Performance Analysis of the Manufacturing Automation Protocol

W. Timothy Strayer

Computer Science Report No. TR-88-03
January 1988

This work was sponsored by the Flight Data Systems Branch of NASA--Johnson Space Center, Houston, Texas.

Performance Analysis of the Manufacturing Automation Protocol

A Master of Science Thesis by

W. Timothy Strayer
Computer Networks Laboratory
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903

Alfred C. Weaver
Thesis Advisor

January 1988

ABSTRACT

The Manufacturing Automation Protocol (MAP) is being developed as an attempt to standardize communications between intelligent manufacturing and controlling devices. Intel Corporation has developed a communications board as a front-end processor to be used in Intel Multibus computers, which with MAPNET2.1 software provides an implementation of MAP version 2.1. The Data Link and the Transport Layers were studied through performance analyses. For the Data Link Layer we measured the one way delay and the throughput for various packet sizes. We also measured the number of messages that could be sent and received per second for a range of Data Link packet sizes. For the Transport Layer we found a buffer configuration which optimized throughput and used it for experiments that measured throughput as the dependent variable. Throughput was measured with respect to TSDU message size. The effects of decreasing the retransmission timer, varying the maximum Transport Protocol Data Unit size, using multiple virtual circuits, and varying the maximum window size are described. One way delay was measured with respect to the TSDU message size. Comparisons between the two layers indicated that there are both benefits and drawbacks to using a front-end processor for communications, largely due to message segmentation.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor, Alfred C. Weaver, whose tireless efforts to find new and interesting research avenues help to keep me employed. Matt Katzer of Intel provided our Computer Networks Laboratory with donated software and hardware, and provided personal support during the initial stages of this project. But most importantly, Matt introduced me to Steve Alten, also with Intel, who spent much time and energy helping me install and trouble-shoot the MAP network. I would also like to thank Kevin Mills' Network Architecture Group at the National Bureau of Standards, especially: Kevin for hiring me over the summer of 1987; Sharon Heatley for insight into a meaningful experiment suite; Dan Stokesberry for being both my manager and a devil's advocate; and Tuan To, who handled the hardware problems with the Intel 310s.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

MANUFACTURING AUTOMATION PROTOCOL

1.1. Introduction

The development of the Manufacturing Automation Protocol (MAP) by General Motors [GENE86] is an event which will change the course of history with regard to factory automation. Before 1980, every plant or assembly line which GM built involved a control system which was supplied by one of the major vendors of industrial automation equipment (e.g. Allen-Bradley, Gould Modicon, Texas Instruments, etc.). However, these products could not communicate across vendor boundaries unless someone, usually GM, financed custom hardware and software to overcome the problem. In some instances the cost of developing the custom interfaces was one-half the cost of automating the operation [WEAV87]. In addition, the custom interfaces took a long time to build, and once in place were inflexible.

Faced with stiff international competition, GM found that it simply could not change its production steps or rates fast enough to respond to a rapidly changing marketplace. Thus in November of 1980 a task force was chartered to facilitate information exchange pertaining to plant-floor computer data communications. Called the "Manufacturing Automation Protocol Task Force", it was formed to investigate and identify a common communications standard for plant floor systems (see Figure 1.1). To this end, Michael Kaminski of General Motors, working with Boeing Computer Systems, developed the Manufacturing Automation Protocol (MAP) in an unusual attempt for the end-user to dictate a standard to its vendors.

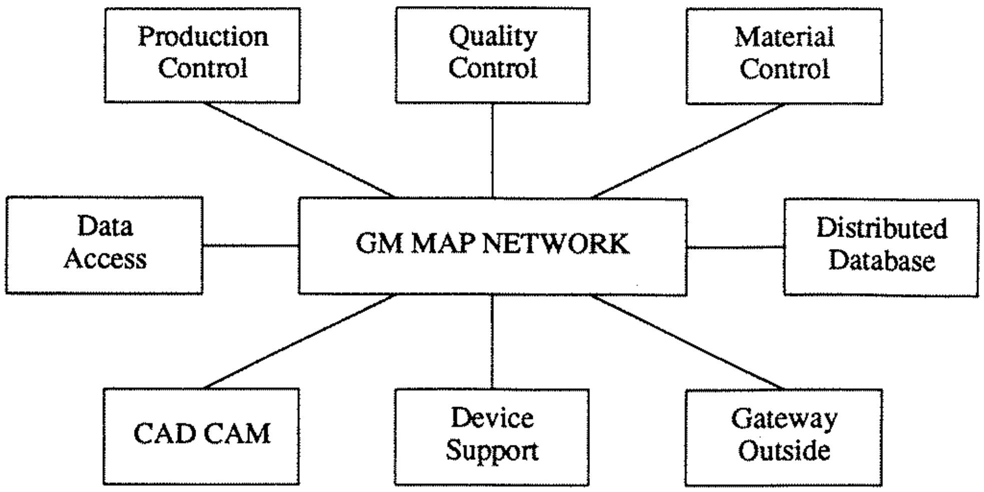


Figure 1.1 — Pictorial Overview of GM MAP

Machinery for a factory floor is supplied by many different vendors. Each vendor typically has its own proprietary way of communicating with its own devices, but seldom do multiple vendors produce systems that can communicate with one another without first developing special hardware and software interfaces. One of the goals of MAP is to provide a method by which any system from any vendor can be inserted into the MAP environment without hardware or software customization, similar to the way that stereo components can be purchased independently, plugged together, and still be expected to inter-operate. MAP is also designed to provide technical services to control various intelligent devices in a consistent and cost-effective manner, contributing to the automation of the factory floor.

Even though MAP is end-user dictated, the equipment vendor's investment in existing protocols also has to be considered. Rather than being a set of newly invented protocols, MAP is primarily a collection of existing national and international standards based on the International Organization for Standardization's Open Systems Interconnection (ISO OSI) reference model as defined by [ISO7498]. The following is a list of the OSI layers and the MAP standards adopted:

OSI LAYER	MAP 2.1 FUNCTIONS
7 - Application	ISO Common Application Service Element (CASE) SASE: MMFS (MMS in MAP 3.0) SASE: FTAM
6 - Presentation	Null (not null in MAP 3.0)
5 - Session	ISO Session Kernel
4 - Transport	ISO Transport Class 4
3 - Network	ISO Connectionless Network Services (CLNS)
2 - Data Link	IEEE 802.2 Link Level, Control Class 1 IEEE 802.4 Token Passing Bus
1 - Physical	IEEE 802.4 Token Bus with Broadband (10 megabits/sec) or Carrierband (5 megabits/sec) Modulation on Coaxial Media

The OSI model is a seven layer architecture in which each layer provides hierarchical support to the layer above it. It provides a structured solution to the reduction of design complexity by isolating functionality into layers, defining the function of each layer completely, and abstracting the layer's implementation decisions to provide modularity. This allows the MAP designers to fill each of these layers by choosing from existing hardware and software standards rather than creating new ones. Knowing that these standards are national or international minimizes the risk to the vendors as well as the end users, and makes all involved more willing to adopt MAP.

Generally a factory floor will consist of several workstations, each with devices dedicated to a specific task. These workstations, in turn, combine at the cell level to provide the cell controller with a set of tasks. These tasks might be assembly operations or machining operations that require little or no human intervention. Several cells are then combined to become the whole factory floor (see Figure 1.2). At each level, computers are needed to provide control and supervision. On the workstation level, a computer might coordinate several robots and a milling machine. The cell controller may need to provide each workstation with the necessary materials and process plans, and the factory floor controller may need to keep track of inventory and provide operator control of products to be manufactured. A Local Area Network (LAN) is the ideal mechanism for providing communications services to these various factory floor devices. MAP specifies the implementation of such a LAN by defining the hardware mechanisms and the software framing of data such that specific types of information can be routed from one type of industrial control equipment to another within the factory hierarchy.

It is the intent of General Motors to purchase industrial devices only from vendors who support MAP, thereby allowing GM to reduce the cost of integrating these devices. It is largely the enormous economic impact that GM has on its vendors that allows GM to dictate this protocol. As a result of vendors producing industrial devices that adhere to MAP, other industries and operations are expected to adopt MAP as well. Hence this protocol will have a significant impact not only on specific manufacturing operations like automobile assembly lines, but on factory automation in general.

GM MAP has been through several versions thus far and will undoubtedly continue to change. Most commercial MAP hardware and software, including the system we tested, conforms to MAP version 2.1. Minor changes were made in version 2.2; version 3.0 has been released in draft form but is not expected to be established as the new specification until mid-1988.

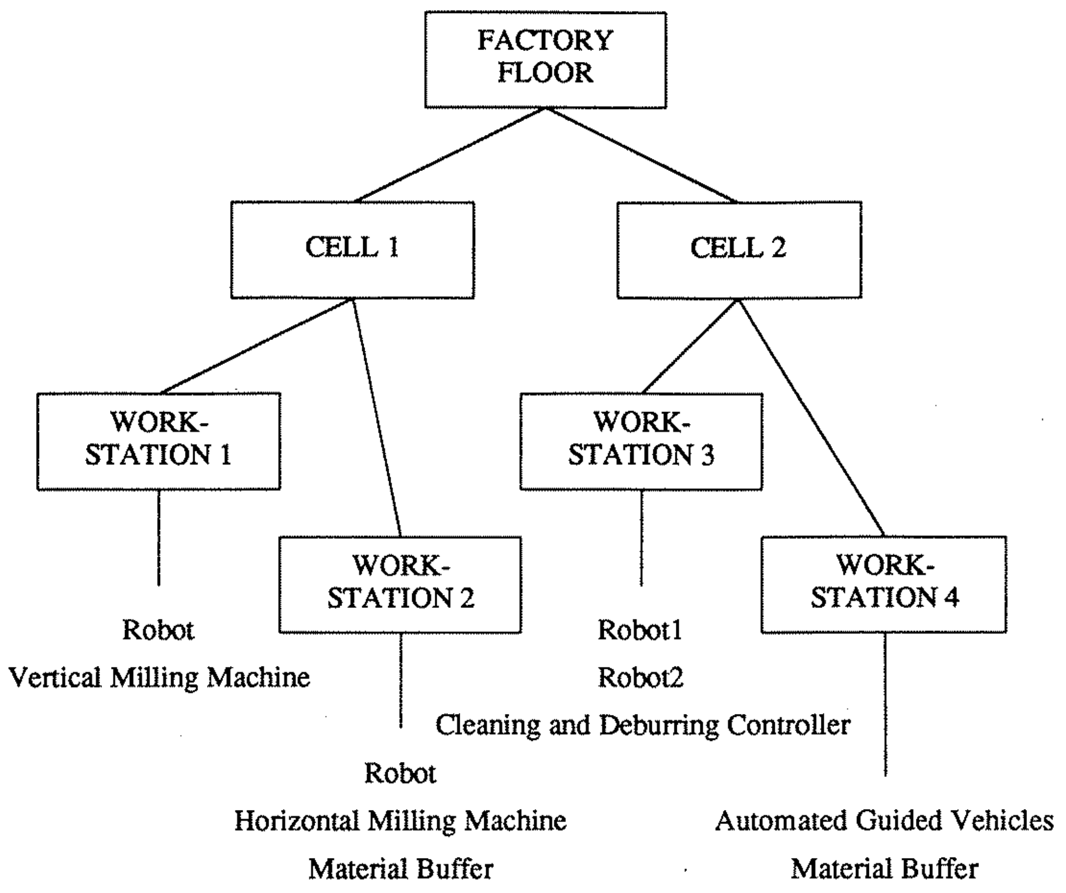


Figure 1.2 — Factory Floor Hierarchy

1.2. MAP Architecture

The Manufacturing Automation Protocol bases its architecture on the ISO OSI reference model. The reference model defines a seven layer hierarchy for providing functionality in a modular fashion. The MAP network can be a collection of segments with or without MAP compatibility. Bridges, routers, and gateways provide access to stations on different segments or different networks. Also, a provision is made for expedited communications using an Enhanced Performance Architecture (EPA). EPA bypasses the upper layers in the OSI model and provides application processes with direct access to the lower level network.

1.2.1. The Open Systems Interconnection Reference Model

One of the goals of GM MAP is to adhere to the networking structure of the ISO OSI reference model. This model is supported by most active national and international standards organizations, and since it provides a common basis for the coordination of standards development, most existing and emerging protocols can be mapped onto one or more of the seven layers of OSI. Furthermore, the model's hierarchical design allows for easy assembling of these protocols on a layer by layer basis, reducing complexity while providing flexibility.

There are seven layers in the OSI model, each layer providing functionality and service to the layer above it. The information passed down through each layer includes data (the message) and control information (the message header), so bits which are control information in the layer above are just treated as data in the layer below. This accounts for the difference in size between the message frame sent over the physical medium and the application's original data unit.

Logically, each layer communicates with its peer layer at another station. This is why control information is added to the data as it is passed down through the layers of the transmitter. As the data passes up through the layers of the receiver, the control information is used and then discarded. At the receiver's application layer, the original message sent by the transmitter's application layer is finally recovered. The only layer that can physically (rather than logically) communicate with its peers is layer 1, the Physical Layer (see Figure 1.3).

