An Access Layer Protocol for Parallel Networks

by

Rajesh Kumar

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED

[illegible]

Dr. S. F. Midkiff, Chairman

[illegible]

Dr. N. J. Davis, IV

[illegible]

Dr. I. Jacobs

May 1993

Blacksburg, Virginia

LY
5655
V855
1993
K863
c.2

c.2

An Access Layer Protocol for Parallel Networks

by

Rajesh Kumar

Dr. S. F. Midkiff, Chairman

Electrical Engineering

(ABSTRACT)

Parallelism can be applied to local area networks to achieve higher data rates using existing hardware technologies. Parallelism can be employed at the different layers of the Open Systems Interconnection (OSI) reference model. This research proposes and analyzes a technique that permits the use of different degrees of parallelism at different protocol layers. A new protocol layer, called the *access layer* is defined. The protocol and functionality for this layer are defined. The definitions make provision for incorporating an error correction coding procedure known as *cross channel coding*. A software simulator was built for the proposed parallel network. The simulator has a detailed model of the access layer and was used to verify the functionality defined for the access layer and to estimate the performance of the parallel network. The simulation results indicate that although the access layer processes data in serial, it is not a bottleneck in the parallel system. Other insights obtained from the experiments are also presented.

Acknowledgments

I would like to express my sincere gratitude to my major professor, Dr. S. F. Midkiff for his willing assistance and guidance throughout my program of study. I am also grateful for the direction he provided my research work, without which it would have been very difficult to complete this thesis.

A special thanks to Dr. I. Jacobs and Dr. N. J. Davis, IV, for their helpful suggestions and for serving on my committee.

Many thanks to Scott Harper and Joe Wiencko for their help and to my friend, Raj Kumar, for his encouragement.

I am grateful for the employment at the Forestry department which helped sustain my studies when other sources were lacking.

Finally, I would to thank my family, Achan, Amma and Akkan, whose encouragement and prayers have made it all possible.

Table of Contents

List of Illustrations

List of Tables

Chapter 1. Introduction

One strategy to meet the demand for higher data rate computer networks is to use parallelism. A joint project between Virginia Tech and Old Dominion University investigated the use of parallelism in local area networks. The research described in this thesis proposes and analyzes a technique that permits the use of different degrees of parallelism at different network protocol layers. A new protocol layer, called the *access layer* (AL) is defined. The protocol and functionality for this layer are defined. A software simulator was built for the proposed parallel network. This simulator was used to verify the functionality of the AL and to obtain performance measures for the entire parallel network.

1.1. Motivation

A number of different strategies are being used to meet the growing demand for higher data rate networks. Most strategies depend on the development of new transmission technologies. One strategy that can make use of currently available technology is to exploit parallelism in computer networks. Apart from the gain in bandwidth, parallelism can also provide improved fault tolerance, scalability and flexibility over a serial approach. Parallelism can be used at the different protocol layers of the OSI reference model. The parallel network project [1] proposed the use of parallelism at the software-intensive upper-layers and at the hardware-intensive lower-layers. Multiple "protocol processors" can be used at the upper layers, while multiple channels can be used at the data link layer.

To achieve the full benefits of parallelism, the parallelism at the upper and lower layers should be transparent to each other. A mechanism is then needed to map between the parallelism at these layers. The *access layer* (AL) is such a mechanism. The AL is positioned between the network and data link layers of the OSI protocol stack.

The AL is a serial operating point in the parallel network. Therefore, it has the potential to be a system bottleneck. A major goal in specifying the AL functionality is to ensure that the AL does not become a bottleneck.

1.2. Research

The first objective of this research was to define the AL protocol and associated functionality for the proposed parallel network. The definition had to satisfy a number of criteria. In keeping with the layered approach, the AL protocol must make no, or few, assumptions of the protocol at the adjacent layers. The AL functionality also had to include a means for using an error correction coding procedure known as *cross channel coding* [2]. The second part of the research was to verify the defined functionality and to study the impact of the AL on the performance of the network.

The AL functionality was defined in two stages. In the first stage the transmission of packets from the upper layers was considered. The second stage developed the steps for processing frames received over the physical channels. The IEEE 802 [5] standards were used as a guide in defining the AL protocol. Criteria such as conforming with existing standards, providing for future needs and the potential for efficient implementation were considered when making decisions such as address widths and header sizes. A software simulator was built for verification and performance estimation. The writing of this simulator often went in parallel with the protocol definition phase. The simulator models one implementation of the entire proposed parallel network. It contains a detailed model of the AL that facilitates functional verification. Parameters such as throughput, latency and utilization are monitored on an end-to-end basis for obtaining performance measures.

The twin objectives of defining the AL protocol and verifying its functionality were achieved. The protocol definition was in terms of a specification of the steps to perform during transmission and reception of packets. The simulator was used to verify the AL functionality. Performance measures were obtained for a prototype of the proposed parallel network.

Analyses of the performance results indicated that the AL was not the bottleneck for most of the cases considered by the simulation experiments. The AL appeared to be the bottleneck only when the average packet size became very small. Larger packet sizes were found to be better for higher throughput and lower latency. Packets are classified according to the destination node and a cross channel coding parameter called "*m*." The average packet latency was reduced, without affecting throughput, if the total number of such classes was reduced. Whenever cross channel coding is used, packets are split over all available physical channels. When it is not used, packets can be sent without splitting them equally over all channels. However, the additional complexity due to unbalanced loading of the physical channels was not found to justify the gains from such a strategy.

1.3. Organization of the Thesis

This chapter has provided a brief introduction to the research. Chapter 2 discusses background relevant to the research, including a discussion of the parallel network project and the architecture of the AL. The chapter also includes a survey of relevant literature. The next chapter presents and discusses the AL protocol and functionality in detail. The functionality for the transmission and reception of packets is discussed separately and examples are provided. Chapter 4 discusses the simulator that was developed to study the AL. The models for the various system components and functions are discussed. A discussion of simulation experiments and results is presented in Chapter 5. The implications of the results are also examined. The final chapter provides a summary and some suggestions for further research.

Appendix A provides the expressions for cross channel coding and decoding for the particular case studied in the simulation experiments. Appendix B lists the relevant parameters of the devices used in the simulator and the timing estimates used in the performance studies. Appendix C provides complete results for the simulation experiments.

Chapter 2. Background and Literature Review

This chapter presents background and a review of the current literature. It begins with a short discussion on gigabit networks. The motivation for the use of parallelism as the basis for building a gigabit network is presented next. This is followed by a literature review. The fourth section provides an overview of the proposed parallel network and the concluding section discusses the architecture chosen for the access layer.

2.1. Gigabit Local Area Networks

There is a trend towards local area networks (LANs) with throughputs of 1 Gbps or greater [15]. A significant number of applications have been identified for such LANs. These include serving as high-speed LAN backbones, enabling a network-based high performance computing environment and permitting large-scale distributed computing [7, 8].

Building a gigabit LAN is not a simple matter of scaling an existing system. They demand a totally new approach [6, 8]. In long-haul gigabit networks, the propagation delay, rather than the capacity of the network, dominates the response time. Kleinrock [6] suggests that this latency be hidden by adopting parallel processing or pipelining at the application layer. Propagation delay is, however, not very significant for LANs since the links are much shorter. Kung [8] identifies several problems specific to gigabit LANs. These include highly bursty traffic and increased mismatches in bandwidth of the network and the host connections to the network. Additional requirements for gigabit LANs are low-latency communication and robust performance. Clearly, new approaches and techniques are needed to make gigabit LANs feasible. One such approach is to use parallelism.

2.2. Motivation for the Use of Parallelism

The standard approach for increasing data rates is to use new technologies to develop faster networks. However, the limitations of this are well known from the design of general purpose computers [16]. An approach with greater potential is the use of parallelism in the network nodes and in the network itself. This provides the potential for large increases in network bandwidth in a manner similar to how parallel computing techniques are used to achieve dramatic advances in computational power.

Apart from bandwidth gains, the use of parallelism gives the network greater flexibility, scalability and fault-tolerance. Some nodes in a network may need only a fraction of the total network capacity. These nodes can use fewer parallel components to satisfy their requirements. Such a node can satisfy future needs for increased capacity by adding more components. This kind of flexibility is either not possible or is very difficult with monolithic systems. Using a similar strategy of adding more components, it is possible to scale the entire network upwards. The degree of parallelism used will change, while the basic network architecture remains the same.

Parallel systems are inherently more reliable than serial systems. If properly designed, a parallel system will fail only when all of the parallel components at some point have failed.

A parallel system can continue to function at partial capacity when some components fail, instead of totally collapsing. In other words, the use of parallelism makes "graceful degradation" possible. In a serial system, the failure of even one component usually causes the entire system to fail.

Parallel networks can also use layering, and parallelism can be introduced at different network layers. Such networks need mechanisms to make the parallelism at each layer transparent to that at adjacent layers. When such techniques are used, parallel networks can take advantage of technology advances at a particular layer as easily as monolithic networks.

2.3. Related Work

A review of related literature is presented in this section. The discussion begins with an alternative to parallelism for building gigabit LANs. This is followed by reviews of several efforts aimed at achieving Gbps rates for transport protocol processing. The final area considered is network access methods for high-speed LANs.

Kung [8] explains why traditional methods in network analysis and design may not be applicable to gigabit LANs. Kung believes that the limited capacity of a shared-medium architecture, such as a bus or a ring, is not suitable for gigabit LANs. The solution he proposes is a switch-based architecture, much like the existing telephone network. Other key features are fast congestion notification by providing feedback from the points of congestion, link-by-link flow control, rather than end-to-end flow control, and cell-level multiplexing. Cell-level multiplexing involves splitting large data packets into small *cells*. This enables small, higher priority packets to be transferred with low latencies even when they are sent behind a large data packet. This switch-based approach is a radical shift from the traditional one, and, as Kung points out, needs to be validated before implemented. An important point is that the implementation of this proposal requires that several new standards be established, including the specification of interfaces to existing standards.

The access layer defined in this thesis assumes the availability of a transport layer which can handle gigabit rates. There are two practical approaches to develop a transport protocol implementation that can handle these rates [12]. These are to develop a dedicated hardware solution for a specific protocol [11] or to use parallel processing [1, 12, 10].

A multiprocessor architecture is proposed by Zitterbart [10] for implementing OSI transport layer protocols at Gbps rates. The author discusses the concept of division of the protocol layers into horizontal sub-layers. Each layer is divided into two parts: a *send part* which is responsible for the data transfer from layer (N + 1) to layer (N - 1), and a *receive part* which transfers data from layer (N - 1) to layer (N + 1). Each part is implemented on separate microprocessors and can function independent of each other. Simulation results for a prototype implementation of the OSI TP4 transport layer protocol are presented. Speed-ups of up to seven times over a single processor implementation were obtained. This approach requires significant amount of special-purpose hardware. However, given that hardware intensive solutions are becoming less expensive, the particular methods presented in this paper are promising.

Another scheme that uses parallel processing to achieve transport protocol processing at Gbps rates is presented by Jain, Schwartz and Bashkow [12]. The authors propose a multiprocessor architecture that assumes hardware support to perform any lower layer protocol processing. The parallelism is at the packet level. A processor-pool is used for processing the transport protocol and each processor has a local memory into which the next packet to be processed is written. Packets are scheduled on these processors in a deterministic manner. A shared memory is used to maintain global state information. Since the packets are stored in local memory, there is not much contention for shared memory access. High-speed buses are assumed between the transport layer processor-pool and the low-level processors which interface to the network. A detailed throughput and delay analysis is presented in the paper which indicates that the proposed architecture will meet the performance requirements for processing received packet at 1 Gbps. The authors suggest that transmission processing at Gbps rates will be possible with four additional processors and modest extensions to the architecture

A hardware-intensive approach is taken by Kanakia and Cheriton [11]. The authors present the design of a network adapter board (NAB) that implements key transport protocol functions in special purpose hardware. The NAB interfaces to the host processor and to the network. Even with special purpose hardware, the NAB uses parallelism in the form of pipelining to process the transport protocol. The main components of the NAB are a processor, buffer memory and the packet pipeline. Buffer memory is used to match speeds between the host bus and the network and as a staging area for transmission and reception processing. Dual-port RAM is used for the buffer memory to reduce contention. A prototype using a 68020 processor was built. The throughput of a single data stream at the user level falls short of the expected performance for networks in the 100 Mbps range or higher. The authors estimate that upgrading the prototype to use a high performance processor, such as MIPS RISC processor, would boost the throughput of a single transaction closer to the 100 Mbps range. They believe that further refinements in the internal network adapter design and possibly host interface and transport protocols are required to push peformance to a higher range.

All three approaches described above utilize parallelism in some form to achieve gigabit rates. Fairly strict ahderence to the OSI layered model is maintained in all three cases. The interface to the data link layer in each case is suitable for connecting to the access layer that is defined in this thesis.

The access layer assumes the availability of a network adapter to interface to the data link layer. Some approaches in developing adapters for high-speed LANs are now discussed. Rege [13] presents the architecture and implementation of an FDDI adapter. This adapter is designed to meet the ultimate performance goal of transmitting 450,000 packets per second. This is an ultimate goal because the maximum packet carrying capacity of FDDI is 450,000 packets of 28 bytes each. The adapter implements all the functions at the physical layer and a major subset of the functions at the data link layer. It uses a three-stage pipeline to process packets without CPU interference. A ring interface is specified for communicating between the pipelined stages. Rings operate as queues that allow buffering between pipelined stages, enabling these stages to proceed in an asynchronous fashion. The memory subsystem on the adapter consists of the packet buffer memory and a packet memory interface unit. This unit has separate buses to connect to the pre-FDDI ring pipeline stage and to the post-host bus stage. A microprocessor subsystem along with local memory forms the intelligent on-board adapter manager (AM). The AM also implements the station management (SMT) functions of the FDDI protocol. One implementation of the adapter architecture is also presented in this paper. This implementation meets the 100 Mbps throughput for transmit and receive streams when the packet size is at least 69 bytes. Below this size, the throughput falls off rapidly.

Ippoliti and Albanese [14] point out that it is not possible, even with custom VLSI implementations, to build a serial media access controller (MAC) for gigabit rates. They propose a parallel MAC to achieve this goal. The physical network is a set of synchronized parallel channels in a fiber optic transmission system. Each channel has a double unidirectional bus topology and a throughput of 139 Mbps. There is a separate MAC entity for each channel. The parallel MAC layer has a fast packet bus (FPB) with the upper-layer service processors. These processors compete for use of the FPB via round-robin arbitration. Every packet from these processors is split over all parallel channels. However, the MAC processing is done on only one channel. The output controller of the adapter splits the packet to be sent among the output buffers of all the channels. Since the channels are synchronized, all fragments of a packet arrive simultaneously at the destination. Hence, it is feasible to reduce overhead by putting the MAC header on only one channel. It must be noted that this MAC connects to a particular physical network. This is in contrast to the access layer which can function with any data link standard.

Martini and Rupprecht [16] point out that a disadvantage of parallel approaches, such as the one used by Ippoliti and Albanese in the Metrocore project [14], is the static distribution of workload. Thus, performance degradation due to temporarily overloaded units cannot be avoided by a dynamic distribution of workload. The authors provide a brief outline of a new parallel network concept based on Petri nets. In their scheme, protocol modules are sequential programs executed on several general purpose microcomputers. They achieve speed-up by taking advantage of all classes of protocol inherent parallelisms. The access layer, as defined in this thesis, can incorporate dynamic load distribution if network loading information is available to it.

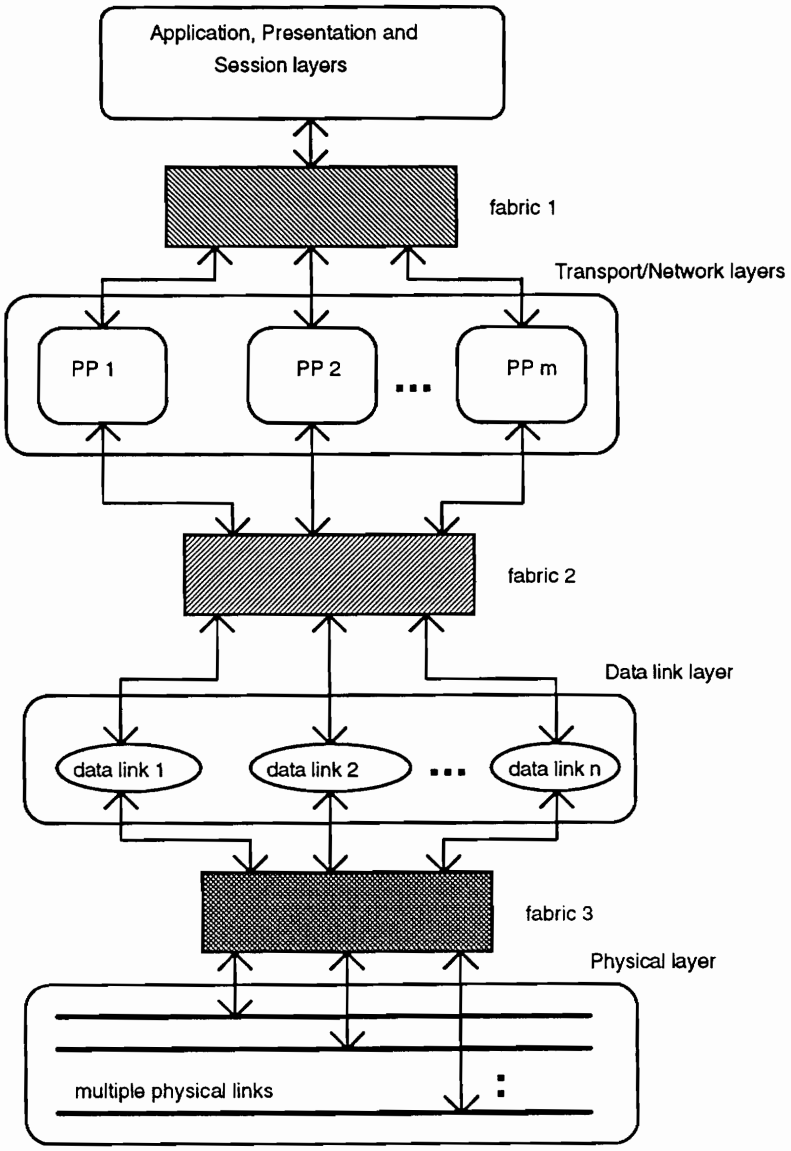

Figure 2.1. Proposed parallel network.

2.4. Overview of Proposed Parallel Network

The project which initiated this research proposed the use of parallelism as the basis for building a gigabit LAN. The network, as visualized in the project, is shown in Figure 2.1. The project uses the OSI seven layer model as a reference for specifying the use of parallelism. Parallelism at the application layer is more in the realm of general-purpose computing and is not considered. The first point for introducing parallelism is at the software-intensive upper protocol layers. These include the network, transport, session and presentation layers. In general, however, session and presentation layer processing are not often cleanly separated from the application layer implementations and can be assumed to be executed by the host processor. Transport and network layer protocols involve a significant amount of processing and are sought to be implemented on multiple *protocol processors* (PPs). Parallelism could be at the connection, packet or process level. The data processed by these parallel PPs are then distributed to multiple entities at the data link layer. This is the second point of parallelism. The data link entities are based on standards such as Ethernet and FDDI. The final point for using parallelism is at the physical layer, where time-division, wavelength-division or space-division multiplexing could be used.

The "fabric" between each point of parallelism keeps the parallelism at each layer or layers transparent to the adjacent points. If this transparency is absent, any change in the parallelism at any layer will impact on the configuration of the adjacent layers. This will negatively affect the network flexibility, scalability and fault tolerance, which are the features sought to be improved by introducing parallelism. Therefore, the fabric is essential to gain the full benefits of parallelism. The access layer is the fabric between the transport/network layers and the data link layer. In a simplistic implementation the fabric at each point could have null functionality.

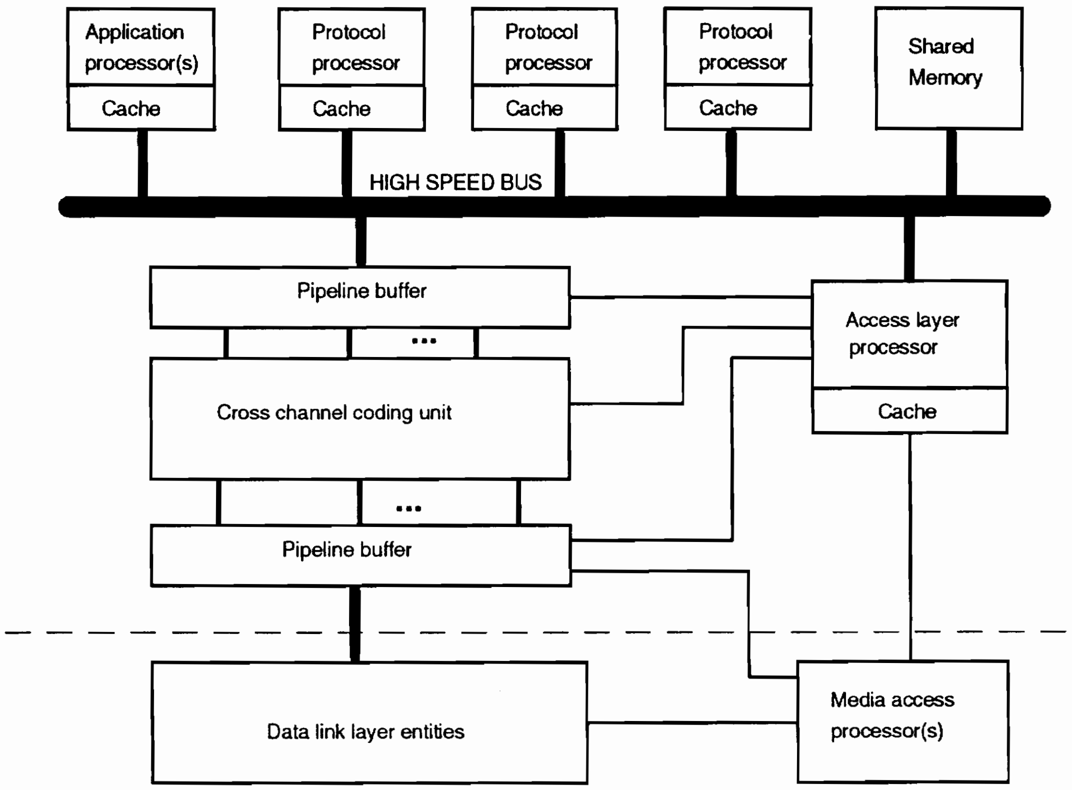

Figure 2.2. Parallel network node implementation.

2.5. Access Layer Architecture

After considering several options, a bus-based architecture was chosen as the most feasible for the AL. The definition and study of the AL protocol which forms fabric 2 in Figure 2.1 is the main objective of this research. The parallel network node implementation assumed for this study is shown in Figure 2.2.

All processors except the media access processor(s) are connected to the common bus. This includes the access layer processor, the application processor(s) and the protocol processors. All data is stored in the shared memory which is also connected to the common bus. The bus controller is not shown in the figure. All processors have sufficient local memory, shown in Figure 2.2 as cache, for storing control information such as data requests and data indications.

