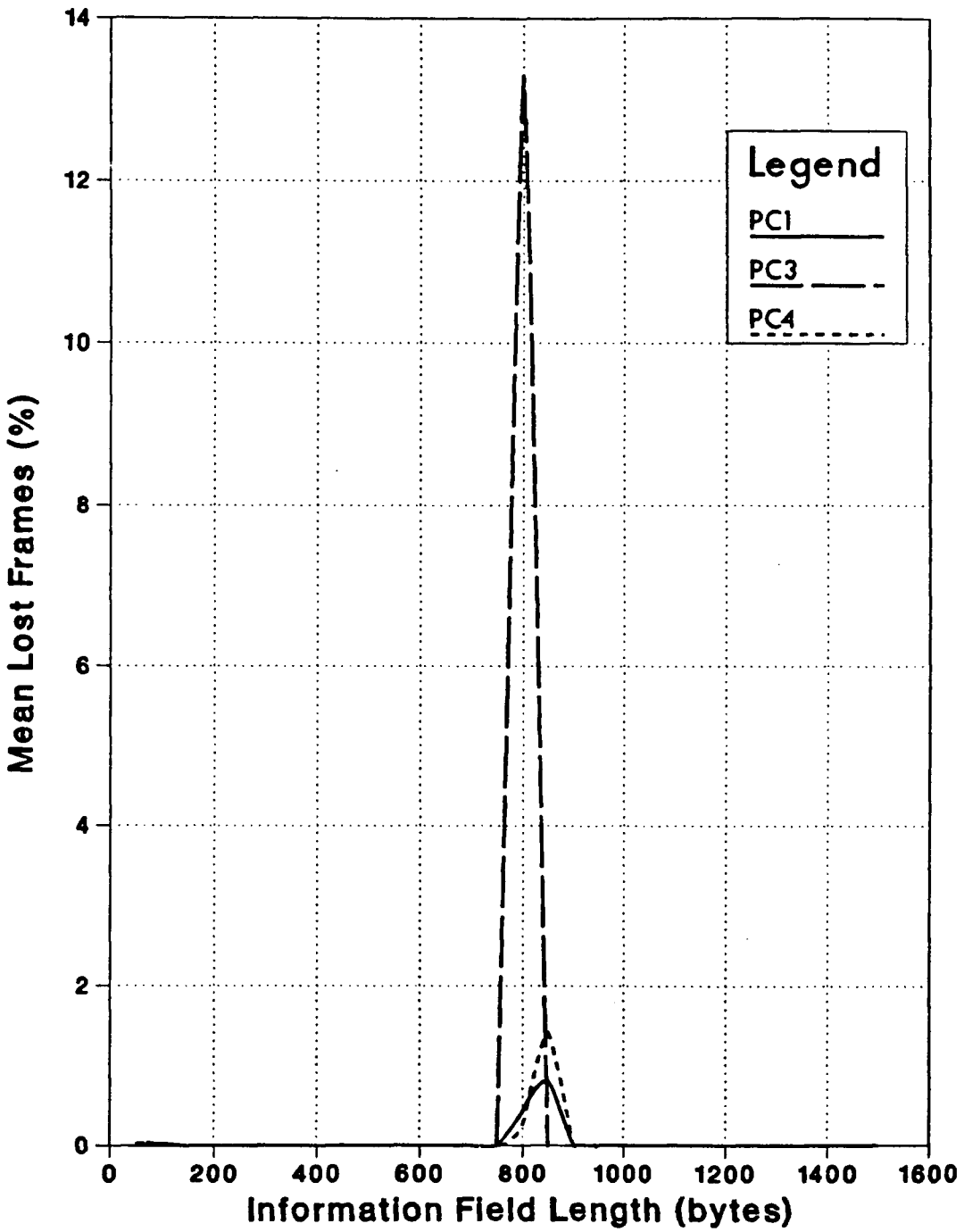


Figure 3.50: Lost frames expressed as a fraction of the total number of received frames.

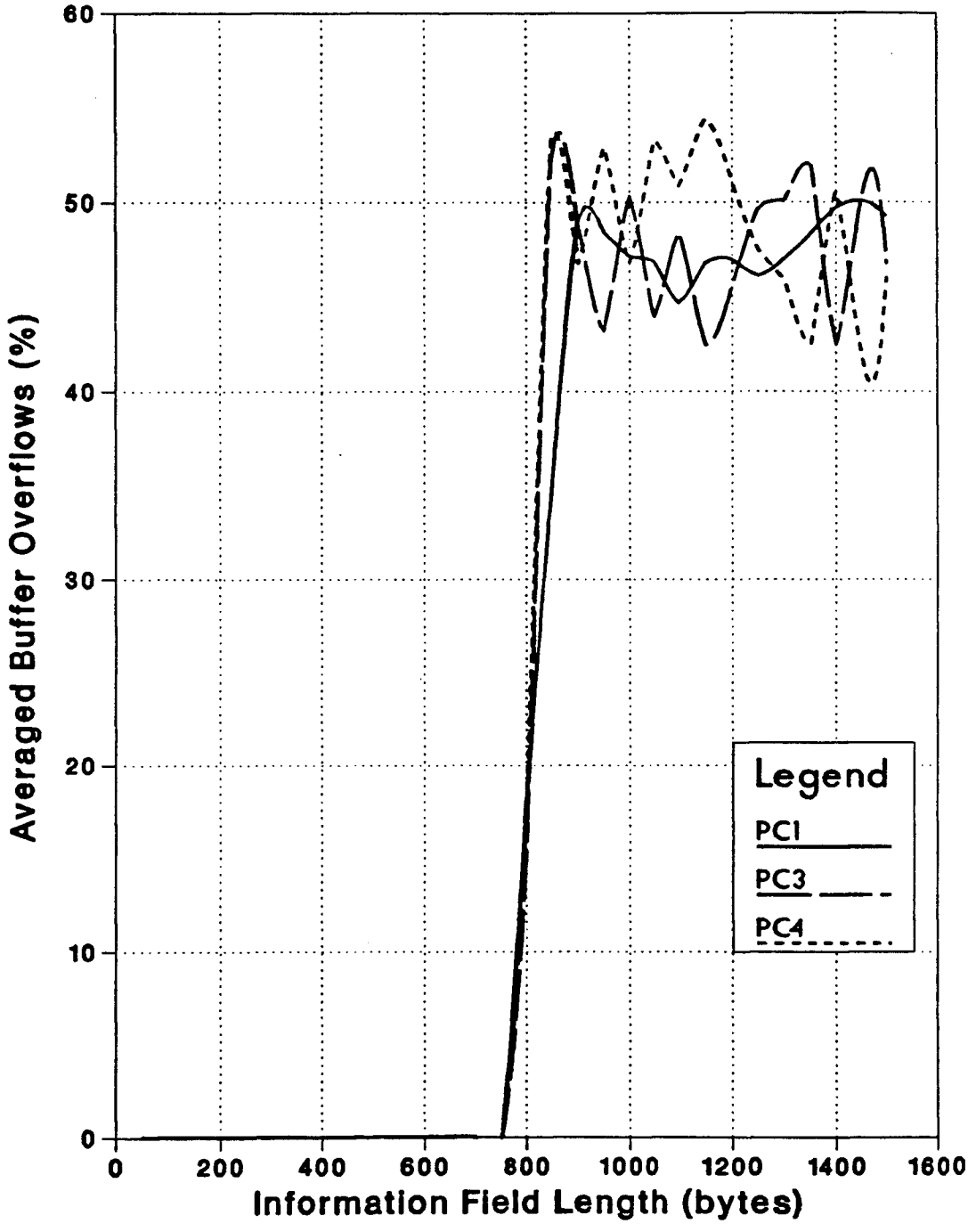


Figure 3.51: Buffer overflows expresssed as a fraction of the total number of received frames.

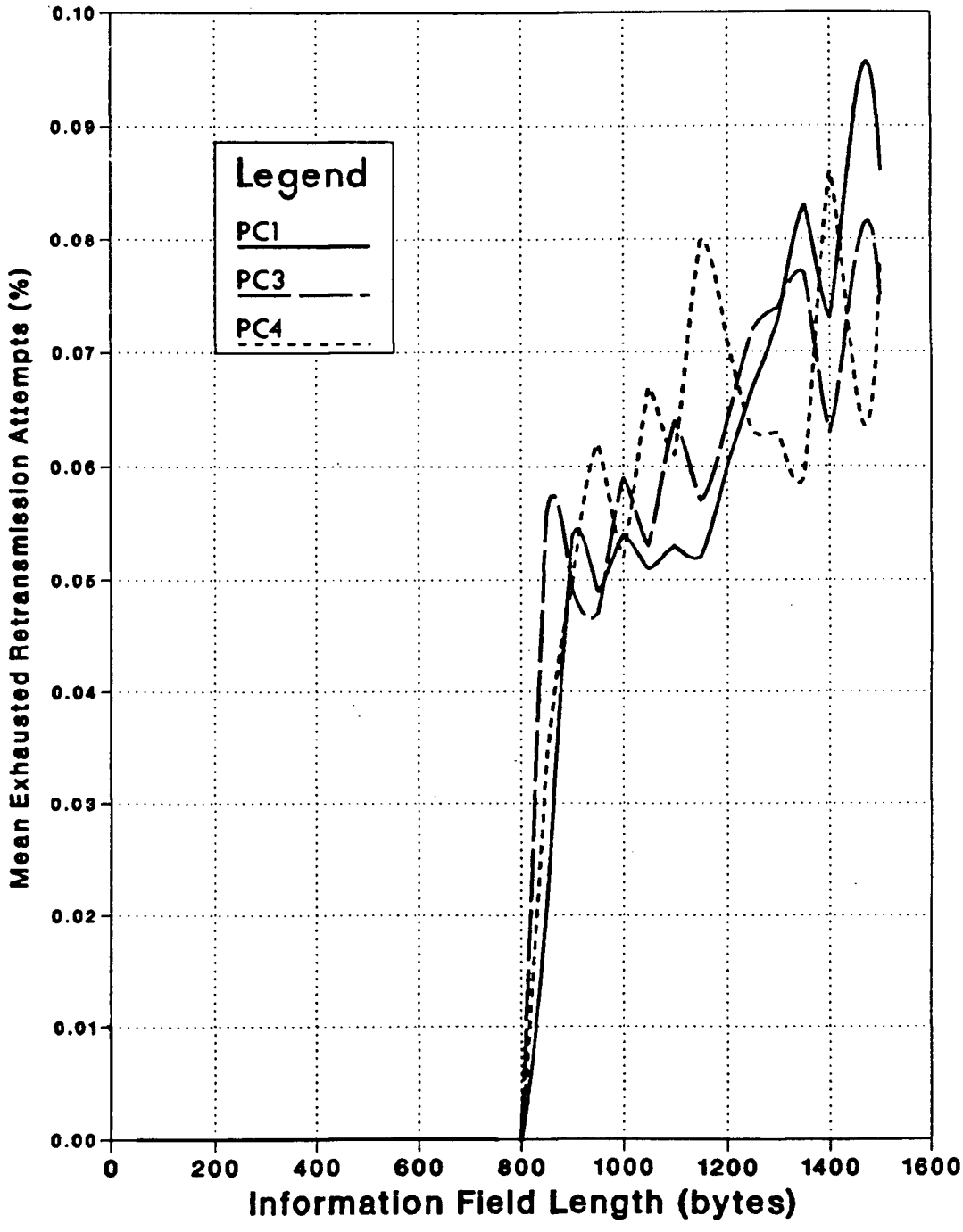


Figure 3.52: Number of unresolved collision intervals as a fraction of offered load

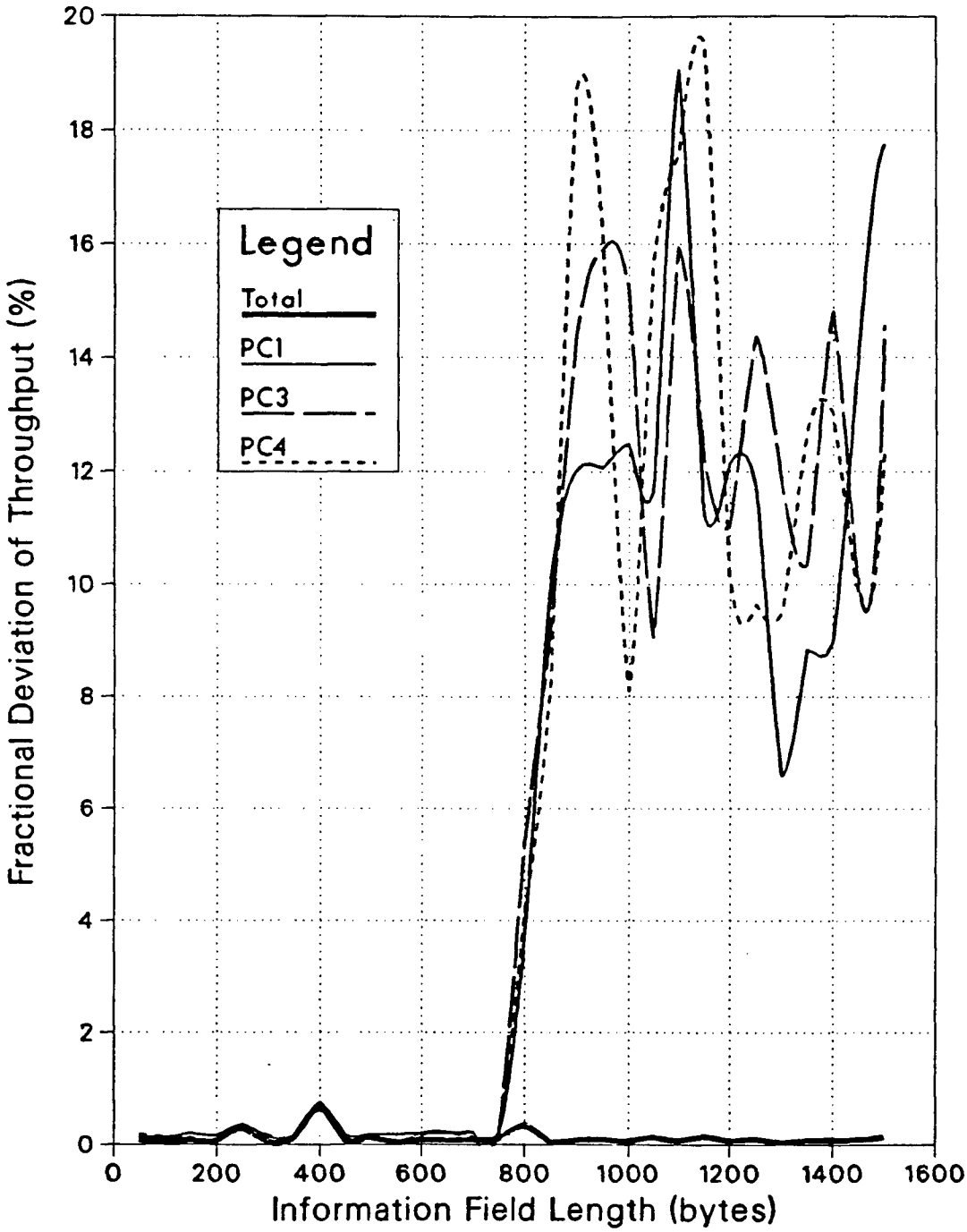


Figure 3.53: Fractional standard deviation of throughput.