Since a network is connected node-to-node by a physical medium, layer 1 must necessarily communicate only with its neighbor nodes (i.e., those nodes physically connected to the same network cable segment). At layer 4, however, communication becomes independent

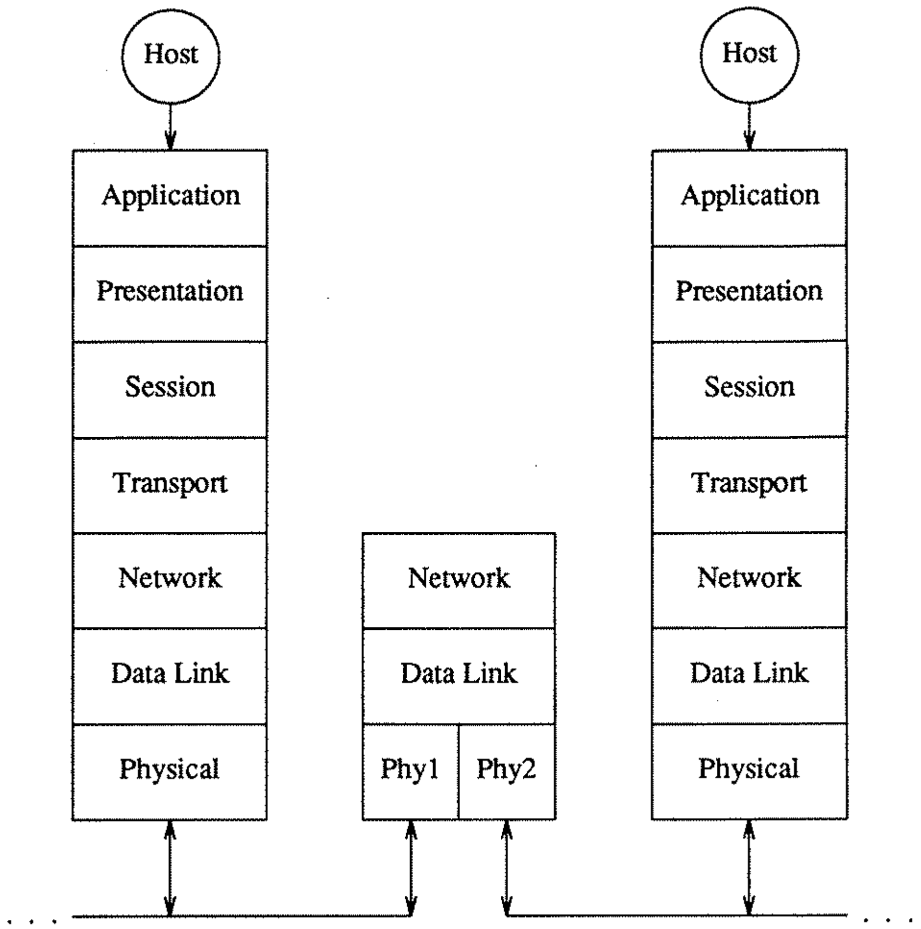


Figure 1.3 — Peer Communications in the ISO OSI Reference Model

of the network topology and communication is said to be end-to-end (i.e., there exists a logical path between transmitter and receiver, without regard to the physical topology required to implement that logical path).

Layer 1 is called the Physical Layer and is concerned with the transmission of raw bits over a physical communications channel. Layer 2 is called the Data Link Layer and uses the raw data transmission service to provide a means by which data frames can be sent and received without errors. The Network Layer is layer 3, and it provides for the routing of packets across networks. The Network Layer examines the name of the destination and decides if it is local or remote. If local, the header appended to the message at this layer is inactive. If remote, the header contains the appropriate addresses used for routing the message to its remote destination. Above the Network Layer, the layers become end-node specific. Layer 4, the Transport Layer, is capable of splitting an arbitrarily large Transport Service Data Unit into multiple smaller Transport Protocol Data Units suitable for the Network Layer. The Transport Layer also provides sequencing and flow control for the end-to-end communications. This is all transparent to the Session Layer, layer 5, which negotiates and establishes connections between end-nodes and manages the dialogue. The Presentation Layer, layer 6, is responsible for negotiating a transfer syntax which is acceptable to both end-nodes. Layer 7, the Application Layer, provides all services directly comprehensible to application programs.

What follows is a more detailed explanation of each layer along with MAP's specification for each.

1.2.1.1. Layer 1: Physical Layer

The Physical Layer provides a connection for transmission of raw data between Data Link entities. It also provides a means by which the physical connection can be activated and deactivated. The intent of the MAP designers is to have the connection be media independent, yet for early MAP implementations two alternatives have been specified for the network's physical media: (1) broadband or (2) carrierband on coaxial cable conforming to the specifications of the IEEE 802.4 Token-Passing Bus Access and Physical Layer Specification [IEEE85c].

In June of 1984 IEEE adopted the 802.4 specifications for local area networks using a token-passing bus access method. This standard defines all of the Physical Layer and part of the Data Link Layer. For the Physical Layer it specifies the electrical and physical characteristics of the transmission media and the electrical signaling method. MAP originally chose the CATV industry standard coaxial cable using broadband, and later added carrierband as a way of reducing the per-station interface costs.

Broadband signaling on coaxial cable uses radio frequencies on the wire itself. Multiple channels are possible by the manner in which the frequencies are divided. The IEEE 802.4 standard recommended by MAP is a 10 megabit per second duobinary broadband coaxial cable with mid-split provision for two-way data flow. This would be MAP's backbone physical medium. Broadband was chosen for its ability to support multiple networks on the same medium. It simultaneously supports real-time voice and video, clearly a direction of future networks, via frequency division multiplexing.

The carrierband alternative supports only one channel. This standard is a 5 megabit per second phase-coherent carrierband. Carrierband is a less complicated technology which does not require the headend remodulator of broadband, hence it is expected to increase reliability and reduce cost. It is appropriate for control and supervisory communications, as one might find on a factory floor.

1.2.1.2. Layer 2: Data Link Layer

The Data Link Layer provides for and manages the transmissions of individual frames of data, ensuring error detection. A *data frame* is a collection of data and control bits delimited at the start and end by special markers. Instead of ensuring that each bit individually arrives at its destination intact, data frames are handled as units. In this manner the Data Link Layer provides a data transfer service to the Network Layer.

The ISO layer specifications allow for any layer to be subdivided, as long as the sum of the divisions can still perform the layer's tasks. The Data Link Layer can further be divided into two distinct sublayers. Part of the IEEE 802.4 standard governs the physical specifications (this is the Physical Layer specification) while the other part governs how that physical medium is accessed. This Medium Access Control (MAC) sublayer specifies the lower half of the Data Link Layer (see Figure 1.4). The upper half of the Data Link Layer is called the Logical Link Control (LLC) sublayer, and it provides for a logical connection between two adjacent nodes by using error detection as specified by the IEEE 802.2 Logical Link Control [IEEE85b] standard. The MAC and LLC sublayers are completely independent of each other, but together provide all of the required layer 2 functionality.

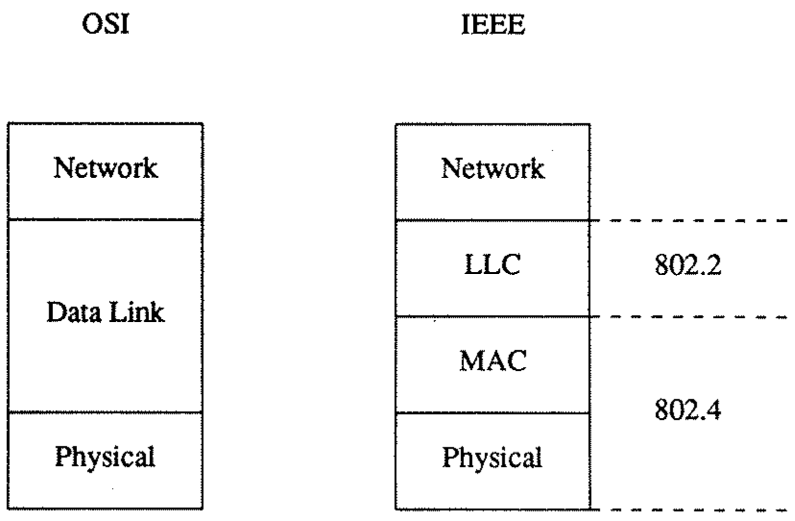


Figure 1.4 — Layer Subdivision in the IEEE Model

The MAC sublayer specified by MAP is the token passing bus access of IEEE 802.4. The MAC is responsible for several actions. First, it provides for the maintenance of the token, which is the "permit" passed between nodes granting transmission privileges. The MAC is also responsible for the maintenance of the logical ring of nodes. Although the physical topology is a bus with nodes attached to one cable, MAC ensures that each node has two logical (not physical) neighbors, a predecessor and a successor. The token passes from neighbor to neighbor, eventually returning to the first node to complete the logical ring. MAC also handles error detection in addition to message and priority management.

The IEEE 802.4 MAC specifications were adopted by MAP for several reasons. Only the IEEE 802.4 token bus presently supports the Data Link protocol on broadband media. Many vendors already have programmable devices that are somewhat based on token bus technology. Messages can be assigned priorities, and the protocol guarantees that the highest priority messages will be delivered within a measurable and deterministic time limit. This is essential for factory floor communications.

The Logical Link Control is specified by the IEEE 802.2 standard. There are three types of service: connectionless mode, connection-oriented, and connectionless mode with acknowledgements (called LLC Type 1, 2, and 3, respectively). Connectionless mode service is sometimes called datagram service, and it provides for the exchange of data between two Logical Link entities without first establishing a connection. This resembles mailing a letter. Connection-oriented service is sometimes called virtual circuit service because it sets up a (seemingly) direct line between Logical Link entities, as a telephone call would between people. Type 3 service is a proposed datagram service with acknowledgement of data frames. MAP specifies using the Type 1 connectionless service, and allows the use of Type 3 service (datagrams with acknowledgements) for improving the reliability of the LLC, as would be needed for the MAP/Enhanced Performance Architecture. The LLC is discussed in more detail in Chapter 2.

A subnetwork is a collection of interconnected nodes for which routing of messages is not necessary for data transfer. Layer 3 handles the routing of messages in the larger interconnection of subnetworks. Ideally, each node in a subnetwork would be a MAP node, that is, it would support the full MAP architecture, but for historical and migratory reasons, this is not a reasonable expectation.

1.2.1.3. Layer 3: Network Layer

The Network Layer provides services to convert global address information into routing information so that a message can be delivered from one end-node to another. To do this it must maintain routing tables and/or algorithms, establish and terminate network connections when appropriate, and provide switching services to incoming messages to route them onto the proper outgoing path.

The MAP specifications allow for four sublayers: the inter-network sublayer, the harmonizing sublayer, the intra-network sublayer, and the link access interface. The inter-network sublayer is responsible for routing information for end-node-to-end-node information flow. Datagrams traverse possibly multiple LANs without regard to an individual LAN's local routing scheme. The standard chosen to do this is ISO 8473 Data Communications Protocol for Providing the Connectionless-Mode Network Service [ISO8473], called CLNP for Connectionless Network Protocol. CLNP describes network service primitive functions and specifies addressing conventions. In a pure MAP environment, this would be the only routing methodology needed, as this sublayer and the lower two layers would comprise all that is necessary for routing and delivery. However, since MAP is also trying to provide for a migration path from existing proprietary systems to this standard, the next two sublayers are concerned with interfaces to non-MAP networks.

Presumably a MAP node could have access to a network where the local addressing and routing schemes are different. The harmonizing sublayer is an implementation-dependent facility for converting inter-network addresses to local addresses and providing intermediate functionality where needed to adapt the global routing requirements to the available local routing services. If this functionality is not required, then this would be a null sublayer.

The intra-network sublayer then provides all the routing and switching of messages to, from, or through a particular node within the immediate local network. This local network would be a set of interconnected nodes communicating with a common, non-MAP, routing protocol. Again, in an ideal MAP environment, this sublayer would also be null.

The link access interface provides the necessary functional interface between the Network Layer and the Data Link Layer. It may have to provide the translation between connection-oriented communications at the Network Layer and a connectionless mode service on the Data Link Layer. Whatever the case, its job is to provide the logical interface between the Network and Data Link Layers.

Above the Network Layer are end-to-end communications in which the concept of a subnetwork should be mostly transparent as should any details of how the data is transferred between peer layer entities.

1.2.1.4. Layer 4: Transport Layer

The Transport Layer exists solely to provide reliable and transparent transfer of data between session or transport user entities. It provides this service without regard to the underlying system of subnetworks or their Network Layer protocols. It relieves the transport user from any concern with how the data is transferred, where it must go to arrive where intended, or the ordering of the data as it arrives.

The Transport Layer provides two types of services: connection management and data transfer. The facility for the creation and deletion of a data path to a peer transport user is provided in connection management. In data transfer, the data may be sent by the normal path or by expedited means when the message is urgent. To provide these services MAP has selected the ISO 8073 Transport Protocol Specification [ISO8073] Class 4. This protocol has provisions for flow control for data transfer efficiency. It can multiplex users, providing each user with access to the network for transmissions. It can detect errors and recover from lost, damaged or out-of-sequence packets of data. An addendum to the standard (ISO 8602 Connectionless-Mode Transport Service [ISO8602]) provides a datagram oriented service, which makes it popular among U.S. computer manufactures for its support of a wide variety of network sublayers. These standards are discussed in detail in Chapter 2.