The AL incorporates the use of an error correction coding scheme called *cross channel coding* (CCC). The coding procedure, discussed in greater detail in Chapter 3, is controlled by the access layer processor, while the actual coding is performed by a separate coding unit. The post-coding pipeline buffers can also serve as the input buffers of the data link layer entities.

Chapter 3. The Access Layer

This chapter discusses the Access Layer (AL) protocol in detail. First, the need for the AL and its basic function are discussed. This is followed by a brief introduction to cross channel coding (CCC), which is incorporated into the AL protocol. The AL protocol is then explained. The chapter concludes with a summation of the implications of CCC on the AL protocol.

3.1. Need and Required Functionality

Parallelism can be applied to the different protocol layers of the OSI reference model. In addition to increased bandwidth, other major advantages of parallelism are scalability and fault tolerance. To realize the full benefits of parallelism and to maintain layering in the system, the degree of parallelism at each layer should be transparent to adjoining layers. Hence, a mechanism is needed to map the degree of parallelism at each layer to that at adjacent layers.

The AL maps upper layer parallelism (multiple protocol processors) to lower layer parallelism (multiple physical channels). During this mapping the AL can also adopt a suitable coding procedure for increased reliability. At the transmitting end, the AL must divide and/or merge the available upper layer packets to create AL packets which are then distributed among the available lower layer channels. At the receiving end, the AL has to reorganize the arriving AL packets into the orginal upper layer packets and deliver these rebuilt packets to the local upper layer. For the purpose of rebuilding split or merged packets, the AL must attach a header to each AL packet.

The AL at each node has all the information needed to translate the node addresses to the physical channel addresses. This information includes the connectivity of the physical channels, the virtual addresses of all the nodes in the network and the mapping between these addresses and the physical channel addresses.

3.2. The AL Protocol

The discussion of the AL protocol begins with an overview of the CCC scheme. The AL protocol is then presented. A step-by-step explanation of its functionality, including transmission and reception processing, is given.

3.2.1. Cross Channel Coding

CCC is an error correction scheme which is particularly suitable for parallel channels. This coding procedure, proposed by Wiencko [2], takes advantage of the simultaneous transmission of multiple packets to the same destination over multiple (parallel) channels. Redundant information is sent as *code bits* in each AL packet so that lost or erroneous packets can be rebuilt at the receiving end rather than being retransmitted. The AL does not guarantee reliable service, so the use of CCC is not essential. However, its use increases reliability because CCC enables the AL to recover a limited amount of data that has been lost due to errors. In the absence of CCC or a similar technique, the upper layers must request retransmission to get such data. Hence, the use of CCC may also reduce the latency in delivering packets to the upper layers.

Consider a case where there are four physical channels between the source and destination nodes. The available data can be split over these four channels to maximize throughput. Consider that the data is transmitted as four packets, with one on each channel. CCC can be applied to these four packets. The set of packets undergoing CCC together is referred to as a *coding set*. The actual encoding process depends on the degree of reliability that is required. For the case of four packets, coding can be done so that at most one or at most two of the four transmitted packets can be rebuilt at the receiving node. In CCC terminology, the number of packets sent is n and of these m can be rebuilt from the other $n - m$ packets. Since one packet from each set of packets undergoing coding is sent over a particular channel, n is usually equal to the number of available channels. In the above case, $n = 4$ and $m = 1$ or 2.

In general, the coding procedure involves first splitting the data in each of the n packets into equal size parts. The number of parts depends on the relative values of n and m. Code bits for each channel are then generated by exclusive-ORing (XOR) specific combinations of certain parts of all packets. The code bits are attached to the ends of the respective packets. At the receiving end, if a packet arrives error free, the data can be forwarded to the upper layers and the code bits can be simply discarded. To rebuild any m or fewer packets, the code and data bits of $n - m$ packets are necessary. The decoding involves a similar splitting and XOR procedure.

The key to CCC is the fact that $a \oplus b \oplus b = a$. The coding and decoding equations for the $n = 4$, $m = 1$ and the $n = 4$, $m = 2$ cases are presented in Appendix A.

The most important implication of using CCC is that all the n packets that undergo coding together, forming a coding set, have to be of equal size.

3.2.2. Definitions and Terminology

This section contains definitions and explanations for terms which have special meaning in the context of the AL protocol. This includes descriptions of the various headers and entities including data requests and data indications.

3.2.2.1. Protocol Processor Header

Protocol processors (PPs) create a *PP header* in accordance with the particular transport/network layer protocols that are used by the upper layers. The actual format of the header depends on the particular protocols, e.g., TCP/IP or UDP/IP.

3.2.2.2. Upper Layer Packets

Upper layer packets are passed to the AL by the network layer for transmission to the destination node. In OSI terminology, these packets form the *service data units* (SDUs) to the AL [3]. These packets are a combination of the data sent by the application processor(s) and the PP header. The AL has to deliver these packets without fragmentation to the upper layers at the destination node.

3.2.2.3. Destination Class

Every upper layer packet belongs to a particular *destination class*, or *dest_class*. A destination class is the combination of destination node and the required m value for CCC. Only packets belonging to the same destination class can be in the same coding set.

3.2.2.4. Data Request

A PP issues a data request (DR) to the AL whenever it has a packet to transmit. The DR format is shown in Figure 3.1.

ID	SVCA	DVCA	SIZE	SERV_CLASS	DATA_PTR
(4)	(6)	(6)	(4)	(2)	(4)

Figure 3.1. Data request format.

The numbers in parentheses in Figure 3.1 indicate the number of bytes in each field. The fields are defined as follows.

ID:	A unique identifier used to distinguish between the DRs generated at a node. The AL uses this value when it is building the header for AL packets. The ID can be any integer in $[1, 2^{32} - 1]$. Mutually exclusive ranges from this set are reserved for each upper layer PP to ensure the uniqueness of the ID.
SVCA:	*Source virtual channel address, i.e.*, the virtual channel address (VCA) of the source node. Addressing is discussed in Section 3.2.3.5.
DVCA:	*Destination virtual channel address, i.e.*, the VCA of the destination node.
SIZE:	Specifies the size of the upper layer packet in bytes.
SERV_CLASS:	Specifies the class of service desired for this packet. This specification could have originated from the application processor. The service

required typically corresponds to the value of m for CCC. This field can also be used by the upper layer to specify particular lower layer channels that are to be used to transmit this packet.

DATA_PTR: Pointer to the shared memory locations in which the upper layer packet is stored.

3.2.2.5. AL Header

The AL attaches an *AL header* to each of the packets that it transmits over the physical channels. This header is of variable size and has the format shown in Figure 3.2.

ID	SCA	DCA	SIZE	SERV_CLASS	CSET	COMP	SPLIT_PAT
(4)	(6)	(6)	(4)	(2)	(4)	(4)	≥ 4

Figure 3.2. AL header format.

As with the DR format, the numbers in parentheses in Figure 3.2 refer to the number of bytes in that particular field. The fields defined are as follows.

ID: A unique identifier that the AL assigns to each packet that it creates.

SCA: The *source channel address* is the physical address of the source node on the physical channel over which this AL packet is to be sent.

DCA: The *destination channel address* is the physical address of the destination node on the physical channel over which this AL packet is to be sent.

SIZE: Specifies the size of the AL packet in bytes, including the AL header and data bits, but not including the cross channel code bits.

SERV_CLASS: This field is split into two sub-fields. The upper twelve bits form the *channel* sub-field, while the lower four bits form the *m* sub-field. Details of the SERV_CLASS field are provided in Section 3.2.3.6.1 dealing with the creation of AL headers.

CSET: A unique identifier for the *coding set* to which this packet belongs. The CSET field is identical for all AL packets belonging to a particular coding set.

COMP: This field is a 32-bit, bit-coded datum. The number of set (1) bits is the number of upper layer packets that have been used for this AL packet. Only a subset of the upper layer packets used in a coding set are used for a given AL packet in that coding set. The positions of the set bits are used to identify this subset. The use of a fixed size COMP field implies that a maximum of 32 upper layer packets can be used in one coding set. The construction of this field is explained in Section 3.2.3.6.1.

SPLIT_PAT: The *splitting pattern* is a variable size field. It is a minimum of 4 bytes and a maximum of 128 bytes in length. The data in an AL packet is made up of parts of different upper layer packets. SPLIT_PAT specifies which part of the AL packet belongs to each upper layer packet used for this AL packet. This field, the COMP field, and an implicit channel ordering are used to rebuild the upper layer packets at the receiver.

With the minimum SPLIT_PAT field of 4 bytes, the minimum size for an AL header is 34 bytes.

3.2.2.6. AL Packets

AL packets are the *protocol data units* (PDUs) created by the AL. These packets form the SDUs for the data link layer. AL packets are composed of the AL header, data from upper layer packets and cross channel code bits.

3.2.2.7. Channel Header and Lower Layer Packets

Depending on the actual physical channel that is used, the channel controller adds a header of its own to the AL packets. This *channel header* usually includes physical addresses and an error detection code. Cyclic redundancy checks (CRCs) are commonly used. The combination of AL packet and channel header forms a *lower layer packet.*

3.2.2.8. Data Indication

A *data indication* (DI) is created by the AL and passed to the upper layer to indicate the availability of a new upper layer packet. The DI is the counterpart of the DR at the receiving end. A DI has the format shown in Figure 3.3.

ID	SVCA	DVCA	SIZE	DATA_PTR
(4)	(6)	(6)	(4)	(4)

Figure 3.3. Data indication format.

The numbers in parentheses in Figure 3.3 represent the number of bytes in each field. The SVCA and DVCA fields have the same meaning as in a DR. The other fields are defined as follows.

ID:	An identifier assigned by the AL to the DI. It is used by the PPs to distinguish between DIs when processing received packets.
SIZE:	The size of the upper layer packet in bytes. It should be identical to the SIZE field of the DR that was used at the transmitting node, to send this packet.
DATA_PTR:	Specifies the address in shared memory where the upper layer packet is stored.

3.2.3. Transmission of Data

In the context of the AL, transmission involves transferring upper layer packets (SDUs), suitably packaged as AL packets (PDUs), to the lower layers. Depending on the relative size of the upper layer packets, a given packet may be split over different AL packets belonging to the same coding set. An overview of the AL transmission protocol is presented first. This is followed by a detailed description of the functionality of the AL during transmission.

3.2.3.1. Overview of Transmission Protocol

Figure 3.4 shows a graphical representation of the various steps involved in the transmission of data from the application layer.

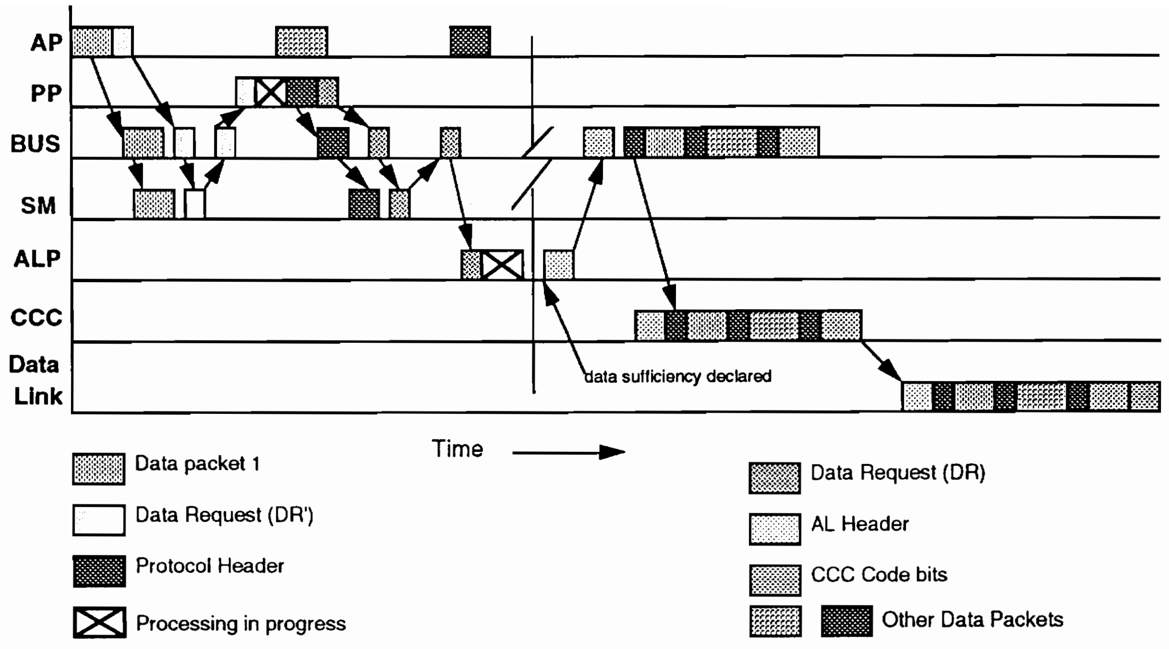

Figure 3.4. Graphical representation of data transmission.

The transmission process starts at the left end of the time scale (x-axis) when the application processor (AP) sends a data packet to the SM over the bus. The AP also writes a data request (DR′) to the SM. DR′ is eventually read by a PP via the bus. After some processing the PP creates a PP header and attaches this to the packet in SM. The PP then creates and writes a data request (DR) to the SM via the bus. DR is read by the access layer processor (ALP) via the bus.

The AL accumulates DRs for upper layer packets until it achieves a condition of *data sufficiency* for some destination class. The break in the time-scale in Figure 3.4 implies the repetition of the steps described in the previous paragraph until data sufficiency is declared. Achieving data sufficiency means that it is decided to send one or more of the accumulated packets in that destination class. The actual algorithm to declare data sufficiency is explained in Section 3.2.3.4. The next step is to create AL packets, one for each channel on which the data is going to be sent. This set of AL packets forms one coding set. A *splitting algorithm* is now run on the upper layer packets marked for transmission. The algorithm must specify how to distribute the data in the upper layer packets among the AL packets of this coding set, such that all AL packets, composed of data, AL header and CCC bits, will be of equal size. Equal sized packets are a requirement of CCC. The AL packets are then built. The first step is to create the header for each of the packets. The data from the upper layer packets is then added. This combination of header and data is sent to the CCC unit, where the CCC code bits are added. The AL packets available at the output of the CCC unit are sent directly to the physical channel buffers, where they await processing by the media access controller (MAC) of each channel. The right extreme of the x-axis in Figure 3.4 represents this point. The phases of data transmission are now described in more detail.

3.2.3.2. Application Layer to Transport/Network Layers

Data almost always originates at the application layer, with the exception of an intermediate layer sending control information. All application processors are connected to the common bus. The data from a particular application is written to the SM via the bus. After the application has written all data to the SM, the associated processor creates and writes a data request to the PPs. This data request (denoted as DR′) is not the data request (DR) sent by a PP to the AL. DR′ is not part of the AL protocol and is specific to the upper layer protocol. Basically, a DR′ specifies the size, destination and address in SM of the data to be transmitted. It may also specify the reliability required for the transmission.

The mechanism by which the PPs are informed about the new DR′ depends on the upper layer protocol. One possibility is having the PPs "snoop" on the bus. Another method is for the PPs to check the SM at regular intervals for a DR′ that may be waiting. In response to the DR′, a PP packages some or all of the data into an *upper layer packet.* The PP may also either use data from one or more previous data requests for this upper layer packet or wait for one or more additional data requests before creating an upper layer packet. This packet includes a PP header. The PP then creates a DR to the AL to request transmission of this packet. The DR is written to SM, where it remains until it is read by the ALP.

3.2.3.3. Transport/Network Layers to AL

The basic function of the AL during transmission is to deliver the upper layer packets, packaged as AL packets to the lower layers. The first step is to read DRs for the upper layer packets from SM. The method chosen for this, in the AL design, is to poll the SM at intervals for any waiting DRs. Consider that the ALP has finished its current task and has not found any incoming packets to process. The ALP initiates a read of the SM for any waiting DRs. If there are none, the ALP goes into an idle state for a predetermined amount of time. The exact length of this time depends on the parameters of the bus. The time should not be so large as to add significantly to the packet latency and it should not be so small as to flood the bus with a large number of unsuccessful reads of SM. If the ALP finds one or more waiting DRs, it reads the oldest one.

3.2.3.4. Data Sufficiency

Before discussing AL transmission processing, the concept of *data sufficiency* is explained. The two methods of achieving data sufficiency are also detailed. As mentioned in Section 3.2.3.1, data sufficiency is achieved for a particular destination class. This means that a decision has been taken by the AL to send some or all of the accumulated upper layer packets belonging to a particular destination class. Determining data sufficiency is governed by two conflicting goals. To reduce per packet latency, the upper layer packets should be sent as soon as possible. But to increase throughput, the size of the physical channel frame, *i.e.*, the lower layer packet, should be made as close as possible to the maximum allowed frame. This ensures that the overhead bits (headers and code) are a smaller percentage of the entire frame.

Two strategies are adopted, in parallel, to address the tradeoff between the two goals of maximizing throughput and minimizing latency. These strategies are based on threshold sizes and timers.

Threshold sizes attempt to meet the first goal of maximizing throughput. Consider that the AL is using P channels for a particular destination. The maximum size of an AL packet is fixed by the particular physical channel. Since the ratio of data to code bits varies with the value of CCC m, there is a maximum size for the data in an AL packet, for each value of m. Relative to these maximum sizes, *threshold sizes* are specified for the data in an AL packet, for each value of m. These sizes are at most equal to the corresponding maximum data size for a given m and physical channel. Let $TSIZE$ be the threshold size for a particular destination class, T be the number of accumulated packets of that destination class, and S_1, S_2, ...,S_T, be the sizes of these packets. Then if

$$\sum_{j=1}^{T} S_j \geq P \times TSIZE$$

data sufficiency is said to be achieved for that destination class. This method is called *threshold data sufficiency*. Threshold sizes are chosen to be close to the maximum data sizes, so as to maximize throughput. An upper layer packet cannot be split across coding sets, so all available upper layer packets need not necessarily be sent when threshold data sufficiency is achieved. Let *MAXSIZE* be the maximum data size for the above destination class. Then the first *M* packets, $M \leq T$, such that

$$\sum_{j=1}^{M+1} S_j > MAXSIZE \times P,$$

are marked for transmission. Note that it is possible that

$$\sum_{j=1}^{M} S_j < P \times TSIZE.$$

This occurs when

$$\sum_{j=1}^{T-1} S_j < TSIZE \times P \text{ and } \sum_{j=1}^{T} S_j > MAXSIZE \times P.$$

In this case, $M = (T - 1)$. The ideal value for *TSIZE*, with respect to maximizing throughput, is clearly *MAXSIZE*.

Timers are needed to minimize latency. Every time the AL reads a DR for a new upper layer packet, it checks if threshold data sufficiency has been achieved for the destination class to which this packet belongs. If not, the AL checks if this is the only packet in its destination class currently accumulated in the SM. If so, the AL starts a timer. The timer value is a function of the maximum packet latency specifications. When more DRs belonging to the same destination class are received, the AL checks for threshold data sufficiency after each one, but does not start another timer. Hence, there is at most one timer for each destination class. If threshold data sufficiency is achieved for a destination class before the timer for that destination class expires, the timer is reset if all available packets are not marked for transmission, or disabled if all packets are used. However, if the timer for a destination class expires before threshold data sufficiency is achieved, all packets belonging to that destination class are marked for transmission. Using the above terminology,

$$\sum_{j=1}^{T} S_j < TSIZE \times P,$$

because threshold sufficiency failed for the same packets. This implies that all T packets can always be used when a timer expires. Whenever a timer expires, data sufficiency is said to be achieved. This method is called *timer data sufficiency.* If *TIMR* is the timer setting for some destination class, then the latency suffered by a packet of this destination class waiting for AL processing can be at most *TIMR*. Hence, the use of timers gives a degree of control over the maximum latency that an upper layer packet can suffer. The best timer setting, with respect to latency alone, is zero, since this will cause the packets to be processed immediately.

The goals of maximizing channel utilization and minimizing latency are conflicting, because the longer a packet is made to wait the greater the possibility that threshold data sufficiency will be met. However, the latency suffered by that packet will also be greater. By incorporating mechanisms to address both goals, albeit separately, the AL protocol provides the capacity to customize the performance of the AL for different applications which may have different needs.

3.2.3.5. Addressing in the AL

Every node has only one AL entity and this AL entity is given an address which is unique across the network. This address is purely for identification purposes and does not directly represent a physical address. The address is called the *virtual channel address* (VCA) of the node. The AL protocol specifies a 48-bit VCA. This represents a very large address space, which may appear unnecessary. However, 48 bits is a standard address size for IEEE 802 protocols [5], and using the same size for the AL simplifies address translation. Another motivation for a long address is that the address space could be divided among several sub-nets, with each sub-net having the use of a specific range of addresses. Even though some sub-nets may not use all of the addresses in their allotted range, none of those addresses will be available for another sub-net. A large address range enables the allotment of a generous range to each sub-net, which ensures that availability of addresses will not constrain future growth. For a particular packet the *source virtual channel address* (SVCA) is just the VCA of the source node, while the *destination virtual channel address* (DVCA) is the VCA of the destination node.

Channel addresses (CAs) are the addresses of the nodes on each of the physical channels to which they are connected. Consider a node N_i which is connected to P physical channels. Let the CA of node N_i on physical channel p be $CA_{i,p}$. The node has physical addresses $CA_{i,1}$, $CA_{i,2}$,..., $CA_{i,p}$,..., $CA_{i,P}$, on the P channels. The size of each of these addresses depends on the type of physical channel used. For a particular packet transmitted on channel p, the *source channel address* (SCA) is the CA of the source node on channel p, while the *destination channel address* (DCA) is the CA of the destination node on channel p.

For data transmission, VCAs must be translated to CAs. The reverse takes place during reception. Both translations are fairly simple table look-up procedures. If an AL packet is to be sent on channel p, from node N_i to node N_k, the SVCA will be translated to $CA_{i,p}$, and the DVCA will be translated to $CA_{k,p}$. For a frame received at node N_i from node N_k, the SCA will be translated to the VCA of node N_k and the DCA will be translated to the VCA of node N_i.

The above translation scheme implies that the AL at every node in a network has full knowledge of the connectivity of the network. This means that every node in the network knows, to which channels each node in the network is connected to, the CAs on each of those channels and the VCAs of the nodes. Whenever a new node is added to the network, the AL at the new node broadcasts a special packet. This packet has an address reserved for broadcasts as the destination address. The existing ALs read this packet and find a new CA in the SCA field. This is an indication of the addition of a new node. The rest of the packet has the VCA and all the CAs of the new node. The size of this packet will depend on the size of the CAs and the number of channels to which the new node is connected. The existing ALs update their translation tables, and then send a single packet to the new node, containing their VCA and CAs. The AL at the new node uses these packets to create its translation tables.

One alternative to storing complete connectivity information is using an *address resolution protocol* (ARP). If an ARP is used the AL needs to store only a subset of the full connectivity information, which it uses frequently. Whenever it encounters a VCA which it cannot translate, it sends a request on the network for resolving the VCA, in accordance with the particular ARP. An AL entity at some other node receiving this request and having the required translation information will send the same to the requesting AL. The use of an ARP requires less space at each AL for storing address translation tables. However, if a large number of ARP messages have to be sent, the average latency of address translations could increase.