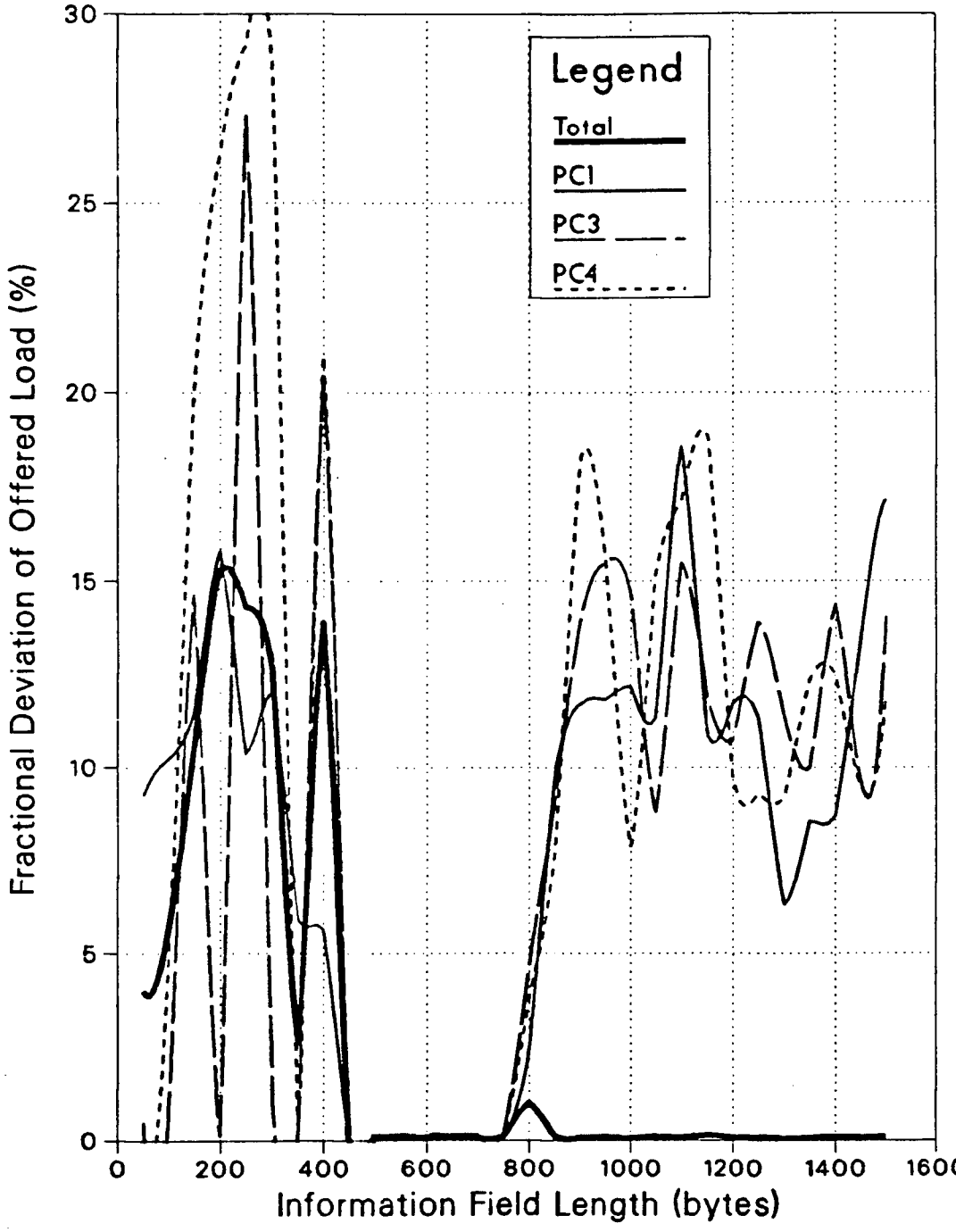


Figure 3.54: Fractional standard deviation of offered load.

as, say, 10 or more nodes). No comparisons were made in either case, but a comparative analysis for a related CSMA/CD standard, Ethernet, is given in [20]. The general results from that reference may be extended to some extent to the case of StarLAN, which is similar in many respects to Ethernet. For Ethernet, the influence of the number of stations on throughput for a given offered load is minor for long frames but can be significant in a network with a small number of nodes (less than 20) if the frames are short and the offered load is large. A discussion on the similarities (and differences) between measurements of and theoretical results for maximum throughput can be found in [20] and will not be given here. In general, the data obtained from the performance measurements are somewhat restricted in the sense that they apply only to a system with a short propagation delay where a few nodes transmit continually. A part of the difficulty here is due to the fact that it is essentially impossible to generalize results from a very small number of nodes. Nevertheless, the results obtained here provide the basis for, if necessary, more exhaustive measurements.

Chapter 4

System Implementation Options

Several alternatives are available to the system designer should StarLAN be selected as the bottom-layer protocol for the FASTBUS Serial Network. These alternatives fall into roughly three areas: topology, bit rate, and access scheme.

4.1 Topology

As was discussed earlier in the Introduction, two topologies are available for StarLAN. The first one is a star topology that uses hubs, as was originally conceived by the IEEE 802.3 Working Group. The second one is a bus topology which does not use hubs. This topology has been promoted most notably by Advanced Micro Devices. The choice of interconnection topology depends on a number of practical considerations and these are discussed below.

The star topology allows nodes and hubs to be easily attached or removed from the network. In a star topology serial network, each FASTBUS crate would contain a hub that connects to higher hub layers in an orthodox star configuration while linking the nodes downstream (the devices plugged into the crate) as a two-line backplane bus. Only nodes are allowed to be connected downstream of a crate hub. This means that FASTBUS crates can be removed from the network by simply disconnecting the crate hub from the upstream hub. There is no need to reconnect lower hub layers back to the network since there aren't any. However, there is a problem with jabber control. If a crate hub cannot stop a node from jabbering, all nodes in the same crate are disconnected

from the network. This form of collective punishment can occur because the nodes are connected together in a bus topology which appears to the hub as just one node input. The crate hub cannot select the node to be disconnected. In fact, it cannot identify the erring node(s) within the crate. Granted that jabbering nodes are expected to be very rare, the only solution to this problem is to provide all nodes with jabber control circuits, transferring the jabber control function from the hub to each and every node. The Am7961 bussed StarLAN transceiver has such a function. However, the chip uses RS-422 (StarLAN) signal levels for the bus data lines which would have to be translated to ECL logic voltages. Another concern centres on the reliability of the hubs themselves. A single hub failure can disable the network to different extents, depending on where it is located. In the worst case, the entire network could be disabled. Hence, all hubs in the network must be very reliable, much more so than the nodes. In fact, the StarLAN specifications require hubs to have a mean time between failures (MTBF) of at least five years of continuous operation.

In a bussed (hubless) network, each node communicates by a bus transceiver, such as the Am7961, which is connected to the LAN controller. In the case of the Am7961, an RS-422–ECL converter is needed since its bus interface uses StarLAN-type RS-422 voltage levels. Insofar as the Am7961 is the only StarLAN bus transceiver on the market today, the need for a voltage converter could prove to be a nuisance. At least one chip will be needed to convert between RS-422 and TTL signals (e.g., TI's SN75109) while two chips will be needed for TTL-ECL conversion (e.g., Motorola MC10K124 for TTL-to-ECL, and MC10K125 for ECL-to-TTL). Hence, at least five chips are needed per node (LAN controller, bus transceiver, three voltage converters) if the Am7961 is used. Note that this figure does not include support chips that may be required by the LAN controller (it does, however, include a clock source since the Am7961 has a clock output). A node in a hub-based network would not require a bus transceiver, although such a device might be

useful if it has a jabber control function, as was mentioned previously. Another aspect of a bus topology serial network is that repeaters are required at each crate if ECL voltage levels cannot be used in the cables that connect the crates together. Even if ECL voltages could be used, repeaters would probably be needed anyway to avoid impedance mismatches between crate-connecting cables and the backplane. Repeaters are crucial elements in the operation of a bussed network. Even a single malfunctioning repeater can disable the entire network, depending on the nature of its defect. Just as the reliability of hubs is a matter of concern in a star-topology network, so is the case with the reliability of repeaters in a bus network. As mentioned in the introductory section on StarLAN, there are two ways for connecting crates together in a bussed StarLAN scheme. The first is a T connection while the second is a daisy chain. For either connection scheme, the task of attaching a crate to (or disconnecting it from) a bussed serial network seems to be less straightforward compared to that in a hub-based network.

4.2 Bit Rate

The decision on whether to use a bit rate (the rate in which data is transmitted and received) of 1 Mbps or 10 Mbps involves a trade-off between implementation complexity and network capacity. A higher bit rate will involve a probably more costly and complex hardware solution since higher clock speeds and faster circuits are involved compared to that for a lower bit rate. The higher bit rate may also result in a somewhat less robust operation in heavily loaded backplanes compared to a lower bit rate. There has to be a serious discussion among FASTBUS system 'experts' as to what data rates are expected for the proposed applications of the serial network. It is important to look not only into average capacity requirements but also into peak demand. Depending on the application, a 1 Mbps network may be sufficient during normal operations but may suffer

from unacceptable delays during peak transmission periods.

4.3 Access Scheme

StarLAN, being a CSMA/CD protocol, is inherently a random access system. There is no guarantee that a frame will be transmitted immediately after it is sent to the LAN controller. This may be a problem in some applications if, for example, a response must be received before a certain time limit. A deterministic access scheme may be required in such cases. StarLAN can still be used for a deterministic access system if higher-layer collision-avoidance protocols are used. Of course, it then becomes a valid question to ask if StarLAN is really suited for implementing the serial network in the first place.

Token-passing is one scheme that allows all nodes in the system to have guaranteed access to the network in a deterministic time interval. The performance characteristics of a token bus (IEEE 802.4) have been compared to a CSMA/CD standard (IEEE 802.3 Ethernet) in [21]. The results from that reference may also apply to the case of StarLAN as well since StarLAN is quite similar to Ethernet in many respects. In general, shorter delays are associated with CSMA/CD compared to token-passing (token bus) while token-passing achieves greater maximum throughput than CSMA/CD. In particular, the maximum throughput for a token bus is much less affected by the network propagation delay than is CSMA/CD. It is not known, however, just how much more complicated the serial network's implementation will become if token-passing is used instead of StarLAN although it is expected to be much more complex. A description of a protocol that effectively implements token-passing will be described later.

Another deterministic scheme is the master-slave method that is currently used in FASTBUS systems. An analogous system is described here for the serial network. At any point in time, the network is composed of slaves and, at most, one network-master. Only

the master node can initiate transmissions. The rest of the nodes in the network can only transmit in response to prompts or commands from the master. When the master has completed its operations, it relinquishes control of the network, after which other nodes can contend for network mastership. All contending nodes 'apply' for network mastership to the master arbitrator by transmitting 'request frames' using ordinary CSMA/CD to a centralized 'master arbitrator'. The arbitrator decides on which node becomes the next master (by a first-come-first-served rule, for example) and broadcasts the address of the next master. This scheme has two attractive features. Firstly, it can be programmed to assign network mastership to all nodes sequentially, thus effectively implementing a token-passing scheme if that is desired. Secondly, it acts as a network 'super master' that settles 'master disputes' where, because of a misunderstanding, more than one node thinks it is the network master, or when one or more non-master nodes initiate transmission without realizing that a master has been assigned control over the network. In such cases, the arbitrator can simply rebroadcast the address of the valid master to correct the erring nodes. However, the centralized arbitrator system is not robust for if anything goes wrong with the arbitrator, the entire master-slave scheme collapses.

One way out of this problem is to do away with the central arbitrator altogether. In this 'distributed' scheme, a node contends for network mastership by broadcasting a master-request frame and relinquishes its mastership by broadcasting a master-release frame. 'Master disputes' (as defined previously) are resolved by a pre-determined prioritization order that is known by each and every node. If a master finds that other nodes are initiating transmissions, it can rebroadcast a master-request frame to reassert its mastership. If it receives a master-request frame from a higher-priority node, then it loses its mastership immediately. To reduce transmission overhead, commands or prompts may be piggy-backed onto master-request frames. Likewise, single-frame responses can be piggy-backed on master-relinquish frames (In this case, the responding node relinquishes

the mastership of the prompting node. This is useful for simple command-response applications.).

Note that it is possible for starvation to occur in the two master-slave schemes described above just as starvation can occur in a FASTBUS system. It should also be noted for either the centralized or the distributed schemes that some time can elapse before a node gains complete mastership of the network (when there is only one master and all other nodes are aware of it). This can occur if the LAN controller of a node happens to be deferring transmission to a master request frame of another node. In this case, the deferring node will transmit after an IFS interval at the end of the master request frame because its transmission cannot be aborted immediately. The received request frame has to be processed first before the node finds out that it can no longer initiate transmissions. Application programs should make allowances for this possibility.

It is expected that some applications running in the FASTBUS Serial Network will involve mainly command-response–type operations (where the response has to be received within a time limit) and other operations which do not require deterministic access to the network. If that indeed is the case, then StarLAN, along with a higher-layer master-slave protocol, can be used to allow both groups of operations to coexist. A node will first have to contend for mastership of the network before proceeding with a time-critical operation. Operations that do not depend on a time limit can proceed (without having to reserve the network) during the intervals when network access is not restricted by a master. The arrangement described above takes advantage of the throughput and simplicity of a CSMA/CD scheme for random access applications, yet will allow deterministic access when required.

Chapter 5

Summary of Results and General Assessment

The following section is given below to summarize the major points of the three previous chapters.

- The prototype network was composed of a FASTBUS crate, a standard StarLAN hub, a StarLAN hub specially constructed for the FASTBUS backplane, and three PC compatible computers which accessed the backplane via RS-422–ECL tranceiver cards.

- Signal jitter and propagation delay in the backplane were measured for different configurations of capacitive loads. The largest amount of signal jitter was measured to be 6 ns while the longest propagation delay was measured to be 16.2 ns. Both of these values fall well within the StarLAN specifications of 62.5 ns for jitter and 4 bit times for node-hub / hub-hub propagation delay.

- Data communications were established by the three PCs through the prototype network without any problems whatsoever. The crate hub was successfully operated either as an intermediate hub (where the standard hub was used as the header hub) or as a header hub. The general behaviour of data transmissions in the three-node prototype network were similar to those for an orthodox StarLAN network, indicating that the FASTBUS hub level appears as just another StarLAN hub level to the rest of the network. It also indicates that the bus topology of a FASTBUS hub level does not cause the network behaviour to differ from that of a star topology.

- Performance measurements of a three-node orthodox StarLAN network show some interesting results. The length of the transmitted frame and the preparation time for frame transmission can have a profound effect on network performance. In particular chaotic behaviour and reduced reliability were observed if the frame transmission time is equal to or greater than the preparation time.

Most of the work for the StarLAN-based serial network has been done as far as the physical and media access control layers are concerned. An implementation of the bussed StarLAN scheme is one area of work that has yet to be done. At this stage however, there does not seem to be an overwhelming advantage in using bussed StarLAN as opposed to standard StarLAN to implement the serial network. Protocols for the higher layers have yet to be developed and are expected to depend very much on what FASTBUS users want to use the serial network for in the first place.

The prototype network was built without much difficulty at all and this shows a certain amount of inherent 'compatibility' between StarLAN and the FASTBUS Serial Network. The work on the prototype network has shown that StarLAN is an effective, simple, affordable and easily adaptable standard to implement the FASTBUS Serial Network. The onus should now be on the detractors, if there are any, to make their case against a serial network based on StarLAN.

Appendix A

PALASM Listing, Notes on Transceiver and Hub Circuits

A.1 PAL Device Logic Equations

A listing of the PAL programming program is given below in figure A.55. The logic equations are equivalent to the programming information given in [9]. A fuse plot and results from simulations produced from the PALASM2 program have not been included. The simulations essentially show that the equations are correct while the fuse plot shows connections in the logic device that were specified by the equations.

A.2 Functional Description of the StarLAN-FASTBUS Transceiver

The following section explains how the transceiver circuit of chapter 2 works (see figure 2.24). The transceiver is composed of two subcircuits: one for signals in the downstream direction (into the backplane) and another for signals in the upstream direction (from the backplane).

U6 and U7 are used for clocking the two subcircuits. RS-422 signals from the upstream source appear at lines UDP and UDN. R1 is an impedance-matching resistor, nominally 100 ohms for twisted-pair telephone wiring. T1 is a dual transformer used for DC isolation of the differential RS-422 signals. U2 is an RS-422 receiver that converts RS-422 signals to TTL. Resistors R2 to R5 are used for voltage squelch (as specified by StarLAN). OUTA of U2 is the unsquelched signal while OUTD has the squelched signal which is used by U4, U7, and U9 to filter the signal from OUTA. U9 also doubles as a TTL to

```
TITLE  Translated PAL program for Intel StarLAN Design Kit.
PATTERN StarLAN.pal
REVISION 2
AUTHOR (orig., Adi Golbert, Intel; translated by Richard Cam)
COMPANY Intel, TRIUMF
DATE    22 May 1987
CHIP    Multifunction PAL16L8
A9NANDA8 A7     A6    A5    A4    IOWR   A0   AEN  REQ1  GND
REQ0     LDPORT RESET DACK3 DACK1 DREQ3 DREQ1 CS  BUSEN  VCC
EQUATIONS
/CS = /AEN * /A9NANDA8 * /A7 * /A6 * /A5 * /A4 * /A0
/LDPORT = /AEN * /A9NANDA8 * /A7 * /A6 * /A5 * /A4 * A0 * /IOWR
/DREQ1 = /REQ0 * /DACK1 + /REQ0 * /DREQ1 + RESET
/DREQ3 = /REQ1 * /DACK3 + /REQ1 * /DREQ3 + RESET
/BUSEN = /DACK1 + /DACK3 + /AEN * /A9NANDA8 * /A7 * /A6 * /A5 * /A4
;/CS.TRST = VCC
;/LDPORT.TRST = VCC
;/DREQ1.TRST = VCC
;/DREQ3.TRST = VCC
;/BUSEN.TRST = VCC
```

Figure A.55: Logic Equations for PAL device of Intel StarLAN board.

ECL translator that drives the Serial Transmit line of the FASTBUS backplane.

ECL signals from the Serial Receive line of the backplane are converted to TTL by U8 and from TTL to RS-422 by U1, an RS-422 driver. Capacitors C2 and C3 are used for trapezoidal modulation (they increase the slew rate of RS-422 signals to reduce rf emissions from unshielded telephone cable. Two JK flip-flops (U5) are used to implement a divide by 4 circuit whose output is used to clock U3, a shift register which is used to enable the output of U1. U3 disables the output of U1 at the end of a StarLAN transmission (when a high logic level is detected for approximately 2 microseconds). The other capacitors are used for power supply bypass purposes. They reduce the magnitude of supply voltage spikes that occur when TTL devices change states.

A.3 Functional Description of the StarLAN Hub Circuit

This section describes how the StarLAN crate hub of chapter 2 works (see figure 2.28). Note that there are two corrections to the original schematic diagram (dated 30 July 1987). First, a 100-ohm resistor across pins 10 and 8 of T1 is missing. Second, the SER-TX (A58) and SER-RX (A57) lines are interchanged (A58 should connect to U5 while A57 should connect to U6).

ECL signals from the Serial Transmit line (going in the upstream direction) are converted to TTL by U6 whose output is fed into U4, the hub controller. The output and output enable pins of U4 are connected to U3, a StarLAN driver/receiver, which converts the TTL output to RS-422 and drives the telephone cable via an isolation transformer, T1. RS-422 signals from the telephone cable (in the downstream direction) are isolated by T1 and filtered by an RCL circuit which has a cut-off frequency of approximately 2 MHz. The filtered signal is then converted to TTL by U3 (which has a built-in squelch circuit). The signal and squelch output of U3 is connected to the upstream port of U4.