1.2.1.5. Layer 5: Session Layer

The purpose of the Session Layer is to enhance the services provided by the Transport Layer with mechanisms for managing and structuring reliable data transfer. Such structuring is achieved in three ways: two-way alternate (half-duplex), one-way interaction (simplex), and two-way simultaneous (full-duplex) data transfer. MAP specifies the ISO Basic Connection Oriented Session Protocol Specification [ISO8327] for this layer, with full-duplex communications.

The process running the Session Layer will be controlled by a state table, where changes in the status of the session will result in state table updates. Typically the state tables consist of a well-ordered series of events and alternatives for establishing, maintaining, and closing a dialogue between session users. These tables monitor the success of session events and

Transport Layer function usage. They also provide major and minor checkpoints for rollback to keep the dialogue synchronized. Upon an error or desynchronization, the Session Layer manages the return to one of these checkpoints (the last place in the dialogue which was mutually agreed to be error-free).

1.2.1.6. Layer 6: Presentation Layer

The purpose of the Presentation Layer is to negotiate a transfer syntax for use by the Application Layer. This layer may actually cause the representation of the data unit to be changed. MAP 2.2 does not specify a Presentation Layer, and thus it is a null layer. However, future MAP versions (beginning with version 3.0) will specify the Presentation Layer as using ISO 8822 Connection Oriented Presentation Service Definition [ISO8822].

1.2.1.7. Layer 7: Application Layer

The Application Layer is the interface between user programs and the network. It defines the way in which communication functions are made available to a user program and manages the details required to support that communication. User programs access the network through service elements, using these elements in a prescribed manner. The interface hides the implementation details of the rest of the network from the user, providing only those elements necessary for user programs to establish connection, transfer data, and terminate the connection.

The MAP specification of the Application Layer is a modified subset of the ISO Common Application Service Elements [ISO8649], called MAP Common Application Service Elements (CASE). MAP CASE provides the five service elements most often used by application processes. Also present at this layer are Specific Application Service Elements, or SASEs, which provide service elements used only in special applications. An example of a SASE specified by MAP is ISO File Transfer, Access, and Management (FTAM).

The User Element is the interface to the area outside of the OSI environment. Outside of this environment are the users of the network. Within the Application Layer are the service elements to provide network functionality. The User Element bridges between the two. The User Element has access to MAP CASE and the SASEs, and provides services through them to the application process.

1.2.1.7.1. Common Application Service Elements (CASE)

The purpose of MAP CASE is to provide an application association between application processes in different nodes in the MAP network. An application association is established when two application processes want to communicate. The building of the underlying connection occurs when lower level protocol entities are invoked to provide such connection establishment. No knowledge of how these lower level protocol entities actually execute this establishment is necessary. Herein lies an important difference between the ISO CASE and the MAP CASE: MAP CASE is mapped directly onto the Session Layer services, whereas ISO CASE provides for the use of the Presentation Layer services. When later versions of MAP are adopted, the Presentation Layer will no longer be null and the mapping of MAP CASE onto the rest of the network will include the Presentation Layer.

MAP CASE provides services and protocols to allow for two application processes to transfer information. A mechanism for presenting and supporting a common MAP addressing and naming scheme allows for logical names to be transparently mapped to network names and addresses. Security is addressed by providing a framework for a MAP authorization and authentication facility. A standard encoding for MAP applications to communicate reasons for refusing an application association, as well as signaling normal and abnormal termination, is also provided.

There are only five service elements in the MAP CASE group. These elements provide the functionality to establish, use, and terminate an application association. They include A-ASSOCIATE, which is used to establish an application association between two application processes; A-RELEASE, which allows either of the two MAP CASE users to terminate the application association without loss of information in transit; A-U-ABORT, which is used by either MAP CASE user to terminate the application association abnormally; A-P-ABORT, which informs each of the two MAP CASE users that the underlying services are terminating abnormally or that MAP CASE detects an error indicating abort; and finally A-TRANSFER, which is used to transfer data between MAP CASE users. A finite state machine controls these service elements.

1.2.1.7.2. File Transfer, Access and Management (FTAM)

The File Transfer, Access and Management (FTAM) [ISO8571] protocol provides services that will support transfer of both binary and ASCII format files, as well as the remote creation and deletion of files. The FTAM services are related to a local filestore, where incoming or outgoing files are buffered. FTAM uses the services of CASE and the Session Layer directly to transfer files across the network. A finite state machine is responsible for monitoring the use of these services.

Files are of two types, binary and ASCII. Binary files are octet string encoded with no embedded structuring recognized. ASCII files, on the other hand, are structured by variable length lines ending in carriage return and line feed. Neither the carriage return nor the line feed can ever appear alone, as they are the end-of-line delimiters.

MAP specifies that ISO 8571 File Transfer, Access and Management be the defining document. There is an allowance for a second phase of implementation of FTAM, where an enriched set of file formats and a limited set of file access capabilities would be included. Other protocol specifications are under review for this next phase. Furthermore, as MAP CASE services mature, it is expected that FTAM will be integrated into the Application Layer functionality.

1.2.2. Bridges, Routers and Gateways

The interconnection of a set of networks is called a Catanet. MAP recognizes the need for a Catanet within a facility, connecting MAP networks to both MAP and non-MAP networks. There are three types of devices used to accommodate the three different layers where points of connection could be established.

Bridges are devices that can connect two or more subnetworks at the Data Link Layer. Their actions are transparent to the network because they actually participate in the token passing in all subnetworks to which they are connected. Upper layers of the communicating nodes have no need to know that the messages they are sending are going through a bridge to get to their destination. Since a bridge is a Data Link connector, the MAC sublayers must be compatible or the same. MAP bridges will be used in very clearly defined situations and will only connect segments with similar MAC sublayers, which for MAP means connecting one IEEE 802.4 token-passing bus to another IEEE 802.4 token-passing bus. Major uses, then, of the bridge device will be connecting carrierband subnets to the broadband backbone network or interconnecting different channels of a broadband system. Figure 1.5 shows an example of a bridge architecture.

A bridge can connect two or more sets of Physical and MAC layers. The Interdatalink that interconnects the layers uses the MAC services to gain access to the data, then it does whatever translation is needed between the Physical Layers. The Interdatalink is not the

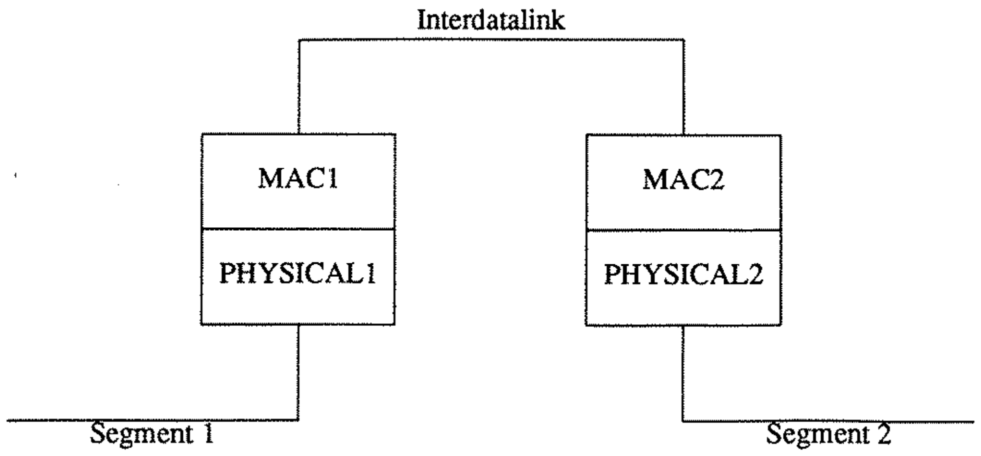


Figure 1.5 — Architecture of a MAP Bridge

Network Layer, or a separate layer at all; it is simply a connector of MAC sublayers.

Routers interconnect networks at the Network Layer when different LLC sublayers are implemented on segments. This means that networks with different Data Link and Physical Layers can be connected together. The router has a known address, and any message needing delivery to another network connected to the router must first traverse the router. Figure 1.6 shows the structure of a MAP router. The Internetlink performs any necessary address translations, but is not a separate layer; it merely interconnects network layers.

Gateways interconnect different network architectures by performing protocol translation. Gateways are used to interconnect completely different networks (either local area networks, or wide area networks, or any combination). Some proprietary and commercial networks are not based on the OSI seven layer architecture, so gateways are also used to cross the boundary between ISO-based and non-ISO based networks. Figure 1.7 shows the architecture of a gateway. Note that layers X1 and X2 represent peer layers of nodes in different networks (i.e. layer TRANSPORT1 might be the OSI Transport Layer of MAP on a local area network, while

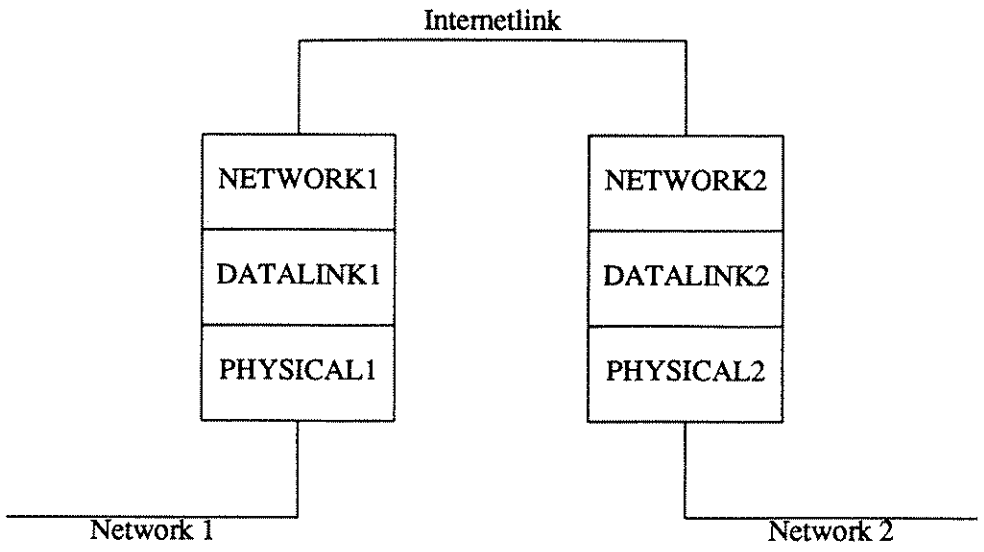


Figure 1.6 — Architecture of a MAP Router

layer TRANSPORT2 might be a TCP Transport Layer in an X.25 wide area packet-switched commercial network).

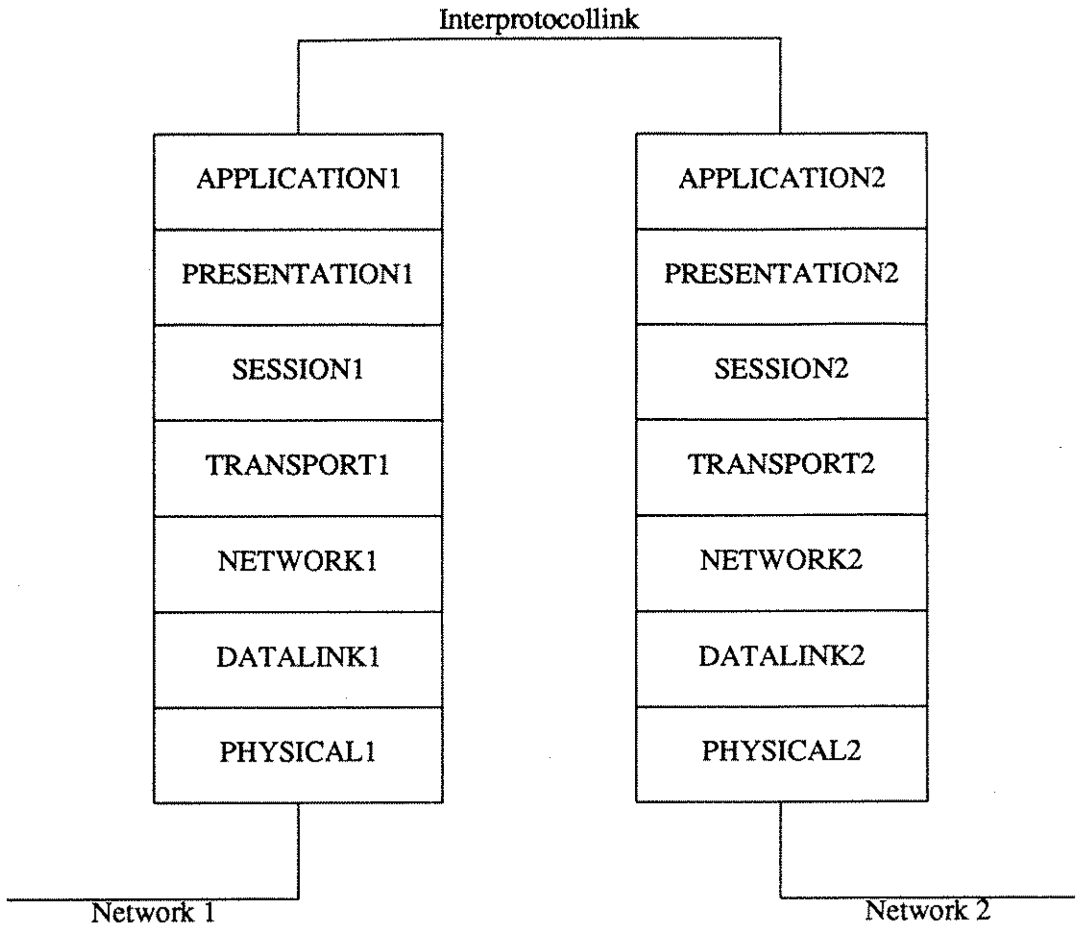


Figure 1.7 — Architecture of a MAP Gateway

1.2.3. EPA - The Enhanced Performance Architecture

The Enhanced Performance Architecture, or EPA, is a subset of the MAP architecture that is designed to provide faster message response times at the sacrifice of upper level functionality. A full MAP node would include all seven layers according to the MAP specification. EPA bypasses the upper layers, connecting the application processes directly to the Data Link Layer in an effort to streamline critical communications. It is expected that most nodes would have both architectures; that is, be able to provide full MAP functionality with the option to provide

EPA in special circumstances. A node that provides only MAP is called Full MAP, a node with both is called MAP/EPA, and a node with only EPA is called Mini-MAP. Figure 1.8 compares the Full MAP, MAP/EPA and Mini-MAP architectures.