3.2.3.6. AL Processing

The ALP classifies each DR it reads according to its destination class. The DEST_CLASS field of a DR indicates the destination class of the upper layer packet for which it was sent. The ALP then does the processing required for the data sufficiency algorithms. It waits until sufficiency is achieved in either fashion. Once data sufficiency is achieved, the ALP begins the process of building the AL packets. For the purpose of the following discussion, it is assumed that U upper layer packets, ULP_1, $ULP_2, \ldots$, $ULP_u, \ldots$, ULP_U, are marked for transmission on P physical channels. Let S_1, $S_2, \ldots, S_u, \ldots S_U$ be the sizes of the U packets. One AL packet is created for each channel and these P packets form one coding set.

3.2.3.6.1. Creation of AL Headers

The first step in the creation of AL packets is to split the U upper layer packets marked for transmission among the P AL packets such that all P are equal in size. The size of an AL packet is the sum of the sizes of the AL header, the upper layer data and the CCC bits. Depending on the actual number of upper layer packets used for a given AL packet, the AL headers for different packets in the same coding set can be different. Hence, the amount of data in different packets of the coding set can also be different. The splitting algorithm is presented in Section 3.2.3.6.2. The input to the algorithm is the sizes of the U upper layer packets and the output is a specification of how these packets are to be split among the P AL packets.

The SVCA and DVCA of the DRs of all the upper layer packets marked for transmission are identical. These are translated to the SCA and DCA for each AL packet as explained in Section 3.2.3.5. The SERV_CLASS field, as mentioned earlier, contains two sub-fields: the *channel* sub-field and the *m* sub-field. The upper twelve bits form the channel sub-field, while the lower nibble forms the *m* sub-field. Figure 3.5 shows these sub-fields.

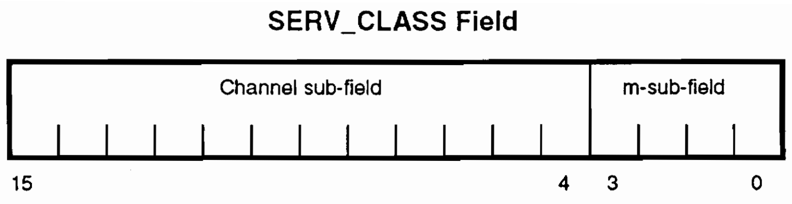

Figure 3.5. Sub-fields of the SERV_CLASS field.

The channel sub-field is bit-coded and indicates the channels to be used for this transmission. The channels are assumed to be numbered uniquely, network wide. The main implication of the channel sub-field is that a maximum of twelve physical channels can be used in a network. Hence, in the above case, $P \le 12$. Let

$$C_{n_1}, C_{n_2}, \ldots, C_{n_p}, \ldots, C_{n_P}, \quad 1 \le n_1 < n_2 < \ldots < n_p < \ldots < n_P \le 12,$$

be the P channels for this coding set. Then bits

$$n_1 + 3,\ n_2 + 3, \ldots,\ n_p + 3, \ldots, \text{ and } n_P + 3$$

of the SERV_CLASS field will be set. The other bits of the channel sub-field will be cleared. The m sub-field has the value of the CCC parameter m needed for this coding set. Since one nibble is used, m values up to 16 can be specified, which is more than sufficient. The CSET field contains the unique identification number allocated for this coding set. The SIZE field is determined by the splitting algorithm and is equal for all the AL packets of this coding set. The header size of the different packets can be different, so and the fraction of upper layer data in the different AL packets could be different.

The COMP and SPLIT_PAT fields are built using the results of the splitting algorithm. Let the splitting algorithm specify that data in a particular AL packet be from J upper layer packets, where $J \leq U$. For any upper layer packet, ULP_u, $1 \leq u \leq U$, belonging to this set of J packets, referred to as the *J-Set*, the AL packet has bytes

$$ULP_u[b_u] \text{ to } ULP_u[e_u], \quad 1 \leq b_u < e_u \leq S_u,$$

of this packet. The upper layer packets are implicitly ordered based on their unique DR ID fields. The ordering is such that, for any l, q, $l < q \leq U$, the DR ID of ULP_l is less than the DR ID of ULP_q. The bit-coded COMP field is now built. The AL packets in this coding set will use the least significant U bits of their COMP field to indicate the presence or absence of data from each of the U upper layer packets. Any bit t - 1, $1 \leq t \leq U$, if set, implies that there is some data from ULP_t. Note that bit 0 corresponds to packet 1. If t - 1 is cleared, then there is no data from ULP_t. For the AL packet considered above, bit t - 1 of its COMP field, for all t such that ULP_t belongs to the *J-Set* are set. The other bits are cleared.

The SPLIT_PAT field uses four bytes for each upper layer packet whose data is used for this AL packet. Two of these bytes are used for the beginning byte position and two are used for the ending byte position. For the data from ULP_u these will be b_u and e_u respectively. The SPLIT_PAT field will occupy $4J$ bytes. The implicit packet ordering is used once again in building the SPLIT_PAT field. Let the packets in the *J-Set* for the AL packet be $ULP_{u_1}, ULP_{u_2}, \ldots, ULP_{u_j}, \ldots, ULP_{u_J}$. Then the SPLIT_PAT field will be:

$$e_{u_J}, b_{u_J}, e_{u_{J-1}}, b_{u_{J-1}}, \ldots, e_{u_j}, b_{u_j}, \ldots, e_{u_1}, b_{u_1}.$$

Example 3.1. Suppose 4 AL packets have to be created using 6 upper layer packets. Then U = 6 and P = 4. The upper layer packets are ULP_1, $ULP_2, \ldots, ULP_6$. The subscripts are based on the implicit ordering. Let the *J-set* for one of the AL packets be $\{ULP_2, ULP_3, ULP_4\}$. Let the splitting algorithm specify that the AL packet has bytes 20 to 55 from ULP_2, bytes 0 to 700 from ULP_3 and bytes 0 to 10 from ULP_4. Then the SPLIT_PAT for the AL packet is 10,0,700,0,55,20. The COMP field has bits 1, 2 and 3, corresponding to ULP_2, ULP_3, and ULP_4, set. All other bits are cleared. The resulting COMP field is shown in Figure 3.6.

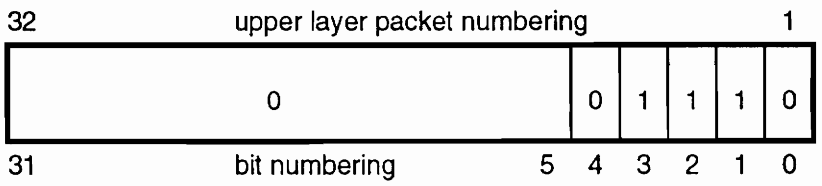

Figure 3.6. COMP field for Example 3.1.

Let the value of CCC parameter m be 1 and let the four channels being used be 1, 2, 3, and 4. Then the SERV_CLASS field is as shown in Figure 3.7.

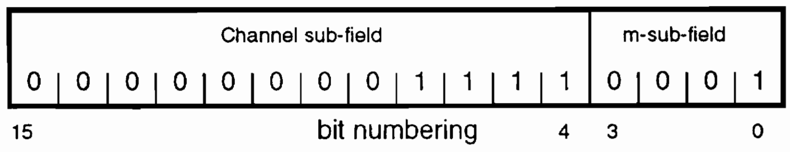

Figure 3.7. SERV_CLASS field for Example 3.1.

3.2.3.6.2. The Splitting Algorithm

The input to the splitting algorithm is the set of U sizes of the upper layer packets to be used for this coding set, the number of channels to be used, P, and the value of CCC m required. In the context of this algorithm, the term *packets,* unless qualified by a prefix, will refer to the combination of upper layer data, AL header and padding, if any. These packets undergo coding to form the AL packets. The CCC procedure requires that the data in each AL packet undergoing coding be split into some d equal sized parts. Hence, the final size of each AL packet must be divisible by d. Another way of stating this requirement is that the sum of the sizes of all the P packets must be divisible by Pd. The value of d depends on the relative values of P and m. For instance, if $P = 4$ and $m = 1$, the size of each of the 4 packets must be divisible by 3, or the total size of all 4 packets must be divisible by 12. The second specification to the splitting algorithm is that all packets be of equal size. This stems from the nature of CCC, where parts of different packets are coded together.

The algorithm is basically an iterative procedure. Each iteration starts with a target size T for the packets. Each iteration attempts to use data from the U upper layer packets such that the size of every packet (data and AL header) equals T. If it succeeds, then the algorithm is complete; if not, another iteration is made. The key obviously is to choose a different value for T for each iteration. The notations to be used in describing the algorithm are now presented. Let Q_k be the sum of the sizes of the U upper layer packets and the P AL headers after the kth iteration. The minimum size header of 34 bytes is referred to as MIN_HDR. Let Pad be the amount, in bytes, of padding. Let T_k be the target size for iteration k. Let A_p, $1 \leq p \leq P$, be the AL packet being built and let c_p be the variable representing the current size of A_p. Let ULP_u be the upper layer packet being used and let s_u, $0 \leq s_u \leq S_u$ be the size of the data available for A_p from ULP_u. Each iteration begins with $p = 1$, and $u = 1$. The starting size of each AL packet is c_p = MIN_HDR. (The symbol "$\leftarrow$" is used as the assignment operator.)

There are a few points to be noted regarding the SPLIT_PAT field. Whenever the current upper layer packet is unable to satisfy the size requirement of the AL packet being built, data from the next upper layer packet has be used. Four bytes are then added to the SPLIT_PAT field of the AL header, and hence to the AL packet, for specifying the amount of data from the next upper layer packet. This can be seen in the first "Else" clause in step 4 in the description below. The COMP field bit corresponding to the next upper layer packet is also set. One possibility is that the AL packet achieves the required size because of these four bytes. Then, no data from the next upper layer packet is actually used. However, the four bytes are retained in the SPLIT_PAT field to satisfy the size requirement. The beginning and ending byte positions contained in these four bytes will be specified as "-1" to indicate that zero bytes are included. The second possibility is that the size requirement of the AL packet is exceeded when the four SPLIT_PAT bytes are added. If the target size is exceeded by r bytes, $1 \le r \le 3$, the last r bytes of the current upper layer packet are not used for this AL packet and are available for the next AL packet. In this case, also, the four bytes added to the SPLIT_PAT field are retained. These two situations are taken care of in the two "Else" clauses of step 5 in the description below. The corresponding COMP field bit remains set in both these situations because the COMP field is used to determine the size of the AL header at the receiver. The splitting algorithm is now described in seven steps.

1. $T_1 \leftarrow (Q_1 + (Q_1 \bmod Pd)) / P$.

2. $u \leftarrow 1$, $s_1 \leftarrow S_1, p \leftarrow 1$, $c_1 \leftarrow$ MIN_HDR and go to step 4.

3. If $u > U, p = P$, then $Pad \leftarrow T_k - c_p$, $c_p \leftarrow T_k$, and go to step 6. Else go to step 4.

4. If $c_p + s_u = T_k$, then $c_p \leftarrow T_k$, $u \leftarrow u+1$, $s_u \leftarrow S_u$ and go to step 6.
   Else if $c_p + s_u < T_k$, then $c_p \leftarrow c_p + s_u + 4$, $u \leftarrow u+1$, $s_u \leftarrow S_u$, and go to step 5.
   Else if $c_p + s_u > T_k$, then $s_u \leftarrow s_u - (T_k - c_p)$, $c_p \leftarrow T_k$, and go to step 6.

5. If $c_p = T_k$, then go to step 6.
   Else if $c_p > T_k$, then $u \leftarrow u-1$, $s_u \leftarrow c_p - T_k$, $c_p \leftarrow T_k$, and go to step 6.
   Else go to step 3.

6. If $p < P$, then $p \leftarrow p+1$, $c_p \leftarrow$ MIN_HDR, and go to step 3.
   Else if $p = P$ and (($u < U$) or ($u = U$ and $s_u > 0$)), go to step 7.
   Else DONE.

7. Calculate target size for new iteration: $T_{k+1} \leftarrow (Q_k + (Q_k \bmod Pd)) / P$ and go to step 2.

Whenever any part of an upper layer packet is assigned to an AL packet, the index of the starting and ending bytes of that part are recorded in a data structure. The contents of this data structure are returned after the algorithm completes. Note that all U upper layer packets are completely used for this coding set. Another point to note is that in step 3, $u > U$, implies that $p = P$. This means that all the upper layer data is exhausted and Ap is still not complete. Padding is resorted to at this point to complete this last packet. The value of *Pad* is not explicitly returned. The manner in which this is handled is explained in Section 3.2.3.6.3. Note that with $u > U$, $p = P$ since iteration k was started with T_k bytes available for each AL packet, and the only reduction was due to reduction of some SPLIT_PAT sizes during this iteration. For p to be less than P when $u > U$, the reduction has to be more than T_k, which is not possible.

3.2.3.6.3. Construction of AL Packets

After headers are created for all P AL packets, the construction of the AL packets begins. The leading portion of the AL packets consisting of the AL header, upper layer data and padding, if any, is built up in the pre-CCC buffers. The CCC unit is aligned with the channels and its implementation is discussed in Section 3.2.3.6.4. Each channel has a pre-CCC buffer. The AL headers are first transferred to the respective buffers over the common bus. The upper layer data from each of the U packets is transferred to the respective AL packets in accordance with the results of the splitting algorithm. The sequence of the transfers follows the implicit upper layer packet ordering. Hence, if an AL packet has bytes b_q to e_q of ULP_q and bytes b_r to e_r of ULP_r, and $q < r$, the data from ULP_q is transferred to the AL packet, being built in the pre-CCC buffers, before the data from ULP_r. The data transfers are scheduled by the ALP. Since these transfers may take a significant amount of time, the ALP can perform other tasks while they are in progress. The transfer of padding, if any, to an AL packet is scheduled after all the data for the packet has been transferred. From the discussion of the splitting algorithm in Section 3.2.3.6.2, it can be seen that padding, if any, will only be for the last AL packet of a coding set. The amount of padding is not returned explicitly by the algorithm. The AL packets are built from the available upper layer packets according to the specifications of the algorithm. If all the upper layer packets have been used and the last AL packet is still incomplete in terms of its size, it is completed with padding data. Padding is transferred to the pre-CCC buffers so that all packets can explicitly achieve the required equal size. This is important since the ALP is not monitoring the transfer of the data from the SM. When the equal size condition is achieved the CCC unit signals the ALP that the packets are ready for coding.

Once the packets are ready for coding, the ALP initiates the CCC process by sending the required control signals to the CCC unit. The CCC unit then proceeds with the coding independent of the ALP. The ALP can now perform other tasks.

3.2.3.6.4. Cross Channel Coding

Coding is done in accordance to predetermined expressions. These expressions depend on the values of P and CCC parameter m. Both software and hardware implementations are possible, although given the simple nature of the coding, hardware solutions may be preferable. In fact, hardware implementations may be essential for the speeds that are required. In that case, the control of the CCC procedure consists simply of properly directing the data through arrays of exclusive-OR gates.

The input to the CCC unit is the leading portion, *i.e.*, header and upper layer data, of each AL packet. The outputs are the completed AL packets. The CCC bits are appended to the end of the input portion of each packet to create the complete AL packet. The output buffers of the CCC unit also function as the MAC buffers of the physical channels. Hence, after the CCC procedure is complete, the AL packets are available to the physical channels.

3.2.3.7. Transfer onto Physical Channels

The MAC at each physical channel adds its own header to the AL packets to create a lower layer frame. The header typically includes error detection bits. The lower layer frames contend for access to the physical medium in accordance with the particular data link protocols. The packets are eventually delivered to the destination node and reception processing starts.

3.2.4. Reception of Data

In the context of the AL, reception processing includes breaking up the incoming AL packets into the original upper layer packets and delivering these packets to the upper layer. The AL may also have to regenerate some of the AL packets using *cross channel decoding* (CCD). Section 3.2.4.1 presents an overview of the reception protocol. This is followed by a detailed description of the functionality of the AL during reception.

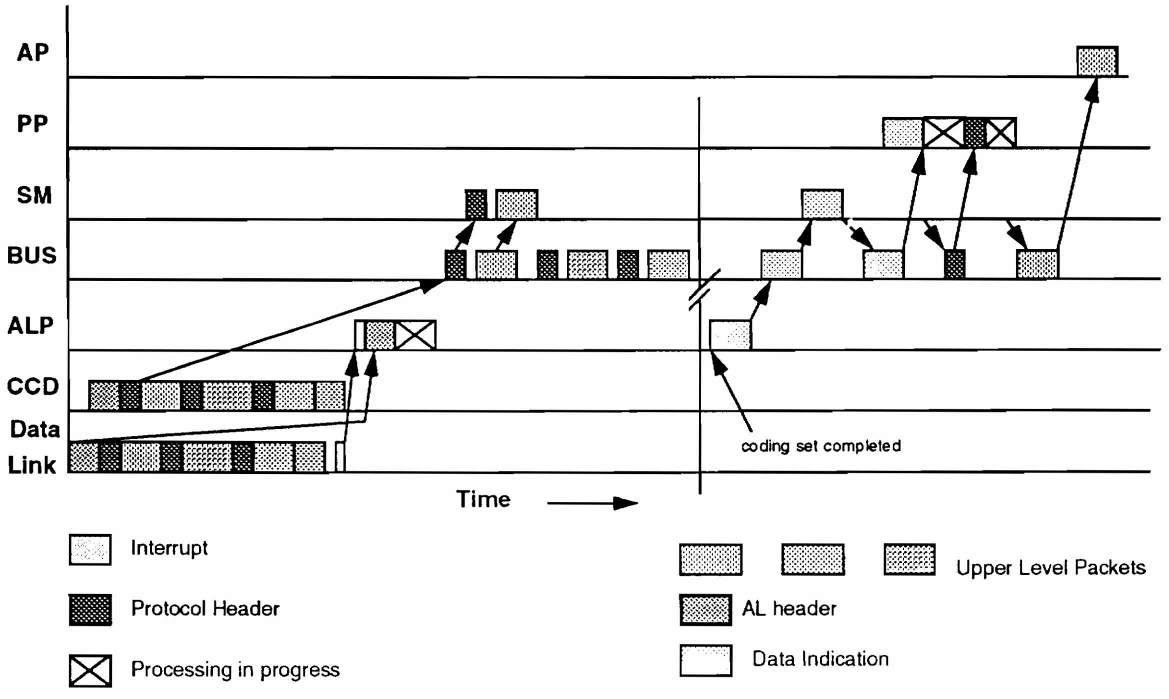

Figure 3.8. Graphical representation of data reception.

3.2.4.1. Overview of the Reception Protocol

Figure 3.8 shows a graphical representation of the steps involved in data reception. The process starts at the bottom left corner of the figure with the reception of a new frame over the physical channels. The MAC entity then issues an interrupt to the AL. The AL processes incoming packets in response to these interrupts issued by the MAC entities at the lower layer. The first step is to recover the AL header from the new packet. This process is fairly straight forward and is detailed in Section 3.2.4.3.1. Once the header is recovered, the ALP knows the coding set, from the CSET field, to which this AL packet belongs. From the channel sub-field of the SERV_CLASS field, the total number of AL packets in this coding set is determined. For every coding set and source node combination, the ALP maintains a list of the arrived packets. This list is maintained until all the packets of the coding set are either received, rebuilt or simply discarded. This is discussed in Section 3.2.4.3.2. Upon receipt of a new packet, the ALP extracts and analyzes the AL header. The number of upper layer packets included is determined. The correct parts of the packet are then placed in appropriate upper layer packets being built in the SM. This is indicated in Figure 3.8 by the arrow showing the data going straight from the CCD unit buffers to the SM via the bus. The ALP then checks if the packet has caused any coding set to be completed. If it has, DIs are created for the upper layer and written to the SM. If the number of missing packets is equal to the CCC parameter *m*, determined from the *m* sub-field of the SERV_CLASS field, a CCD timer is started. CCD timers are discussed in Section 3.2.4.3.6.

If a CCD timer triggers, the ALP initiates CCD to rebuild the missing *m* packets of the coding set for which the timer was started. The rebuilt packets are also re-organized into their constituent upper layer packets. The AL packets from each coding set are maintained in the pre-CCD buffers until the entire coding set is received. Once the DIs for all the upper layer packets of the coding set are transferred to the SM, the AL reception processing is complete. In the figure, the break in the time scale indicates the wait for the coding set to be complete. The DIs, one of which is shown in the figure, are created after this and transferred to the SM via the common bus. The figure then shows the DI being read and processed by a PP. The upper right corner of the figure shows the original data packet arriving at the application processor.

3.2.4.2. Physical Layer to AL

Lower layer frames are delivered by the physical medium to the destination node. The MAC entity of each channel strips its header from the incoming packet. The error detection code in the header is used to check for any detectable errors in the packet. If any are found the packet is typically discarded and the processing of the next packet is started. Otherwise, the packet is buffered in the CCD buffers and an interrupt is issued to the ALP to indicate the arrival of a new packet.

3.2.4.3. AL Processing

The interrupts issued by the MAC entities are of a very low priority. The interrupts are merely queued and do not actually interrupt any task of the ALP. However, the interrupt queue is checked after the ALP completes every task. This approach ensures that arrivals are processed fairly expeditiously, while keeping the AL processing simple. The first step is to recover the AL header.

3.2.4.3.1. Recovery of AL Header

A low data rate bus is provided between the CCD buffers and the ALP. This bus is used to read the leading 30 bytes of each new packet. These bytes contain the first seven fields of the AL header. The COMP field is then used to recover the SPLIT_PAT field. Let the COMP field have J bits set, $1 \leq J \leq 32$. Then the SPLIT_PAT field occupies $4J$ bytes. These are copied by the ALP to complete the recovery of the header.

3.2.4.3.2. The Coding Set and Source Node List

A coding set is identified by the CSET field in the AL header. The CSET number is common to all the AL packets which are part of the same coding set. Since the same coding set number can be used at different nodes, a packet cannot be uniquely identified only by its coding set number. To uniquely identify the coding set of received packets, the combination of CSET and SVCA are used. A list of the different CSET-SVCA combinations received are maintained at each node. Whenever a packet belonging to a new CSET-SVCA combination arrives, an entry is added to this list. The list also contains information regarding the number of AL packets received in each coding set and the channels on which they arrived, using the channel ordering. This information is maintained as a simple bit-encoded field, *ARRVD,* which is similar to the channel sub-field of the SERV_CLASS field.

Entries are deleted from this list for two reasons: either the coding set is complete, with some packets possibly being rebuilt using CCD, or the entry has timed out. The second case is discussed first. Every entry in the CSET-SVCA field has a time stamp as to when it was added to the list. The ALP checks this time stamp at regular intervals. If the entry persists after some predetermined maximum waiting time, it is deleted, and all the packets from that coding set are assumed to be lost. The packets of this CSET-SVCA combination that did arrive are deleted from the CCD buffers. Any data from them, transferred to the SM are overwritten by the ALP during subsequent transfers and no DIs are sent to the upper layers. The alternative to creating time stamps is to overwrite old packets in the CCD buffer when the buffer overflows. In this case, however, it will be difficult to identify which packets, if any, can be overwritten in the SM.