The retimed output is then converted to ECL by U5, which drives the Serial Receive line of the backplane. U5's output is enabled by a high logic level through R7. U2 is used to clock U4. A debounced switch (to reset U4) is implemented by S1, R5, D1, C1, and U1. U4 has three outputs for driving LED's (pins 15, 16, and 19) and these are buffered by U1, which drives the LED's through dropping resistors R2 to R4. R1 and DS1 implement a power-on indicator. The remaining capacitors are used for bypassing the -5.2V and 5.0V power supply lines.

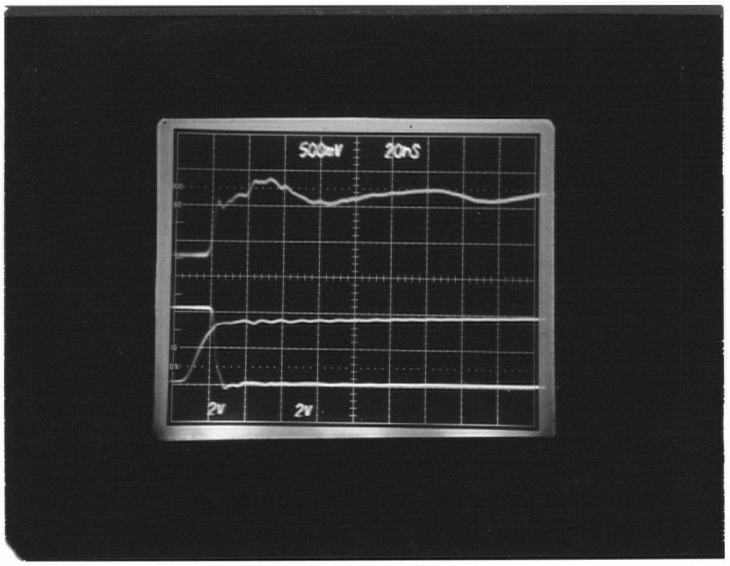
Appendix B

Photographs and Table of Electrical Measurements

Measurements of propagation delay and pulse jitter are tabulated in table B.1. The corresponding oscilloscope photographs are shown in figures B.56 to B.73.

Reference Number	*Propagation Delay (nsec.)*		*Pulse Jitter (nsec.)*
	first edge	*second edge*	
7	11.6	9.4	2.2
8	7.8	9.4	1.6
9	6.2	7.6	1.4
10	9.2	8.4	1.2
11	7.0	6.2	0.8
12	9.4	9.6	0.2
13	12.2	12.2	0.0
14	14.8	12.6	2.2
15	10.8	10.2	0.6
16	9.2	9.6	0.4
17	14.6	12.4	2.2
18	13.2	7.2	6.0
19	13.8	12.8	1.0
20	9.6	9.0	0.6
21	16.2	16.0	0.2
23	7.8	6.8	1.0
24	12.8	12.2	0.6
25	15.2	14.2	1.0

Table B.1: Propagation delay and pulse jitter measurements.



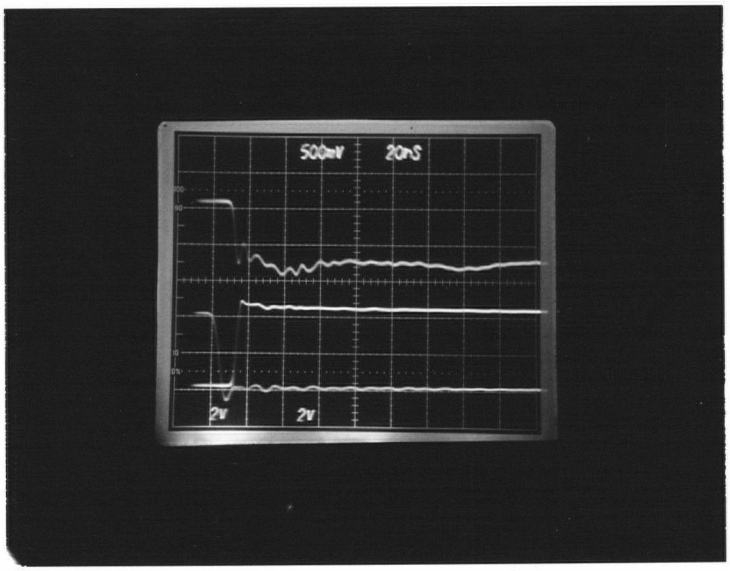
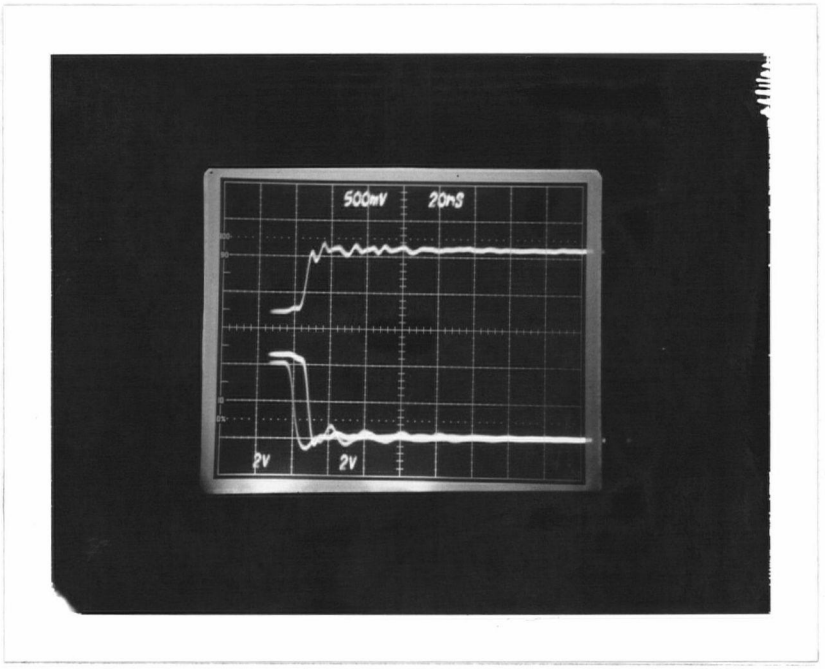


Figure B.56: Reference number 7



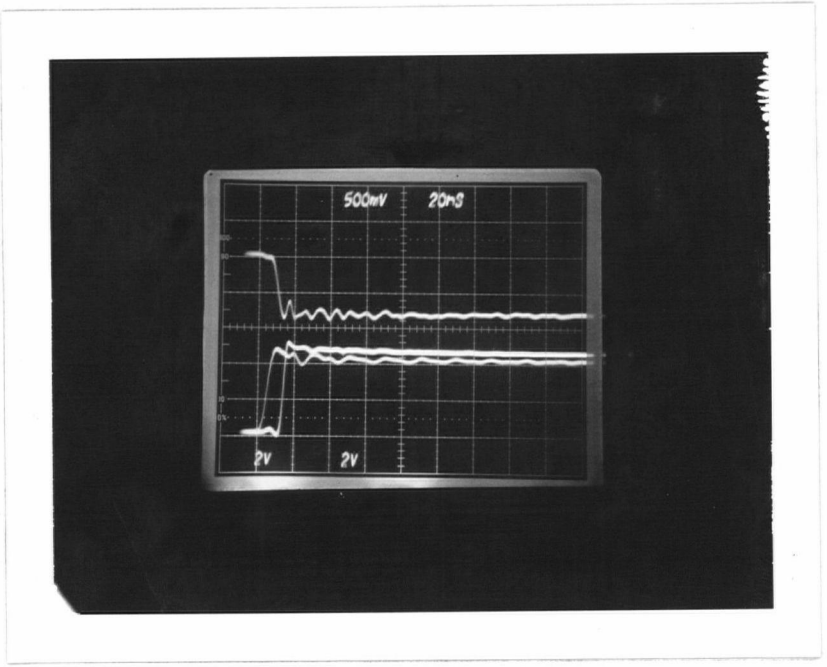
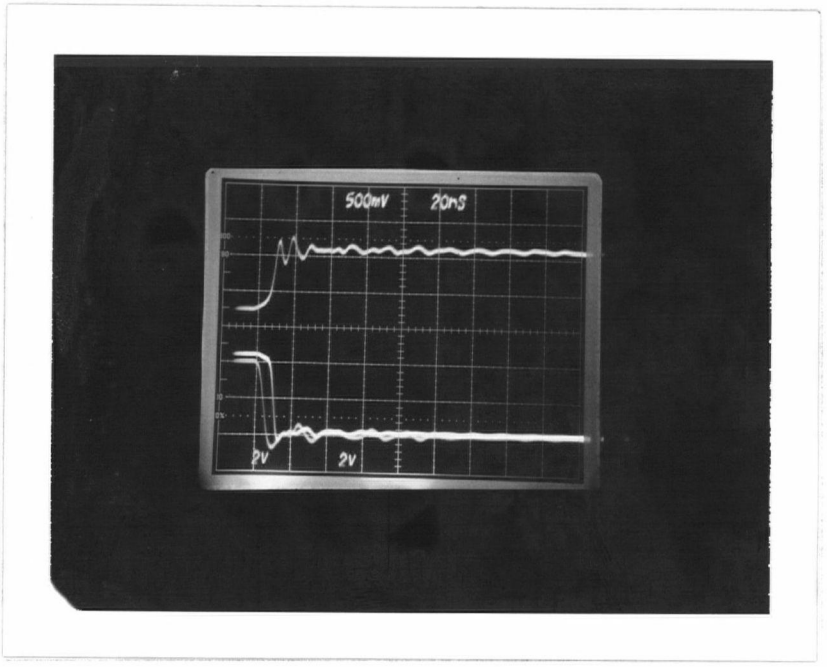


Figure B.57: Reference number 8



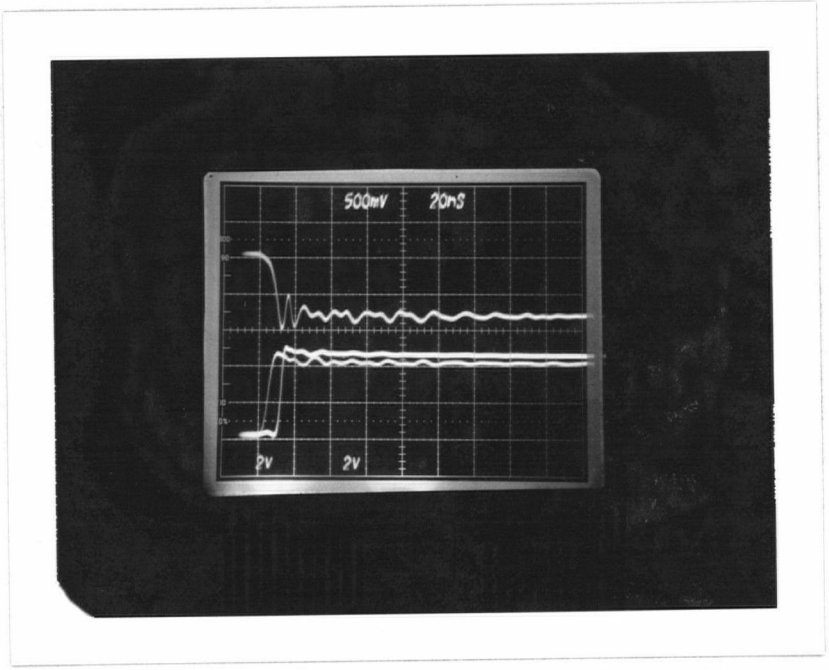
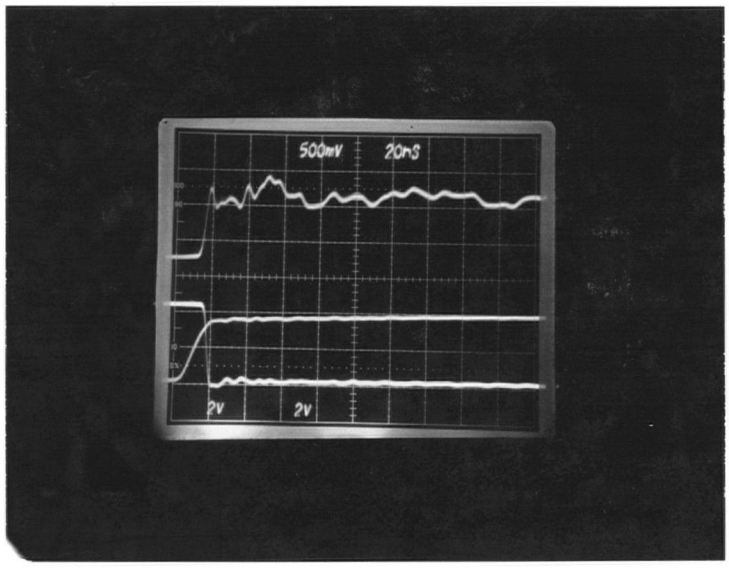


Figure B.58: Reference number 9



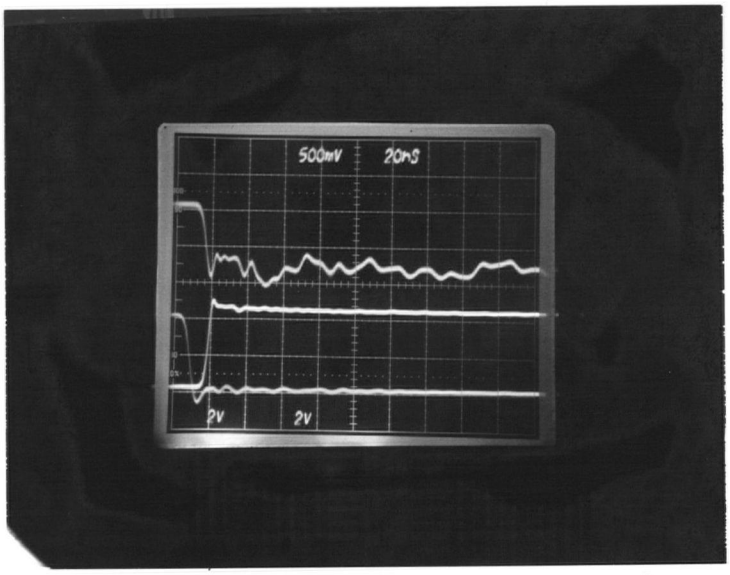
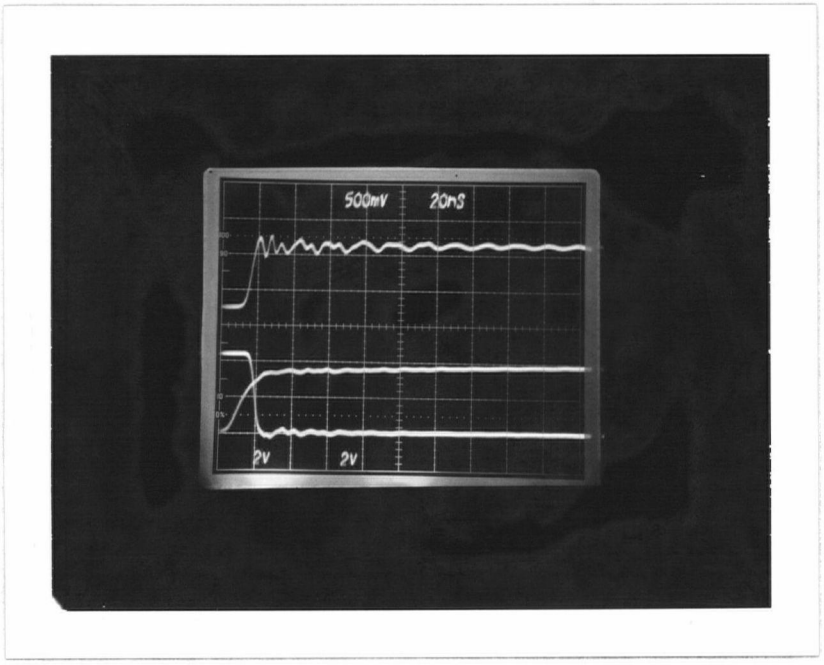