In the Full MAP implementation, each layer provides a service through which peer-to-peer communication of differing computing devices can be achieved. All seven layers are implemented. MAP/EPA provides Full MAP capabilities during normal factory floor communications as well as a method for obtaining faster response time for use on control and

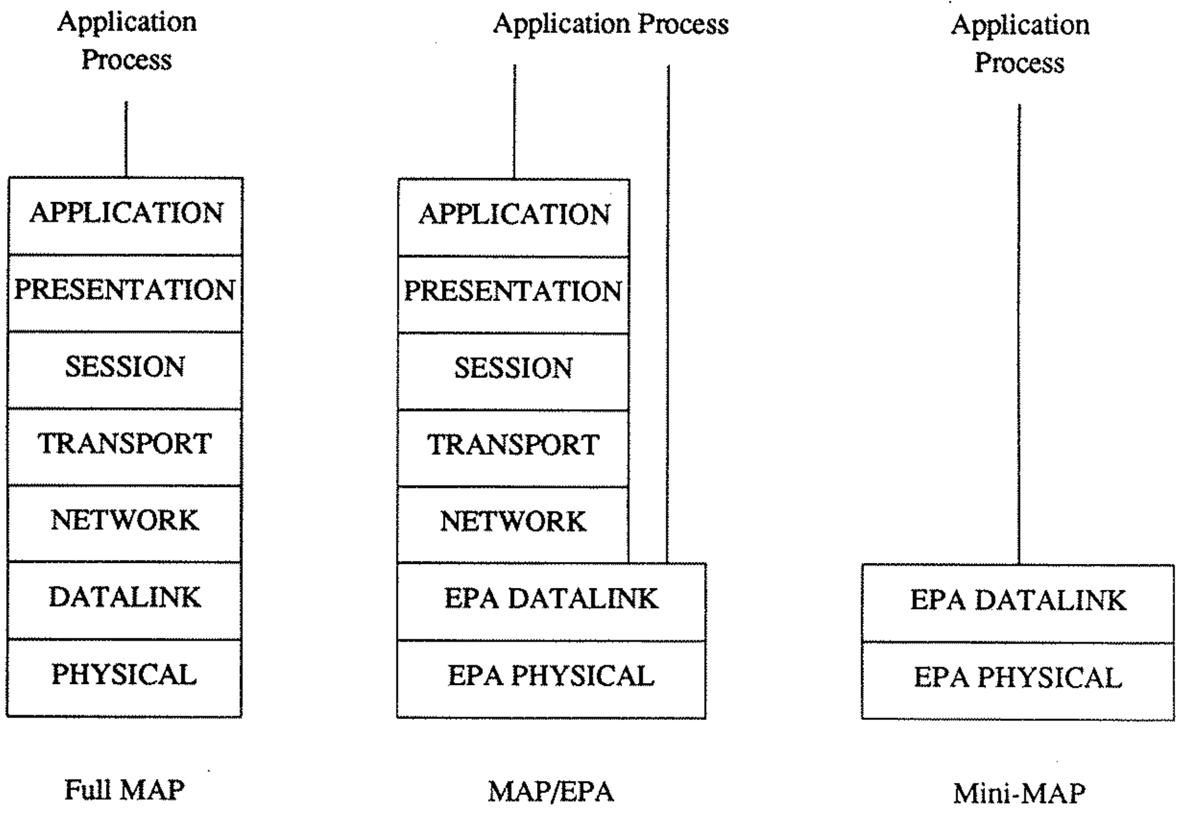


Figure 1.8 — Comparison of Full MAP and MAP/EPA

time-critical networks. It is designed to exist in specific environments for specific types of communications. This special case is usually a small, tightly coupled group of devices which require rapid response, use small messages, and need little or no support from a directory server.

An example of where MAP/EPA may be used effectively would be a workstation with several intelligent devices and a computer controller. A robot may be in charge of taking a part blank from the materials tray and loading it into the vice of a milling machine. The robot must report its status very often, informing the controlling computer where it is and what it is doing. These status data frames must be delivered quickly, as the robot is moving in real-time and the information is short-lived. Small messages, rapid response time and no worry about the occasional loss of a message are characteristics of this application. These are also the characteristics of EPA. If the controlling computer is connected to the MAP backbone, it must have the full MAP architecture. However, to facilitate monitoring the status of the robot, EPA is more appropriate. Thus, the computer controller should have the dual MAP/EPA architectures.

In order to achieve the quickened communications promised by EPA, several layers in the OSI reference model have been bypassed. These layers provide functionality necessary for more complex and leisurely communications, which increases the processing and response times. By eliminating the layers, the procedural interface between the layers and the control information used by the layers is also removed. This lessens the processing time by reducing the amount of software to actually handle the message and reducing the message overhead due to headers. There is a price to pay for such expedience, however.

In obtaining this streamlined architecture, the application processes interface directly with the Data Link Layer. This means functionality is lost and restrictions apply. Skipping the Presentation Layer means no negotiation or changing of the message encoding scheme or its transfer syntax. The presentation syntax must be known *a priori* for applications that use EPA services. Data streams cannot be monitored by the checkpoints and the resynchronizing services that the Session Layer normally provides. The Transport Layer is not present to guarantee message delivery. The only assurance given is that a "best effort" attempt will be made to deliver the message. High quality guaranteed message delivery is therefore sacrificed. The messages are restricted to transfer across a single network segment because the Network Layer internet services are missing, so there can be no global delivery of messages. Only one unacknowledged message no larger than the Data Link protocol data unit may be sent. This reduces throughput if errors occur frequently, and restricts the length of the message because no facility exists for disassembling and reassembling messages. The type of service can only be connectionless (datagram, or Type 1), with the option of immediate acknowledgement (Type 3). If the Type 3 acknowledged message is used, this becomes a "stop-and-wait protocol" (the sender stops and waits after each message transmission until an explicit acknowledgement of the last message sent is received).

The specifications for EPA include the IEEE 802.4 phase coherent carrierband for the physical medium. The Data Link Layer is IEEE 802.4 MAC and IEEE 802.2 LLC, the same as for a full MAP architecture. These layers will be managed by the same network management scheme as the full MAP, as specified in IEEE 802.1B Station Management [IEEE85a].

It is important to realize that the Mini-MAP is not a MAP node, and thus must be connected to MAP by either routers, gateways or bridges. A Full MAP node cannot communicate with a Mini-MAP node except through one of these devices. In this respect, EPA is not MAP compatible, but actually a separate subnetwork of devices which can be connected to a MAP network.

1.3. Network Management

The purpose of Network Management is to gather information on the usage of the network media, ensure the correct operation of the network, and provide reports. This information is then made available for three types of users: those concerned with the maintenance of the network, those concerned with the operations within the network, and those concerned with the planning of network design. The maintenance aspect uses the data to provide problem detection and diagnosis, preventive maintenance, and the installation of new nodes. The operations aspect uses the data to provide performance monitoring and network access management. The concern in planning is to design, model, and simulate new network configurations.

Three entities exist within a network for gathering all of this information: a human operator, a management applications processor, and management agents. The human operator invokes the management applications processor (manager) and requests services from it. This manager is an application program, so it uses the services of the Application Layer to communicate to various nodes in the network. For each node there is an agent which receives a message from the manager and reacts or responds to it.

The type of information MAP has specified for network management falls under the categories of configuration, performance, event, and fault. Configuration management is concerned with the current system state. Performance management reports on system statistics, such as the frequency of event occurrence. The event processor reports the occurrence of a state changing event. Fault management verifies the system state, isolates and corrects faults. There is a management applications processor for each of these categories, allowing the human operator to collect, control, store or present the information requested by use of the manager's functionality. The types of data returned may be real-time, as in the current state of a node, or administrative, as in a software version number. The manager's functions access this data by using management messages and parameters.

The typical System Management Protocol Data Unit (SMPDU) is comprised of (1) the type of the message, either request, response, or event; (2) the layer to which the request is directed; and (3) one or more fields for information to be sent or received. These SMPDUs are given to the Application Layer for delivery to the agent, which also responds via the Application Layer. The manager makes all of the connections, initiates all the transactions, and closes all channels. The agent must maintain a storage and retrieval system at each level for the information that may be requested.

MAP specifies that only layers 3 through 7 are responsible for such Network Management dialogue. Layers 1 and 2 will be handled, at least initially, by another management scheme. The domain of the Network Management extends only so far as the environment is MAP-based, and never extends through gateways or routers to non-MAP networks.

1.4. Migration Path

General Motors recognizes the fact that vendors have been making their own proprietary communications systems for some time. Given that change takes time, MAP has laid out a set of interim recommendations for both equipment manufacturers and GM MAP system architects. The MAP environment will be present from the start in the form of a MAP backbone network, and from it the evolution of devices will occur. As a device becomes more MAP-like, it moves closer to becoming a full MAP node on the MAP backbone or on a MAP segment.

In the short term, the MAP backbone will consist of a token passing bus network on broadband coaxial cable. Non-MAP subnetworks will be attached to the backbone by way of gateways, which will perform the necessary translation from non-MAP to MAP network protocols. Any device on the non-MAP subnet will be an indirect participant on the MAP network. In the long term, these non-MAP subnetworks will be replaced by MAP subnetworks or the devices will be placed directly on the backbone as participating MAP nodes.

Chapter 2

STANDARDS

2.1. Introduction

The suite of experiments described in Chapter 4 measured the performance of Intel's MAPNET2.1 [INTE87a] implementation of MAP 2.1 at the Data Link and Transport Layers. The following is a detailed review of the standards used at these two layers, with particular emphasis on the type or class specifically required for MAP 2.1.

2.2. Transport Layer Introduction

The Transport Layer is the fourth layer in the ISO OSI reference model. It provides transparent transfer of data between transport users, relieving the user of any concern with the details of how the data is transferred. There are degrees of reliability offered and the techniques for ensuring this reliability are also transparent. This transparency is reflected in the network independence characteristic of the Transport Layer.

There are two types of Transport Layer Services defined by ISO. The first is specified by ISO 8073 Connection-Oriented Transport Layer Service [ISO8073]. It provides the transport user with connection management and data transfer services through *virtual circuits*. The second is specified by ISO 8602 Connectionless Mode Transport Layer Service [ISO8602]. It provides a *datagram* service for data transfer without connection maintenance.

2.2.1. Connection-Oriented Transport Layer

The Connection-Oriented Transport Layer provides two kinds of services to the transport user: connection management and data transfer. The transport connection management service provides the facility to create, maintain, and destroy a data path between peer transport processes. The data transfer service provides normal and optionally expedited transfer of data between peer transport processes.

The transport entity will negotiate with its peer for the quality of service to be provided by this layer. The criteria are: what is required by the transport user, what is provided by the underlying network, and of that which is requested and available, which is affordable with respect to the cost of providing that service. The negotiations take place during connection establishment, and in part account for the multi-way handshake involved.

A transport user accesses the Transport Layer through a Transport Service Access Point (TSAP). The user passes service data units to the Transport Layer and receives service data units from the Transport Layer through the TSAP. These data units are called Transport Service Data Units (TSDUs). The Transport Entity operates on incoming TSDUs, preparing them for delivery to the user by processing header information, and on outgoing TSDUs, preparing them for departure on the network. The TSDUs are transformed into one or more Transport Protocol Data Units (TPDUs). The Transport Layer specified by the ISO 8073 document assumes the presence of a Network Layer with certain services. The TPDUs that are passed down to the Network Layer then become the Network Layer's Network Service Data Units (NSDUs), and they come and go at one or more Network Service Access Points (NSAPs) (see Figure 2.1).