Let *n* be the total number of AL packets in a given coding set. Then this coding set can be completed in two ways: either all *n* packets are received error free, or at least *n* - *m* are received error free and at most *m* are rebuilt using CCD. If either of these cases occurs, the corresponding entry in the list is deleted after the creation of the DIs. The space occupied by these packets in the CCD buffers is then available for other packets.

3.2.4.3.3. AL Header Analysis

After the ALP recovers the complete header, it analyzes some of the fields. The SCA is translated to the SVCA, as detailed in Section 3.2.3.5, and the ALP checks if the CSET-SVCA combination has been previously entered in the coding set-source node list. If it has been, the *ARRVD* field of the entry is updated by setting the bit corresponding to the channel on which this packet arrived. The channel ordering used in building the channel sub-field of the SERV_CLASS field of AL headers is used here. If this SVCA-CSET combination has not been entered, the entry is made now and the *ARRVD* field is initialized by first clearing all bits and then setting the bit corresponding to the channel on which this AL packet arrived.

The ALP initiates the reconstruction of the constituent upper layer packets by enqueueing correct parts of the AL packet for writing to the SM. The *ARRVD* field is then compared to the channel sub-field of the SERV_CLASS field. If the number of AL packets that have not arrived is greater than the *m* used for this coding set, the ALP does nothing more at this point. If the number not arrived is exactly *m*, a CCD timer is started. The use of this timer and criteria for determining its setting are discussed in Section 3.2.4.3.6. Basically, when a CCD timer expires, the ALP starts the packet regeneration process.

If the *ARRVD* and channel sub-fields match exactly, then all the AL packets of this coding set have arrived. The ALP now creates DIs for all the upper layer packets contained in this coding set. Since no upper layer packet is split over more than one coding set, the processing of all the AL packets of a coding set ensures that all the constituent upper layer packets have been completely received.

3.2.4.3.4. Reconstruction of Upper Layer Packets

Irrespective of the actual number of packets that have been received from a particular coding set, the data contained in each AL packet is always forwarded to the SM. The entire data is not forwarded as a single transfer; instead, parts of the packet which belong to different upper layer packets are separately transferred. For some AL packet, let *J* bits of the COMP field of the AL header be set and let the SPLIT_PAT field be

$$e_{u_J}, b_{u_J}, e_{u_{J-1}}, b_{u_{J-1}}, \ldots, e_{u_j}, b_{u_j}, \ldots, e_{u_1}, b_{u_1}.$$

The notation is the same one used in the discussion on AL header construction in Section 3.2.3.6.1. Let *k*, $1 \le k \le 32$, be the least significant bit that is set in the COMP field. Then, bytes $ULP\left[b_{u_1}\right]$ to $ULP\left[e_{u_1}\right]$ from upper layer packet *k*, are contained in this AL packet. This numbering of the upper layer packets is a result of the implicit ordering used during transmission. The ALP then forwards the proper chunk of data for each upper layer packet to the SM. The storage has to be allocated such that parts of the same upper layer packet in other AL packets can be stored to SM in the correct sequence. During this upper layer packet reconstruction process, the ALP also totals the amount of data received for each upper layer packet. When all AL packets have been processed, this sum will represent the size of the upper layer packet. Any data left over in the AL packet after all chunks, as specified by the SPLIT_PAT, have been transferred, is discarded, since it is padding.

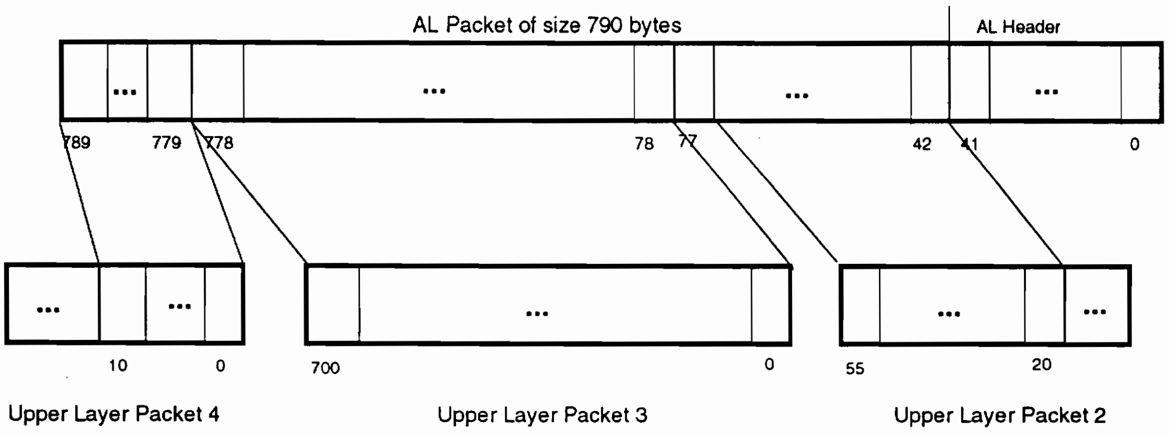

Figure 3.9. Reconstruction of upper layer packets in Example 3.2.

Example 3.2. The AL packet considered in Example 3.1 is used here. The COMP field (Figure 3.6), shows the use of ULP_2, ULP_3 and ULP_4, since bits 1, 2 and 3 are set. This implies that the SPLIT_PAT field is 12 bytes long. Hence, the AL header size is 42 bytes (34 + 8) long. From Example 3.1 the SPLIT_PAT field for this packet is 10,0,700,0,55,20. Then byte 42 to 77 of this AL packet are written as byte 20 to 55 of ULP_2 in the SM. Byte 78 to 778 are written as byte 0 to 700 of ULP_3 and byte 779 to 789 are written as byte 0 to 10 of ULP_4. Figure 3.9 shows this graphically.

3.2.4.3.5. Building DIs

The DIs for each of the reconstructed upper layer packets are built in a straight-forward manner. The ID field is filled with a unique number assigned by the ALP. This ID is used by the PPs for transferring all the data associated with a DI to a particular application. The SVCA and DVCA are filled by address translation. The DATA_PTR field is filled with the address of the respective upper layer packet in SM. Since the packet was written to SM by the ALP, this address is readily available. Each DI after creation is transferred to the SM. This completes the reception processing done by the AL in the typical case. The processing done when a CCD timer expires is detailed in the next section.

3.2.4.3.6. CCD Timers

A CCD timer is started for a particular CSET-SVCA combination when exactly n-m packets from this coding set have arrived, where m is the number of packets that can be rebuilt by CCD in this coding set. Since the packets are processed by the AL sequentially, a timer is started for every CSET-SVCA combination, except when $m = 0$. If the remaining m packets arrive before the timer expires, the timer is disabled. If the timer expires, the missing packets, which are at most m, are rebuilt using CCD. An interrupt is issued to the ALP by the CCD unit for each rebuilt packet. These packets are then processed by the ALP similarly to the packets that arrive without error over the physical channels. The need for CCD timers is first discussed. Some guidelines for determining their settings are then presented.

In general, CCD can be used to build the last m packets of a coding set as soon as n - m packets from that coding set are received. However, it is highly likely that some or all of the last m packets have arrived, or will be arriving very shortly, on other channels. Since the AL processes packets sequentially, it is advisable to allow some time for the AL to process other packets, and perhaps complete the coding set, before starting CCD. The CCD timers are used for this purpose. The frame size on a typical physical channel is large enough so that it is faster to process the packet arrival, rather than rebuild the packet using CCD. However, there is a need to limit the maximum time that the AL waits for the last m packets and this limit forms the timer setting.

The settings for the CCD timers depend on the typical amount of time required for AL to process the remaining *m* packets of the coding set. This will also have to take into consideration any skew between the channels. In general, the larger the timer setting, the smaller will be the number of rebuilt packets. This is because the longer the AL waits before starting CCD, the greater the probability that the packet may arrive over one of the channels and, hence, not need reconstruction. However, the greater the waiting time, the greater the latency suffered by the constituent upper layer packets, especially if some of the last *m* packets do not arrive and CCD is used. A smaller number of re-built packets also results in a smaller ***rebuilt list*** (see Section 3.2.4.3.8). A smaller timer setting may give lower latency, especially in situations where CCD has to be resorted to often. However, the number of rebuilt packets could become very large and contention for the CCD unit could become an issue.

3.2.4.3.7. CCD

CCD can be implemented in software, perhaps as a process running on the ALP, or in hardware. Given its simplicity, a hardware implementation is attractive. The CCD unit requires arrays of XOR gates like the CCC unit. An implementation could time multiplex several large arrays of XOR gates for both the CCC and CCD units.

3.2.4.3.8. Handling Very Late Arriving Packets

The use of CCD to rebuild some packets imposes an additional responsibility on the AL: that of recognizing a rebuilt packet, if it later arrives over a physical channel. In this context a *very late arriving packet* is one which arrives after CCD has been used to rebuild it. After the rebuilding process, the relevant entry in the CSET-SVCA list is deleted. If the late arriving packet is not identified, it will result in a fresh entry in the

CSET-SVCA list. To avoid this, a similar list is maintained for rebuilt packets. This is called the *rebuilt list.* When a new packet arrives, this list is first searched to check if the coding set to which this new packet belongs has been completed by using CCD. If it has been, the packet is discarded. Entries in this list can be deleted on an as needed basis. When there is no space for a new entry, the earliest entry in the list is deleted. Then the onus is on ensuring that the maximum size of this list is large enough, so that after an entry is deleted, the chances that it will be needed are extremely low.

3.2.4.4. Transport/Network Layer to Application Layer

The PPs read the DIs from the SM and begin processing the new upper layer packet. They use the PP header contained in each packet to perform the transport/network layer protocol functions. The data, once received correctly, is forwarded to the proper application over the common bus.

3.3. Implications of CCC

A point to note about CCC is that it is only an *error correcting code.* It cannot detect errors. This implies that there must be some other scheme to detect errors. The only error the AL can detect is the non-arrival of a packet. This can occur under two situations: either the packet actually does not arrive over the channels or the packet arrived, but was not delivered to the AL by the MAC entity. Most data link layer protocols use some kind of coding for error detection. When a packet is received at the destination node, this code is used to check for errors. If any are found, the packet is discarded. This appears as a non-arrival of the packet to the AL. There are two situations in which the MAC error detection code will not work. One is that the number of errors is greater than the capacity of the code. There is very little that can be done in this case, except to increase the capacity of the code. The second is that the error occurs during the transfer from AL to physical channels or physical channels to AL. However, this possibility is remote since the post-CCC buffers serve as the MAC transmit buffers and the pre-CCD buffers serve as the MAC receive buffers. In other words, data is not moved in the AL except when it is being sent to the SM or read from it. This small error probability does not justify the time and computational penalty to be paid for adding an error detection code at the AL.

CCC is designed for systems where data is transferred in parallel, so it is, in a sense, the ideal coding scheme for the proposed parallel network. It makes use of the fact that data is being sent to the same destination over parallel channels. When CCC is used, code bits are added to every channel and there is no separate *code channel.* Hence, all the packets in a coding set are identical as far as rebuilding capacity is concerned. When packets are received correctly, no special processing is required to recover the data bits. After the AL header has been analyzed, the data portion of the AL packet, which form the leading bytes, are forwarded to the SM, while the code portion is discarded.

The cost of CCC is the loss in throughput due to transmission of the rather large amount of code bits that CCC mandates. This has to be balanced with the fact that the use of CCC allows up to m channel failures and may reduce the number of retransmissions that a system must request.

The AL views all the packets in one coding set as one data block. Hence, if the AL could request retransmission, it would do so for the entire coding set, even if only one packet of the coding set did not arrive. This is a severe penalty, especially if the error rates are high. In fact, if the error rates become very high, the use of CCC may actually increase the throughput because of reduced retransmission requests from the PP layer. The impact of the use of CCC has been studied analytically by Wiencko [2]. For the case of an optical fiber medium, it has been shown that the retransmission rate will fall significantly because of the use of CCC.

The other capability afforded by CCC is that of graceful degradation. If one physical channel fails completely, the AL can continuously rebuild data on that channel by making *m* equal to at least one. Meanwhile, the AL can inform the upper layers of a loss in capacity and request a reduction in the arrival rate. The user will experience a certain degree of reduced throughput, but will not lose any data.

A significant implication of CCC is that only packets of equal size can undergo coding together. This means that the data being sent in parallel has to be divided into equal parts before the coding can be done. This splitting problem is complicated by the fact that the AL header size is variable. Taking care of this requirement is one of the major tasks of the AL and is completely transparent to the other layers. One simple solution would be to impose the equal-sized packet condition on the upper layer packets. However, this would interfere with the transport/network protocols and defeat the idea of a layered architecture.

The packet recovery ability requires the use of timers at the receiving end. Information about rebuilt packets has to be maintained so that late arriving packets are not unnecessarily processed. Note that this is not required for the typical error correcting code, since it cannot rebuild entire packets. Any packet received at the AL is immediately forwarded to the SM. However, a copy is maintained in the CCD buffers since it may be needed for rebuilding some packets. This implies that buffers have to be larger than those in a standard system.

Therefore, CCC has advantages and disadvantages. The main advantage is increased reliability and reduced packet latency in high error rate environments, while the disadvantages are an increase in complexity of the AL protocol and reduced throughput in low error rate situations.

Chapter 4. A Simulator for the Access Layer

This chapter presents a simulator for the AL. The simulator is designed to study the effects of the AL on the end-to-end performance of the proposed parallel network and to verify the functionality of the AL protocol. The simulation model is fine-grained and most of the AL functionality is simulated. The simulator also incorporates limited models for the other network layers. The objectives of the simulator are presented first. The simulation scheme is then discussed. This is followed by discussions of the models for the various system entities, timing calculations and finally the implementation of the system functionality.

4.1. Objectives of the Simulator

The objectives of the simulator are two-fold: to perform a functional verification of the AL protocol and to study the performance limits of a parallel network using such an AL. Functional verification involves ensuring that the defined functionality of the AL is sufficient to achieve the goals of the AL. For this verification, the simulator was made fine-grained, as far as the AL is concerned. The entire functionality of the AL protocol, except handling of late arriving packets and certain other special cases, is simulated in software. Data packets are formed, headers are created and attached to the packet, cross channel coding is done and transmission on physical channels is simulated. The common bus protocol is also simulated. At the receiving end, the headers are stripped from the packets, analyzed and the data is suitably delivered to the upper layers. The creation of such a simulator helped clarify the definition of the AL protocol, while its successful execution gives greater confidence in the correctness of the protocol.

The second goal was to study the performance limits of a network incorporating such an AL. Since end-to-end performance is of interest and since the behavior of the other layers affects the performance of the AL, all other layers of the network are also simulated, albeit in a limited way. The main quantities of interest are the data latency, data throughput and the utilization of various system resources. State and timing information is captured at various points to measure these quantities. These measurement methods, the study itself and the results obtained are discussed in the next chapter.

4.2. Simulation Scheme

The choice for the type of simulator and the design approach are discussed in this section. The section concludes with a high-level discussion of the simulator.

4.2.1. Type of Simulator

In any network simulation project, there are always three options: either to use some standard package such as NETWORK II [4], to use a simulation language like SLAM, or to custom design the simulator. Each approach has its advantages and disadvantages. Standard packages and simulation languages are generally geared towards performance estimation and not so much to functional simulation. They are also more suitable for standard (serial) architectures. Modifying them for a new kind of architecture is either not possible or would involve significant amount of programming.

For the task at hand, simulation of the AL, the custom design approach was considered more suitable. In the first place, it provides a great amount of flexibility and imposes no constraints on the simulator design. The greater degree of detail required for functional verification can be incorporated. Finally and perhaps most importantly, a simulator whose code was well understood and completely accessible was considered ideal for this project.

4.2.2. Design Approach

Keeping in view the objective of functional verification, the key design approach was to model an actual implementation as closely as possible. In a sense, a "literal" model was created, as opposed to a "mathematical" model. The question that was asked before modeling any function is, *"How would this be done in an implementation?"* The answer was then used as a guide to modeling the function. Several times a conceptually simple solution would turn out to be highly complex to implement and less intuitive solutions had to be found. As a result the process of defining the AL protocol and writing the simulator often ran in parallel.

4.2.3. High-Level View of the Simulator

The AL simulator was designed as an event-driven simulator [23]. This implies that, after start up, no event is unilaterally scheduled; future events are scheduled only by currently executing events. A single global event queue is maintained for the entire system. The event queue is a singly linked list and each record in the list has three parameters: the network node at which this event is to take place, the time at which it is to take place and the type of event. All events are processed by a function call, so the event field has a pointer to the relevant function. A separate routine is provided to schedule events. The above three parameters are passed to the scheduling function. In addition, relevant global data structures are updated to provide data for the event when it is executed. The scheduling function creates a new record containing the parameters that it has been passed and places this record in the global event queue in time sorted order. If there are other events in the queue which are to occur at exactly the same time as this new event, the new event is placed immediately after all such events. The primary loop of the simulator is simply:

```
While (Event_Queue NOT Empty)
    Begin
        Current_time = ( First_Node (Event.Queue) ).Time;
        Return_Value = ( First_Node (Event.Queue) ).Event();
        If (Return_Value = Error)
            Exit (Error);
        else
            Delete ( First_Node (Event.Queue) );
    End;
```

There are two concepts of time in the simulator: real time and simulation time. Real time corresponds to the time actually taken by the simulation on a host machine, while simulation time (*Current_time* in the above pseudo-code) is the theoretical time for which the system being simulated has run. Typically, real time is several orders of magnitude larger than simulation time. Real time depends on the host processor and the quality of the software model. While it is advantageous to reduce real time, it has no impact on the results of the simulation as long as steady state is achieved. Hence, a great deal of attention was not paid to lowering real time. Simulation or system time runs in parallel for the different nodes in the network and for the different network layers in each node, so simulation time must advance only when it has advanced for all layers in all nodes in the network. This means that if simulation time is to be advanced from T_l to $T_l + t$ then there must be no pending event at any layer, in any node, which occurs before $T_l + t$. To ensure this, the event queue is time-sorted without regard to which node or layer the event is supposed to occur.

4.3. Choice of Components

Performance studies depend on the parameters of the components of the system. While generic processors are sufficient assumptions for the PPs and the ALP, specific choices have to be made for the common bus, the data link layer standard, and the CCC/CCD units.

4.3.1. The Common Bus

The M Bus standard [24] designed by Sun Microsystems was chosen for the common bus. This is a high speed bus, capable of peak transfer rates of 2,560 Mbps. The M Bus standard permits multiple bus masters and is designed primarily for multiprocessing applications. It has protocol support for cache memory and an arbitration unit which is implementation dependent. The combination of speed and features makes it a good choice for the common bus. The M Bus standard is augmented with a locked-bus cycle. During a locked-bus cycle, the bus master can perform two consecutive transfers on the M Bus, usually with some time for data processing in between, without relinquishing the bus to other potential masters. This feature is necessary for preventing duplicate processing of the same data by different processors. Specifically in the AL protocol, this is necessary for the PPs to process data requests from the application layer and data indications from the AL. Apart from this addition, the M Bus protocol is modeled as defined in the standard.

While the M Bus functionality is large, only the necessary subset of functions is implemented.

4.3.2. Data Link Layer Standard

The Fiber Distributed Data Interface (FDDI) standard [17] was chosen for the physical channels. FDDI is the first fiber-based local area networking standard [20] and is a 100 Mbps LAN. High transmission speed and low error rates make optical fibers the natural choice for the next generation of computer networks. FDDI represents one order of magnitude increase in throughput over the Ethernet standard. FDDI is a strictly packet switched LAN, although one enhancement, called FDDI-II, proposes to integrate circuit switched operation with the traditional packet switched operation. FDDI uses a ring topology, with a typical network having two counter-rotating rings. If a break occurs in one of the rings, it can wrap around and combine with the other ring to form one ring. This ability increases the reliability of FDDI networks. Stations that are connected to both rings are called *dual attach stations* (DAS), while those connected to only one of the rings are called *single attach stations* (SAS). FDDI uses a timed token protocol. The protocol guarantees that the token will arrive at each station within some maximum interval. This maximum interval is called the *target token rotation time* (TTRT), and is negotiated by all the nodes when the network is initialized. This timed token protocol is essential for time-critical and synchronous applications.

Numerous studies have characterized FDDI performance [21, 22]. A number of suggestions for the application of FDDI have also been presented [18, 19, 20]. Given the objective of designing a gigabit network, the high throughput of FDDI channels was attractive. Using FDDI meant that fewer physical channels would be needed. For these reasons, choosing FDDI for the physical channels was fairly appropriate.

4.3.3. CCC/CCD Units

Cross channel coding and decoding consist only of exclusive-OR operations and can be implemented in either software or hardware. Given the speed requirements, a hardware implementation was modeled. In terms of the simulation, this choice impacted only the time estimated for cross channel coding and decoding.

4.4. Models for System Components

The simulation models for the various physical system components are discussed in this section.

4.4.1. Application Layer Model

The application layer is modeled as processes that generate packets according to the arrival distribution and transfer them over the common bus to the SM. These transfers are followed by the creation and transfer of data requests to the PPs. The various choices for destination node, reliability and size are made in accordance with pre-specified distributions for those quantities. At the receiving end, no particular processing is done, except to tabulate the amount of data received. The simulation code for these processes monitor data latency and throughput.

4.4.2. Processor (PPs and ALP) Model

All processors are modeled as a collection of subroutines. Each routine will perform some part of the functionality required of the processors. Most of these routines are events that can be scheduled in the global event queue, while the rest are called by these events. Global data structures are used to determine the data to operate on when one of these events occur. The ordering of the entries in these data structures is significant. In the general case, fresh entries are appended to the end of the list, while the one at the head of the list is processed next.

Both the PPs and the ALP have an *idle* routine. This routine, assumed to take zero, or negligible, simulation time, is called at the end of every indivisible processing task. The various queues are checked for fresh data in a prescribed order. The checking itself takes finite time. If there is no data to be processed, these routines schedule themselves to occur after some time. Thus these routines form a kind of looping that is internal to the simulator.

Context switching has not been considered for any of the processors and all of them are assumed to have sufficient local cache memory.

4.4.3. M Bus

The M Bus by itself is not modeled. The only model is that of the M Bus queue and the arbitration for access to the bus. The latency involved in M Bus transfers is also modeled. Since a number of resources are connected to the M Bus there must be an arbitration scheme to decide the next bus master. For the simulation, round robin access is used. Also, all potential bus masters are assumed to have equal priority. The M Bus protocol does have provision for different priority levels, which could be useful when time-critical data like control information must be transferred. To ensure fair access for all of the connected resources, the M Bus permits a single transfer to be a maximum of 128 bytes. Data blocks greater than 128 bytes require multiple transfers. To facilitate this, data packets are maintained as linked lists of records which have at most 128 bytes of data. The M Bus queue functions much like the global event queue except that there is a separate M Bus queue at each node. After each transfer, the head node of the M Bus queue is deleted and the next master is selected after arbitration. If there are no entries in the queue, the M Bus goes into an idle state and waits for a fresh request to be enqueued.