Figure B.59: Reference number 10



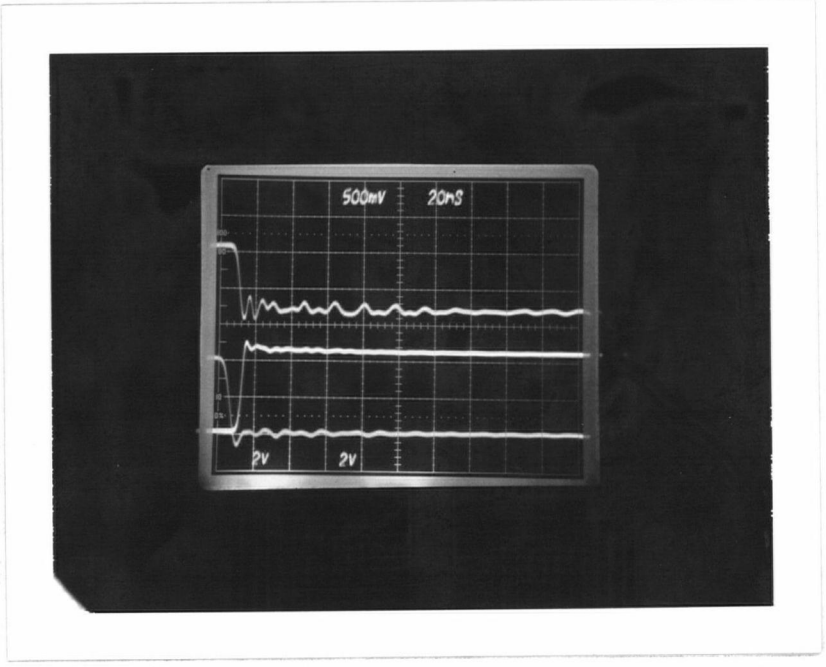
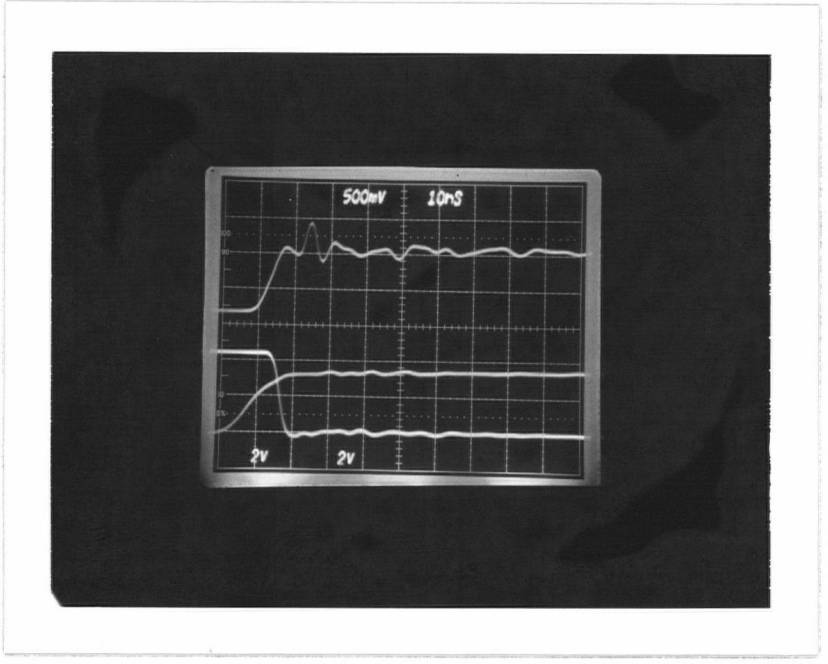


Figure B.60: Reference number 11



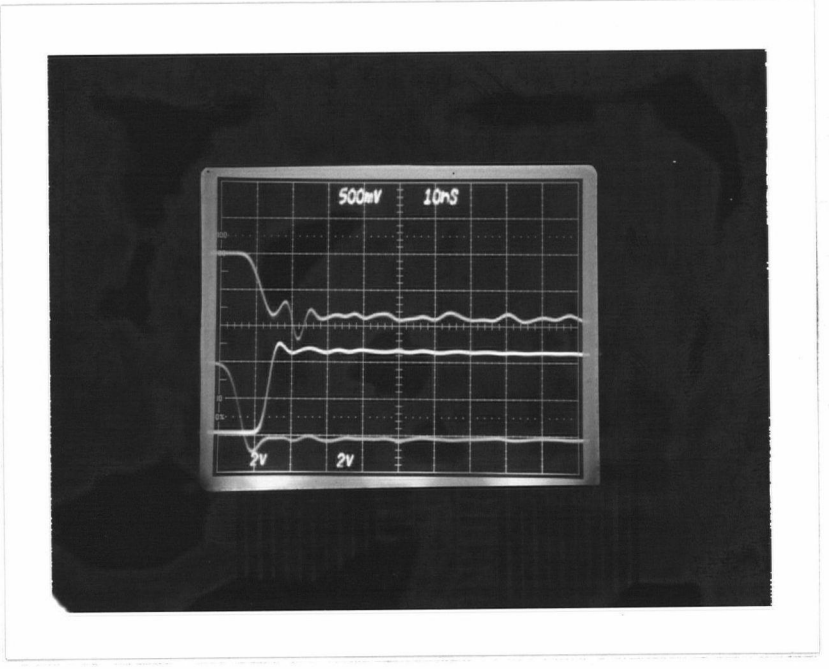
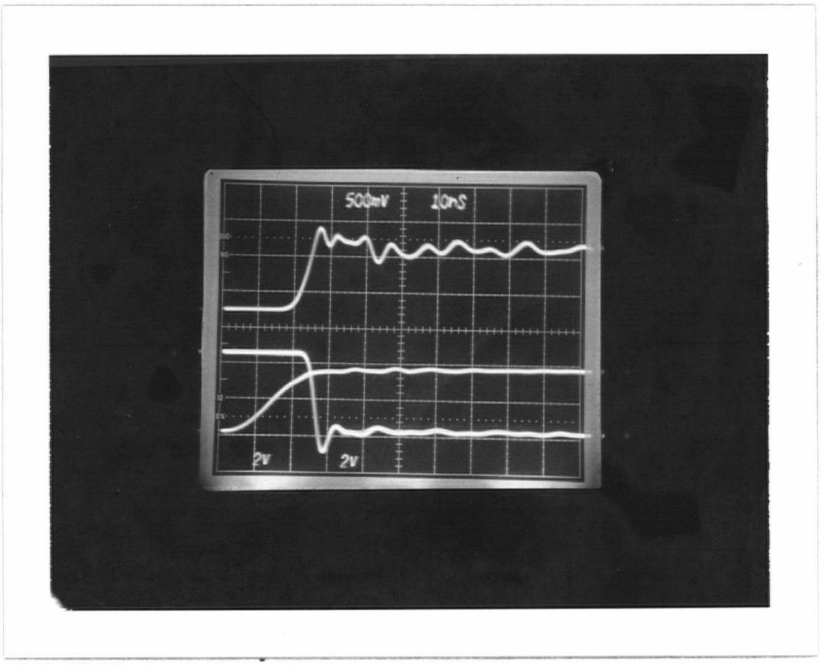


Figure B.61: Reference number 12



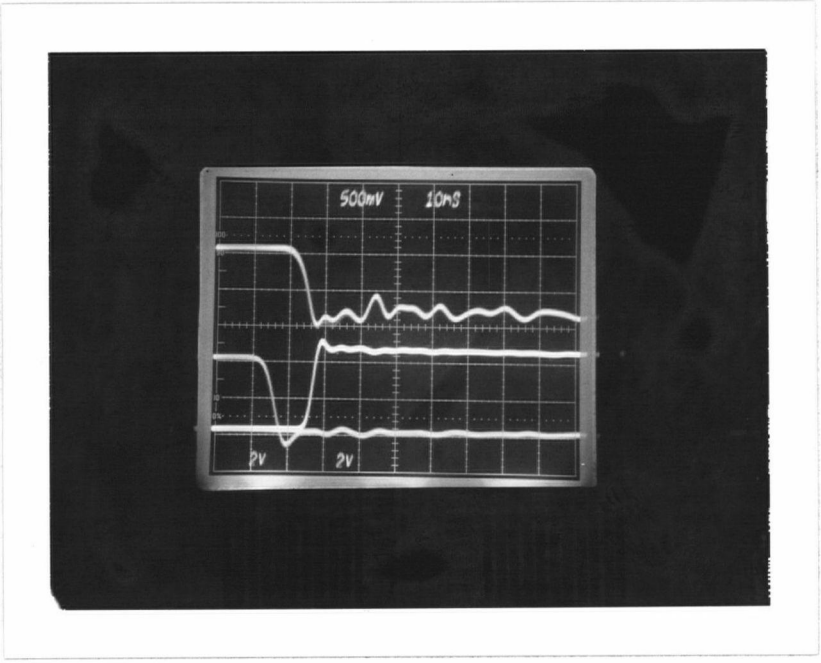
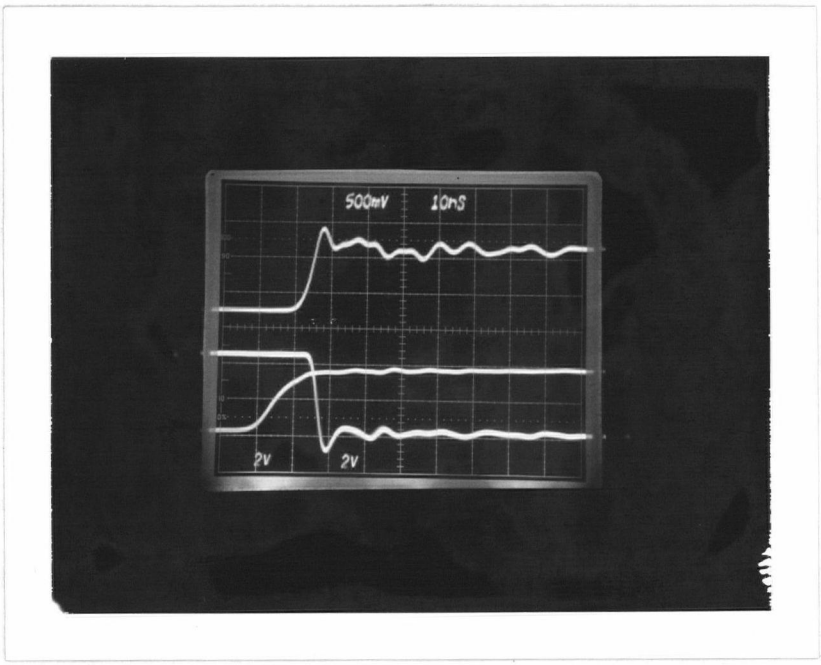


Figure B.62: Reference number 13



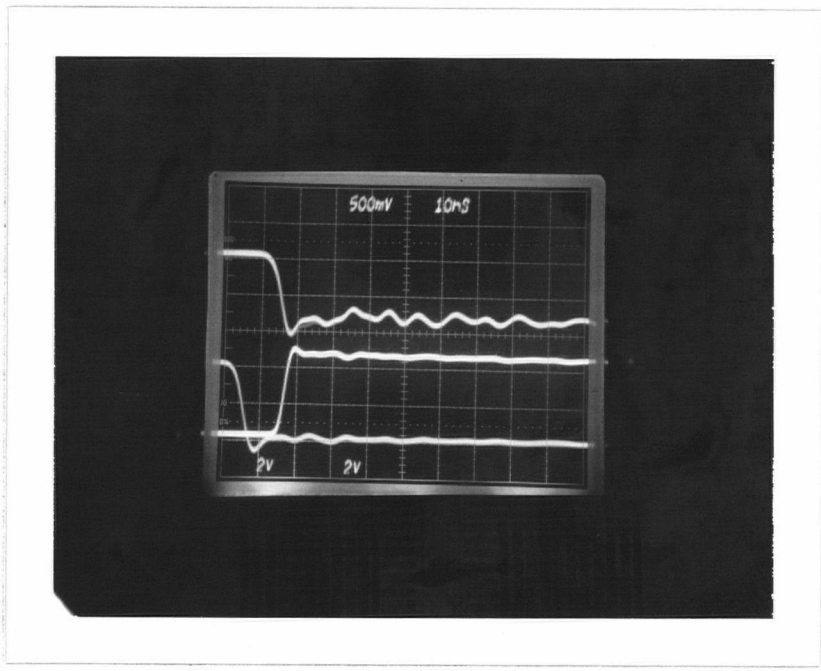
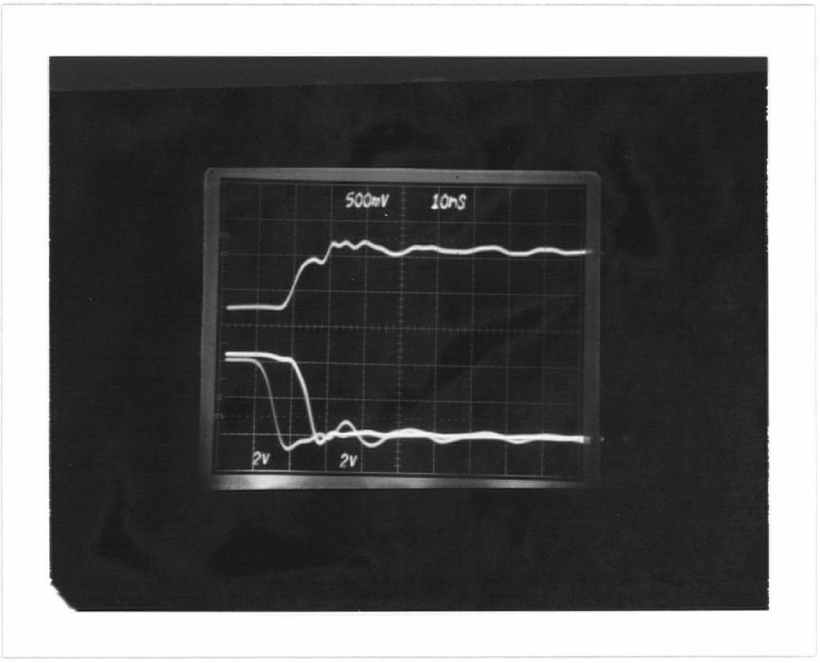


Figure B.63: Reference number 14



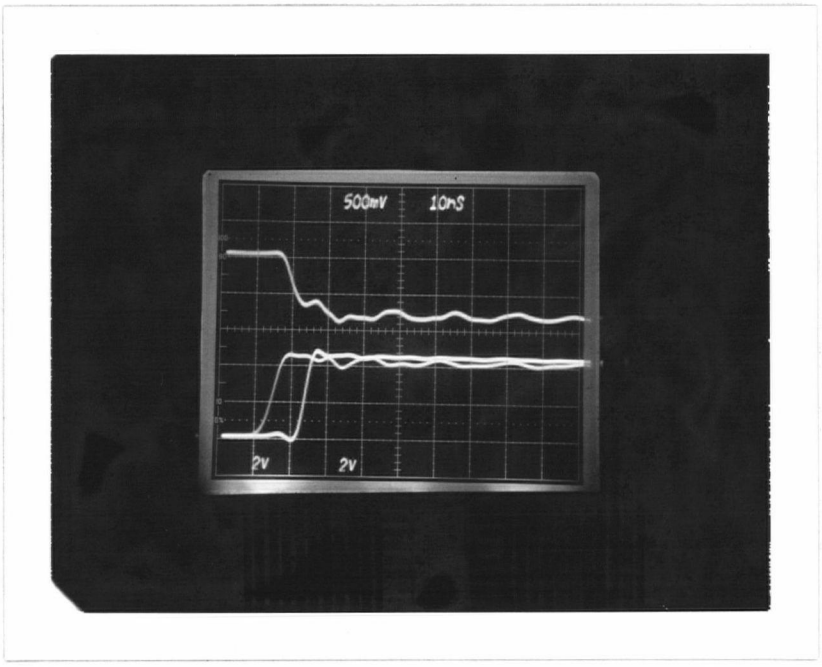
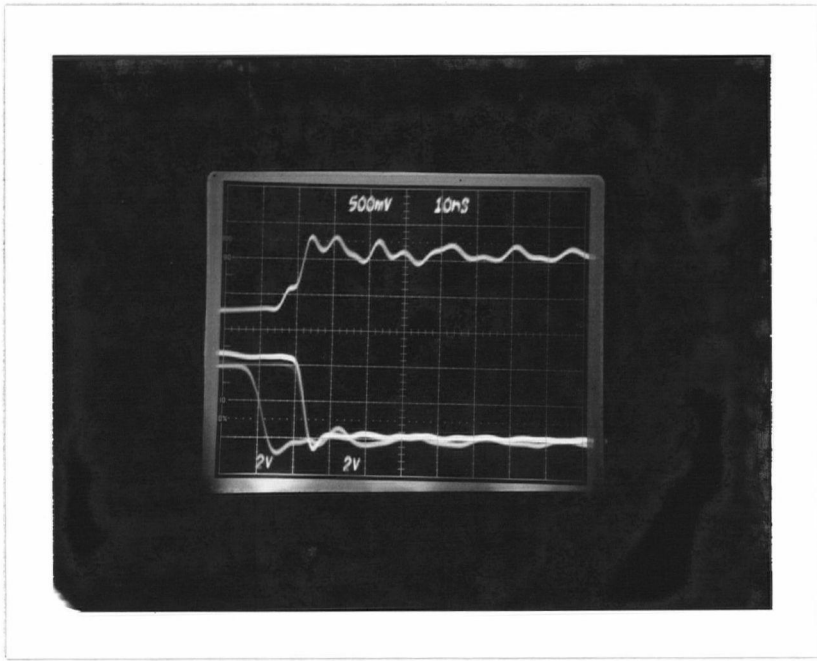