When a large TSDU is passed to the Transport Layer, it may have to be broken down into smaller, more manageable TPDUs. This is called *segmenting*, and it is the transport provider's

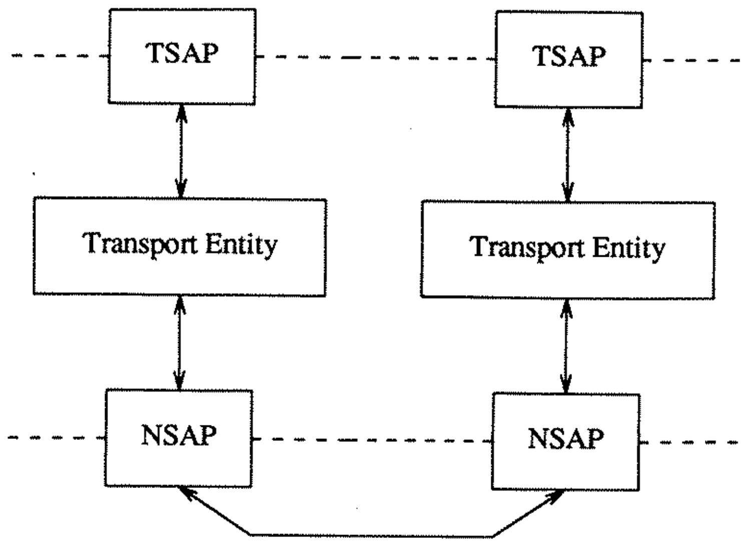


Figure 2.1 — Model of Service Access Points

responsibility to *reassemble* these pieces into the original TSDU upon delivery to the peer transport user.

One transport connection may use two or more network connections to improve throughput. Because the user is unaware of how the Transport Layer provides the cost-effective, reliable data transfer, it is not the user's task to manage these multiple connections. When the transport provider does this, it is called *splitting* and *recombining*. As in the case of segmenting and reassembling, the transport provider must recombine the pieces of the message before it can be delivered to the user.

Multiplexing and *demultiplexing* allow two or more transport users to use the same network connection. Again, this is transparent to the users and it is the responsibility of the transport provider to make this service transparent.

Sometimes the size of several TSDUs is small enough that they can be placed into the same TPDU. This is called *chaining*. Of course, it is the job of the transport provider to *decouple* these messages before final delivery.

2.2.1.1. Classes Of Service

A class of service is a set of functions providing a level of quality of service to the transport user. The options are functions within a class which may or may not be included. There are five classes of service available:

Class 0 - Simple Class
Class 1 - Basic Error Recovery Class
Class 2 - Multiplexing Class
Class 3 - Error Recovery and Multiplexing
Class 4 - Error Detection and Recovery with Multiplexing

Class 0 is the simplest of the classes. It provides the barest of transport services, allowing the maximum level of errors acceptable. Residual and signaled errors are acceptable without detection or recovery.

Class 1 provides the basic transport services with minimal overhead. It is designed to be recoverable from network layer disconnect or reset, but still allows residual errors to go undetected.

Class 2 provides Class 0 functionality with the addition of multiplexing. As stated above, multiplexing is the use of a single network connection to provide service to two or more transport connections. No detection or recovery from residual or signaled errors is provided.

Class 3 provides Class 1 functionality with the addition of multiplexing. Now the system can recover from signaled errors, but the residual errors still go undetected.

Class 4 provides the capability to multiplex, as well as to detect and recover from errors which are the result of a low grade of service from the network provider. Errors detected include TPDU loss, TPDU delivery out of sequence, TPDU duplication, and TPDU corruption. By providing these services efficiently, increased throughput and additional resilience to network failure can be observed.

2.2.1.2. Class 4 Transport

Class 4 provides the highest degree of reliability for an error-prone network. Many mechanisms and timers are required to provide this level of service.

2.2.1.2.1. Class 4 Mechanisms

Sequence numbers tag the TPDUs with identifying and ordering information. Each TPDU is stamped with a unique number, one more than the last TPDU processed, and one less than the next TPDU in line. This number is then used to acknowledge receipt and order TPDUs before handing them up to the transport user.

A *window* is an ordering of sequence numbers that are termed active. This may be the list of sequence numbers on TPDUs that can be transmitted or received. The window extends from the next sequence number to be processed to one greater than the last sequence number for which processing is possible.

A TPDU with the special purpose of acknowledging the receipt of one or more data TPDUs is called an *acknowledgement* (ACK). ACKs carry the sequence number of the next active data TPDU (one greater than the sequence number that is being acknowledged) and a *credit* field. The credit field in an ACK is the means for specifying how many sequence numbers after this one are considered active by the receiving user. As a consequence ACKs also represent permission to transmit. A *closed window* has a credit of 0, which effectively turns off the transmitting node. An ACK with updated credit information may be sent at any time to control, or "throttle", the transmitting node.

2.2.1.2.2. Class 4 Timers

There are several timers that are useful in flow control and error recovery. These timers "timeout" when they reach certain set values. The values for the timers are determined from negotiations or supplied by the underlying network. These values are expected lengths of time for certain events, and if the event does not occur within that expected time an exception is signaled.

Since the Transport Layer uses the services of the Network Layer, a Transport Protocol Data Unit becomes a Network Service Data Unit when it is passed to the Network Layer. The Network Layer provides the value of the length of time an NSDU can remain in the network. These values, called NSDU lifetimes, are referred to as M(rl) for remote to local, and M(lr) for local to remote. This is a guarantee from the Network Layer that if this time has passed then the NSDU will be discarded.

A value for the expected maximum transmit time is the total amount of time necessary for a TPDU to be transmitted from the sending node to the receiving node. It is referred to as E(lr) for local to remote transmit, and E(rl) for remote to local transmit. The amount of time a receiving node can wait before it must acknowledge a received TPDU is called A(l) for the local value, and A(r) for the remote. A value for the local retransmission timer (T1) can be calculated using the the above values.

$$T1 = E(rl) + E(lr) + A(r) + X$$

where X is an amount of time to allow for system latency.

The number of retransmissions of a particular TPDU is bounded to prevent the Transport Layer from sending infinitely many retransmitted TPDUs over a bad network. This number is called N. The persistence time (R) is the length of time before the Transport Layer stops trying to transmit over the network, and is calculated as:

$$R = (T1 * N) + Y$$

where Y is system latency.

The number of sequence numbers is finite, so they will have to be reused after some finite number of TPDUs have been transmitted. There is a bound (L) on the amount of time that should pass before a sequence number can be reused:

$$L = M(rl) + M(lr) + R + A(r).$$

To ensure that peer Transport Layers are still communicating, an inactivity timer (I) is used. When the amount of time specified by this timer has passed without a response from the remote Transport Layer, the local Transport Layer initiates a release due to inactivity. To keep the window information current, and to ensure that flow control is affected, a window timer (W) signals for periodic transmission of window and credit information.

2.3. Connectionless Mode Transport Layer

Connectionless Mode transfer of data at the Transport Layer provides the transport user with a single-access data transfer without the overhead of transport connection establishment. This service is intended for the benefit of those applications that require a one-time, one-way transfer of data, towards one transport user, taking full advantage of mechanisms which are simpler than those used in Connection-Oriented Transport Services.

2.3.1. Functions

The functions provided are at least those necessary to bridge the gap between the service available from the Network Layer and the service to be offered to the Transport User. They are concerned with the enhancement of quality of service in a cost optimizing manner. These functions include selection of the Network Service that best fits the requirements of the Transport Service User, address mapping for determining the destination address from the function parameters, TSDU delimiting, and error detection.

2.3.2. Model of Connectionless Mode Transport Service

The model of the connectionless mode Transport Service is similar to the model for connection-oriented Transport Service. The transport entity communicates with the Transport Service User through one or more TSAPs by means of transport service primitives. These primitives cause or are the result of the exchange of TPDUs between peer transport entities. Protocol data units are exchanged by making use of the Network Layer services.

Transfer of data may occur over a connectionless mode Network Service, or a connection-oriented Network Service. The purpose of using a connectionless Network Service is to provide the one-time, one-way transfer of a TSDU between Transport Service Users without confirmation of receipt, without transport connection establishment and release, and without network connection establishment and release. Connection-oriented Network Service satisfies the purpose of connectionless mode Network Service with the added reliability of a network connection at the added cost of that connection overhead.

2.4. Data Link Layer

The Data Link Layer is the second layer in the ISO OSI reference model. In the IEEE family of standards, the Data Link Layer is divided into two sublayers, the Medium Access Control (MAC) and the Logical Link Control (LLC). The IEEE 802.4 Token-Passing Bus Access Standard [IEEE85] describes the Physical Layer and the MAC sublayer. The LLC sublayer is described in the IEEE 802.2 Logical Link Control Standard [IEEE84b], to be used in conjunction with the MAC of the IEEE 802.4. It is the LLC sublayer which provides the service interface to the Network Layer or any other Data Link user, and thus it is discussed in detail here.

The services of the Data Link Layer are the capabilities that it offers to the data link user. These services are provided at the LLC sublayer, and are built upon the services provided by the MAC sublayer below it. Services are specified by describing the information flow at the interface between the service user and the service provider, or Data Link Entity. This is implemented by passing service primatives to the Data Link Entity through a Link Service Access Point (LSAP).

2.4.1. Types of Service

The services provided by the LLC to its user include peer-to-peer protocol procedures that are defined for the transfer of information and control between any pair of LSAPs on a subnetwork, independent of the particular MAC being implemented on that subnetwork.

There are two types of operations provided by the LLC sublayer. The first type provides unacknowledged data link connectionless service across a data link. This is a means by which users can exchange Data Link Service Data Units (DLSDUs) without first establishing a Data Link Level connection. Since there is no connection, there are only data transfer primatives associated with this type of service. This is called Type 1.

The second type provides a data link connection-oriented service across data links, supporting sequenced delivery of data units and error recovery techniques. Primatives are provided that establish the connection, allow transfer of DLDSUs, reset a connection to an initial state, terminate the connection, and control the flow of data associated with a specified connection across the interface from user to provider. The Data Link Protocol Data Units (DLPDUs) will be assigned sequence numbers, and acknowledgements verifying receipt will indicate what sequence number is expected next. The LLC Entity is also responsible for error recovery and flow control. This collection of services is called Type 2.

Driven by the new RS-511 Manufacturing Message Service (MMS) [ELEC86] specification for MAP 3.0, a third type is proposed. This would be a hybrid of the first two types, combining connectionless service with acknowledgements. Any DLPDU received that is of Type 3 would send back an acknowledgement. This type of service would be for the important commands that would need a verification of receipt, as are present in some MMS primatives used in a MAP/EPA environment.

2.4.2. Classes of Procedure

Various Classes of Procedure incorporate one or more of these types. Class I LLC is the support of Type 1 services only. Class II LLC is the support of Type 1 and Type 2 services, with the ability to switch between the two on a DLPDU-by-DLPDU basis, if necessary.

2.4.3. Logical Link Control Protocol Data Unit Structure

IEEE 802.2 specifies the format to which all LLC PDUs must conform, as shown in Figure 2.2. The address representation is in the form of two one-octet fields. The Destination

Information	Control	SLSAP Address	DLSAP Address
8*M bits	y bits	8 bits	8 bits

where y is either 8 or 16 and M is an integer value equal to or greater than 0.

Figure 2.2 — Format of the Logical Link Control Protocol Data Unit

Link Service Access Point (DLSAP) has 7 bits for the actual address and 1 bit for identifying the address as either an individual or group address. The Source Link Service Access Point (SLSAP) also has 7 bits for the actual address and 1 bit for identifying if the LLC PDU is a response or command. The Control field consists of one or two octets and is used to designate command and response functions, and contains sequence numbers when required. The Information field consists of any integral number (including 0) of octets.

Chapter 3

ENVIRONMENT

3.1. Introduction

A performance analysis was designed and experiments were performed on Intel's implementation of GM MAP, using Intel hardware and software products. These experiments were intended to measure various aspects of performance at the Data Link and Transport Layers in the ISO stack. The hardware environment included Intel computer hardware, timing hardware and communications hardware. The specific hardware and software are described below.

The experiments were performed using two Intel 286/310 computer systems serving as nodes on a MAP network. MAP communications services were provided by MAPNET2.1 [INTE87a,b], Intel's implementation of MAP version 2.1. MAPNET2.1 software provided the OSI Application, Session, Transport, Network, and Logical Link Control Layers. The iSXM554 hardware provided the Medium Access Control (MAC) sublayer and the interface to the 10 megabit/second IEEE 802.4 token bus broadband network. The Intel product iNA960 [INTE86a,b,c] provided a subset of the MAPNET2.1 services, specifically the Transport, Network, and Logical Link Control Layers. iNA960 was the product actually used in the experiments because it provided Transport and Data Link services in exactly the same manner as MAPNET2.1 without the complication of the upper layers. All the MAPNET2.1 or iNA960 software resided on the front-end processor.

Our application processes, written in 'C' and 'PLM' and resident on the host processor, communicated with MAPNET2.1 and with iNA960 using another Intel convention called Multibus Interprocessor Protocol (MIP) [INTE86c]. MIP allows the host processor (the 80286) and the front-end processor (the 80186 on the iSXM554) to communicate request blocks by simply passing pointers to shared memory via a Message Delivery Mechanism (MDM). If the block to be transferred is not in shared memory, MDM will copy the block from host memory into the front-end processor's packet buffer. MIP isolates user tasks from the complexities of communicating across the Multibus. MIP handles the interaction between the host processor, the front-end processor, and other intelligent Multibus devices. MIP supports functions such as locate a port, attach/de-attach a task to/from a port, and transfer/receive a buffer to/from a port. Figure 3.1 shows the relationship between MAPNET2.1 and iNA960.