The M Bus model has two associated enqueueing routines. One is for data that is already in 128 or smaller byte chunks and the other is for data that is in bulk form. To enqueue a new request, the source, destination and the size of the transfer have to be supplied to the relevant enqueueing function. Due to the special nature of the bulk enqueue, it requires some additional information. Note that these enqueueing functions, like the arbitration scheme, are not part of the M Bus protocol and were defined for this simulation.

4.4.4. Shared Memory Model

There are two options for simulating memory. The first is to declare a large byte array and treat this array like the actual memory in a real system. In this case, the location of any given datum will be identified by its address. The other option is to allocate memory as it is required. In this case, there is no single entity representing the memory. Instead, several separate memory blocks are used for the different types of data stored in the memory. The main reasons for modeling SM are, first, to simulate its affect (storage) and, secondly, to get an estimate of the required size of this memory. Both these objectives are met by the more efficient, second option. The SM is maintained as a set of singly linked lists. Each list corresponds to a different type of data, e.g., DRs, DIs, and actual data. Since all the different types of data are maintained in a common linked list format at the different layers, transferring data in the model is a simple matter of detaching and attaching records. This makes for a more efficient implementation.

4.4.5. FDDI Model

Physical entities have two effects on data: they can transfer the data to another physical location, and they can process the data, usually changing it in some way. Both of these operations involve finite latencies, which are not necessarily sequential. The transfer latency depends on the properties of the physical medium. The change in the data could be desirable (processing) or undesirable (errors). When a communication channel such as FDDI is the physical entity, the transfer latency is a very significant parameter, while any change in the data is usually due to the appearance of errors and is hence undesirable. The model for the FDDI rings takes into account the latency involved in data transfer. Since the AL protocol does not allow for re-transmission of missing packets, the only effect of errors on the FDDI rings is to cause the packets to be rebuilt by CCD, if the number of errors are within the CCC capacity. This functionality was separately verified during the simulation. Any errors will affect the performance results, but for a meaningful study, a good error model for the parallel channels is needed. This is beyond the scope of this study.

In a real FDDI system, the stations on the ring agree upon a value for target token rotation time (TTRT) when the network is initialized. In the simulation, the value of TTRT is pre-specified. When any station releases the token, it puts the TTRT value into a timer called the token holding time (THT) timer and starts the timer. When the token arrives again, the station can hold on to it for what ever time is left on the THT timer. If the value is zero or negative, then the station has no time slot during this rotation of the token. The token rotation and THT are modeled in a literal manner. Whenever a station releases a token, either due to lack of data to transmit or because the THT has run out, it schedules a token arrival at its downstream neighbor after the latency for token transfer. THT is calculated by recording the time at which the token was last released and comparing this value with the time when the token is next received. Each of the FDDI channels in the network are separately simulated. As in the case of all system components, other than the AL, FDDI rings are simulated only to the extent of their effects on the data. In the case of FDDI, this happens to be the latency that it introduces.

4.5. Timing Assumptions

Estimates must be made for the time required for the execution of the various operations. These estimates have a significant impact on the performance results, especially utilization. Estimates were made for some key functions. To keep the simulator as general as possible, the number of instructions required for these functions was estimated. These values are then be multiplied by the time/instruction rating of any processor to get a time estimate for that processor. All time estimates used for the simulation are provided in Appendix B.

4.5.1. M Bus

The latency involved in M Bus transfers has a significant impact on the performance results since all data and control information has to pass over the bus. The time for M Bus transfers is deterministic and does not have to be estimated. The peak rate possible, 2,560 Mbps, was used for all transfers and error free operation was assumed. Each M Bus cycle transfers a 64-bit word, or 8 bytes, in 25 ns. Larger transfers take integral multiples of 25 ns. There is a dead cycle, of 25 ns, between every change of bus master, to avoid contention problems.

4.5.2. FDDI

The latency in FDDI transfers is also assumed to be deterministic. The peak data rate of 100 Mbps was used, and error-free operation was assumed. It takes 80 ns to transmit one byte at this rate. A token of 11 bytes [22] is used for each ring. The token latency is, therefore, 880 ns. The TTRT for the network was set at 2 ms, which is recommended in [22]. The link lengths were assumed to be 30 meters, for which an average propagation delay of 15 μs was used [21].

4.5.3. Protocol Processors

The time-critical tasks performed by the PPs are the processing of data requests from the application layer to create PP headers and build upper layer packets, and the processing of data indications from the AL to deliver received data to the application layers. The time for each of these tasks depends on the particular transport/network protocol being used. For the simulation, TCP/IP was assumed to be used. Hence, timing estimates for the two critical PP tasks were based on the results of a study of the TCP/IP protocol suite [25]. Transmission processing was estimated to take 250 instructions, while reception takes 200 instructions. The time to build a data request is included in the time for transmission processing.

The *idle* process (see Section 4.4.2) on the PPs schedule themselves to occur after a pre-determined period of time, if no data is available for processing at this time. If this time is too small, the M Bus could be flooded with too many unsuccessful reads of the SM. If it is too large, fresh data may wait unnecessarily long in the SM. For the simulations, the value of this time was set at that required for 175 instructions. The chosen value is a tradeoff between the two extremes.

4.4.4. Access Layer Processor

The critical tasks at the ALP are transmission and reception processing, timer checking and the *idle* routine. Transmission processing at the ALP has two major processing steps. Each DR from the upper layers has to be analyzed. When data sufficiency is achieved, the splitting algorithm has to be run and the AL packets have to be built. Processing a received AL packet is viewed as a single process for time estimates.

Processing DRs, running the splitting algorithm and then building the AL packets is similar to TCP transmission processing. However, some extra per-DR processing is needed for data sufficiency checks. Processing for each DR is estimated to take 200 instructions. Each iteration of the splitting algorithm was estimated to take 40 instructions. Taking an average of 2.5 iterations for each coding set, the splitting procedure is estimated at 100 instructions for each coding set. Finally, building the AL headers involves similar steps as building the TCP headers. Since this task is a sub-set of the TCP functionality, it is estimated to take 100 instructions. Reception processing is estimated to take 150 instructions. Timer checking is estimated to take 20 instructions. This estimate is also based on the results in [25]. Based on the criteria used for the PPs, the *idle* routine period for the ALP is also estimated to take 175 instructions.

While the estimates are fairly crude, an attempt was made to err on the side of caution by choosing larger numbers. In fact, the authors of [12] estimate that TCP should take fewer instructions than those presented in [25].

Time estimates for ALP and PP functions do not have as much impact on the performance results as might initially appear. The latency of the FDDI channels dominates the performance results (see Chapter 5).

4.4.5. CCC/CCD Units

A model of a hardware implementation of the cross channel coding and decoding units was used for the simulation. Both CCC and CCD units require arrays of XOR gates. These gates are estimated to cause a per-byte latency of 1 ns for both coding and decoding.

4.6. Implementation of Node Functions

In general, the simulator implements the different processing steps in a literal manner. This section provides some specific information on the implementation of some of the important tasks at each node. Details of the AL functions were discussed in Chapter 3.

4.6.1. Transfer of Data from Application to Transport/Network Layers

Data is written by application processors to SM. This is followed by a data request (DR′) to the PPs, which represent the transport and network layers, to transmit this data. The DR′ contains the size of the data block and its destination node. There may be other information also, such as desired reliability level, priority, etc. The DR′ is read and processed by one of the PPs.

All the PPs access the SM over the M Bus. The various potential bus-masters are given access on a round-robin basis. During each access, the master can perform either one read or one write. This creates a potential problem with DR′. Suppose PP_1 reads DR′ during the current M Bus transfer. After analyzing the DR′, PP_1 has to write to the SM either updating the size field of the DR′, if it is going to use only part of the data, or deleting the DR′, if it is going to use all the data. However, to do this, PP_1 has to enqueue for M Bus access, which it will acquire only after round-robin arbitration. During this waiting period, it is possible for another PP to read the same DR′ from the SM. To prevent this situation, a locked bus cycle is added to the M Bus functionality. A PP initiates a read to the SM for a new DR′ as a locked-bus cycle. During the first phase, it reads the first DR′ that is available. If none are present, the PP releases the bus. If a DR′ is found, the second phase of the locked-bus cycle is started. During this phase, the PP analyzes the DR′. The M Bus remains idle during this phase. The final phase is after the analysis when the PP writes back to the SM, either updating or deleting the DR′. The M Bus is released after this third phase.

The addition of the locked-bus cycle eliminates potential duplicate processing by the different PPs. The facility for using only part of the data block sent by the application layer is significant. This permits the data block size sent by the application processors to be independent of the particular transport/network protocols. This independence is important for achieving layered operation.

4.6.2. Data Sufficiency

Checking for threshold data sufficiency is done in an obvious manner. Every time a new DR is read, a function is called to check if threshold sufficiency has been achieved for the destination class to which the DR belongs. If it is, the ALP checks if all of the accumulated upper layer packets of that destination class can be used. As explained in Section 3.2.3.4, the last packet may not be usable since its addition may violate the maximum AL packet size. The splitting function is then called.

After the ALP reads a DR, it checks if this is the only DR in its destination class. If so, a timer is started. Starting a timer implies scheduling a timer event, which is entered in the global event queue. If *curr_time* is the current simulation time, and the timer value is *TIMR_VAL*, the timer event is scheduled to occur at time *curr_time* + *TIMR_VAL*. When scheduling the timer an entry is made into an associated data structure. This entry contains the destination class for which the timer is being scheduled. The time at which the timer is to expire, *curr_time* + *TIMR_VAL*, is also stored in this structure. This is used to identify the entry when the timer expires. Potentially different timer values are used for different destination classes. If the timer expires, all available packets of this destination class are marked for transmission. This is always possible, as explained in Section 3.2.3.4. However, if threshold sufficiency is achieved, the timer has to be disabled or reset. Since the timer is merely another process in the global event queue, it is not easy to access the relevant entry in the event queue. A separate field is added to the timer data structure to handle this. This field is called the *active flag* and is either true or false. Initially, when the timer event is scheduled, the flag is set to true. Later, if threshold sufficiency is achieved, this flag is set to false. The relevant entry in the timer data structure is easy to locate since there is at most one active entry for each destination class. When a timer event occurs, the related active flag is checked. If it is false, the process is aborted. If not, timer data sufficiency is declared.

As mentioned, it is possible for the last packet not to be used when threshold sufficiency is declared. For this situation the AL protocol specifies that the related timer be reset. This means that the existing timer event must be removed from the global event queue and re-scheduled. There is again the same problem of accessing a specific event in the event queue. The approach used is to first disable the active flag of the existing entry and then schedule a new timer event. In this situation there are really two timer events for the same destination class, but only one of them is active.

4.6.3. The Splitting Algorithm

The splitting process is implemented in two phases. The function implementing the first phase takes the sizes of the upper layer packets to be used, the value of CCC parameter *m* and the number of AL packets to be formed as input. This function implements the algorithm presented in Section 3.2.3.6.2. During each iteration, the upper layer packets are split in potentially different ways. This splitting is, however, not stored. The only result required from this function is a target size for the AL packets which uses all the available upper layer data. Hence, during the internal looping in each iteration of the algorithm, the only information maintained is the AL packet number and the size of data allotted to this packet. When this size reaches the target size of the iteration, the current internal loop terminates and a new one is started for the next AL packet.

The second phase function takes, as input, the target size returned by the first phase, in addition to all parameters passed to the first phase function. This function does one single iteration of the splitting algorithm since it uses the target size passed to it. During this iteration, the splitting patterns of the upper layer packets are stored as nodes in a linked list. Each record identifies a chunk of data from a particular upper layer packet, for a particular AL packet. Each record, therefore, has the AL packet number, the DR ID of the upper layer packet, and the starting and ending indices of the chunk of data to be used. This linked list is then used to build the AL packets.

The splitting process is done in two phases to ease implementation. If the splitting patterns were maintained during the first phase, memory for the list records would have to be allocated and freed during each iteration.

4.6.4. M Bus: Enqueueing and Transfers

The maximum size of a single M Bus transfer is 128 bytes. If a data block greater than 128 bytes has to be transferred, multiple M Bus transfers are required. The bus master requesting this transfer has to contend for M Bus access for each transfer of 128 or fewer bytes. Two issues must be considered: the manner in which data is maintained and the manner in which a transfer is enqueued for M Bus access.

Data is maintained using linked lists. Every entity except packets is less than 128 bytes in length. Hence, the entire entity can be transferred over the M Bus in a single transfer. Data packets are often greater than 128 bytes in length. To facilitate easy M Bus transfers, each packet is split into several list records, each of which has at most 128 bytes of data. Consider a packet of size S bytes. If $S \leq 128$, it is stored as a single data record.

If $S > 128$, the packet is stored as $\left\lceil S/_{128} \right\rceil$ records, where the last record may contain less than 128 bytes. This scheme makes data transfer very simple using the enqueueing method described below. However, if only parts of the packet are to be read or written, additional processing is needed. Each list record contains the start and end indices of the data block that it contains. These have to be used and adjusted during partial transfers. Separate enqueueing and transfer functions are used for such partial transfers.

The standard enqueueing function is discussed first. The M Bus queue is a singly linked list. Each record represents the request for the transfer of at most 128 bytes. All the information required for the actual transfer, such as source, destination, type and size is available in each record. This information is passed to the enqueueing function. Let *TR_SIZE* be the size, in bytes, of the requested transfer. If $TR\_SIZE \leq 128$, then one M Bus queue record is created and appended to the end of the M Bus queue list. The fields of the new record are filled with the parameters passed to the enqueue function. If TR_SIZE > 128, at most $\left\lceil TR\_SIZE/_{128} \right\rceil$ transfers of 128 bytes each are scheduled. If (*TR_SIZE* mod 128) > 0, the last transfer is of only (*TR_SIZE* mod 128) bytes. Because of the manner in which data is stored, there is no need to calculate start or end points for each transfer. For each transfer, the data specified by the first record is transferred. This is the typical form of enqueueing and data transfer.

The other case is that of a partial transfer. This occurs when data is not stored in the 128-byte format at either the source or destination point. Typical cases are when data is transferred to or from the AL. When an AL packet is being built, it can have data from several upper layer packets. Since the AL packet has to undergo CCC, it has to be stored as a single array. A separate *partial enqueue* function is used to enqueue such transfers. It is similar to the regular enqueue function discussed above, except that it uses additional parameters to specify the start and end indices of the data to be moved. The destination resource has to be qualified with a specific packet number. For example, if an AL packet is being built, the destination is one of the CCC buffers. However, since a number of packets could potentially be in the buffers, the ID of the particular packet being built also has to be passed. When upper layer packets are being reconstructed at the receiver, the SM is the general destination and the specific packet being built is the particular destination.

Separate functions are used to transfer different types of data. Except for partial transfers, all transfers involve detaching a record from the source list and attaching it to the destination list. For partial transfers, the specified amount of data is deleted from the source and written to the destination. In the case of SM to CCC buffer transfers, this means using as many of the records in the SM list as required. It is quite possible that the data from a particular record is only partially used. The start, end and size fields of such records have to be properly adjusted. This data is then written into the array which represents the CCC buffer. When upper layer packets are being re-constructed, the data from the CCD buffers is stored using records that contain 128 or fewer bytes.

Round robin arbitration is used for granting M Bus access. All potential masters are assumed to be arranged in circular order. Suppose there are four masters, *A*, *B*, *C* and *D*. Let the round-robin order be *A B C D A*. Let the current master be *C*. After this transfer is over, the M Bus controller checks if any request has been queued by *D*. If it has, this request is transferred to the head of the M Bus queue and processed as the next transfer. If there is no request by *D*, the queue is checked for requests from *A*, *B* and *C* in that order. The first master which has a request is then given access. If the current master is *B*, the checking order will be *C*, *D*, *A* and finally *B*.

4.6.5. FDDI Token Rotation

A token of 11 bytes is assumed for each FDDI ring [22]. The basic property of this token is that it takes finite time to go from one node to another. This time is purely overhead as far as FDDI utilization is concerned. Ideally, control should pass instantaneously from one node to another. Token rotation is simulated by scheduling the arrival of the token at the downstream node. Suppose the node which currently has the token has a frame of S bytes to transmit. Assuming that sufficient time is available on the token holding time (THT) timer, this packet is transmitted in SR seconds, where R is the transmission rate in seconds/byte on the physical medium. For the peak FDDI rate of 100 Mbps, this works out to be 80 ns for one byte. Ignoring propagation delay, this frame arrives at the downstream node after SR seconds. If this is the only packet to be sent, or if sufficient time is not available to transmit another packet, the token is transmitted immediately after this packet. The transmission takes $11R$ seconds since a token is 11 bytes long. A *token arrival* event is scheduled in the global event queue, to occur at *curr_time* + SR + $11R$ seconds, at the downstream node, where *curr_time* is the current simulation time.

When a token arrival event occurs, the transmit buffers at the node at which the token arrives are checked. If the node has data to transmit on the channel, it uses the value in its THT timer to determine how long it can retain the token. The node then sends as many complete frames as possible. The token is then transmitted to the downstream node. It should be noted that the tokens on each FDDI ring rotate independently of the tokens for the other rings.

4.6.6. Transfer of Data from AL to Transport and Network Layers

The ALP enqueues the chunks of data belonging to different upper layer packet from each AL packet for transfer to the SM. After the coding set has been completed, DIs are also sent for each upper layer packet being rebuilt. Because of the M Bus arbitration scheme, it is likely that a DI, which needs only one transfer, will reach the SM before all the data for its associated upper layer packet have fully arrived in the SM. Hence, a PP cannot start the processing for the upper layer packet immediately after reading a DI. The data from each AL packet is enqueued for transmission to the SM immediately after the AL has processed the associated AL header. These transfers go on independent of the ALP. Hence, it is very difficult for the ALP to determine if all the data from all the AL packets belonging to a particular coding set have reached the SM. This is the reason for writing the data indications for the all the upper layer packets in a coding set without ensuring that the data has arrived in the SM.

To ensure that a DI is read by only one PP, the locked bus transfer described in Section 4.6.1 is used. The DI is deleted during the write-back phase. The PP then initiates a read to the SM to check if the associated upper layer packet has been completely copied to the SM. During each check, the PP reads the address of the last byte of the upper layer packet that is in memory. The starting address is supplied in the data indication for this upper layer packet. The difference between this address and the starting address and the expected size for the upper layer packet are compared. The PP performs this checking at regular intervals, until it finds the complete packet in the SM. At this point it enqueues on the M Bus to read the PP header. After analyzing the header, it forwards the data to the application layer by means of M Bus transfers. The PP header analysis is not actually implemented, since it is not part of the AL protocol.

Chapter 5. Simulation Experiments and Results

The AL simulator was used to conduct experiments to study the effects of the AL on performance. The basic objective of the experimental study and the quantities measured during each simulation are explained first. Then the system configuration used for all the experiments is described. A "base case" is identified on the basis of the choices to be made for a given simulation. For each subsequent simulation, the value of one of the parameters is varied from that in the base case. These cases are discussed. The experimental results are then presented and analyzed. The chapter concludes by discussing the implications of the results.

5.1. Objective and Measured Quantities

The basic objective of the experiments was to study the performance limits of the system rather than to plot performance curves. This approach was considered suitable for the first study of a new network architecture. Each experiment, therefore, yielded one value for each of the measured quantities. The idea was to obtain guidelines for the choice of parameters for an implementation or even a more complete set of experiments.

The basic quantities measured during each simulation are throughput, latency, utilization, and memory or queue length. The time average of the utilization of the M Bus, ALP and the FDDI channels is calculated. End-to-end throughput and latency are calculated on a per-packet and per-byte basis. The "end points" are the application layers at the source and destination nodes. The average and maximum amount of storage required in the SM are also calculated. The maximum and average sizes of the FDDI transmit and receive queues in terms of number of packets are also calculated.

5.2. System Configuration and Monitoring Methods for Experiments

The particular system configuration used for all experiments is first explained. This is followed by a discussion of the monitoring methods and detection of steady state.

5.2.1. System Configuration

The performance of a four-node network was studied. All four nodes are identical in all respects. Each node has three upper layer protocol processors. Each of these processors has performance of 50 MIPS. Each node has one ALP, also with 50 MIPS throughput. The simulated network used four FDDI rings, with all four nodes connected to all four FDDI rings. The four FDDI rings give a total data transfer capacity of 400 Mbps at the physical layer. The application layer is represented by one processor, whose capacity is assumed to be as large as required. This means that the application layer can perform its functions in zero time. At the transmitting end, the application layer has to generate data according to the arrival distribution chosen for the experiment. It has no function at the receiving end.

The use of four FDDI channels implies that for CCC, n can be at most four. Since all nodes are connected to all channels, n is always four whenever CCC is used. When n is four, m can be zero, one or two.

5.2.2. Monitoring Methods and Steady State Detection

Throughput and latency calculations are made whenever the application layer receives data from one of the protocol processors. Calculation of throughput is a simple matter of adding up the size of data received and averaging this total over the simulation time.

Calculation of latency requires the packet to be specifically identified. This identification also enables data collection for packets according to their size and *m* values. When the data for each packet is created, a unique identifier is used as the leading four bytes of the data. This integer forms the index into a data structure that stores the transmission time, size, and *m* value for the packet. When the packet is received, the identifier is extracted from the data. Since the current time is always known, the latency can be calculated using the contents of the relevant entry in the data structure.

The queue lengths and shared memory sizes are calculated on a continuous basis. Each of these values is maintained as a global variable. Whenever any value changes, the corresponding global variable is updated. The time at which the last change was made is also stored. This value is used to calculate the time average for the queue lengths and SM sizes.

Detection of steady state in network simulations is always a contentious issue [23]. For the experiments two separate checks were used. Both had to succeed simultaneously for the system to be declared in steady state. The first check is that the difference between the throughput for two successive windows of packets has to be less than a predetermined small value, ε_1. The second check is that the difference in packet latency for the two successive windows of packets has to be less that another predetermined small value, ε_2.

For every simulation, steady state was not tested until after 400 packets were processed. This prevented any start-up conditions from being falsely detected as steady state.

5.3. Experimental Cases

The parameters that are varied for the experiments are discussed first. The base case is then presented. This is followed by the ten variations of the base case which were studied.

5.3.1. Parameters Varied for the Study

Five groups of parameters were varied to generate the test cases. These are:

1. **The application layer packet size distribution**: A bimodal distribution was used for the size of the packets at the application layer. The packets could be either *large* or *small*. Large packets were 3,500 bytes and small packets were 100 bytes. The large-packet size was chosen to be close to the threshold size for the $m = 1$ case. The small-packet size was chosen to be smaller than the maximum allowed, single M Bus transfer, 128 bytes. The fraction of large and small packets was varied.

2. **The destination node distribution**: For each node in the network, the percentage of application layer packets that go to each of the other three nodes is specified. This percentage varies between 0, which means no packets are sent to that node, and 100, which means that all packets are sent to that node.

3. **The CCC *m* parameter distribution**: Each network node is connected to four physical channels. This implies that $n = 4$ whenever CCC is used and hence, m can be 0, 1, or 2. The percentage of packets with each value of m is specified. This

percentage varies between 0, meaning no packets have this value of *m*, and 100, meaning all packets have this value of *m*.

4. **Data sufficiency timer setting**: The data sufficiency timers have three possible settings. The first is infinity, meaning timers are not used, the second is zero, meaning timers always determine data sufficiency and the third is a value between the first two. For each experiment only one of these settings was used.