Figure B.64: Reference number 15



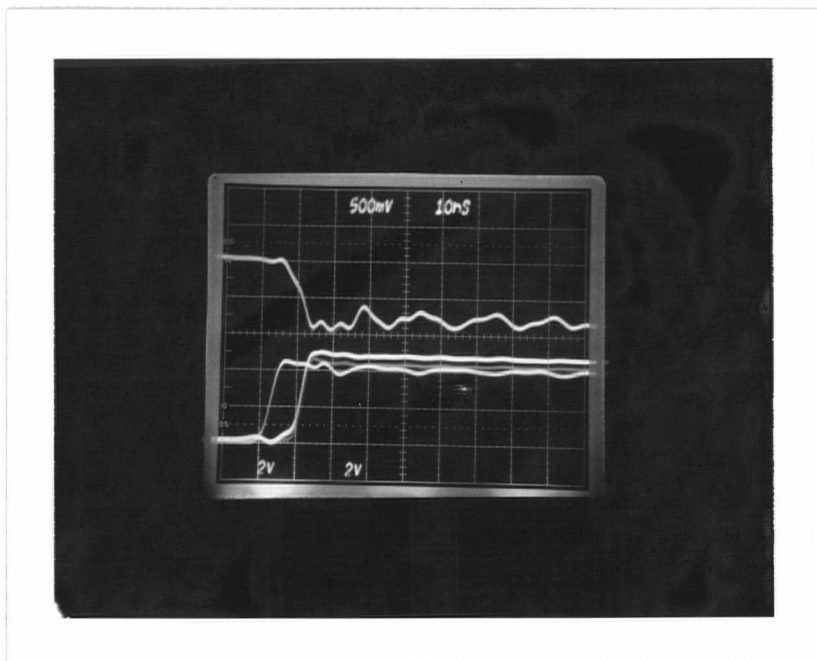
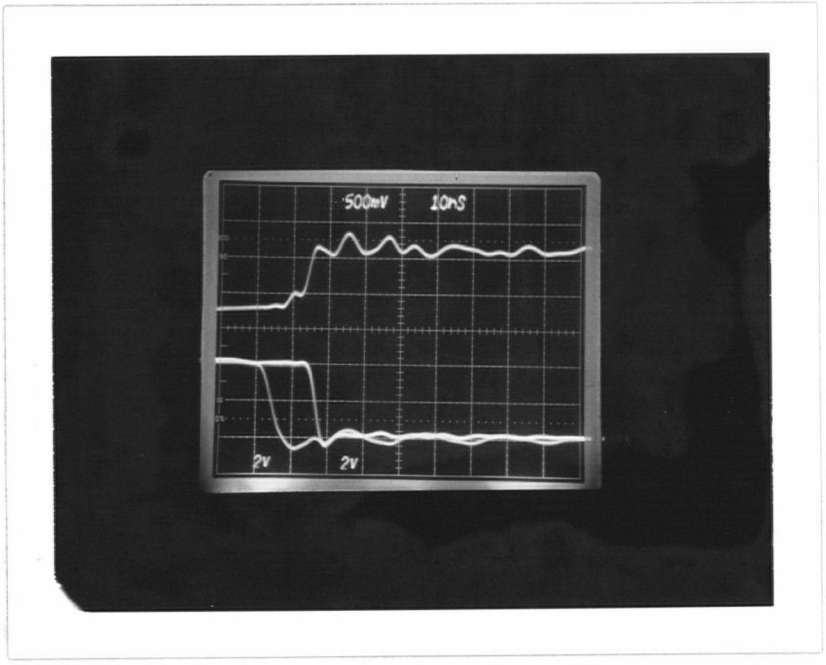


Figure B.65: Reference number 16



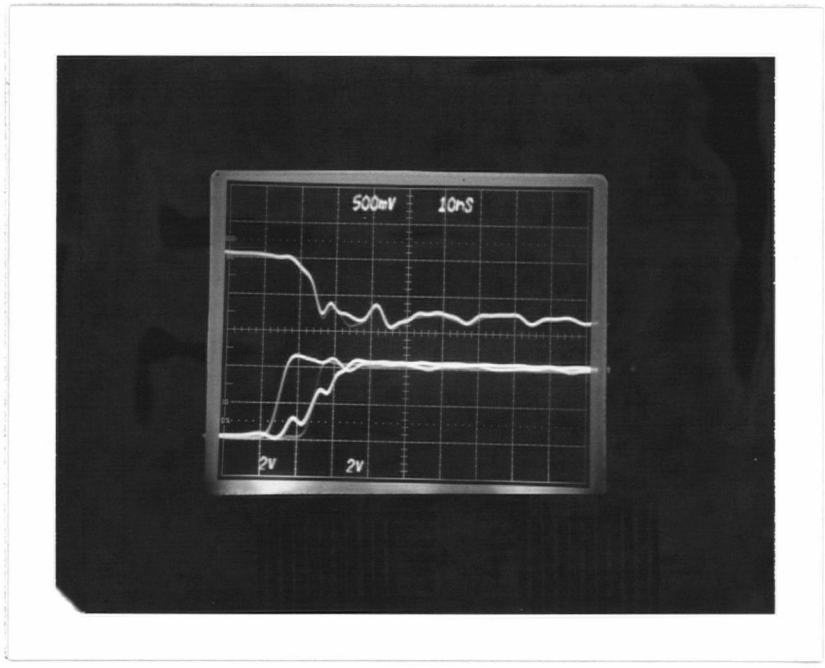
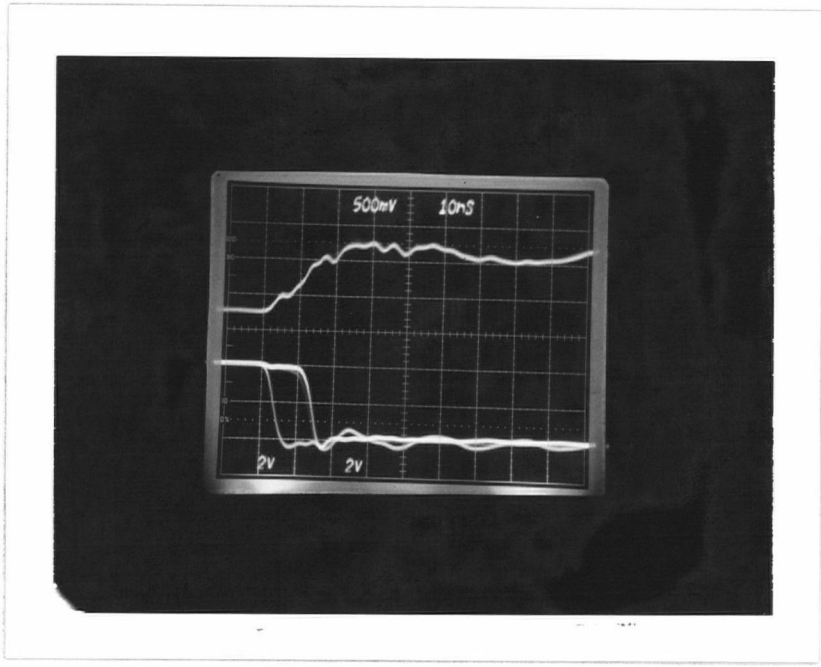


Figure B.66: Reference number 17



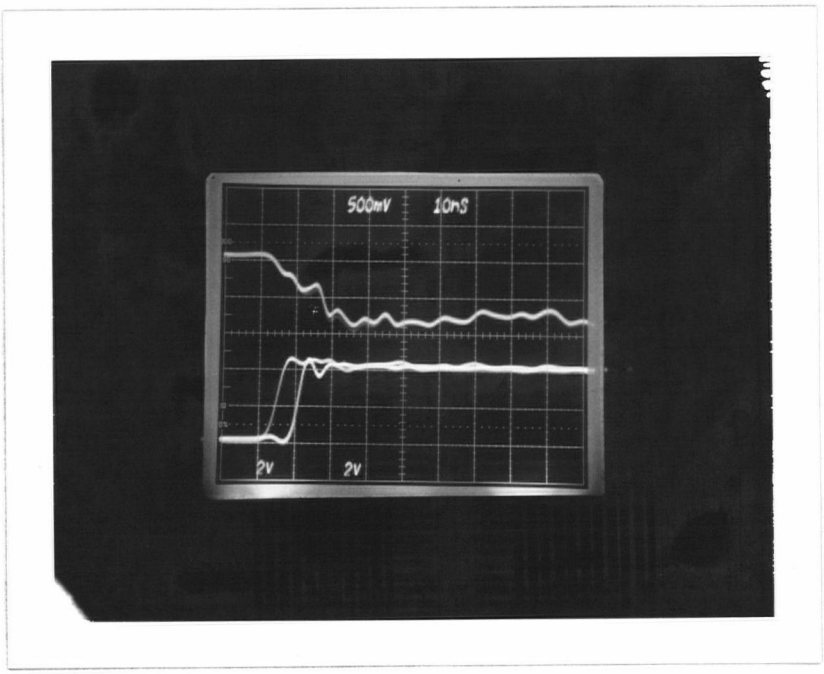
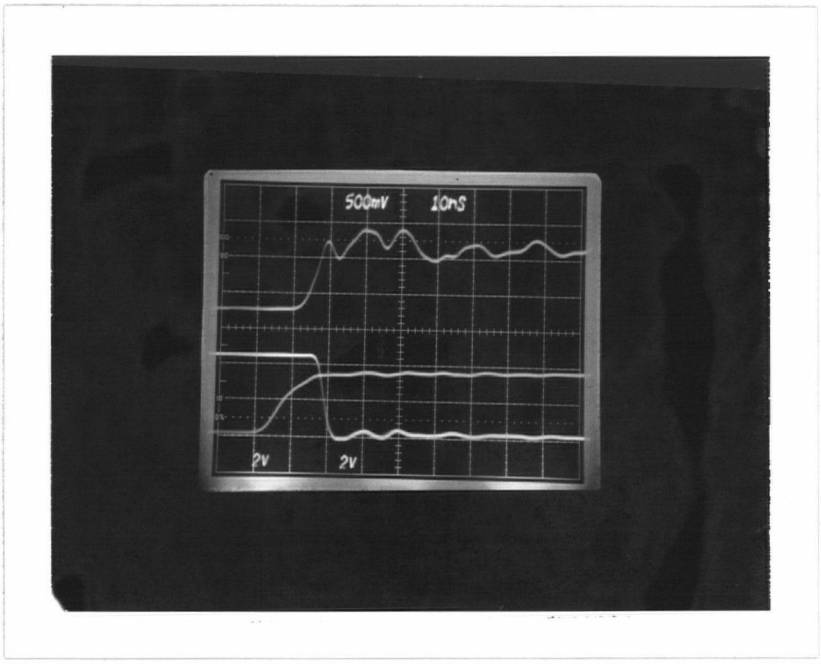


Figure B.67: Reference number 18



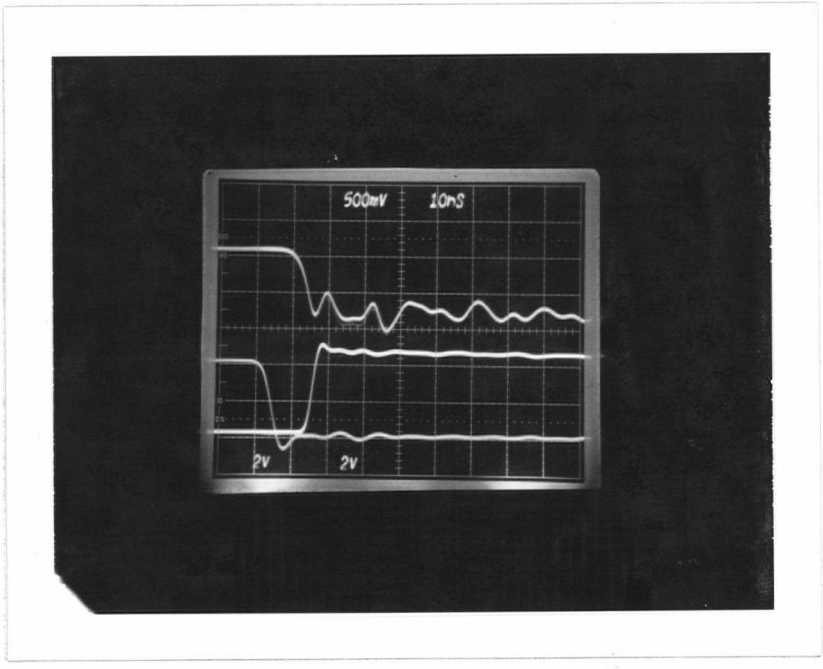
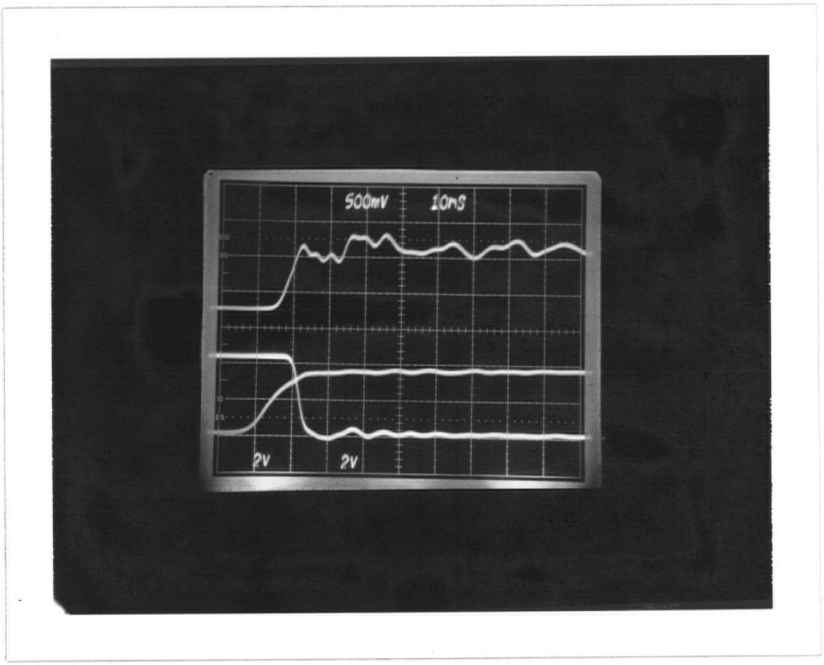


Figure B.68: Reference number 19



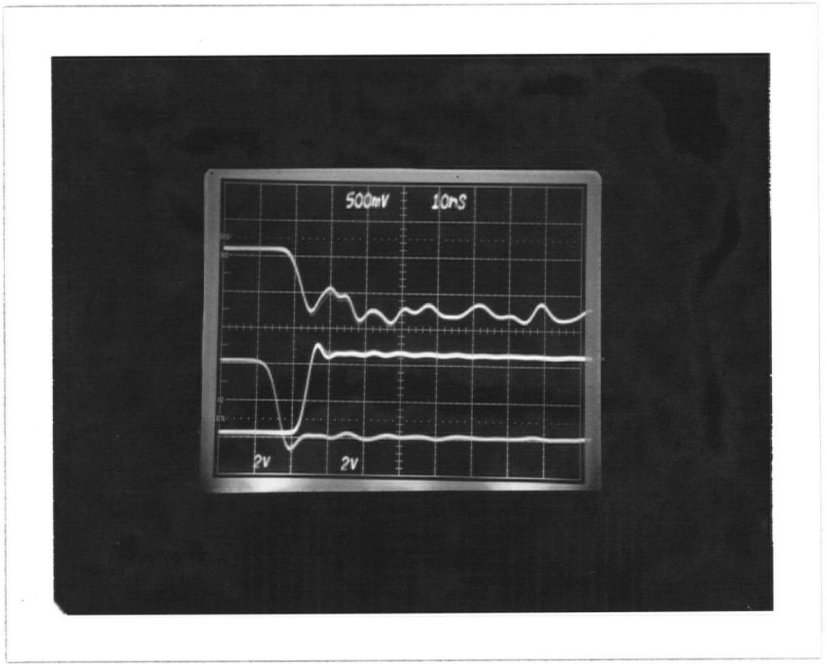


Figure B.69: Reference number 20

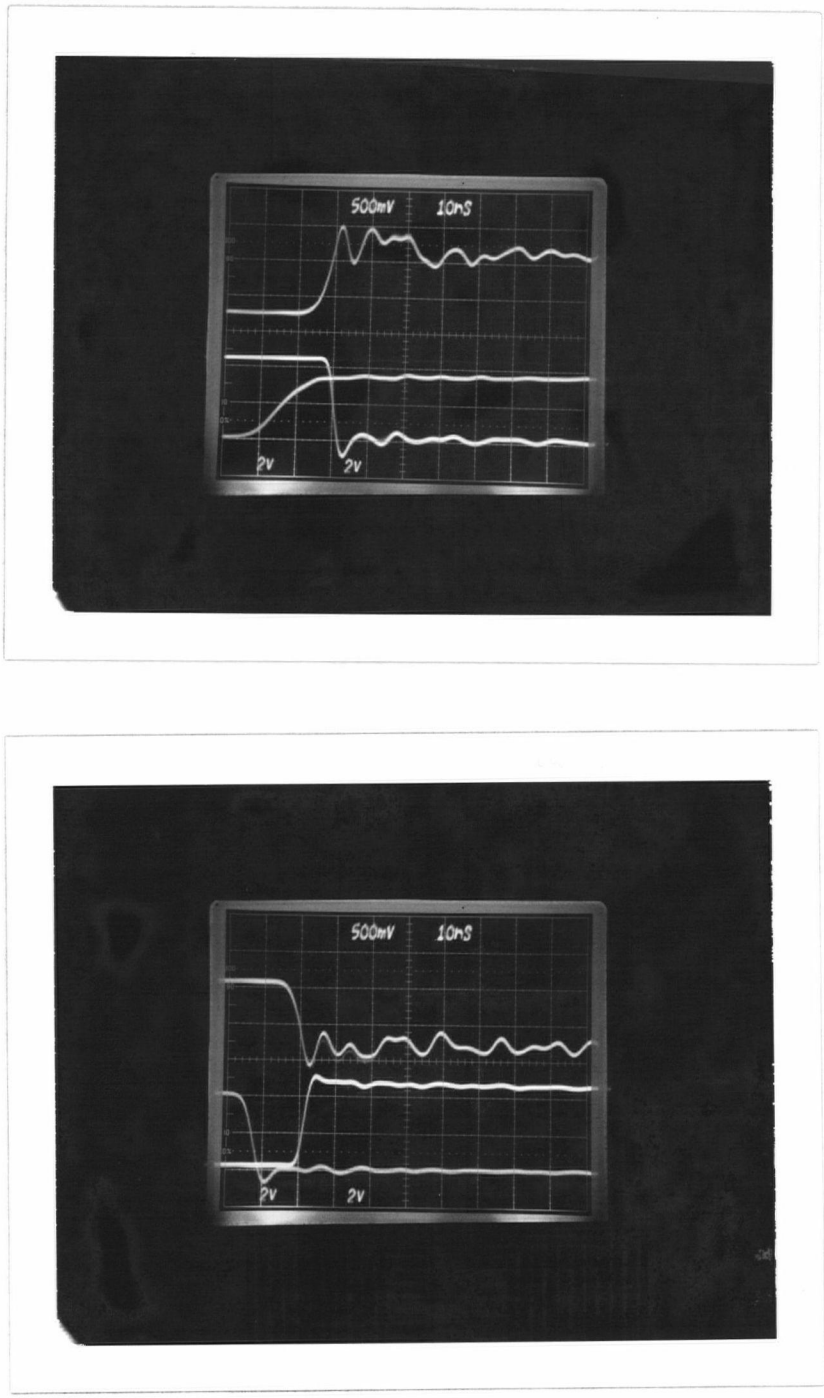
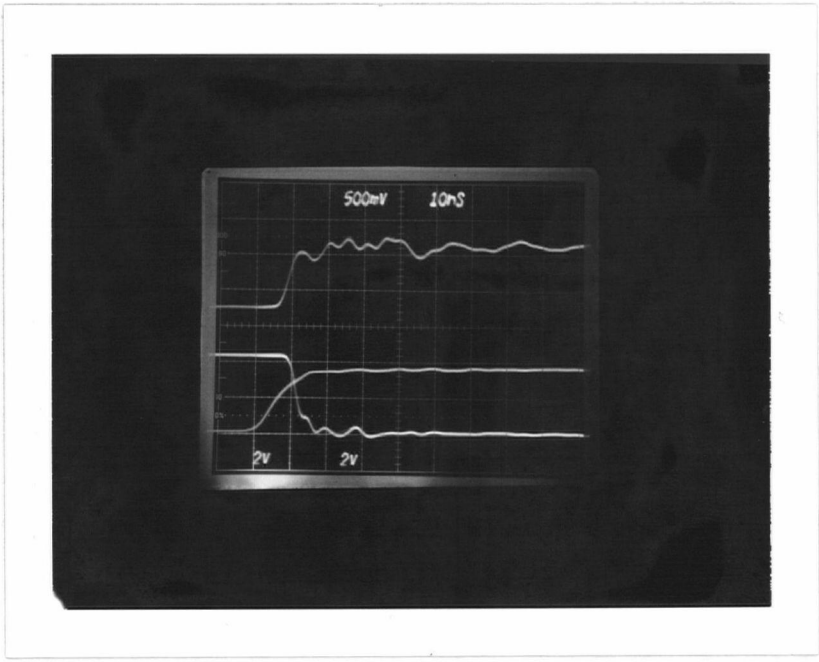