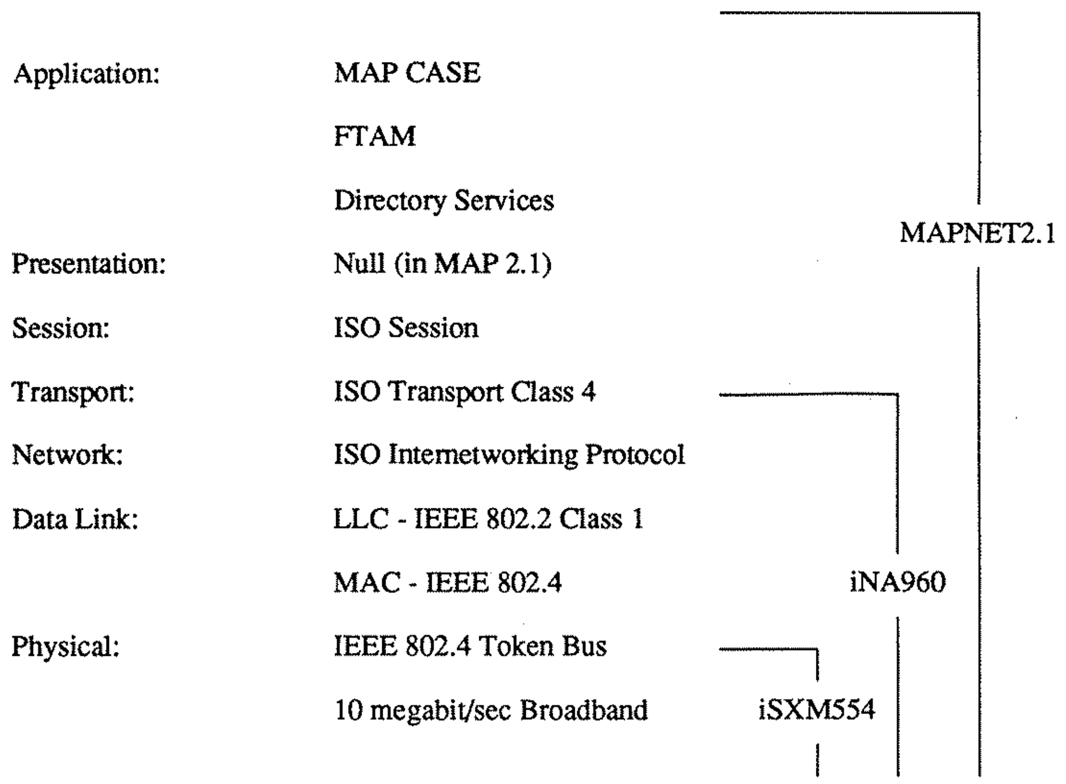
3.2. Intel 286/310 Computer System

Each of the Intel 286/310 computer systems was used as a station or node on the Local Area Network. Each consisted of a chassis with a Multibus backplane, a power supply, a floppy disk drive and a Winchester hard disk drive. An iSBC 286/10 single board computer (host board) was placed into the backplane to complete the computer system. Included on this board was:

1) 80286 CPU -- 6.0 MHz
2) 80287 Math Co-processor -- 6.0 MHz
3) Two RS-232 serial I/O ports
4) One Centronix parallel I/O port
5) 1 megabyte RAM addressable by Multibus.

The software functions performed on this board consist of:

1) iRMX86 real-time multi-tasking operating system [INTE85]
2) Multibus Interprocessor Protocol (MIP) driver
   (enables communications with COMMengine boards)
3) Application Performance Programs



Application:	MAP CASE
	FTAM
	Directory Services
Presentation:	Null (in MAP 2.1)
Session:	ISO Session
Transport:	ISO Transport Class 4
Network:	ISO Internetworking Protocol
Data Link:	LLC - IEEE 802.2 Class 1
	MAC - IEEE 802.4
Physical:	IEEE 802.4 Token Bus
	10 megabit/sec Broadband

Figure 3.1 — Intel's OpenNET Products

There was also a memory board with 896 kilobytes of RAM on the backplane. This memory belonged to the Multibus, and could be addressed by any intelligent device attached to the Multibus at the backplane, including the host board. Additional Multibus boards could be placed in the chassis. These would provide specific hardware functions such as communications.

3.3. The Synchronous Clock System

All delays recorded for experiments were measured using a global synchronous clock [FRAN86]. This clock was external and hardwired into each of the Intel 286/310 computers serving as experiment stations, thus assuring absolute synchronization.

The Synchronous Clock System consisted of a specially designed clock board connected via coaxial cable to a Program Interval Timer (PIT) on each of the Intel iSBC 286/10 CPU boards. The clock board generated a pulse with a period of 0.1 millisecond. These pulses were counted by the PIT in a 16 bit word. When this word overflowed, which occurred about every 6.55 seconds, the PIT generated an interrupt. A special interrupt handler caused a 16 bit software clock word to be decremented; thus a synchronous clock value contained in a 32 bit double word was available for delay measurements in each of the connected Intel system.

Hardware modification was necessary to connect the Synchronous Clock System to the Intel systems. This consisted of rearranging PIT jumpers on the iSBC 286/10 CPU board and installing the coax wire into the input of the local PIT.

Software modifications included installing the new interrupt handler at the start of an application performance program using the clock, and removing it when done. Clock readings were made by a simple function call. Since the clock was being decremented, the start time was subtracted from the stop time to yield the elapsed time.

The procedures used within the software for measuring the times were as follows:

install$sync$intr -- installed the Synchronous Clock interrupt handler
remove$sync$intr -- removed the Synchronous Clock interrupt handler
init$sync$clk -- initialized the Synchronous Clock to 0xFFFFFFFF
read$sync$clk -- returned a 32 bit double word clock reading

3.4. MAPNET2.1

MAPNET2.1 is Intel's implementation of the Manufacturing Automation Protocol version 2.1 (MAP 2.1). MAP 2.1's architecture is based on the ISO OSI reference model. This model specifies the standards to be used at each of the seven layers in the architecture. MAPNET2.1 implements each of these standards at each of these layers to provide the MAP networking functions. Furthermore, MAPNET2.1 allows programmatic access to several layers in the architecture, not just the Application Layer.

MAPNET2.1 is the complete seven layer implementation and simply incorporates iNA960's Transport and Networking services with the Session, Presentation, and Application Layers. MAPNET2.1's programmatic interface is similar to iNA960's. Both products are configurable and are loaded onto front-end processors that provide the communications hardware. These boards are called COMMengines; since MAP 2.1 calls for 10 megabit/sec broadband bus using token-based network access, the iSXM554 COMMengine board was used.

In MAPNET2.1, CASE and Session Layer data transfer primatives map directly onto the Transport Layer data transfer primatives. Since iNA960 provides the transport services without the overhead of the higher layers, it was used for the Transport and Data Link experiments. Only the additional upper layers provided with MAPNET2.1 are described here; the Transport, Network and Data Link Layers are described in the iNA960 section. The MAC and Physical Layers are described in the COMMengine Communications Board section.

3.4.1. Application Layer

MAPNET2.1 includes a programmatic interface to the Application Layer. This layer provides the network user with the highest level of abstraction. MAP Common Application Service Elements (CASE) provide association and data transfer primatives. The File Transfer, Access and Management (FTAM) services provide file manipulation at remote stations. The Directory Services provide CASE and FTAM will the name-to-address conversion services necessary for name-based associations.

3.4.1.1. Common Application Service Elements (CASE)

The MAPNET2.1 Common Application Service Elments (CASE) facility provides MAP 2.1 compatible CASE associations between application processes for information transfer, independent of the nature of the application. Remote process connection may be made using only the process name; CASE will access the Directory Services to make the logical name-to-address conversion. Included in the CASE services is a reliable, full duplex data transfer with a graceful close. This graceful close ensures that all data in transit is actually received before the connection is broken.

The CASE protocol exists primarily to establish and terminate associations. These associations are the major enhancement over the Session Layer services. Application titles are used rather than session addresses during the establishment of an association. Directory Services are used to resolve these application titles into session addresses, relieving the CASE user of this responsibility. The data transfer commands do not require the Directory Services so they are mapped directly onto the session data transfer commands.

3.4.1.2. File Transfer and Access Management (FTAM)

The File Transfer and Access Management (FTAM) facility provides functions for the retrieval of file attributes and the creation, reading, writing, and deletion of files at remote stations. FTAM operates over the Session Layer using Directory Services to make the logical name-to-address conversions.

Any implementation of FTAM contains an FTAM consumer at the local node and an FTAM server at the remote node. The FTAM server responds to the requests for services on its local files. The user requests an FTAM consumer to establish a connection with a remote FTAM server. This request is honored only if the session entities can establish a connection. The two FTAM entities carry on a dialogue consisting of FTAM requests and responses. Finally the FTAM and Session connections are terminated.

The FTAM commands are as follows:

FTAM Connect:	Establishes an FTAM/Session connection with a remote FTAM server.
FTAM Release:	Gracefully terminates the FTAM/Session connection with the remote server.
FTAM Abort:	Abruptly terminates the FTAM/Session connection with the remote server.
FTAM Create:	Creates a new file at the remote server's "virtual filestore".
FTAM Delete:	Deletes an existing file at the remote server's "virtual filestore".
FTAM Select:	Selects an existing file at the remote server's "virtual filestore".
FTAM Deselect:	Detaches a previously selected or created file at the remote server's "virtual filestore".
FTAM Get Attributes:	Reads and returns to the user the MAPNET2.1 file attributes of a previously selected or created file at the remote server's "virtual filestore".

FTAM Open:	Opens a previously selected or created file at the remote server's "virtual filestore" for the purpose of reading from or writing to the file.
FTAM Close:	Closes a previously opened file at the remote server's "virtual filestore".
FTAM Read:	Reads and returns file data to the user from a previously opened file at the remote server's "virtual filestore".
FTAM Write:	Writes file data supplied by the user to a previously opened file at the remote server's "virtual filestore".
FTAM Write End:	Indicates that the user has finished the file write.
FTAM Cancel:	Abruptly ends a file read or write.

3.4.1.3. Directory Services

The MAPNET2.1 Directory Services are designed to support MAP 2.1 Directory Services, which defines a global directory called the Directory Information Base (DIB). The DIB is accessed via the Directory Service Agent (DSA). The Client Service Agent (CSA) provides an interface between the user and the DSA. The CSA also maintains a Local DIB.

The Directory Services provides a mechanism for storing, retrieving, and maintaining information about *network objects*. These objects may be of different types, but each object has a unique name and one or more properties.

Network objects are described by *object names* and *object types*. The object name is used to distinguish separate objects. The object type is an optional field that acts as an adjective to describe the object in more detail. For MAP networks, the object types are used to specify the communication service used by that object, either FTAM or DT (Data Transfer). Objects have one or more *properties* associated with them. These properties consist of a property type and one or more property values. A property type is a general attribute, such as ADDRESS. The property value is the actual information.

The DIB is the collection of all the visible network objects. For the typical MAP network there is one centralized directory. The Local DIB must contain the name and address of the DSA in order to use its centralized services. When the DIB is accessed the information is automatically entered into the Local DIB like a cache.

The commands used to access the directory are: *Add New Name*, *Delete Name*, and *Return Property*. MAPNET2.1 provides some Human Interface commands built on top of these. They are used to add a name (Addname), add an object (Addobj), delete a name (Delname), return an address (Rtnaddr), and return a property (Rtnprop).

3.4.2. Session Layer

The MAPNET2.1 Session Layer is an implementation of the ISO 8327 Session standard. It is built on the Transport Class 4 service, and provides reliable, full duplex communications with a graceful close sequence. One transport connection is used per active session connection. First a transport connection is created with various connection parameters negotiated. Then a session connection is established if the transport connection satisfies the needs of the session user. Now data can be transferred using mostly transport functionality with little Session Layer processing. At the end of the session, the session connection is terminated prior to the termination of the transport connection. These transport connections are not reusable for multiple session connection.

The session connection establishment and data transfer commands are very similar to the analogous transport commands. The additional functionality provided by session appears in the connection termination. There are three ways to terminate connections: *orderly release*; *connection abort*; and *connection termination notification*. With orderly releases, both sides agree to the release and the data being processed is not lost. This graceful close is accomplished by request/accept handshaking. The abrupt terminate closes the connection without regard to data loss. Connection termination notification tells the session user when a close has happened or is being requested.

3.5. iNA960

Intel's iNA960 is a general purpose local area network software package. It implements the ISO OSI reference model for the Transport, Network and Data Link Layers. Also, a Network Management Facility (NMF) is implemented to monitor and adjust the network's operation in order to maintain the network and optimize its performance.