5. **Distributing packets with *m* = 0**: Packets with $m = 0$ do not undergo CCC. Hence, the standard algorithm of dividing available upper layer packets uniformly over all channels does not always have to be used. One alternative is studied in one of the experiments.

One significant parameter which was not varied is the arrival distribution. Saturation arrivals were assumed for all cases. This is to ensure that the system is not idle for lack of data from the application layer. This decision is in keeping with the objective of studying performance limits. The other major assumption was that all packets are delivered error-free by the physical medium. The CCD functionality was tested separately.

5.3.2. Base Case

A base case was established for the experiments. In this case, only two nodes, *node* 1 and *node* 2, are active. *Node* 1 sends 100 percent of its data to *node* 2, while *node* 2 sends back acknowledgments (acks) to *node* 1. *Node* 2 does not transmit any data of its own. This is the limiting case in terms of channel access, since one node has the maximum possible access to the FDDI channels. 50 percent of the application layer packets at *node* 1 are *large*, while the other 50 percent are *small*. 25 percent of the packets have

$m = 0$, 25 percent have $m = 2$, and the remaining 50 percent have $m = 1$. These two choices reflect an estimate of what a typical situation might be.

In the base case, packets with $m = 0$ are divided equally over all four channels like other packets. As a result, the four tokens rotate at the same rate. The threshold sizes are chosen such that they are less than the maximum sizes by less than one small-packet. For instance, if *MAXSIZE* is the maximum size for some destination class, and S_{small} is the size of the small packet (100 bytes), then the threshold size for that destination class, *TSIZE* is such that

$$MAXSIZE - TSIZE < S_{small}.$$

Let T upper layer packets be accumulated for this destination class. Let $S_1, S_2, \ldots, S_t, \ldots, S_T$ be their sizes. Then if

$$\sum_{t=1}^{T} S_t \geq TSIZE \times 4,$$

threshold data sufficiency is declared. For this particular choice of *TSIZE*, there is no benefit in waiting for the next packet to try to come closer to *MAXSIZE* for each channel. This is because,

$$\sum_{t=1}^{T+1} S_t > MAXSIZE \times 4, \text{ even if } S_{T+1} = S_{small}.$$

This gives the benefits of setting the threshold sizes nearly equal to the respective maximum sizes while reducing the latency penalty that such a setting involves. The timer settings for the base case are determined by considering the packet size and m distributions. In both these choices, the knowledge of the other parameter values have been used. Guidelines for making these choices are provided in Section 5.6.

5.3.3. Cases 2 and 3: Packet Size

The packet size distribution is varied in cases 2 and 3. Case 2 has 100 percent large-packets, while case 3 has 100 percent small packets. Case 2 is expected to have higher throughput than the base case because of the better data to overhead ratio. Similar reasoning indicates that case 3 should have lower throughput than the base case.

5.3.4. Cases 4, 5 and 6: Destination

Cases 4, 5, and 6 vary the destination distribution. In case 4, one node sends its data uniformly to the other three nodes, each of which send acknowledgments back to the first node. These three nodes do not transmit any data to each other. In case 5, three nodes send 100 percent of their data to the fourth node, which sends back acknowledgments to the first three nodes, but no other data. In case 6 all four nodes distribute their data uniformly (one-third each) between the other three nodes. Acknowledgments are not sent explicitly and are assumed to be piggy-backed on the regular packets.

Case 4 represents broadcast and case 6 represents uniform traffic distribution. An example for case 5 is a high performance video display receiving inputs from a number of sources.

5.3.5. Cases 7 and 8: CCC m Parameter

The value of CCC parameter m is varied in cases 7 and 8. 100 percent of the packets in case 7 have $m = 1$, while in case 8 all packets have $m = 0$. Case 7 represents a possible general case, while case 8 studies the performance in the absence of CCC. For both cases, data sufficiency should be achieved more quickly than in the base case since all packets belong to the same destination class. This should reduce latency.

5.3.6. Cases 9 and 10: Data Sufficiency

Data sufficiency options are varied in cases 9 and 10. In case 9 the timers are set to infinity. This implies that only threshold data sufficiency is used. In other words, the throughput is maximized at the expense of latency. Case 10 has all timers set to zero. This means that all upper layer packets are sent immediately upon reception. This case minimizes latency at the expense of throughput.

5.3.7. Case 11: Distributing Packets with No CCC

An alternative scheme for distributing packets with $m = 0$ on the physical channels is used for case 11. For packets with $m = 0$, only one AL packet is built at a time and sent on the physical channel with the shortest transmit queue. This causes the tokens to rotate at different rates. The latency for packets with $m = 0$ is expected to be significantly reduced for this case.

The section now concludes with Table 5.1 which shows all the parameters for the base case and cases 2-11.

Table 5.1. Parameters for Each Case

	Pkt size		CCC *m* value			Destination				D.S.	m = 0	
	small	large	0	1	2	A	B	C	D	timer	1	2
Base case	50	50	25	50	25	√				I	√	
Case 2	0	100	25	50	25	√				I	√	
Case 3	100	0	25	50	25	√				I	√	
Case 4	50	50	25	50	25		√			I	√	
Case 5	50	50	25	50	25			√		I	√	
Case 6	50	50	25	50	25				√	I	√	
Case 7	50	50	100	0	0	√				I	√	
Case 8	50	50	0	100	0	√				I	√	
Case 9	50	50	25	50	25	√				∞	√	
Case 10	50	50	25	50	25	√				0	√	
Case 11	50	50	25	50	25	√				I		√

Notes:

1. All numbers in the table represent percentages.
2. Four choices are possible for the destination distribution. These are identified in the table as follows.

   A: 100 percent *Node* 1 to *Node* 2. Acks from *Node 2* to *Node* 1. *Nodes* 3 and 4 idle.

   B: *Node* 1: one-third each to *Nodes* 2, 3, and 4. *Nodes* 2, 3, 4: Acks to *Node* 1.

   C: *Nodes* 1, 3, 4: 100 percent to *Node* 2. Node 2: Acks to *Nodes* 1, 3, and 4.

   D: One-third from each node to other three nodes. No explicit acks from any node.
3. For packets with $m = 0$, option 1 stands for splitting equally on all channels while option 2 represents sending one AL packet at a time, on the least busy channel.
4. "D.S." stands for "data sufficiency." The three options for timer are "∞" for infinity, "0" for zero and "I" for the intermediate setting.

5.4. Experimental Results and Implications

Utilization results from the base case are presented first. Selected results from each set of related experiments are then presented. Each time, the results are discussed relative to those from the base case. The complete listing of results from all experiments is provided in Appendix C.

5.4.1. Base Case: Utilization Results

The utilization of the three key resources, the M Bus, the ALP and the FDDI rings is presented in Table 5.2a and Table 5.2b. The utilization of *nodes* 3 and 4 is not important since these nodes do not send or receive any data. The only processes running at those nodes are the "idle" processes of the ALP and the PPs which check for any data to process at regular intervals.

Table 5.2a. M Bus and ALP Utilization for the Base Case

	Node 1	Node 2	Node 3	Node 4
M Bus	45.57%	43.17%	17.35%	17.35%
ALP	24.68%	24.78%	2.1%	2.1%

Table 5.2b. Utilization of FDDI Rings for the Base Case

FDDI Ring 1	FDDI Ring 2	FDDI Ring 3	FDDI Ring 4
93.43%	93.43%	93.43%	93.43%

Comparing the results in Table 5.2a and Table 5.2b, it is apparent that the bottleneck is not the AL. The network throughput is limited by the physical channel capacity.

5.4.2. Cases 2 and 3: Packet Size

The packet size distribution is varied for cases 2 and 3. The interesting results are the throughput and latency, which are presented in Table 5.3. Figure 5.1a compares the throughput of these cases with that of the base case.

Table 5.3. Throughput and Latency for Base Case and Cases 2 and 3

	Base case	**Case 2**	**Case 3**
Throughput (Mbps)	238.53	246.26	72.05
Latency (ms/pkt)	3.236	3.721	44.718

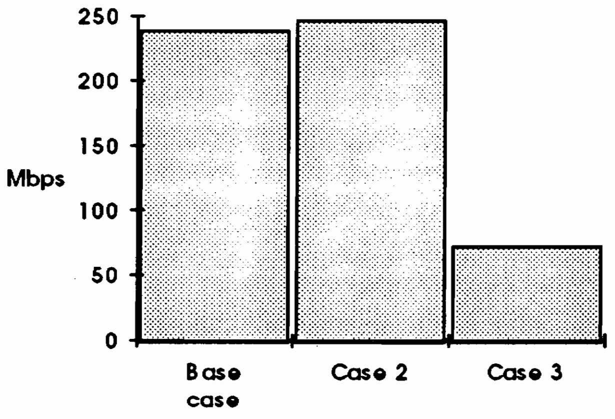

Figure 5.1a. Throughput for base case and cases 2 and 3.

As expected, case 2 has slightly higher throughput than the base case while case 3 has much lower throughput than the base case. Case 3 is discussed further below.

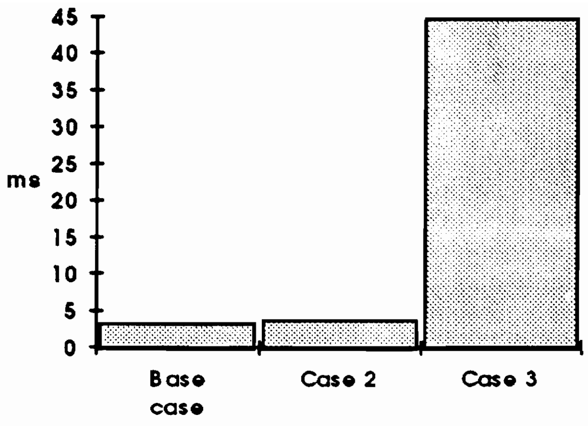

Figure 5.1b. Packet latency for base case and cases 2 and 3.

Figure 5.1b compares the packet latency for the base case and cases 2 and 3. Packet latency for case 2 is slightly greater than that for the base case. This is because of two reasons. One is that there are certain situations in which packets suffer greater latency in case 2 than in the base case. One instance is considered now. Let the total size of the accumulated packets of some destination class be *ACCSIZE*. Let *TSIZE* be the threshold size for this destination class. S_{small} is the size of the small packet and S_{large} is the size of the large packet. Four channels are used and suppose,

$$S_{small} \leq 4TSIZE - ACCSIZE < S_{large}.$$

Since $ACCSIZE < 4TSIZE$, data sufficiency is not declared now. However, the next packet belonging to this destination class, irrespective of its size, will cause threshold sufficiency to be declared. If this last packet is a "small" packet, it will be used immediately in the base case. Hence, the overall latency comes down. In case 2 all packets are "large" and therefore the last packet is not used with the existing packets. Hence there can be no improvement in the average latency of the earlier set of packets. The second reason for greater latency for case 2 is that the average size of packets in case 2 is larger than that for the base case, and larger packets suffer greater latency in all transfers.

One reason for the large increase in per-packet latency for case 3 is that a large number of packets have to be accumulated to come close to the maximum physical frame size. Decreasing the data sufficiency timer setting may be a solution. However, the impact of this on the throughput would need to be considered.

To understand the dramatic drop in throughput for case 3, consider Table 5.4 and Figure 5.2, which show the utilization of the ALP and M Bus at *node* 1 and of all four FDDI rings.

Table 5.4. Utilization Results for Base Case and Cases 2 and 3

	M Bus, node 1	ALP, node 1	All FDDI Rings
Base case	45.57	24.69	93.42
Case 2	41.25	16.38	93.39
Case 3	100.0	100.0	57.78

In the base case and case 2, the FDDI rings are quite clearly the bottleneck. However, in case 3, the FDDI rings are utilized only 57.78 percent, while both the M Bus and the ALP are utilized 100 percent. Hence, in this case, the throughput is limited by the AL. The AL is unable to process enough packets to keep the FDDI rings busy. This explains the low throughput for case 3.

The 100 percent utilization of the ALP in case 3 is also partly responsible for the high packet latency. Larger number of application layer packets are created in case 3 than in the base case because the average packet size is much smaller in case 3. Since all processing times are specified on a per-packet basis, rather than per-byte basis, this implies that the packets in case 3 have to wait longer on the average for ALP service.

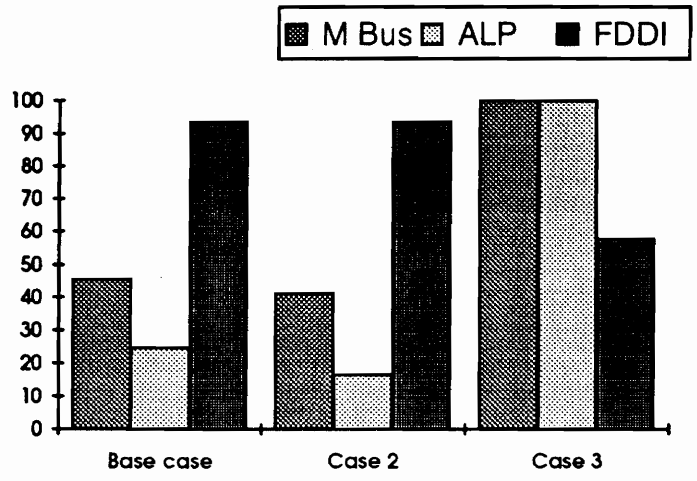

Figure 5.2. Utilization Results for Base Case and Cases 2 and 3.

The throughput results emphasize the advantage of larger sized packets. The surprising result is the increase in packet latency for case 3, since the average packet size for this case is much smaller than that for the base case. One way to alleviate this problem is at the expense of the throughput, by reducing the threshold sizes. In general, however, the AL is quite clearly the bottleneck for case 3.

5.4.3. Cases 4, 5 and 6: Destination Distribution

Case 4, 5, and 6 deal with variations in the destination node distribution. In addition to throughput and latency, the average size of the SM at each node and the FDDI receive and transmit queues for each channel are discussed. Tables 5.5, 5.6a and 5.6b present the results. Figures 5.3a, 5.3b, 5.4a, and 5.4b provide comparisons.

Table 5.5. Throughput and Latency for Base Case and Cases 4, 5 and 6

	Base case	Case 4	Case 5	Case 6
Throughput (Mbps)	238.53	235.62	245.20	254.34
Latency (ms/pkt)	3.236	3.464	8.019	9.221

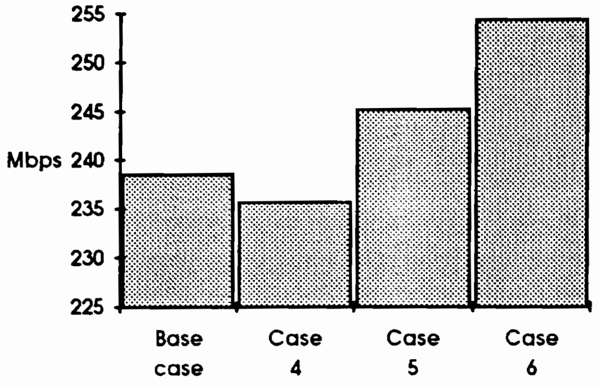

Figure 5.3a. Throughput for base case and cases 4, 5, and 6.

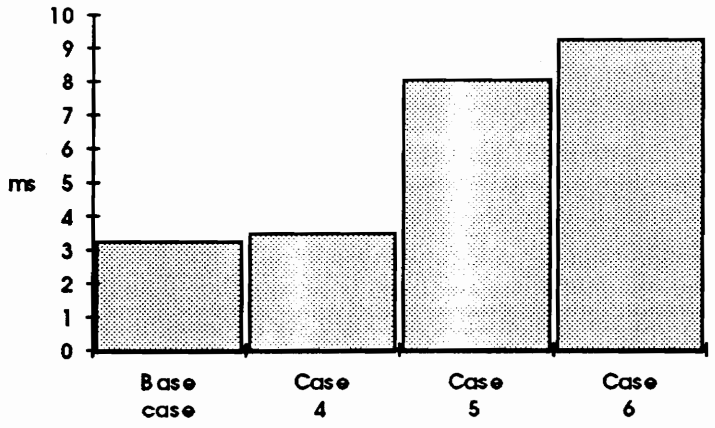

Figure 5.3b. Packet latency for base case and cases 4, 5, and 6.

Table 5.6. Utilization of FDDI Rings for Base Case and Cases 4, 5, and 6

Base case All Rings	Case 4 All Rings	Case 5 All Rings	Case 6 All Rings
93.42	93.27	95.49	95.81

The utilization result of an FDDI ring indicates the fraction of the available throughput (100 Mbps) that has been used for transmitting frames. The remaining has been used for token rotation. Since the token has to rotate at least once every TTRT, some loss in throughput is unavoidable. The token overhead decreases as the average token rotation period approaches TTRT.

The base case and case 4 have almost identical throughput at the channel level. However, the end-to-end throughput for case 4 is less than that for the base case. In both of these cases the same number of packets are being generated at *node* 1. In the base case, all of them are sent to *node* 2. However, in case 4, the packets are uniformly distributed among the other three nodes. Hence, it is rather difficult for threshold data sufficiency to be achieved in case 4, and timer data sufficiency occurs more frequently in case 4, than in the base case. This causes the reduction in throughput for case 4.

In case 5, three nodes are transmitting at full capacity as compared to one in the base case. This implies that each rotation of the token lasts longer in case 5 than in the base case. As a result the overhead due to token rotation is reduced. This results in the greater utilization at the FDDI ring level for case 5. Similar reasoning can be used to explain the FDDI utilization for case 6. In fact, in case 6 all four nodes are transmitting at full capacity and hence, the FDDI utilization is marginally better than for case 5. The greater utilization of the FDDI rings is reflected in the higher throughput for cases 5 and 6.

The traffic due to acknowledgments (acks) does not have much impact. Each ack is 60 bytes long, and hence a large number of them have to be accumulated to create one coding set of four AL packets. This implies that ring utilization by acks is very marginal.

In the base case, there are situations when threshold data sufficiency is achieved before the data sufficiency timer for the particular destination class expires. However, this occurs very rarely, if at all, in case 4, as explained earlier. This causes the average packet latency in case 4 to be higher than that in the base case.

As the utilization of the rings increase, frames at a particular node have to wait longer, on the average, for access to the ring. This causes the higher latency for cases 5 and 6 when compared to the base case.

Table 5.7a. Average Size of the Shared Memory in KB for the Base Case and Cases 4, 5 and 6

	Node 1	**Node 2**	**Node 3**	**Node 4**
Base case	569.79	562.81	0	0
Case 4	525.88	168.31	173.4	173.3
Case 5	183.53	539.96	178.28	183.57
Case 6	139.53	142.54	144.84	141.4

For the base case, *nodes* 3 and 4 do not take part in data transmission or reception. Hence, the required size of shared memory at those nodes is zero. All the data from *node* 1 goes to *node* 2 in the base case. Therefore, the size of the shared memory is approximately equal for both nodes.

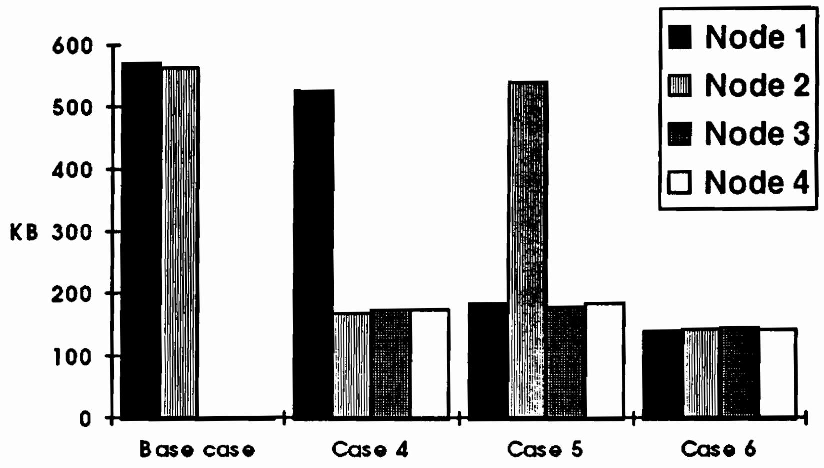

Figure 5.4a. Average size of shared memory for base case and cases 4, 5, and 6.

In case 4, *node* 1 sends one-third of its data to each of the other three nodes. This causes the average size of the shared memory at *nodes* 2, 3 and 4 to be approximately one-third the size of the shared memory at *node* 1. In case 5, *node* 2 receives from the other three nodes. Hence, *node* 2 needs about three times the memory that each of the other nodes need. In case 6, because of uniform traffic distribution, all nodes need almost equal size shared memories.

It can be observed that as the network capacity is distributed among more nodes, the size of the shared memory needed at each node decreases. Further, the total amount of shared memory needed is also reduced. For instance, case 6 needs only half the total shared memory needed for the base case. This reduction is compensated by an increase in the

FDDI transmit and receive queue lengths. The results for these queues are presented in Tables 5.7b and compared in Figures 5.4b and 5.4c. Each node has a separate queue for each FDDI ring. The lengths shown are for each of these queues, at each node.

Table 5.7b. Average Length, in Number of Packets, of FDDI Transmit Queues for the Base Case and Cases 4, 5 and 6

	Node 1	Node 2	Node 3	Node 4
Base case	7.72	1.47	0.0	0.0
Case 4	12.85	0.77	0.78	0.82
Case 5	9.04	2.55	8.91	8.90
Case 6	12.47	13.0	12.46	12.39

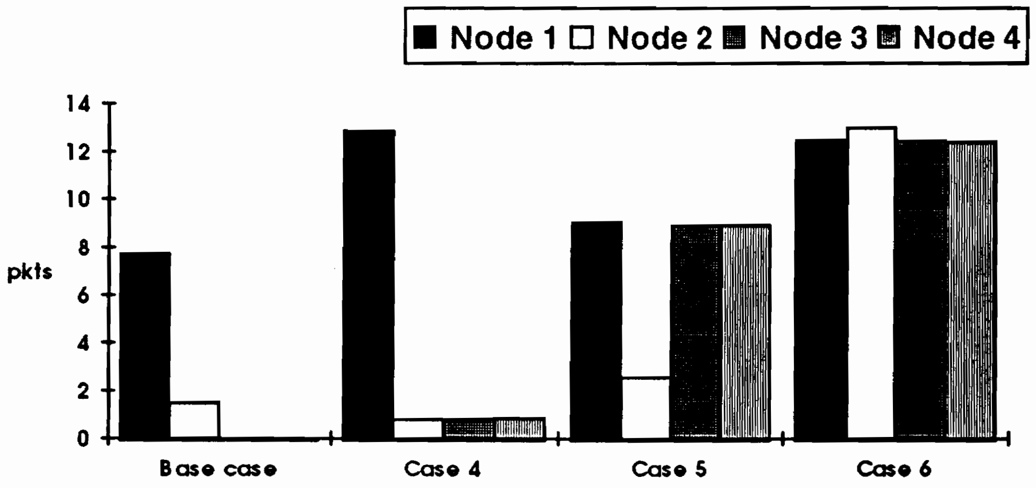

Figure 5.4b. Average size of FDDI transmit queues for the base case and cases 4, 5, and 6.