Figure B.70: Reference number 21



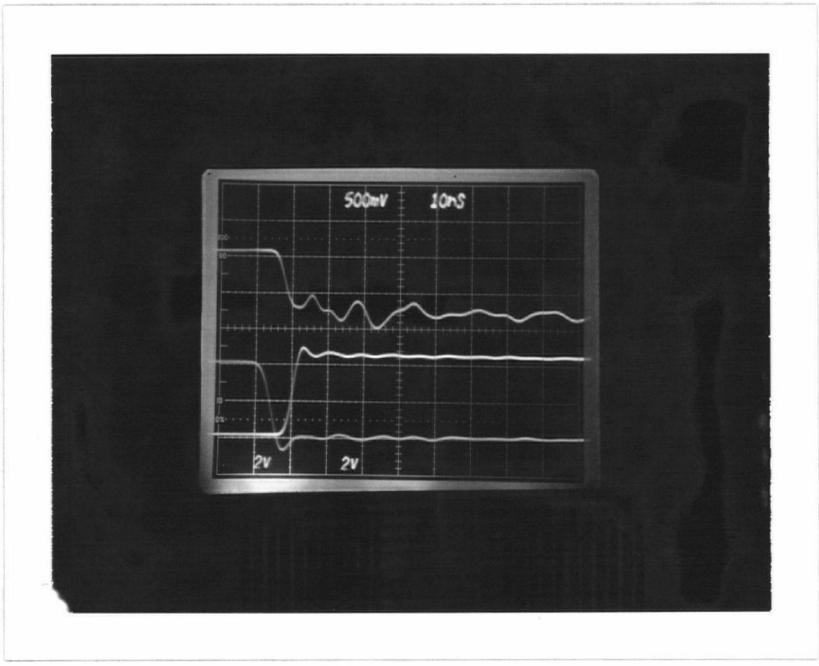
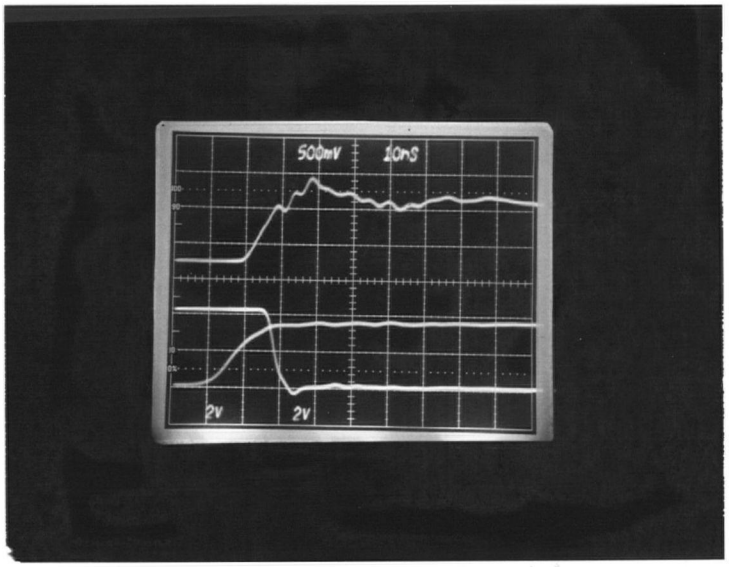


Figure B.71: Reference number 23



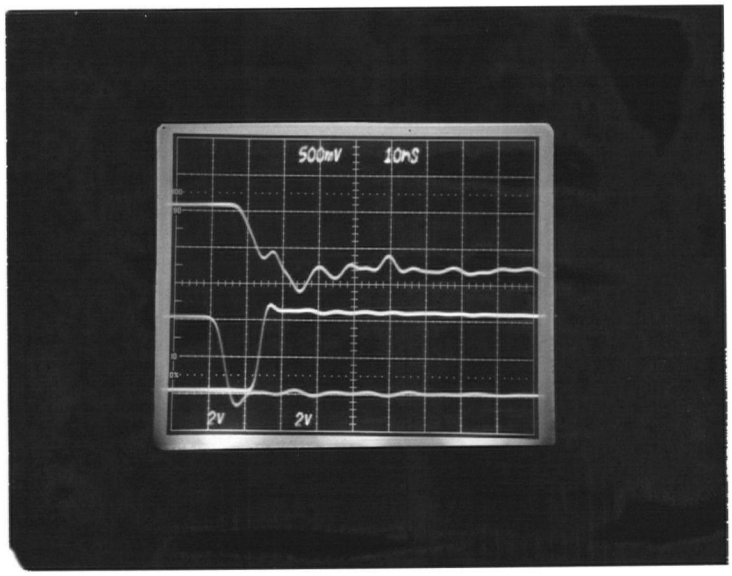
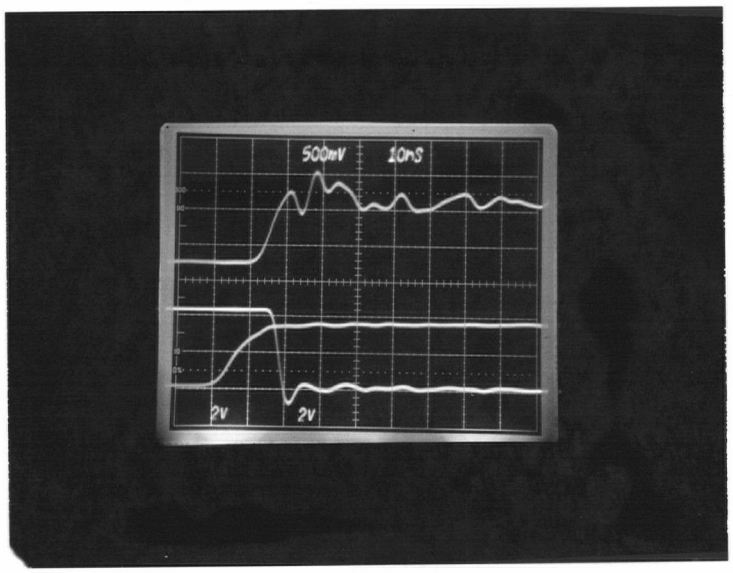


Figure B.72: Reference number 24



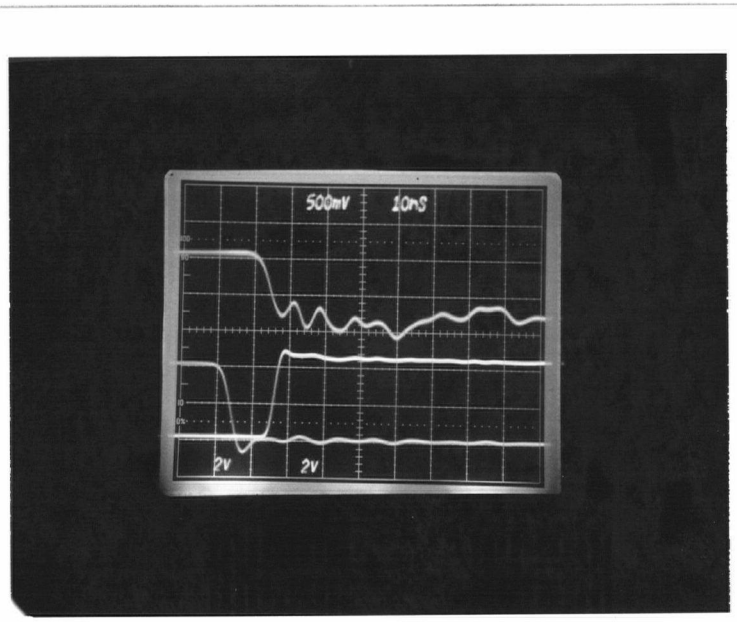


Figure B.73: Reference number 25

Appendix C

Tables of Preliminary Performance Data

This appendix contains tables of preliminary performance data that was obtained with the Intel program. Three PC compatibles were used to drive the network. Tables C.3, C.4, and C.5 show general transmission statistics for three frame lengths while the detailed collision data are shown in tables C.6, C.7, and C.8. All numbers are in base 10. The 'mode' field code is used to identify the network configuration of the data and is described below in table C.2. The data show that each PC behaves similarly under different network configurations. There are only minor differences in performance, and they should not be a cause for concern at all.

Mode	*Description*	*Figure*
A	crate header hub	2.23c
B	crate intermediate hub with Retix HB-6 header hub	2.23c
C	simple repeater	2.23b
D	orthodox StarLAN network	2.23a

Table C.2: Network configurations for preliminary performance measurements.

PC	Mode	*Total tx*	*Collisions*	*Good tx*	*Deferred tx*
PC1	A	155904	1727	154177	142438
PC1	B	140551	1861	138690	127256
PC1	C	155606	1733	153873	141697
PC1	D	139762	1826	137936	126490
PC3	A	220139	1318	218821	217307
PC3	B	218546	1318	217228	215763
PC3	C	220886	1345	219541	218062
PC3	D	221227	1369	219858	218462
PC4	A	89026	2201	86825	75860
PC4	B	103543	2127	101416	89727
PC4	C	89939	2184	87755	77257
PC4	D	105835	2153	103682	93044

Table C.3: General transmission data for 72-byte frames.

PC	Mode	*Total tx*	*Collisions*	*Good tx*	*Deferred tx*
PC1	A	26197	1460	24737	21850
PC1	B	25747	1564	24183	21359
PC1	C	26326	1540	24786	21835
PC1	D	26715	1480	25235	22337
PC3	A	25245	1489	23756	23044
PC3	B	24651	1542	23109	22416
PC3	C	25175	1527	23648	22859
PC3	D	25599	1472	24127	23368
PC4	A	25946	1519	24427	21542
PC4	B	27048	1452	25596	22684
PC4	C	25972	1529	24443	21585
PC4	D	25356	1568	23788	20695

Table C.4: General transmission data for 532-byte frames.

PC	Mode	*Total tx*	*Collisions*	*Good tx*	*Deferred tx*
PC1	A	10181	1398	8783	7260
PC1	B	10321	1401	8920	7400
PC1	C	9916	1465	8451	7021
PC1	D	10532	1436	9096	7560
PC3	A	9583	1463	8120	7474
PC3	B	9903	1439	8464	7766
PC3	C	9885	1407	8478	7756
PC3	D	9881	1431	8450	7731
PC4	A	10417	1415	9002	7478
PC4	B	10032	1484	8548	7041
PC4	C	10456	1430	9026	7456
PC4	D	9828	1487	8341	6917

Table C.5: General transmission data for 1526-byte frames.

PC	Mode	*Collision Resolution Interval (number of retransmission attempts)*															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Failed
PC1	A	698	100	11		1		1									49
PC1	B	663	67	9	2	1											64
PC1	C	687	91	8	2												52
PC1	D	610	84	8													64
PC3	A	962	44	25	8	14	8	5	1								
PC3	B	979	51	27	8	8	4	5	2								
PC3	C	1030	51	25	9	7	5	4		1							
PC3	D	1024	51	38	6	4	3	6	2	1							
PC4	A	441	43	7	3	2										1	101
PC4	B	499	67	14	3	1						1					89
PC4	C	408	44	9	2	1	1						1			1	101
PC4	D	506	56	17	8	2	1						1				89

Table C.6: Distribution of Collision Resolution Interval for 72-byte frames.

PC	Mode	*Collision Resolution Interval (number of retransmission attempts)*															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Failed
PC1	A	452	15	23	15	7	5										49
PC1	B	487	17	31	14	10	2										52
PC1	C	458	19	27	16	9	1										53
PC1	D	489	14	15	20	11	4	1									47
PC3	A	469	14	26	14	9	1	1									50
PC3	B	451	22	12	25	8	3	3									52
PC3	C	484	23	22	20	11	2										49
PC3	D	480	22	22	15	10	1								1		47
PC4	A	502	18	19	21	10	1										49
PC4	B	474	17	21	19	10	2	1									46
PC4	C	495	23	32	13	9						1					49
PC4	D	450	17	26	13	12	5										54

Table C.7: Distribution of Collision Resolution Interval for 532-byte frames.

PC	Mode	*Collision Resolution Interval (number of retransmission attempts)*															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Failed
PC1	A	378	21	2	12	17	11	3									47
PC1	B	361	23	1	9	20	11	3	2								47
PC1	C	382	25	1	5	22	11	6	3								48
PC1	D	407	22	5	12	18	7	10	2								44
PC3	A	357	20	2	11	21	16	3			1						49
PC3	B	404	30	1	6	25	12	1	1								46
PC3	C	351	22	3	7	20	13	6	1			1					46
PC3	D	397	19	4	14	20	9	2	1								47
PC4	A	391	23	1	11	17	14	6	1			1					45
PC4	B	366	23	1	11	17	14	6									50
PC4	C	390	18	2	11	18	14	4									47
PC4	D	353	21	2	19	15	9	3	2				1				52

Table C.8: Distribution of Collision Resolution Interval for 1526-byte frames.

Appendix D

Calculation of Software Overhead Time

Software overhead time is defined as the time required to prepare a frame for transmission. For a system that uses an Intel 82588 LAN controller, several operations have to be done before a frame can be transmitted by the LAN controller. These include, as a minimum requirement, initialization of a DMA controller (for data transfer to the LAN controller) and sending a TRANSMIT command to the 82588. The software overhead time per frame transmitted, t_{oh}, can be estimated by computing the difference between the total time to prepare and transmit a frame, t_{tot}, and the time required to transmit a frame through the network (by the LAN controller), t_{tx}.

$$t_{oh} = t_{tot} - t_{tx}$$

The frame transmission time, t_{tx}, in seconds per frame, is given by

$$t_{tx} = \frac{L + I}{C}$$

where L is the frame length in bits, I is the interframe spacing (96 bits), and C is the channel bit rate (1 Mbps for StarLAN). The total transmission time, t_{tot}, in seconds per frame is given by

$$t_{tot} = \frac{L}{S}$$

where S is the measured throughput of the transmitter.

This calculation does not take into account the time involved in deferring transmission or in resolving a collision. Hence, it can only be used for data obtained from a single-node

network where there is no contention for channel use. The overhead time for different information field lengths is given in table D.9. (to obtain the frame length, add 26 bytes to the *field* length).

The average overhead time was calculated to be 5632 microseconds, which can be converted to the field length of a frame that would take roughly the same amount of time to transmit. For a 1 Mbps channel, this would correspond to a frame whose information field length is 678 bytes $((678+26\,bytes)(8bits/byte)/(1Mbps) = 5632microseconds)$. The longest overhead time was 5661 microseconds while the shortest was 5596 microseconds. These values correspond to transmission times for frames with field lengths of 682 and 674 bytes respectively. So, the mean field length is bound to within approximately 4 bytes from the data obtained.

Field Length (bytes)	*Throughput*	t_{oh} *(μsecs.)*
50	0.096	5629
100	0.149	5661
150	0.197	5643
200	0.240	5629
250	0.279	5610
300	0.313	5628
350	0.345	5615
400	0.373	5633
450	0.399	5640
500	0.423	5644
550	0.445	5651
600	0.467	5620
650	0.486	5624
700	0.504	5620
750	0.521	5612
800	0.536	5624
850	0.550	5638
900	0.564	5631
950	0.576	5652
1000	0.589	5631
1050	0.600	5643
1100	0.612	5615
1150	0.622	5621
1200	0.632	5615
1250	0.640	5646
1300	0.649	5641
1350	0.657	5651
1400	0.665	5651
1450	0.673	5641
1500	0.682	5596

Table D.9: Overhead times for different field lengths.