3.5.1. Transport Layer

The Transport Layer provides the message delivery service. It relieves the transport user of any concerns with the detailed way in which reliable, cost effective transfer of data is achieved. iNA960 provides two methods of data transfer at this layer: the Virtual Circuit Service and the Datagram Service. Virtual Circuits are logical "hard wire" connections, with the characteristics of high reliability and in-order delivery. Datagrams are more like postal letters, arriving with no ordering requirement. They should therefore be less expensive to use.

3.5.2. Network Layer

The Network Layer provides routing between subnetworks to provide the network user with a view of the whole network rather than its many subnetworks.

3.5.3. Data Link Layer

The Data Link is divided into two parts, the Logical Link Control (LLC) and the Media Access Control (MAC). At the LLC layer the IEEE 802.2 standard is used, which is the common LLC for any of the IEEE 802 series network protocols. At the MAC layer the IEEE 802.4 Token Passing Bus Access is provided.

Access to the Data Link Layer is provided through an interface called External Data Link (EDL) interface. Data Link datagrams may be sent and received using these EDL commands.

3.5.4. Network Management Facility

The Network Management Facility (NMF) supplies a network with planning, operation and maintenance facilities. A network manager can make adjustments according to the type and volume of activity loading the network. It is a distributed function that is built into every layer on every system, and whose activities are being performed constantly to ensure proper operation of the network. There does not exist a centralized network control station. NMF can change some configuration parameters at most layers, for example, the retransmission time. The Network Layer has no parameters that can be set using NMF.

3.6. iNA961: iNA960 Preconfigured Files

The preconfigured package of iNA960 is called iNA961. It comes in various versions representing the various hardware and software most likely to be used. Since these are standard

products, these are the files that were loaded onto the COMMengine for running the Transport and Data Link experiments.

The COMMengine used for Token Bus/MAP 2.1 network was the iSXM554. The iNA961 preconfigured file used was iNA961.24, which included the following configurations:

NMF	- Local, Remote, and Subnet NMF functions.
Transport	- Virtual Circuit (100 VCs), Expedited and Datagram Services (30 Datagram TSAPs).
Network	- Internet Protocol with MAP 2.1 addressing.
Subnet	- iSXM554 subnet. External Data Link interface with 10 LSAPs.

3.7. iNA960 Programmatic Interface

The user of the communications services provided by the properly installed communications board and iNA960 software accesses these services through procedural interfaces. These interfaces, discussed in more detail in the MIP Overview section, allow the user to access the various iNA960 layers directly or indirectly. Memory segments, called request blocks, are allocated from the user's memory pool and passed by reference to iNA960 through the appropriate procedure.

A request block consists of several fields, some fixed format and some variable format. Those in fixed format remain constant throughout the use of the services of iNA960, such as data relevant to the user and the target layer. An example would be the mailbox for which the user will block, waiting until the request block has been serviced. The variable format fields allow flexibility in accessing different layers and different kinds of services within these layers.

The user is responsible for the proper allocation and formatting of the request block. After the request block has been properly specified, the user calls a special procedure that delivers the request block to the iNA960 software for execution. Upon completion, the request block is returned to the user at the place he has specified to receive it (the mailbox).

3.8. Communication between the COMMengine and the Host

A COMMengine is a communications board which has a separate processor and operating system to run the iNA960 software. Since this is a separate processor, running at a different speed and with a different operating system than the host processor, there are therefore two environments that must be connected to provided communications services to the user applications (see Figure 3.2). The host environment consists of the host processor and its operating system, the memory accessible to it, and a message delivery mechanism (MDM). The iNA960 environment includes the COMMengine board and associated data link hardware to interface iNA960 with the underlying network, the iNA960 communications software running on a processor other than the host's, and an MDM as well. MDM bridges the two environments, providing the means by which request blocks are transferred between the host and iNA960. Since both environments are connected on the backplane by the Multibus system bus, the MDM used by both environments is the Multibus Interprocessor Protocol (MIP). The implementation of this on the COMMengine is built into the iNA960 software. In the host environment it is the first level user job called the MIP device driver.

3.8.1. Message Delivery Mechanism (MDM)

The iNA960 CPU must have direct access to all locations in host memory where request blocks can be allocated. The Message Delivery Mechanisms (MDMs) of the host and iNA960 may simply pass the pointers of the request blocks to the iNA960 CPU. This may be too simplistic, however. In another scheme the MDMs copy the request block into the memory space directly accessible by iNA960. iNA960 will operate on this copy and then return it to the MDM to be returned to the host. This copy then overwrites the original.

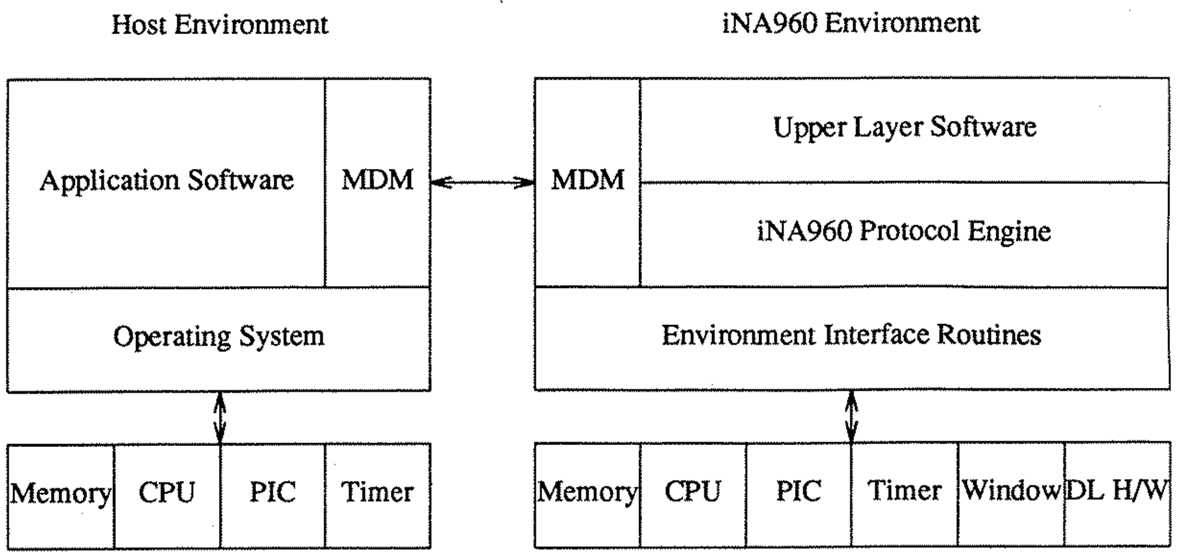


Figure 3.2 — Detailed View of the Communications World

3.8.2. Data Link Hardware

The data link hardware on the COMMengine board for the Token Bus/MAP 2.1 network is the iSXM554 Token Bus Controller. This hardware includes the data link controller, the means by which the data link controller is initialized or reset, the means by which it is signaled, the means by which the host identifier is read, and the initialization of any other data link hardware which may be present. The data link controller is the central feature in this environment and is considered by iNA960 as a black box.

3.8.3. Protocol Engine

The configurations of the iNA960 software can be considered as the Protocol Engine, which consists of one or more of the following:

- Data Link Layer (subnets)
- Network Layer
- Transport Layer
- Network Management Facility
- Optional upper-level software
  (such as MAPNET, RMXNET, and Xenix)

Each layer has configurable options, and some subset of them is linked into the loadable iNA960 software.

3.8.4. Environment Interface Routine (EIR)

The Environment Interface Routines (EIR) are a set of software routines that allow a protocol engine to operate in the environment provided. These routines interface the Protocol Engine to the environment, and they include the routines to handle the MDM, windowing, interrupts, timers, and other data link hardware drivers.

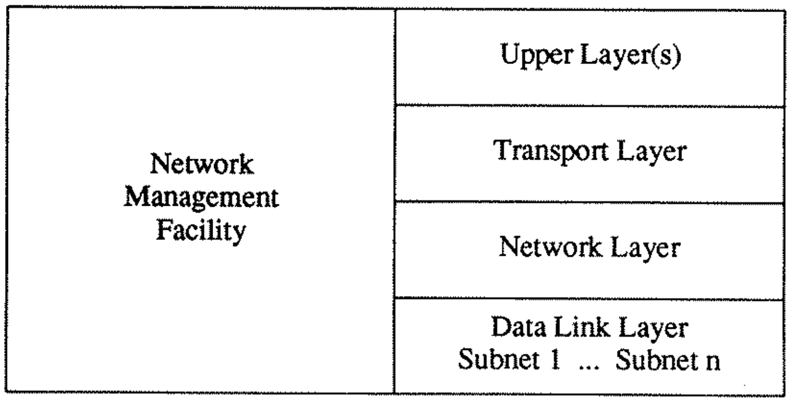


Figure 3.3 — Protocol Engine Model

3.8.5. Multibus Interprocessor Protocol (MIP)

There is a specification for a set of mechanisms and protocols that enable reliable and efficient exchange of data among tasks executing on various different single board computers connected to a common Multibus system bus. This is called the Multibus Interprocessor Protocol, or MIP. The MIP specification ensures compatibility among MIP implementations, called MIP facilities.

MIP isolates user tasks from the complexities of communicating across the Multibus, relieving them from such issues as:

-- different processors
-- different operating systems
-- different Multibus signaling mechanisms
-- different memory space
-- different addressing to the same shared memory
-- interference in the same shared memory areas.

MIP supports single board computers, called devices. Such a device is the COMMengine, complete with CPU and operating system. MIP handles the interaction between devices attached to the backplane, providing the use of the services of the device at a higher level of abstraction.

These devices may generate tasks within their operating system that desire communications with tasks from other devices. A task is a functional unit; it may be a program, part of a program, or a system of related programs. Each processor on each device must be running a MIP facility that conforms to the MIP specification. The functionality provided can then bridge the gap between processors and allow tasks to interact.

Tasks interact via shared memory, called a buffer. This memory is called Multibus memory because it is simply a memory board attached to the backplane. A port is a logical delivery mechanism that enables delivery in FIFO order. It is represented as a queue. Another name for a port is a mailbox. A socket is an ordered pair of (device, port). At higher levels of abstraction, ports are referenced by a function name.

There are three levels of interface within a MIP facility. The virtual level allows the user to interact with the MIP facility itself. The physical level allows the MIP facilities on different boards to interact with each other. The logical level translates between the virtual and the physical levels. MIP facilities must be compatible at the physical level only.

3.8.5.1. Physical Level

The physical level provides a communications mechanism between devices. This mechanism is called a request queue. It is a fixed-size, unidirectional, first-in-first-out queue. The ends of the queue are attached to ports. Two way communication can be implemented by placing two request queues between two devices. This pair is called a channel.

3.8.5.2. Logical Level

The logical level uses request queues to transfer requests between source and destination MIP facilities. A request is either a command or a response. Commands are orders sent from the source to the destination, and a response is the return indication of the result of the attempt to deliver the command.

3.8.5.3. Virtual Level

There are five procedures which use logical level entities to provide functionality to the user interface. On the sending side, FIND locates a port given a function name, and TRANSFER initiates a transfer of a buffer to a port channel by sending a command and waiting on the response. On the receive side, ACTIVATE attaches a task to a port and enables reception of messages at that port. RECEIVE allows the completion of the transfer of a buffer by accepting the command sent by TRANSFER and responding to it. DEACTIVATE disconnects a task from its port and terminates reception of commands at that port.

3.8.6. iRMX86 MIP Driver Implementation

The iRMX86 MIP driver implementation is based on the passing of request blocks between the two processor environments to facilitate interactions. A request block is a formatted segment of memory with some fixed fields for identification and some variable fields for the arguments to the services desired. Services are requested and completion responses are returned via formatted request blocks.

There are three types of procedures within the iRMX86 MIP driver. These are the device independent procedures, the conversion procedures, and the device dependent procedures.

The device independent procedures are responsible for the initialization of local data structures, loading and reporting the status of the communications software to the front-end processor, creating and deleting users, and transferring formatted requests to the front-end processor. Some of these routines are used in Human Interface commands like "Load" and "Status". Others form the procedural interface for applications software, like the *CqCommRb* and *CqCreateCommUser* routines. Others are just used internally.

The conversion procedures allow this MIP driver implementation to be operating system independent. By converting an iRMX86 address reference within a request block into an absolute address reference, buffers used by the user application can be referenced by the different processor with its different addressing scheme. When the request block is returned, if it carries the absolute address of some filled buffers, these addresses must be reconverted. These procedures are available to the applications user.

The device dependent procedures are not available to users. They are used internally, by each other and by the user interface routines, to provide the desired interactions. These procedures directly manipulate the hardware to perform specific functions. Such procedures include interrupt generation, MIP send and receive procedures, and queuing procedures.

3.8.7. Configuration Process

There are two steps needed for the MIP driver configuration in iRMX. The first is the MIP job generation (device driver) and the second is the generation of the iRMX Operating System to include the MIP job.

A file called MIPCFG.A86 is edited to reflect parameter setting specific to the system for which the MIP job is being generated. Such parameters include:

- the COMMengine board name
- the wake up port*
- MIP interface address*
- interrupt level*
- number and address of MIP input and output queues
- number of mailboxes that the MIP driver can support.

*jumpered on the COMMengine

A file called MIPXXX.CSD, where XXX is the descriptor of the communications board, is submitted. This links and locates the MIP job.