As the network is utilized by more nodes, the average transmit queue length increases, since each node is now receives a smaller fraction of the total bandwidth. Hence, packets must wait longer for transmission, giving higher latency (see Table 5.5 and Figure 5.3b) and requiring larger transmit queues. The queue size at *node* 2 in the base case, *nodes* 2, 3 and 4 in case 4 and *node* 2 in case 5 are very small since these nodes only transmit acks. Further, the transmit queue lengths for all channels at a particular node are equal because of balanced loading of the FDDI rings. In Table 5.7c and Figure 5.4c, "R" stands for ring.

Table 5.7c. Average Length, in Number of Packets, of the FDDI Receive Queues for the Base Case and Cases 4, 5 and 6

	Base case	Case 4	Case 5	Case 6
Node 1, R 1	0.07	0.05	0.02	0.00
Node 1, R 2	0.08	0.07	0.02	0.01
Node 1, R 3	0.09	0.08	0.02	0.01
Node 1, R 4	0.10	0.09	0.03	0.02
Node 2, R 1	0.01	0.01	0.01	0.00
Node 2, R 2	0.03	0.01	0.02	0.01
Node 2, R 3	0.04	0.02	0.04	0.01
Node 2, R 4	0.05	0.02	0.05	0.02
Node 3, R 1	0.0	0.01	0.02	0.00
Node 3, R 2	0.0	0.01	0.02	0.01
Node 3, R 3	0.0	0.02	0.03	0.01
Node 3, R 4	0.0	0.02	0.03	0.02
Node 4, R 1	0.0	0.01	0.02	0.00
Node 4, R 2	0.0	0.01	0.02	0.01
Node 4, R 3	0.0	0.02	0.03	0.01
Node 4, R 4	0.0	0.02	0.03	0.02

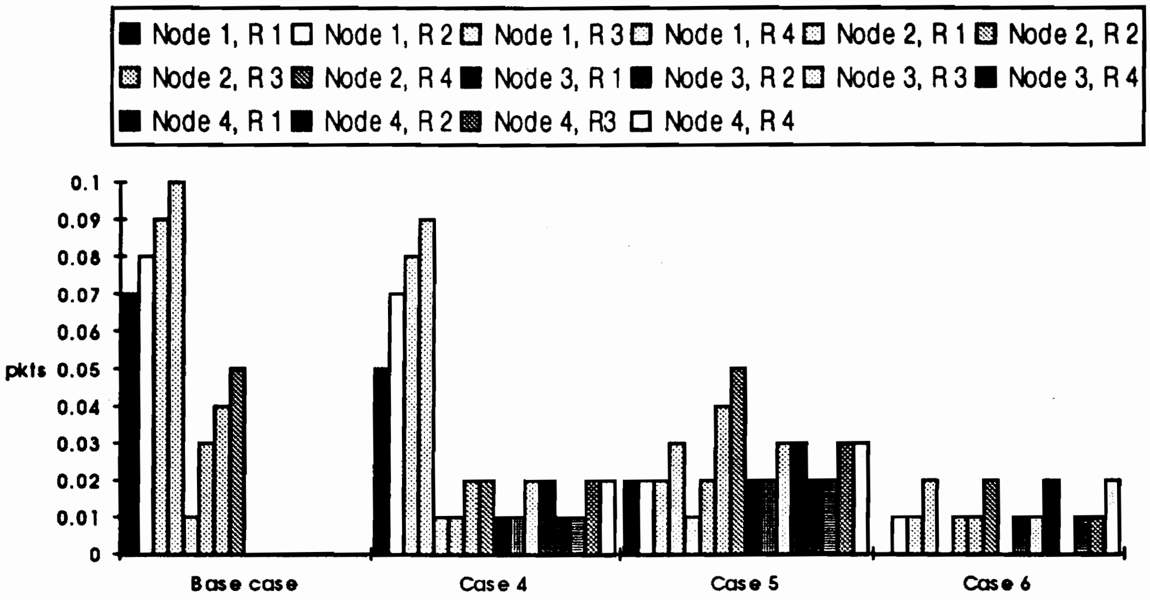

Figure 5.4c. Average size of FDDI receive queues for base case and cases 4, 5, and 6.

The receive queue sizes are small compared to the transmit queue sizes because priority is given to processing received packets. The queue size tends to increase from ring 1 to ring 4 because the AL processes packet arrivals on the lowest order ring (ring 1) first. Since the average value is rounded off to the second decimal place, the table shows that for case 6, ring 1 needs zero receive buffers.

The priority given to reception processing causes the receive queue size to be nearly constant for all cases. However, a decreasing trend is evident as more nodes receive packets since the transmitted packets are now distributed among a larger number of nodes.

5.4.4. Cases 7 and 8: CCC m Value

Cases 7 and 8 vary the distribution of the value of the CCC parameter *m*. In case 7, 100 percent of the packets have $m = 1$, while in case 8, all packets have $m = 0$. The interesting measures are again throughput and latency because the value of m has the greatest impact on the number of overhead bits and the declaration of data sufficiency. Relevant results are tabulated in Table 5.8, and compared in Figures 5.5a and 5.5b.

Table 5.8. Throughput and Latency for Base Case and Cases 7 and 8

	Base case	Case 7	Case 8
Throughput (Mbps)	238.53	255.37	338.18
Latency (ms/pkt)	3.23	2.92	2.93

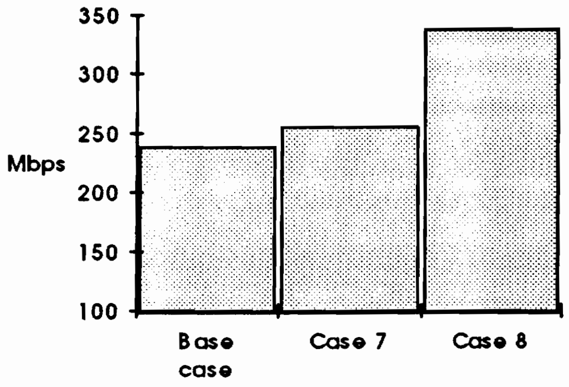

Figure 5.5a. Throughput for base case and case 7 and 8.

The throughput for case 7 is greater than for the base case because of the absence of $m = 2$ packets which have 50 percent CCC code bits. The throughput for case 8 is the maximum possible, since CCC is not being used at all. The loss in throughput from the 400 Mbps available at the FDDI ring level is due to token rotation and header bits only.

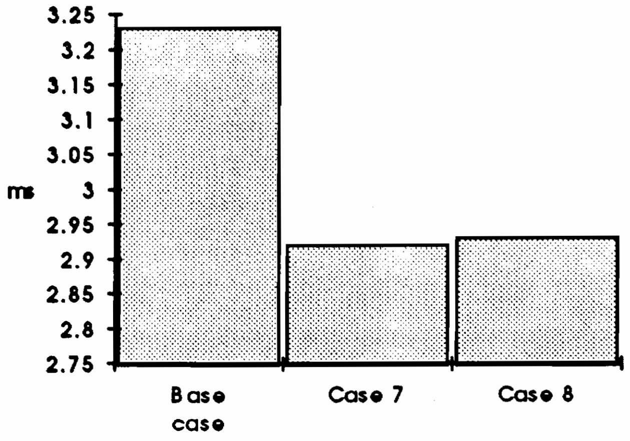

Figure 5.5b. Packet latency for base case and cases 7 and 8.

In Cases 7 and 8 the number of destination classes are reduced to one-third of those in the base class because *m* has only one possible value, rather than three. Hence, data sufficiency can be achieved more quickly. This is reflected in the lower packet latencies for cases 7 and 8, as shown in Figure 5.5b.

The significant implication of the results for these cases is that reducing the number of destination classes is a useful way of reducing packet latency without reducing throughput. The number of destination classes can be reduced by either reducing the number of *m* values or the number of destination nodes. Since the number of possible destination nodes depends on the network size, reducing the number of different *m* values may be the best approach.

5.4.5. Cases 9 and 10: Data Sufficiency

Two data sufficiency strategies are employed by the AL to try to maximize throughput and minimize latency. Cases 9 and 10 study the extreme settings for these two strategies.

Case 9 seeks to maximize the throughput by setting the data sufficiency timers to infinity, while Case 10, seeks to minimize the latency by setting the timers to zero. The relevant results are throughput and packet latency. The results are presented in Table 5.9 and graphically in Figures 5.6a and 5.6b.

Table 5.9. Throughput and Latency for the Base Case and Cases 9 and 10

	Base case	Case 9	Case 10
Throughput (Mbps)	238.53	243.47	216.97
Latency (ms/pkt)	3.23	3.85	2.13

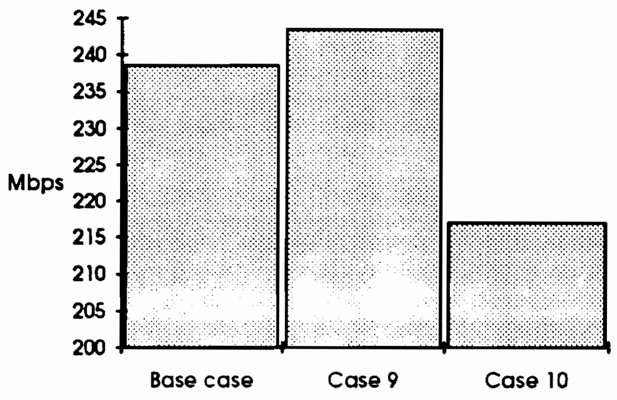

Figure 5.6a. Throughput for base case and cases 9 and 10.

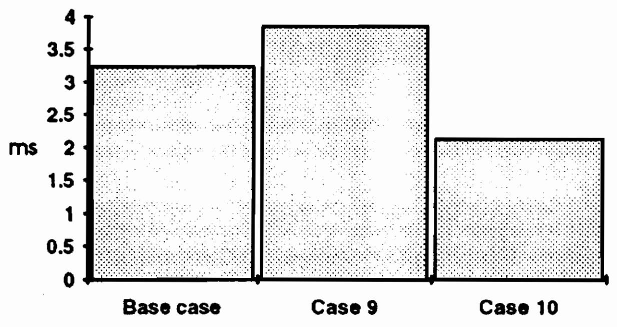

Figure 5.6b. Packet latency for base case and cases 9 and 10.

As expected, the throughput for case 9 is greater than that for the base case. Case 10 sees a fall in throughput because threshold sizes are almost never achieved, resulting in a greater overhead to data ratio. Case 9 sees increased latency because the AL always waits for threshold sufficiency to be achieved. The effect of setting the timers to zero can also be obtained by setting the threshold sizes to very small values.

These results demonstrate that the two data sufficiency strategies used by the AL are effective. Both latency and throughput can be varied over a range of values by adjusting the threshold sizes and/or timer settings.

One additional noteworthy result is the utilization of the FDDI rings. Table 5.10 and Figure 5.7 compare these results for the base case and cases 9 and 10. The utilization of all four rings in these cases are identical because all four tokens rotate at the same rate in all three cases.

Table 5.10. Utilization of FDDI Rings for Base Case and Cases 9 and 10

Base case	Case 9	Case 10
93.42%	93.54%	81.83%

Most of the timing estimates for the AL functions are on a per-packet basis. This implies that the rate of processing of AL packets is largely independent of its size. Hence, the average number of AL packets processed in the base case, case 9 and case 10 will be almost equal. However, the utilization of the FDDI rings depend only on the size of the frames. The frames in case 10 are smaller than those in the base case or case 9, since every packet is individually divided over all four rings. In other words, while the average number of frames for all three cases are almost equal, the average frame size for case 10 is less than that for the other two cases. This explains the reduced utilization for case 10 when compared to case 9 and the base case.

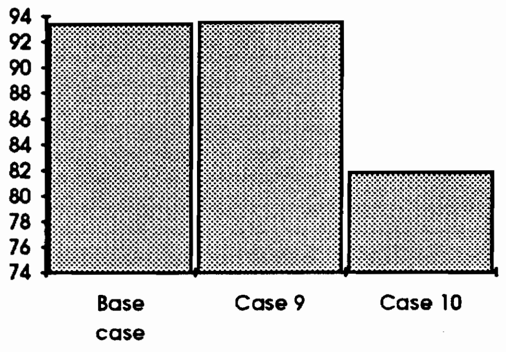

Figure 5.7. Percentage utilization of the FDDI rings for base case and cases 9 and 10.

5.4.6. Case 11: Distribution of Packets with no CCC

The final case uses a different algorithm for packets with $m = 0$. Data sufficiency is checked on a per-channel basis for these packets. In other words, the AL attempts to achieve data sufficiency for only one AL packet which is then sent on the first available channel. This lead to uneven loading of the four FDDI rings and the tokens rotated at different rates. In addition to the overall throughput and packet latency, the latency of packets belonging to each packet size and m value combination is also presented. Table 5.11 lists the overall throughput and latency. Throughput values are compared in Figure 5.8a, while latencies are compared in Figure 5.8b. The latency results for the different packet size and m value combinations are presented in Table 5.12 and in Figure 5.9.

Table 5.11. Throughput and Latency for the Base Case and Case 11

	Base case	Case 11
Throughput (Mbps)	238.53	239.60
Latency (ms/pkt)	3.23	3.13

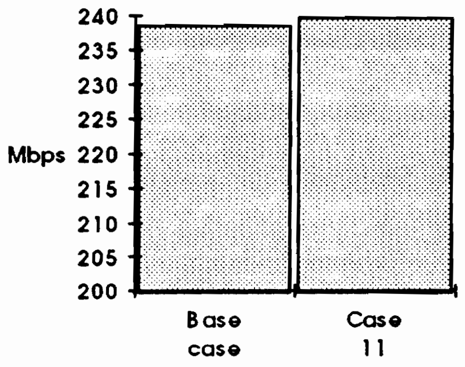

Figure 5.8a. Throughput for base case and case 11.

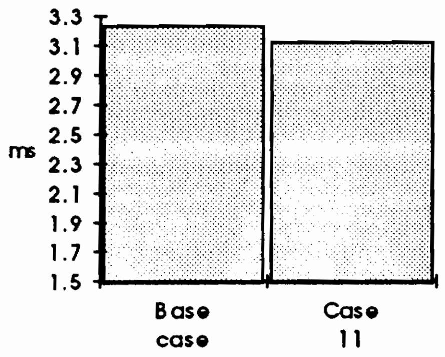

Figure 5.8b. Latency for base case and case 11.

The total amount of data sent in both the base case and case 11 are the same because they have the same destination and CCC *m* distributions. This explains why the throughput remains approximately equal for these cases. The main impact of sending packets with

$m = 0$ quickly is to reduce the latency of such packets at the expense of the latency of packets with $m = 1$ or 2. As a result the overall packet latency for case 11 is almost equal to that for the base case, as is evident from Figure 5.8b. Figure 5.9 compares the latencies for the packets with different size and CCC m value combinations.

Table 5.12. Latencies Based on Packet Type for the Base Case and Case 11

	Large Pkt $m = 0$	**Large Pkt** $m = 1$	**Large Pkt** $m = 2$	**Small Pkt** $m = 0$	**Small Pkt** $m = 1$	**Small Pkt** $m = 2$
Base case	3.73	3.11	3.26	3.67	2.96	3.13
Case 11	2.58	3.53	3.64	2.25	3.32	3.45

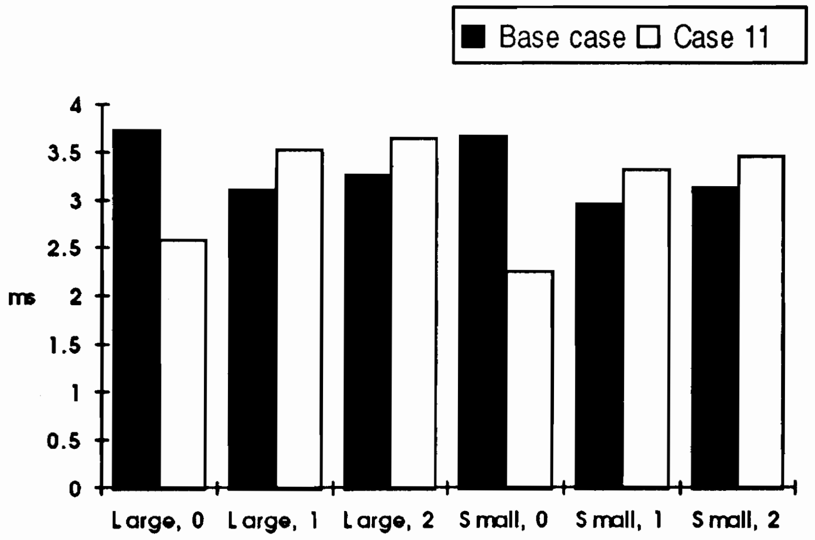

Figure 5.9. Packet latency from Table 5.12.

It is clear that the latency for packets with $m = 0$ is reduced at the expense of the latency for packets with $m = 1$ and $m = 2$. Packets with $m = 0$ are sent fairly quickly because threshold data sufficiency must be satisfied for only one channel. This delays packets which use CCC.

This experiment suggests that the additional complexity of uneven loading may be too high a price for reducing the latency of packets with $m = 0$. This is especially so when the increase in the latency of the other packets is considered.

5.5. Verification of CCD Functionality

All packets passed to the FDDI rings are assumed to arrive error-free at the destination node in the above experiments. Hence, the CCD unit and its associated functionality were not tested. To obtain meaningful performance results when using CCD, a good error model for the parallel channels is needed. In the absence of this, only the functionality of the CCD unit was verified.

The base case was used for verification. The packet arrival scheduling function was modified such that any frame being sent on FDDI Ring 1 did not arrive at the destination node. This simulated a simple failed ring error situation. By using a debugger to step through the reception processing and by observing the performance measures, the functionality associated with the CCD unit was verified. The strategy for handling late arriving packets is not implemented in the simulator.

5.6. Summary

The experimental results were largely as expected. Detailed inferences are in the subsections of Section 5.4. The main points are summarized here.

The most significant inference from the simulation experiments is that the AL, as defined, is not a bottleneck in the parallel network. The functionality of the AL protocol was verified in detail, except for some special cases.

Cases 2 and 3 studied the effects of packet size on throughput and latency. Larger packets have higher throughput because of the lower overhead-to-data ratios that frames created from such packets have. For a small decrease in the average packet size, the packet latency is reduced slightly, as can be seen by comparing results for case 2 and the base case. However, for a large reduction in the average packet size, the latency actually increases very significantly. This is largely because as the packet size reduces, the number of packets increases and the AL tends to become the bottleneck. For very small packets, the utilization of the FDDI rings is reduced and the AL is clearly the bottleneck.

Cases 4, 5 and 6 indicate how the throughput and latency increase as more nodes utilize the network. This increase in active nodes also causes the data storage in the FDDI transmit queues to increase. Since the total data processed at a node does not change significantly, the required shared memory sizes are reduced. The receive queues are not significantly affected by the change in the destination distribution because of the high priority given to processing receptions.

Cases 7 and 8 demonstrate the advantage of having fewer values for the CCC *m* parameter, and hence fewer destination classes. Reducing the number of destination classes reduces packet latency without adversely affecting throughput.

Cases 9 and 10 suggest how the two data sufficiency strategies can be used to control packet latency and throughput. Latency can be minimized by setting the data sufficiency timers to zero (case 10) or by setting the threshold values to very small values.

Throughput can be maximized by setting the timers to infinity or in other words, by not using the data sufficiency timers.

The final experiment, case 11, seems to indicate that there is not much benefit in having an alternative algorithm to handle $m = 0$ packets. The use of an alternative algorithm increases the complexity of the AL functionality. The only gain is a slight reduction in the latency of packets with $m = 0$. This reduction is, however, obtained at the expense of the latency of packets which use CCC.

Chapter 6. Conclusion

A summary of the research described in this thesis is presented in the first section of this chapter. This is followed by suggestions for further research.

6.1. Research Summary

The research described in this thesis proposed and analyzed a technique that permits the use of different degrees of parallelism at different network protocol layers. The technique is to introduce a new protocol layer called the *access layer* (AL) between the network and data link layers of the OSI protocol stack. The AL maps between the parallelism at transport/network layers and that at the data link layer.

The protocol and associated functionality were defined for the AL. The functionality is described in terms of the step-by-step operations to be performed for transmitting and receiving packets. The defined AL protocol does not make any assumptions regarding the protocols being used at the adjacent layers. It is a data request/data indication type of protocol. In other words, layer N + 1 sends a *data request* to layer N for transmitting some data, while layer N sends a *data indication* to layer N + 1 when it has some data for the upper layer. All transfers in the AL take place over a common bus, and all the data is stored in a shared memory. The AL protocol also has provision for the use of an error correction coding procedure known as cross channel coding (CCC) [2]. CCC is designed for situations where data is transferred over parallel channels to the same destination. Depending on the level of coding used, CCC can be used at the receiver to recover the frames on one or more of the parallel channels. All packets undergoing CCC together have to be of equal size. A division algorithm was developed to divide the available data into packets of equal size before undergoing coding.

A software simulator was developed to verify the functionality of the AL protocol and study the impact of the AL on performance. The simulator modelled the proposed parallel network [1] and included a detailed model of the AL. It was used to verify that the functionality defined for the AL is sufficient to meet the requirements for the AL. The simulator was also used to obtain performance measures for the parallel network. These performance studies provided insights into the impact of the AL.

The key result is that the AL is not a bottleneck in the parallel network for most of the cases considered in the simulation experiments. However, as the average packet size decreases, the AL appears to become a system bottleneck. Larger packets were found to give better throughput since their overhead-to-data ratio is small. For small reductions in the average packet size, the packet latency is reduced since shorter packets are transferred faster over the common bus and the physical medium. If the average packet size becomes very small, the per-packet processing dominates the packet transfer time, and the packet latency becomes fairly high.

The results indicated that reducing the number of choices in CCC gives improved latency measures without affecting throughput. This is because the data required to start CCC can be accumulated faster. The AL uses two strategies in attempting to maximize throughput and minimize latency. These strategies, based on threshold sizes and timers, were found to provide a range over which the data throughput and packet latency can be varied.

The number of nodes utilizing the network was also found to have a significant impact on the throughput, latency, required size of shared memory at each node and the lengths of the FDDI transmit and receive queues. When more nodes have data to transmit, the FDDI token rotation period increases. This reduces the token overhead and hence, increases the data throughput. Since more nodes are sending data, the data at any particular node has to wait longer for access to the physical medium. This causes the packet latency to increase. As the number of active nodes increases the required size of the shared memories at each node decreases while the FDDI transmit and receive queues increase in length.

Whenever CCC is used the data is divided equally over all four channels. However, when it is not used, all four channels do not have to be used. The final simulation experiment examined the impact of sending packets not using CCC on the least-busy channel, rather than dividing over all four channels. The only gain was a reduced latency for the packets which did not use CCC. However, the other packets were slowed down. The use of such a strategy also adds additional complexity to the AL functionality, which does not seem to be justified by the gains it provides.

6.2. Suggestions for Further Research

The AL protocol and functionality described in this thesis were developed for a particular network architecture. One potential research activity is to study how this defintion can be used for other architectures for a parallel network. For the purpose of implementation, it may be advantageous to simplify the defined AL protocol. One simplification is using fixed size AL headers. This could eliminate the need for the elaborate division algorithm at the expense of throughput. This, and other avenues for simplification of the AL protocol could be studied.