Appendix E

Notes on the Automated Network Driver Program

Flowcharts of the routines in the program are shown in figures E.74 to E.76. A brief description of each routine is given in the header files of the different program modules as shown in figures E.77 to E.83. Figures E.84 and E.85 show a sample master and slave program respectively. A sample output file (as produced by a master program) is given in figure E.86. Apart from the flowcharts, no documentation has been produced for this program. However, there are many comments (in the source files) at device-dependent subroutines which interact with the 82588 LAN controller of the Intel board (these are expected to be more difficult to understand than other parts of the program). Interested readers should contact the author for more details.

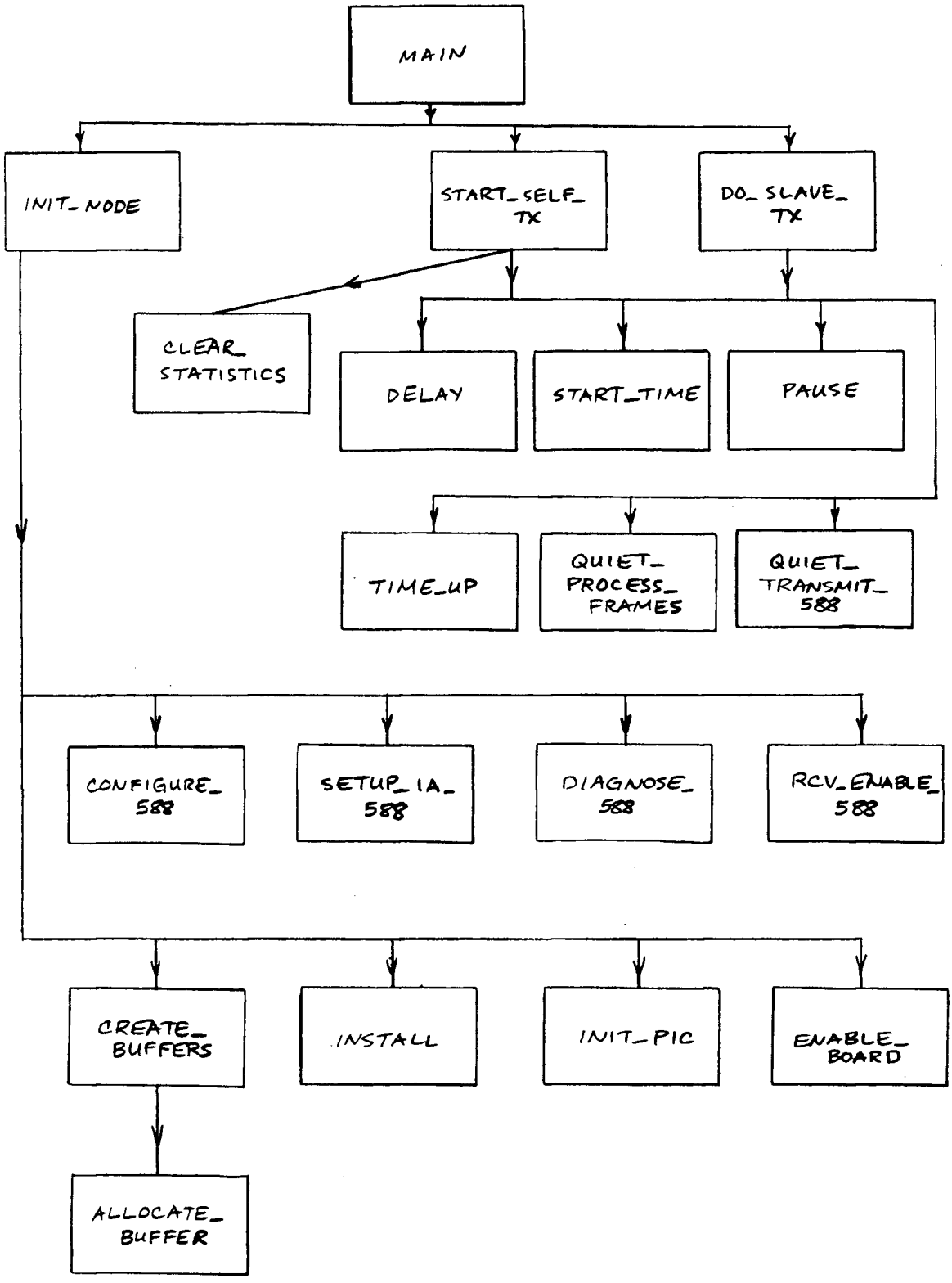


Figure E.74: Subroutine Flowchart of Driver Program, Part 1 of 3

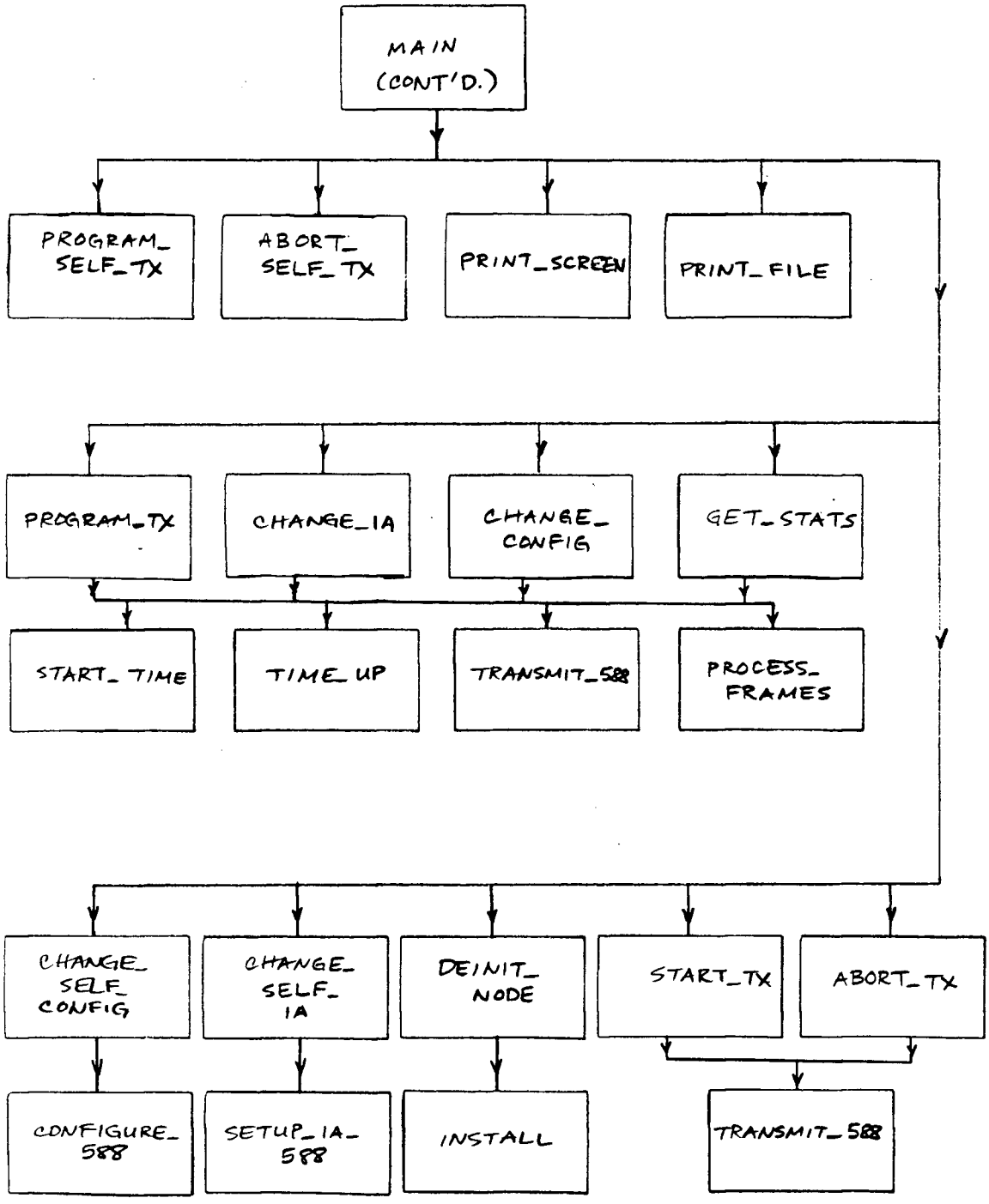


Figure E.75: Subroutine Flowchart of Driver Program, Part 2 of 3

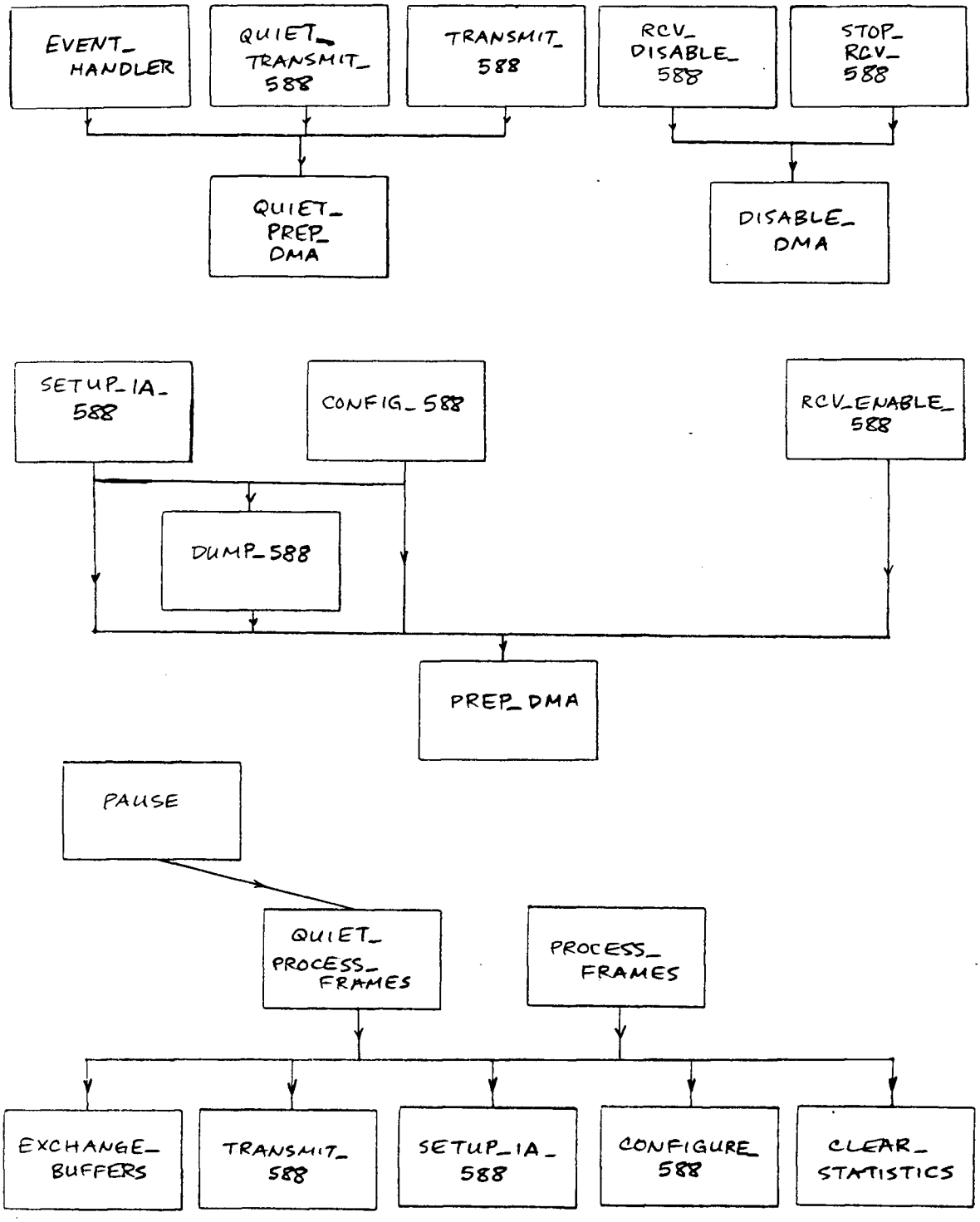


Figure E.76: Subroutine Flowchart of Driver Program, Part 3 of 3

```
/*
      Richard Cam
      93523827
                        Primary set of local application routines.
                        Header file: autoutil.h
                        System: IBM PC's, compatibles, and variants thereof.
                        Compiler: Turbo C version 1.5 w/ Compact memory model
                        Version: 2.1   20 June 1988
*/


void init_node ();
/* initialize buffers and 82588  */

void change_self_config ( int conf0, int conf1, int conf2, int conf3,
                          int conf4, int conf5, int conf6, int conf7,
                          int conf8, int conf9 );
/* change 82588 configuration of local node  */

void change_self_ia ( int new_ia0, int new_ia1, int new_ia2, int new_ia3,
                      int new_ia4, int new_ia5 );
/* change 82588 individual address of local node  */

void program_self_tx ( unsigned int tx_duration,
                       unsigned int delay_period, unsigned int info_field_len,
                       int targ0, int targ1, int targ2, int targ3,
                       int targ4, int targ5 );
/* change repeated-transmission parameters of local node  */

void start_self_tx ( int master );
/* initiate repeated transmissions  */

void do_slave_tx ( int master );
/* slave version of start_self_tx  */

void abort_self_tx ();
/* forcefully halt local nodes transmissions  */

void deinit_node ();
/* normal termination of program  */
/*
      Richard Cam
      93523827
                        Buffer Management Routines.
                        Header file: buffmgr.h
                        System: IBM PC's, compatibles, and variants thereof.
                        Compiler: Turbo C version 1.5 w/ Compact memory model
                        Version: 2.1   06 June 1988
*/


#define MAX_ALLOC_TRIES 3  /* max. number of attempts for make_buffer to
                              allocate memory space in the heap for the
                              specified buffer size without crossing page
                              boundaries.  */


void allocate_buffer ( address *buff, unsigned int num_elements );
/* allocate memory space for a block of size num_elements bytes such that
   it does not cross page boundaries  */
```

Figure E.77: Header Files

```
void create_buffers ( address *config_buff, address *ia_buff,
                      address *dump_buff, address *tx_buff,
                      address rx_buff[], int num_rx_buffs,
                      address *sub_tx_buff, address *sub_rx_buff );
/* allocate memory space for buffers used in the program.  */


void exchange_buffers ( address *buff_a, address *buff_b );
/* exchange contents of address structures.  */
  /*
        Richard Cam
        93523827
                      Global Declarations.
                      Header file: globals.h
                      System: IBM PC's, compatibles, and variants thereof.
                      Compiler: Turbo C version 1.5 w/ Compact memory model
                      Version: 2.1   06 June 1988
  */


  #define TRANSMIT 1
  #define RECEIVE  0

  #define CHANNEL_1 1  /* equate CHANNEL_1 to channel 1 of 8237A DMAC  */
  #define CHANNEL_3 3  /* equate CHANNEL_3 to channel 3 of 8237A DMAC  */

  #define MAX_RX_BUFFS 17  /* max. number of receive data buffers */


  #define CONFIG_BUFF_SIZE 12  /* size of 82588 configuration block  */
  #define IA_BUFF_SIZE 8  /* size of 82588 6-byte indi. addr. block  */
  #define DUMP_BUFF_SIZE 63  /* size of 82588 register dump block  */
  #define MAX_TX_BUFF_SIZE 1508  /* size of largest 82588 transmit data block */
  #define MAX_RX_BUFF_SIZE 1514  /* size of largest 82588 receive data block  */


  #define OCW2 0x20  /* 8259A EOI or specific EOI register  */


  #define UNMASK_LINE_5 0xDF  /* AND mask bit pattern to enable line 5  */
  #define SEOI_LINE_5 0x65  /* specific EOI code for line 5  */


  #define INTR_VECT_13 13  /* vector table entry for hardware
                              interrupt line 5  */
  #define INTR_LINE 5  /* hardware interrupt line  */


  typedef struct {   /* memory block information record  */
     unsigned char *ptr;  /* pointer to block  */
     unsigned int segment, offset;  /* 8088/8086 segment-offset of block's
                                       starting address  */
     unsigned char lowaddr, highaddr, page;  /* 8237A DMA paged-memory form of
          block's starting address. lowaddr and highaddr are the low and high
          bytes for the DMA start address; the page register
          (of the DMA channel) in the IBM PC is loaded with the value
          of page.  */
     unsigned char lowcount, highcount;  /* low and high bytes of block size  */
```

Figure E.78: Header Files (continued)

```
    unsigned int size;  /* size of block in bytes  */
    unsigned int frame_len;  /* actual size of block  */
  } address;
/*
      Richard Cam
      93523827
                    Routines for simple application-layer commands.
                    Header file: midutil.h
                    System: IBM PC's, compatibles, and variants thereof.
                    Compiler: Turbo C version 1.5 w/ Compact memory model
                    Version: 2.1    10 June 1988
*/


#define NO_OPERATION 0  /* nop  */
#define PROGRAM_TX 1  /* command code for program transmission parameters  */
#define CHANGE_IA 2  /* ... for remote indiv. address change  */
#define CHANGE_CONFIG 3  /* ... for remote configuration parameter change  */
#define GET_STATS 4  /* command code for tx/rx statistics fetch operation  */
#define START_TX 5   /* ... for initiation of transmission session  */
#define ABORT_TX 6  /* ... for forced termination of transmission session  */


typedef struct {
   unsigned int src_addr[6], dest_addr[6], configuration[10];
   unsigned int delay, duration;
   unsigned int frame_len;
   unsigned long good_tx, bad_tx, good_rx, bad_rx;
   unsigned long collisions[17], defers, lost_cts, lost_crs, under_runs;
   unsigned long buffer_overflows;
   unsigned long crc_errs, srt_frms, alg_errs, no_eofs, over_runs;
} stats_record;


void clear_statistics ( stats_record *stblk );
/* clear performance data part of statistics block  */


void program_tx ( int master, int dest0, int dest1, int dest2, int dest3,
                  int dest4, int dest5, unsigned int tx_duration,
                  unsigned int delay_period, unsigned int info_field_len,
                  int targ0, int targ1, int targ2, int targ3, int targ4, int targ5 );
/* remote-programs the repeated-transmission parameters of a node  */

void change_ia ( int master, int dest0, int dest1, int dest2, int dest3, int dest4,
                 int dest5, int new_ia0, int new_ia1, int new_ia2, int new_ia3,
                 int new_ia4, int new_ia5 );
/* changes the 82588 indiv. address of another node  */

void change_config ( int master, int dest0, int dest1, int dest2, int dest3, int dest4,
                     int dest5, int conf0, int conf1, int conf2, int conf3,
                     int conf4, int conf5, int conf6, int conf7, int conf8,
                     int conf9 );
/* changes the 82588 configuration of another node  */

void get_stats ( int master, int dest0, int dest1, int dest2, int dest3, int dest4,
                 int dest5, stats_record *tx_rx_stats );
/* fetches statistics from another node  */

void start_tx ( int dest0, int dest1, int dest2, int dest3, int dest4,
```

Figure E.79: Header Files (continued)

```
                int dest5 );
/* initiates repeated-transmission sequence of other node(s)  */

void abort_tx ( int dest0, int dest1, int dest2, int dest3, int dest4,
                int dest5 );
/* halts repeated-transmission session of other node(s)  */

void print_screen ( stats_record tx_rx_stats );
/* prints contents of statistics block to console  */

void print_file ( FILE *fileptr, stats_record tx_rx_stats );
/* prints contents of statistics block to an output file  */
/*
      Richard Cam
      93523827
                    Routines for 82588 Operations and Events.
                    Header file: opevents.h
                    System: IBM PC's, compatibles, and variants thereof.
                    Compiler: Turbo C version 1.5 w/ Compact memory model
                    Version: 2.1   08 June 1988
*/


#define LAN_CTRLR 0x300  /* I/O port address of 82588  */

#define DONE 1  /* flag value: operation was acknowledged by 82588  */
#define PENDING 0  /* flag value: acknowledgement pending  */
#define PASSED 2  /* flag value for diagnose-passed event  */
#define FAILED 3  /* flag value for diagnose-failed event  */
#define SUCCESSFUL 4  /* flag value for successful transmission  */
#define COLLISION 5  /* flag value for collision during transmission  */
#define NOT_OK 6  /* flag value for unsuccessful transmission not caused by
                     a collision  */

#define NOP 0  /* 4-bit operation field code for no-operation command  */
#define IA_SETUP 1  /* ...code for ia-setup command  */
#define CONFIGURE 2  /* ...code for configure command  */
#define TX 4  /* ...code for transmit command  */
#define DUMP 6  /* ...code for dump command  */
#define DIAGNOSE 7  /* ...code for diagnose command  */
#define RCV_ENABLE 8  /* ...code for enable receiver command  */
#define RCV_DISABLE 10  /* ...code for disable receiver command  */
#define STOP_RCV 11  /* ...code for stop receiver command  */
#define RETX 12  /* ...code for retransmit command  */

#define IA_SETUP_DONE 1  /* 4-bit event field code for ia_setup-done event  */
#define CONFIGURE_DONE 2  /* ...code for configuration-done event  */
#define TRANSMIT_DONE 4  /* ...code for transmit-done event  */
#define DUMP_DONE  6  /* ...code for dump-done event  */
#define DIAGNOSE_PASSED 7  /* ...code for diagnose-passed event  */
#define END_OF_FRAME 8  /* ...code for end-of-frame event  */
#define RECEPTION_ABORTED 10  /* ...code for reception-aborted event  */
#define RETRANSMIT_DONE 12  /* ...code for retransmit-done event  */
#define DIAGNOSE_FAILED 15  /* ...code for diagnose-failed event  */

#define CHNL_588_0 0x00  /* 1-bit CHNL field to select channel 0 of 82588  */
#define CHNL_588_1 0x10  /* ...channel 1 of 82588  */


typedef void interrupt (*intrfn_ptr)();
```

Figure E.80: Header Files (continued)

```
                /* intrfn_ptr <--> pointer to an interrupt  function  */
typedef intrfn_ptr *ptr_intrfn_ptr;
                /* ptr_intrfn_ptr <--> pointer to interrupt function pointer  */


void install ( intrfn_ptr newhandler, unsigned int int_num,
               intrfn_ptr *oldhandler );
/* save old interrupt vector of table entry int_num and replace it with
   new interrupt vector  */

void interrupt event_handler ();
/* 82588 interrupt handler  */

void setup_ia_588 ( address ia_buff, address dump_buff );
/* set up and verify 82588 individual address  */

void configure_588 ( address config_buff, address dump_buff );
/* configure and verify 82588 configuration  */

void dump_588 ( address dump_buff );
/* dump 82588 registers  */

void diagnose_588 ();
/* run internal 82588 diagnosis test  */


void rcv_enable_588 ();
/* enable 82588 receiver and prepare first receive buffer  */


void rcv_disable_588 ();
/* immediately disable 82588 receiver  */


void stop_rcv_588 ();
/* disable 82588 receiver as soon as no frame is being received  */


void quiet_transmit_588 ( address tx_buff );
/* no-screen-echo version of transmit_588  */


void quiet_process_frames ( int master );
/* no-screen-echo version of process_frames  */


void transmit_588 ( address tx_buff );
/* transmit frame  */


void process_frames ( int master );
/* process received frames (if any)  */
/*
      Richard Cam
      93523827
                    Timer Utilities
                    Header file: sysclock.h
                    System: IBM PC's, compatibles, and variants thereof.
                    Compiler: Turbo C version 1.5 w/ Compact memory model
                    Version: 2.1   10 June 1988
```

Figure E.81: Header Files (continued)

```
*/

void pause ( int master, unsigned int interval );
/* pause for interval seconds ( and continue to process incoming frames).  */

void delay ( unsigned int interval );
/* delay for interval for loops  */

void start_time ( unsigned int interval );
/* initialize the timer  */

long time_up ( );
/* check if interval seconds have elapsed, return -1 if interval seconds
   have not yet elapsed, return the excess seconds otherwise  */
/*
      Richard Cam
      93523827
                  Utility Routines for DMA/Interrupt Operations.
                  Header file: sysutil.h
                  System: IBM PC's, compatibles, and variants thereof.
                  Compiler: Turbo C version 1.5 w/ Compact memory model
                  Version: 2.1   06 June 1988
*/

#define DMA_MASK 0x0A  /* 8237A single mask register  */
#define FLIPFLOP  0x0C  /* 8237A first/last byte pointer flip-flop  */
#define CH1_ADDR 0x02  /* 8237A channel 1 base address register  */
#define CH1_COUNT 0x03  /* 8237A channel 1 block length register  */
#define CH3_ADDR 0x06  /* 8237A channel 3 base address register  */
#define CH3_COUNT 0x07  /* 8237A channel 3 block length register  */
#define DMA_MODE 0x0B  /* 8237A mode register  */

#define CH1_PAGE 0x83  /* page register in PC for channel 1  */
#define CH3_PAGE 0x82  /* page register in PC for channel 3  */

#define CH1_TX 0x49  /* memory-to-I/O transfer on channel 1  */
#define CH1_RX 0x45  /* I/O-to-memory transfer on channel 1  */
#define CH3_TX 0x4B  /* memory-to-I/O transfer on channel 3  */
#define CH3_RX 0x47  /* I/O-to-memory transfer on channel 3  */

#define MASK_CHANNEL_1 0x05  /* mask channel 1  */
#define UNMASK_CHANNEL_1 0x01  /* unmask channel 1  */
#define MASK_CHANNEL_3 0x07  /* mask channel 3  */
#define UNMASK_CHANNEL_3 0x03  /* unmask channel 3  */

#define OCW1 0x21  /* 8259A mask register  */

#define BOARD_PORT 0x301  /* I/O address of StarLAN board enable port */
#define ENABLE_ALL 0xFF  /* enables all board functions  */

void disable_dma ( unsigned int dma_channel );
/* disable dma channel  */
```

Figure E.82: Header Files (continued)

```
void prep_dma ( address buffer,
                unsigned char dma_channel, unsigned char direction );
/* prepare dma channel for transmission or reception  */


void quiet_prep_dma ( address buffer,
                      unsigned char dma_channel, unsigned char direction );
/* prepare dma channel for transmission or reception  */
/* no screen echo version  */


void init_pic ( unsigned int interrupt_line );
/* initialize programmable interrupt controller  */


void enable_board ();
/* enable all functions in the Intel 82588 StarLAN board  */
```

Figure E.83: Header Files (continued)

```
/*
      Richard Cam
      93523827
                    Master Driver Program for 82588 StarLAN Network.
                    Source file: runlm.c
                    System: IBM PC's, compatibles, and variants thereof.
                    Compiler: Turbo C version 1.5 w/ Compact memory model
                    Version: 1.0   24 June 1988
*/


#include <stdio.h>  /* standard I/O library  */
#include "globals.h"  /* global definitions  */
#include "opevents.h"  /* interrupt handler & loader; 82588 commands  */
#include "midutil.h"  /* remote programming calls  */
#include "autoutil.h"  /* local control calls  */
#include "sysclock.h"  /* timer calls  */


#define MASTER 100


extern stats_record stats_blk, saved_stats_blk;
extern int repeat_transmissions;


stats_record foreign_stats;

unsigned int ia_block[ IA_BUFF_SIZE ] = { 6, 0, 1, 1, 0, 0, 0, 0 };
unsigned int config_block[ CONFIG_BUFF_SIZE ] = { 10, 0, 0x08, 0x00, 0x26, 0x00,
                                  0x60, 0x00, 0xF2, 0x04, 0x88, 0x40 };

FILE *firoutfile, *secoutfile, *thioutfile;
unsigned int i, j;

main()
{
   puts(" Run #1: ring pattern tx data for 50 - 1500-byte info. fields \n");
   puts("         in 50-byte increments.\n");

   firoutfile = fopen("runlm.m01", "wt");
   secoutfile = fopen("runlm.s03", "wt");
   thioutfile = fopen("runlm.s04", "wt");
   init_node();
   if ( MASTER ) {
      for (j = 0; j < 10; j++) {
         for (i = 50; i <= 1500; i += 50) {
            program_tx( MASTER, 1, 3, 0, 0, 0, 0, 300, 0, i,  1, 4, 0, 0, 0, 0 );
            program_tx( MASTER, 1, 4, 0, 0, 0, 0, 300, 0, i,  1, 1, 0, 0, 0, 0 );
            program_self_tx( 300, 0, i,  1, 3, 0, 0, 0, 0 );
            start_tx( 0xFF, 0xFF, 0xFF,   0xFF, 0xFF, 0xFF );
            start_self_tx( MASTER );
            get_stats( MASTER, 1, 3, 0, 0, 0, 0,  &foreign_stats );
            print_screen( foreign_stats );
            print_file( secoutfile, foreign_stats );
            get_stats( MASTER, 1, 4, 0, 0, 0, 0,  &foreign_stats );
            print_screen( foreign_stats );
            print_file( thioutfile, foreign_stats );
            print_screen( saved_stats_blk );
            print_file( firoutfile, saved_stats_blk );



         }
      }
   }
   puts(" >>> End of runlm.");
}
```

Figure E.84: Sample Master Program

```
/*
      Richard Cam
      93523827
                     Slave Driver Program for 82588 StarLAN Network.
                     Source file: runis3.c
                     System: IBM PC's, compatibles, and variants thereof.
                     Compiler: Turbo C version 1.5 w/ Compact memory model
                     Version: 1.0   25 June 1988
*/


#include <stdio.h>  /* standard I/O library  */
#include "globals.h"  /* global definitions  */
#include "opevents.h"  /* interrupt handler & loader; 82588 commands  */
#include "midutil.h"  /* remote programming calls  */
#include "autoutil.h"  /* local control calls  */
#include "sysclock.h"  /* timer calls  */


#define MASTER 0


extern stats_record stats_blk, saved_stats_blk;
extern int repeat_transmissions;


stats_record foreign_stats;

unsigned int ia_block[ IA_BUFF_SIZE ] = { 6, 0, 1, 3, 0, 0, 0, 0 };
unsigned int config_block[ CONFIG_BUFF_SIZE ] = { 10, 0, 0x08, 0x00, 0x26, 0x00,
                                    0x60, 0x00, 0xF2, 0x04, 0x88, 0x40 };

int i;

main()
{
   puts(" Slave Program #1: for use with a runim master.\n");
   puts(" Station Number 1 3 0 0 0 0.\n");

   init_node();
   if ( MASTER == 0 ) {
      i = 0;
      for ( ; ; ) {
         process_frames( MASTER );
         if ( repeat_transmissions ) {
            i++;
            printf("\n repeated tx session number: %d.\n", i );
         }
         do_slave_tx( MASTER );
      }
   }

   deinit_node();
   puts(" >>> End of Slave Program #1 for station 1 3 0 0 0 0.");

}
```

Figure E.85: Sample Slave Program

```
Sample Output File (part of run1m.m01):
Each `paragraph' of numbers corresponds to data
for one PC from a transmission session.

1  1  0  0  0  0
8  0  38  0  96  0  242  4  136  64
1  3  0  0  0  0
300  50  0
40156  253  39966  0
28869  0  0  0
253  126  49  8  0  1  0  0  0  0  0  0  0  0  0  0  0
0
0  0  0  0  0


1  1  0  0  0  0
8  0  38  0  96  0  242  4  136  64
1  3  0  0  0  0
300  100  0
37649  23  37749  0
26309  0  0  0
23  12  4  1  0  0  0  0  0  0  0  0  0  0  0  0  0
21
0  0  0  0  0


1  1  0  0  0  0
8  0  38  0  96  0  242  4  136  64
1  3  0  0  0  0
300  150  0
35556  0  35657  0
20456  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
19
0  0  0  0  0


1  1  0  0  0  0
8  0  38  0  96  0  242  4  136  64
1  3  0  0  0  0
300  200  0
33934  0  34034  0
11994  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
17
0  0  0  0  0
```

Figure E.86: Sample Output File

Bibliography

[1] Costrell, L., Dawson, W.K., "FASTBUS Modular High-Speed Data Acquisition System", repr. from IEEE Int'l Conf. on Communications, ICC '82, June 13-17, 1982, Phila., PA.

[2] Costrell, L., Dawson, W.K., "FASTBUS for Data Acquisition and Control", *IEEE Trans. on Nuclear Science*, Vol. NS-30, No. 4, August 1983.

[3] *IEEE Standard FASTBUS Modular Data Acquisition and Control System*, IEEE Inc., 1985.

[4] personal communication from David Gustavson to W.K. Dawson and R.W. Dobinson, 31 July 1987 electronic mail.

[5] *802.3-1985 (ANSI/IEEE) (Draft International Standard) Standard for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD)*, IEEE Standards Catalog (Fall 1988) order number SH09738.

[6] *802.3a,b,c and e - 1988 (ANSI/IEEE) Supplements to Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Standards Catalog (Fall 1988) order number SH11411.

[7] Golbert, A., Gandhi, S., *Implementing StarLAN with the Intel 82588*, Intel Application Note AP-236, Intel Corp., 1986.

[8] Derfler Jr., F., "Making Connections: Fast Performance over Telephone Wire", *PC Magazine*, Vol. 7, No. 15, 13 Sept. 1988.

[9] *82588 Design Kit Users Guide*, Order Number 296098-001, Intel Corp., 1986.

[10] *Local Area Networking (LAN) Component User's Manual*, Order Number 230814-003, pp. 6-1 to 7-60, Intel Corp., Sept. 1985.

[11] Retix, 1547 Ninth St., Santa Monica, CA 90401, U.S.A., phone: (213) 829-4922.

[12] WD83C510 StarLAN Hub Controller, Document Number 79-000104, Western Digital Corporation, 1987.

[13] Gustavson, D., Theus, J., "Wire-OR Logic on Transmission Lines", *IEEE Micro*, June 1983.

[14] Wilson, R., "The Physics of Driving Backplane Buses", *Microprocessors and Microsystems*, Vol. 10, No. 2, March 1986.

[15] Southard, R., "Interconnection System Approaches for Minimizing Data Transmission Problems", *Computer Design*, March 1981.

[16] Blood Jr., W., *MECL System Design Handbook*, 4th ed., 2nd printing, Motorola Inc., 1983.

[17] Del Corso, D., et al., *Microcomputer Buses and Links*, Academic Press, 1986.

[18] Mokhoff, N., "Transceiver Does Away With Central Hub in StarLAN", *Electronic Design*, June 25, 1987.

[19] personal communication with Ray Duley (AMD Austin, Texas).

[20] Hammond, J., O'Reilly, P., *Performance Analysis of Local Computer Networks*, Addison-Wesley, 1986. pp. 381-390.

[21] Ibid., pp. 390-401.