There is significant potential for further research in the performance studies of the parallel network. The first activity could be to repeat the experiments described in Chapter 5 after incorporating a meaningful error model for the FDDI rings. The detailed AL model could be interfaced with similar models for the other layers to obtain more accurate performance results. The impact of varying the degree of parallelism can also be studied. Finally, a set of simulation experiments can be performed to meet the more conventional goal of plotting performance curves.

References

[1] J.A. Wiencko, S.J. Harper, R. Kumar and S.F. Midkiff, "Parallelism as an Architecture for High Data Rate Networks," *Proceedings Southeastern Symposium on System Theory and Annual Symposium on Communications, Signal Processing, Expert Systems, and ASIC VLSI Design,* 1992, pp. 566-569.

[2] J.A. Wiencko, "Cross Channel Coding," Ph.D. Dissertation, Virginia Polytechnic Institute and State University, in preparation.

[3] M. Schwartz, *Telecommunication Networks: Protcols, Modeling and Analysis,* Addison-Wesley Publishing Company, 1988, pp. 331-397.

[4] *Network II.5 User's Manual,* CACI Products Company, 1990.

[5] Token Ring Access Method and Physical Layer Specifications: ANSI/IEEE Std 802.5, ISO/DP 8802/5, 1985.

[6] L. Kleinrock, "The Latency/Bandwidth Tradeoff in Gigabit Networks," *IEEE Communications Magazine,* Vol. 30, No. 4, April 1992, pp. 36-40.

[7] C.E. Catlett, "In Search of Gigabit Applications," *IEEE Communications Magazine,* Vol. 30, No. 4, April 1992, pp. 42-51.

[8] H.T. Kung, "Gigabit Local Area Networks: A Systems Perspective," *IEEE Communications Magazine,* Vol. 30, No. 4, April 1992, pp. 79-89.

[9] N. Modiri, "The ISO Reference Model Entities," *IEEE Network Magazine,* July 1991, pp. 24-33.

[10] M. Zitterbart, "High-Speed Protocol Implementations Based on a Multiprocessor Architecture," *Protocols for High-Speed Networks*, H. Rudin and R. Williamson (Editors), North-Holland, 1989, pp. 151-163.

[11] H. Kanakia and D.R. Cheriton, "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors," *Proceedings SIGCOMM Symposium: Communications, Architectures and Protocols*, 1988, pp. 175-187.

[12] N. Jain, M. Schwartz and T.R. Bashkow, "Transport Protocol Processing at Gb/s Rates," *Proceedings SIGCOMM*, 1990, pp. 188-199.

[13] S.L. Rege, "The Architecture and Implementation of a High-Performance FDDI Adapter," *Digital Technical Journal*, Vol. 3, No. 3, Summer 1991, pp. 48-63.

[14] A. Ippoliti and A. Albanese, "Parallel Media Access Controller for Packet Communications at Gb/s Rates," *Proceedings International Conference on Communications*, 1990, pp. 991-996.

[15] D. E. Tolmie, "Local Area Gigabit Networking," *Proceedings 11th IEEE Symposium on Mass Storage Systems*, 1991, pp. 11-16.

[16] P. Martini and M. Rupprecht, "Designing High Speed Controllers for High Speed Local Area Networks," *Proceedings IEEE Global Telecommunications Conference*, 1989, pp. 170-174.

[17] Floyd E. Ross, "An Overview of FDDI: The Fiber Distributed Data Interface," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 7, September 1989, pp. 1043-1051.

[18] S. Kapoor and G. M. Parulkar, "Design of an ATM-FDDI Gateway," *Proceedings SIGCOMM*, 1991, pp. 173-183.

[19] M. J. Strohl, "High Performance Distributed Computing in FDDI Networks," *IEEE LTS*, Vol. 2, No. 2, May 1991, pp. 11-15.

[20] M. P. Kovacs, "FDDI Networking: Where will it go from here?," *Laser Focus World*, Vol. 26, No. 9, September 1990, pp. 169-174.

[21] M. J. Johnson, "Performance Analysis of FDDI," *Proceedings 6th European Fibre Optic Communications and Local Area Networks Exposition*, June 1988, pp. 295-300.

[22] R. Jain, "Performance Analysis of FDDI," *Digital Technical Journal*, Vol. 3, No. 3, Summer 1991, pp. 78-88.

[23] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, 1991.

[24] Kevin M. Kitagawa, *An M Bus Tutorial*, SPARC Technical Marketing, Sun Microsystems Inc., 1991.

[25] D. D. Clark, V. Jacobson, J. Romkey and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications Magazine*, Vol. 27, No. 6, June 1989, pp. 23-29.

Appendix A. Cross Channel Coding Equations

CCC Encode and decode equations for the case of $n = 4$ are provided in this appendix. With $n = 4$, m can be 0, 1, or 2.

Notation

- Channels are A, B, C, and D
- "$\oplus$" is modulo-2 addition, *i.e.*, the exclusive-OR operation
- Check bits A' are transmitted along with data on channel A, and similarly for the other channels.

Encode Equations

Case for $m = 0$

- Data on each channel passes through unmodified
- No check bit calculations take place

Case for $m = 1$

- Data on each channel is split into 3 parts, for example, A0, A1, A2

- One check bit per three data bits
- Check bit calculations:

$$A' = B0 \oplus C1 \oplus D2$$

$$B' = A2 \oplus C0 \oplus D1$$

$$C' = A1 \oplus B2 \oplus D0$$

$$D' = A0 \oplus B1 \oplus C2$$

Example A.1. Let the data in each channel be 12 bits long. Then the packet in channel A is split into three parts as shown in Figure A.1. Each part is 4 bits long which means that the code portion, A′, is also 4 bits long. Hence, the AL packet for channel A will be 16 bits long.

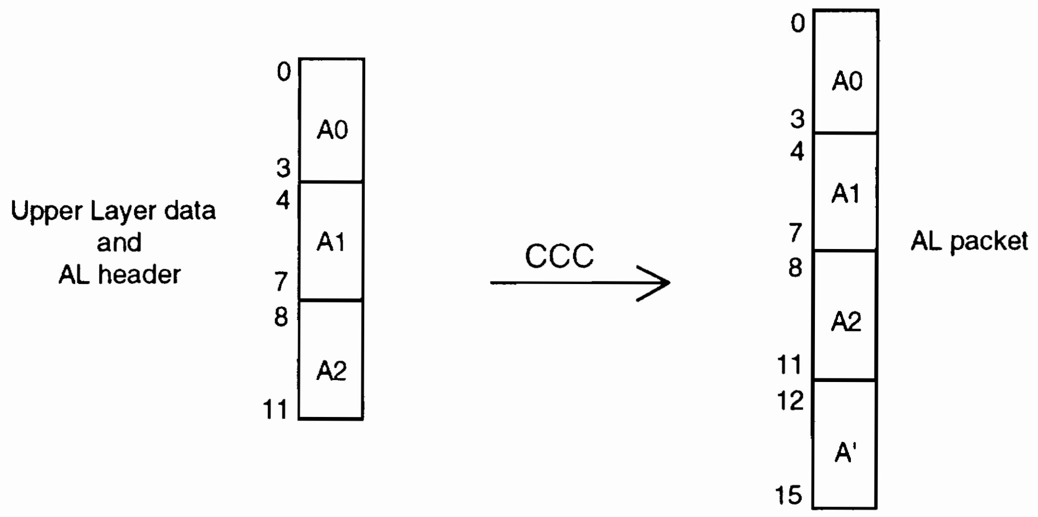

Figure A.1. Addition of CCC code bits to form an AL packet for m = 1.

Case for m = 2

- Data on each channel is not split
- One check bit per data bit

- Check bit calculations:

$$A' = B \oplus C$$

$$B' = C \oplus D$$

$$C' = D \oplus A$$

$$D' = A \oplus B$$

Example A.2. Using the same packet size as in Example A.1, the code in each AL packet is of the same length as the data bits, *i.e.*, 12 bits. Figure A.2 shows how CCC bits are added to build the AL packet for channel A.

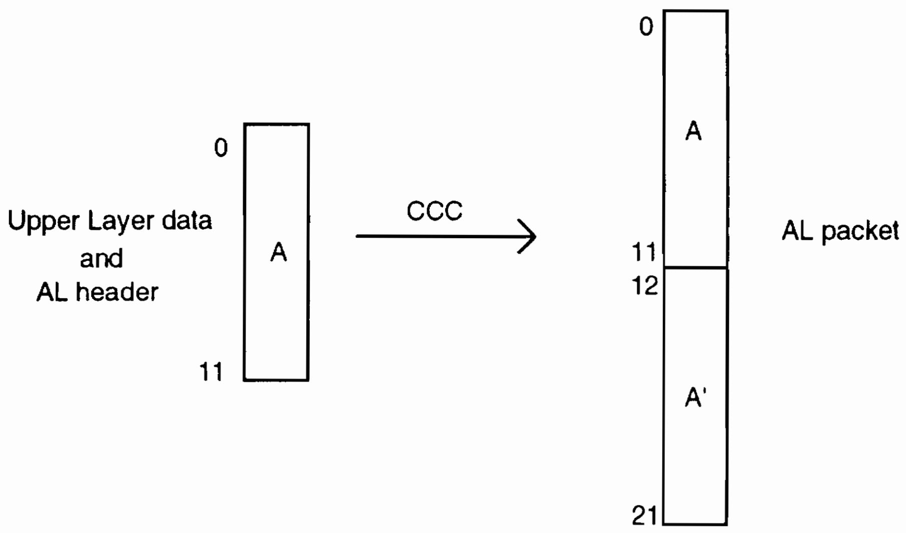

Figure A.2. Addition of CCC bits to form an AL packet for $m = 2$.

Decode Equations

Case for *m* = 1; Channel A is missing

- Channel A packet is rebuilt as follows:

$$A2 = B' \oplus C0 \oplus D1$$

$$A1 = C' \oplus B2 \oplus D0$$

$$A0 = D' \oplus B1 \oplus D0$$

- Similar sets for channels B, C, and D missing

Case for *m* = 2; Channels A and B missing

- The missing channel packets are recovered as follows:

$$A = C' \oplus D' \oplus C$$

$$B = D' \oplus C$$

- Similar sets for channels BC, CD, and AD missing

Case for *m* = 2; Channels A and C missing

- The missing channel packets are rebuilt as follows:

$$A = B' \oplus D$$

$$C = D' \oplus B$$

- Similar set for channels BD missing

Appendix B. System Parameters and Timing Estimates

The system parameters assumed for the simulation studies are presented in this appendix along with the complete list of timing estimates. The system parameters are listed first.

ALP and PPs

- 50 MIPS processors. This translates to 20 ns/instruction.

M Bus

- 64 bit bus, with peak rate of 40 MHz which translates to a peak throughput of 2,560 Mbps
- Each active M Bus cycle transfers 8 bytes, which requires 25 ns.
- Each time any master gains control of the M Bus, it can transfer a maximum of 128 bytes.
- Round-robin arbitration is used to decide the next bus master with one idle cycle of 25 ns between change of masters.

FDDI Rings

- 100 Mbps throughput, which means that one byte needs 80 ns to be transmitted
- Error-free operation
- Token of 11 bytes
- TTRT: 2 milliseconds

Physical network

- Number of nodes: 4
- Propagation delay: 15,000 ns

Table B.1. Timing Estimates for Simulation Experiments

Task	Number of instructions	Time in ns
ALP DR processing	200	4,000
Data sufficiency analysis	100	2,000
AL header building	100	2,000
Check timer status	20	600
ALP idle wait	175	3,500
PP idle wait	175	3,500
PP create DR	250	5,000
ALP build DI	50	1,000
PP DI processing	200	4,000
ALP received pkt processing	150	3,000
CCC and CCD/byte	-	1

Appendix C. Simulation Results

Simulation results from all the experiments are presented in this appendix. Table C.1 lists the packet latency, data throughput and packet throughput for all eleven cases. Packet latency is listed on a per-class basis in Table C.2, where "class" refers to the combination of application layer packet size and value of the CCC parameter *m*. Each class is denoted by "L" for *large* packets, "S" for *small* packets, and 0, 1, or 2 for the value of the CCC parameter *m*. The entry "NA" statnds for *not available* because no packets from that class were used in the simulation case. Tables C.3, C.4, and C.5 list the utilization of the M Bus, ALP, and the FDDI rings, respectively. The Average and maximum shared memory sizes are presented in Table C.6. In Table C.7 through Table C.10b, "R" stands for FDDI ring. Table C.7 lists the maximum required length of the FDDI transmit queues, while the maximum lengths for the receive queues are presented in Table C.9. Tables C.9a and C.9b present the average length of the FDDI transmit queues while the average lengths for the receive queues are listed in Tables C.10a and C.10b.

Table C.1. Latency and Throughput

	Packet Latency (ms)	Throughput (Mbps)	Pkt Throughput/s
Base case	3.23	238.53	16415.29
Case 2	3.72	246.26	8795.02
Case 3	44.718	72.05	89600.12
Case 4	3.464	235.62	16260.39
Case 5	8.019	245.20	17037.30
Case 6	9.221	254.34	17432.81
Case 7	2.916	255.37	17563.93
Case 8	2.929	338.18	23496.83
Case 9	3.849	243.47	16853.16
Case 10	2.135	216.97	14892.02
Case 11	3.125	239.60	13608.91

Table C.2. Packet Latency (ms) for Different Packet Classes

	S 0	S 1	S 2	L 0	L 1	L 2
Base case	3.672	2.964	3.129	3.731	3.105	3.262
Case 2	NA	NA	NA	4.125	3.550	3.676
Case 3	44.249	44.741	45.130	NA	NA	NA
Case 4	4.045	3.100	3.373	4.111	3.250	3.485
Case 5	9.417	7.317	7.818	9.294	7.553	7.993
Case 6	11.789	7.751	9.035	11.882	8.180	9.278
Case 7	NA	2.880	NA	NA	2.951	NA
Case 8	2.887	NA	NA	2.970	NA	NA
Case 9	4.320	3.566	3.735	4.365	3.665	3.943
Case 10	2.064	2.072	2.066	2.170	2.214	2.200
Case 11	2.255	3.322	3.449	2.581	3.525	3.635

Table C.3. Utilization of the M Bus (percent)

	Node 1	Node 2	Node 3	Node 4
Base case	45.57	43.17	17.36	17.36
Case 2	41.25	39.68	17.36	17.36
Case 3	100.0	56.82	17.36	17.36
Case 4	42.79	25.17	25.3	25.34
Case 5	27.07	43.96	27.14	27.27
Case 6	29.56	29.5	29.56	29.52
Case 7	44.8	42.05	17.36	17.36
Case 8	53.76	49.58	17.36	17.36
Case 9	46.11	43.43	17.36	17.36
Case 10	43.7	42.24	17.36	17.36
Case 11	45.73	43.87	17.36	17.36

Table C.4. Utilization of the ALP (percent)

	Node 1	Node 2	Node 3	Node 4
Base case	24.69	24.79	2.1	2.1
Case 2	16.38	16.67	2.1	2.1
Case 3	100.0	80.76	2.1	2.1
Case 4	25.13	9.43	9.57	9.56
Case 5	9.6	25.54	9.52	9.57
Case 6	9.4	9.42	9.36	9.32
Case 7	22.64	22.79	2.1	2.1
Case 8	28.18	27.79	2.1	2.1
Case 9	22.2	23.2	2.1	2.1
Case 10	40.01	37.48	2.1	2.1
Case 11	26.31	27.21	2.1	2.1

Table C.5. Utilization of FDDI Rings (percent)

	Ring 1	Ring 2	Ring 3	Ring 4
Base case	93.42	93.42	93.42	93.42
Case 2	93.39	93.39	93.39	93.39
Case 3	61.67	61.67	61.67	61.67
Case 4	93.27	93.27	93.27	93.27
Case 5	95.49	95.49	95.49	95.49
Case 6	95.81	95.81	95.81	95.81
Case 7	93.42	93.42	93.42	93.42
Case 8	93.51	93.51	93.51	93.51
Case 9	93.54	93.54	93.54	93.54
Case 10	81.83	81.83	81.83	81.83
Case 11	93.43	93.28	93.38	93.51

Table C.6. Average and Maximum Shared Memory Sizes

	AVERAGE				MAXIMUM			
	Node 1	Node 2	Node 3	Node 4	Node 1	Node 2	Node 3	Node 4
Base case	569.79	562.81	0.0	0.0	1089.64	1058.24	0	0
Case 2	299.49	291.51	0.0	0.0	591.49	559.26	0	0
Case 3	1711.20	288.77	0.0	0.0	3207.46	542.65	0	0
Case 4	525.87	168.31	173.40	173.29	1052.83	333.34	342.30	340.40
Case 5	183.53	539.96	178.27	183.57	384.58	1054.30	380.56	386.60
Case 6	139.52	142.54	144.84	141.40	325.33	318.68	326.60	333.45
Case 7	555.91	553.30	0.0	0.0	1096.48	1085.31	0	0
Case 8	737.47	731.60	0.0	0.0	1470.55	1441.85	0	0
Case 9	552.91	532.23	0.0	0.0	1070.97	1045.14	0	0
Case 10	403.87	400.73	0.0	0.0	747.37	718.85	0	0
Case 11	564.78	573.22	0.0	0.0	1079.39	1078.82	0	0

Table C.7. Maximum FDDI Transmit Queue Lengths

	Node 1				Node 2				Node 3				Node 4			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
Base case	21	21	21	21	7	7	7	7	0	0	0	0	0	0	0	0
Case 2	22	22	22	22	7	7	7	7	0	0	0	0	0	0	0	0
Case 3	3	3	3	3	2	2	2	2	0	0	0	0	0	0	0	0
Case 4	37	37	37	37	5	5	5	5	5	5	5	5	4	4	4	4
Case 5	17	17	17	17	13	13	13	13	17	17	17	17	18	18	18	18
Case 6	27	27	27	27	27	27	27	27	26	26	26	26	27	27	27	27
Case 7	15	15	15	15	7	7	7	7	0	0	0	0	0	0	0	0
Case 8	15	15	15	15	6	6	6	6	0	0	0	0	0	0	0	0
Case 9	15	15	15	15	2	2	2	2	0	0	0	0	0	0	0	0
Case 10	74	74	74	74	40	40	40	40	0	0	0	0	0	0	0	0
Case 11	22	21	24	27	7	6	6	7	0	0	0	0	0	0	0	0

Table C.8a. Average FDDI Transmit Queue Lengths

	Node 1				Node 2			
	R1	R2	R3	R4	R1	R2	R3	R4
Base case	7.72	7.72	7.72	7.72	1.47	1.47	1.47	1.47
Case 2	8.43	8.43	8.43	8.43	1.6	1.6	1.6	1.6
Case 3	0.58	0.58	0.58	0.58	0.25	0.25	0.25	0.25
Case 4	12.85	12.85	12.85	12.85	0.77	0.77	0.77	0.77
Case 5	9.04	9.04	9.04	9.04	2.55	2.55	2.55	2.55
Case 6	12.47	12.47	12.47	12.47	13.0	13.0	13.0	13.0
Case 7	6.12	6.12	6.12	6.12	1.66	1.66	1.66	1.66
Case 8	5.77	5.77	5.77	5.77	1.51	1.51	1.51	1.51
Case 9	5.92	5.92	5.92	5.92	0.39	0.39	0.39	0.39
Case 10	18.63	18.63	18.63	18.63	7.73	7.73	7.73	7.73
Case 11	7.87	7.37	8.06	9.10	1.56	1.57	1.61	1.71

Table C.8b. Average FDDI Transmit Queue Lengths

	Node 3				Node 4			
	R1	R2	R3	R4	R1	R2	R3	R4
Base case	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 4	0.78	0.78	0.78	0.78	0.82	0.82	0.82	0.82
Case 5	8.91	8.91	8.91	8.91	8.9	8.9	8.9	8.9
Case 6	12.46	12.46	12.46	12.46	12.39	12.39	12.39	12.39
Case 7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table C.9. Maximum FDDI Receive Queue Lengths

	Node 1				Node 2				Node 3				Node 4			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
Base case	4	4	4	5	3	3	3	3	0	0	0	0	0	0	0	0
Case 2	4	4	5	5	1	1	1	1	0	0	0	0	0	0	0	0
Case 3	1	1	1	1	2	2	2	2	0	0	0	0	0	0	0	0
Case 4	5	5	5	5	2	2	2	2	2	2	2	2	1	2	2	2
Case 5	4	4	4	4	3	3	3	3	5	5	5	5	4	4	4	4
Case 6	1	2	2	2	2	2	2	2	1	2	2	2	1	2	2	2
Case 7	4	4	4	15	1	2	2	2	0	0	0	0	0	0	0	0
Case 8	4	4	4	4	2	2	2	2	0	0	0	0	0	0	0	0
Case 9	1	1	1	1	1	1	1	2	0	0	0	0	0	0	0	0
Case 10	24	24	24	24	6	6	6	7	0	0	0	0	0	0	0	0
Case 11	4	4	4	5	2	2	3	2	0	0	0	0	0	0	0	0

Table C.10a. Average FDDI Receive Queue Lengths

	Node 1				Node 2			
	R1	R2	R3	R4	R1	R2	R3	R4
Base case	0.07	0.08	0.09	0.10	0.01	0.03	0.04	0.05
Case 2	0.04	0.05	0.06	0.07	0.01	0.02	0.03	0.04
Case 3	0.02	0.03	0.04	0.05	0.04	0.06	0.09	0.11
Case 4	0.05	0.07	0.08	0.09	0.01	0.01	0.02	0.02
Case 5	0.02	0.02	0.02	0.03	0.01	0.02	0.04	0.05
Case 6	0.00	0.01	0.01	0.02	0.00	0.01	0.01	0.02
Case 7	0.05	0.05	0.06	0.07	0.01	0.02	0.03	0.04
Case 8	0.05	0.06	0.07	0.08	0.01	0.02	0.03	0.04
Case 9	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.04
Case 10	1.01	1.06	1.10	1.15	0.11	0.16	0.20	0.25
Case 11	0.03	0.03	0.03	0.03	0.02	0.02	0.02	0.02

Table C.10b. Average FDDI Receive Queue Lengths

	Node 3				Node 4			
	R1	R2	R3	R4	R1	R2	R3	R4
Base case	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 4	0.01	0.01	0.02	0.02	0.01	0.01	0.02	0.02
Case 5	0.02	0.02	0.03	0.03	0.02	0.02	0.03	0.03
Case 6	0.00	0.01	0.01	0.02	0.00	0.01	0.01	0.02
Case 7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Case 11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Vita

Rajesh Kumar was born in Nagpur, India on July 10, 1968. He received his high school diploma from the Vimanapura High School, Bangalore, India in May 1983, and his Pre-University Certificate from the St. Joseph's College, Bangalore in June 1985. He joined the University Visvesvaraya College of Engineering, Bangalore in November 1985 and received the Bachelor of Science degree in electrical engineering in February 1990.

He began work towards his Master of Science degree in electrical engineering at Virginia Tech in January 1991 and completed the requirements for the degree in May 1993.

[illegible]