Next, the iRMX86 operating system is generated to include the MIP driver interface as a first level user job. This informs iRMX86 to service this device driver when communications with the device are taking place. Several things must be done to the old operating system definition in order to include this MIP driver. First, the part of memory where the job has been located is removed from the contiguous free heap. Next, the root job must be informed of the MIP driver job. This is done by declaring the user job and its parameters using the user jobs screen. Finally, the full pathname of the module that is to be the user job is specified. Now the definition file can be used to generate the new operating system, which includes the MIP driver.

Once the computer system is booted using this new operating system, it is ready to service requests on the COMMengine through the MIP device driver interface. If everything has been configured, linked, and located properly, and the COMMengine device is installed, interaction between the application software (host task) and iNA960 (the device task) can occur with little or no knowledge of the type or design of the device. This interaction is further discussed in the section on iNA960.

3.9. iSXM554 COMMengine Multibus Communications Board

The interaction between user application software and communications software depends upon the hardware environment selected. A communications board that has the capacity to provide the user with a front-end communications processor is called a COMMengine. These communications boards each have a CPU, an operating system, and on-board memory, and can run communications software in parallel with the host board. This COMMengine configuration allows the host board to offload the communications concerns to the communications board (see Figure 3.4).

By placing a communications board into the Multibus backplane, the system can be used as a node on a local area network. The communications board and the host board interact across the Multibus backplane using the Multibus Interprocessor Protocol (MIP) facility. By using the functionality provided by MIP, the user can access the network services and yet be separated from the network by a layer of abstraction.

The iSXM554 COMMengine used consisted of the following components: an iAPX 186 Microprocessor running at 8 MHz; an RF Broadband 10Mbps TBM-15 Token Bus Modem; and

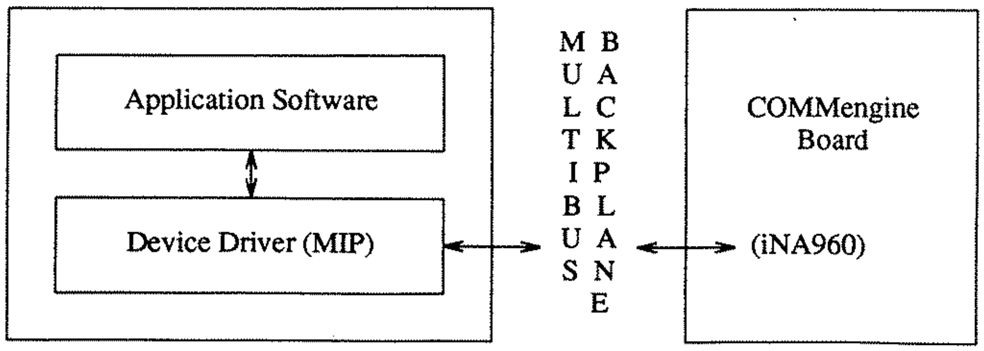


Figure 3.4 — HOST with COMMengine attached to Multibus

the iNA960/961 software.

3.10. The Network

Two Intel 286/310s were used as nodes on the network. Each node was connected to the bus by a coaxial cable from the COMMengine modem to a 4-way multitap as shown in Figure 3.5. The bus was connected to an INI headend remodulator [INDU85a,b].

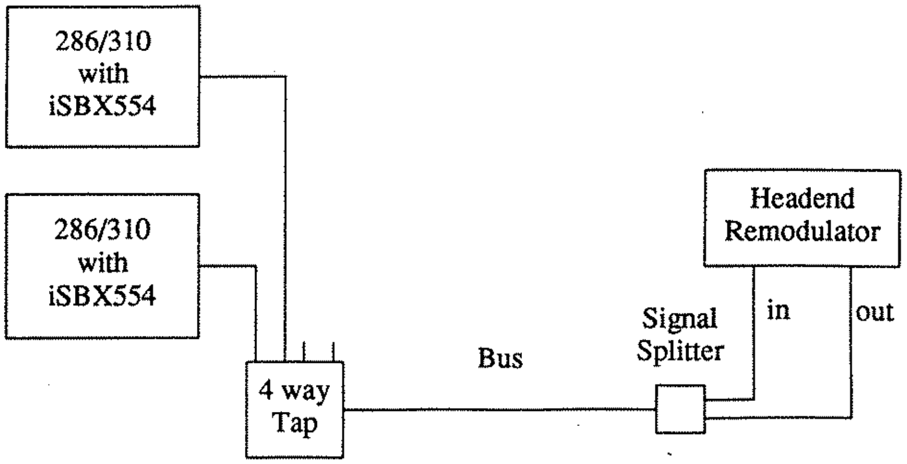


Figure 3.5 — The Token Bus Network

Chapter 4

EXPERIMENTS

4.1. Introduction

Since MAP is designed to operate on the factory floor, we developed a set of tests which was intended to measure its performance characteristics. In MAPNET2.1, CASE and Session Layer data transfer primatives map directly onto Transport Layer primatives. The Transport virtual circuit service is built upon the datagram service of the Data Link Layer. For this reason our measurements were targeted toward the Transport and Data Link Layers.

All applications programs which access the communications services of iNA960 or MAPNET2.1 include several common components. These components are reviewed below. A detailed description of the program written to access the External Data Link services and a brief description of the program used to access the Transport Layer services are discussed. Finally, the results of the experiments are presented.

4.2. Components of Applications

The procedures and libraries which provide the programmatic interfaces must be included in any program to use the communications services of iNA960. These procedures create a communications user for the communication session, translate addresses between the Host and the COMMengine, and submit request blocks which deliver requests for service to iNA960.

There are several steps involved in using the communications services. A communications user and its associated resources must be allocated. Request blocks must be formatted and submitted. These request blocks provide a vehicle for requests to iNA960. Timing routines are included at several places in the experiment programs but are not necessary for the proper use of the communications services of iNA960. When the communications session is terminated, the resources must be deleted. The following describes each of these steps in more detail.

4.2.1. Allocating Resources

A call to *CqCreateCommUser* must be made to create a user_id for identifying resources and requests for the whole communications session. This is only done once. The user_id remains active and is placed into the fixed format section of every request block during the communications session.

Since the iRMX86 mailbox is the structure used as the point of access for the services of iNA960, a mailbox dedicated to this purpose must be created. The system call used to allocate a mailbox is *RqCreateMailbox*. One or more mailboxes can be used throughout the communications session, however, only one is necessary when the services requested follow a sequential order. A mailbox is used because its structure is a queue. Since processes must communicate asynchronously across the Multibus using MIP, requests are enqueued at the mailbox while waiting for iNA960 to accept them, and they are again enqueued there upon return from iNA960. A user is blocked waiting on this mailbox until the request block is returned from iNA960 through the mailbox.

Request blocks are segments of memory, therefore a block of memory must be allocated. A program should use *RqCreateSegment* to get a block of memory on a segment boundary (required by iNA960 because of the address translations). This block then becomes the request block, and is filled and submitted.

4.2.2. Formatting the Request Block

A request block has three parts, a set of identification fields, a set of system fields and a set of arguments. Formatting a request block requires three steps: (1) the process identification information is inserted into the identification fields; (2) the subsystem of iNA960 and the operation to be performed within that subsystem are placed into the system fields; and (3) the arguments are supplied. The identification field structure is called a header:

1. Length	-- total length of the request block
2. User_id	-- from CqCreateCommUser
3. Response_port	-- always 0xFF
4. Return_mailbox	-- from RqCreateMailbox
5. Segment_token	-- from RqCreateSegment

The Return_mailbox and the Segment_token fields are used by the inter-task communications facilities of iRMX86 to return processed request blocks to the user application program that originated them. The Return_mailbox is the mailbox that was allocated, and the Segment_token is the segment part of the address of the memory block that was allocated for the request block.

The system fields are the subsystem field and the operation field. The subsystem is an identification number for use by the External Interface Routine (EIR) of iNA960 to direct the request block to the proper layer within iNA960. This is how iNA960 is able to provide services at its several layers. The operation field is another identification number for the type of service requested, for instance, Connect, Transmit, or Post Receive Buffers.

There are also operation-dependent arguments which are of variable length. An example of variable field arguments is the several fields that hold the number of transmit buffers and their addresses. iNA960 uses these arguments when it copies the buffers to the COMMengine for processing and transmission.

4.2.3. Submitting a Request Block

By making the *CqCommRb* call with the segment address of the request block as the argument, the request will travel through the Message Delivery Mechanism (MDM) to iNA960 for servicing. *CqCommRb* is actually a MIP procedure; it is on the virtual level of the MIP facility (see MIP, section 3.8.5) and provides the service of the MDM in an encapsulated form. *CqCommRb* enqueues the request on the mailbox provided, where it is delivered to the communications board to be serviced by iNA960.

When the request block is returned by the MDM, it is placed on the queue at the same mailbox. It waits there until the procedure *RqReceiveMessage* is issued which returns the address of the request block. This address is the address of the original request block. This returned request block may have some return information left in some of the fields, such as whether the request was successfully serviced. If *RqReceiveMessage* is called before the request block has been processed, the application program (or task) blocks waiting for that mailbox to receive the message in the form of the returned request block.

4.2.4. Timing Routine Calls

Timing routines [FRAN86] are placed at various points in the application programs to report the amount of time specific operations take. For example, a call to the routine to read the clock, *read$sync$clk*, could be placed before and after a *CqCommRb* call to measure the amount of time it takes to submit a request block.

4.2.5. Deleting Resources

When the program terminates successfully, the resources it acquired for use should be returned. This includes the segment of memory used for the request block and the mailbox.

The memory is returned using *RqDeleteSegment*, and the mailbox is returned using *RqDeleteMailbox*.

4.3. EDLP — External Data Link Performance Program

The External Data Link (EDL) is an interface used to access directly the services of the Data Link Layer, allowing users to circumvent the Transport and Network Layers. A user of the EDL uses the general request block programmatic interface common to all layers and commands, and described in the iNA960 section (3.7) above. Through request blocks, commands can be issued to establish and terminate Data Link connections, transmit data, post receive buffers for incoming data, configure the Data Link Controller, reassign an individual address for a node, and add and remove multicast addresses to and from a list maintained by the data link node.

Data Link users communicate via Link Service Access Points (LSAPs). This LSAP is a code that identifies a specific user process or another layer on the same node. Each receiving process is identified by a Destination LSAP (DLSAP), and each sending process is identified by a Source LSAP (SLSAP). Packets must contain the DLSAP of the receiving process in order to be received into the buffers posted at that point. The EDL Connect command adds a processes LSAP to the active list, and the EDL Disconnect command removes it.

The following is the flow of control of the *EDLP* program used for Data Link performance measurements using the EDL interface.

EDLP Psuedo-Code

```
{
        install clock interrupt handler
        create resources {
                CqCreateCommUser
                RqCreateMailbox
        }
        input: (T)ransmit; (R)eceive; (Q)uit
        if 'Q' {
                remove interrupt handler
                exit
        }
        read$sync$clk(start_experiment)
        submit EDL Connect for a specific LSAP {
                RqCreateSegment
                format Connect Request Block
                CqCommRb
                RqReceiveMessage
                RqCreateSegment
        }
        if 'R' {
                RqCreateSegment
                format Post Receive Buffer Request Block for that LSAP
                read$sync$clk(start)
                CqCommRb
                read$sync$clk(stop)
                receive_CqCommRb_delay = stop - start
                RqReceiveMessage
                read$sync$clk(received_timestamp)
                get sent_timestamp from packet
                one-way packet delay = sent_timestamp - received_timestamp
                total_time = first sent_timestamp - last received_timestamp
                RqDeleteSegment
        }
        else {
                RqCreateSegment
                format packet with receive LSAP (DLSAP)
                format Transmit Request Block
                read$sync$clk(start)
                sent_timestamp = start
                put sent_timestamp into packet
                CqCommRb
                read$sync$clk(stop)
                transmit_CqCommRb_delay = stop - start
                read$sync$clk(start)
                RqReceiveMessage
                read$sync$clk(stop)
                time_to_transmit = stop - start
```

```
            RqDeleteSegment
        }
        submit EDL Disconnect for a specific LSAP {
            RqCreateSegment
            format Connect Request Block
            CqCommRb
            RqReceiveMessage
            RqDeleteSegment
        }
        RqDeleteMailbox
        read$sync$clk(stop_experiment)
        length_experiment = stop_experiment - start_experiment
        print report to screen
    }
```

4.4. PerformTB - Transport Token Bus Performance Program

The applications program *PerformTB* was used for the Transport Layer experiments. This program accessed the Transport Layer directly via the programmatic interface in much the same way as the program *EDLP*. The flow of control of *PerformTB* is much like *EDLP* and the delays were measured in a similar fashion.

Before starting the transfer of data, the user of *PerformTB* is allowed to provide several experiment set-up parameters. Data may be transferred in one direction or both. A user-defined delay may be inserted between successive messages. Message (TSDU) size is actually the size of the buffer holding the message. The number of messages sent is determined by the total amount of data to be transferred. The number of user buffers determines the size of the queue for sending messages. The number of virtual circuits is the number of separate connections that will be used to transfer the total amount of data. There is also a protocol option for the number of bits used for the sequence numbers and whether checksumming is turned on. For all results reported here, checksumming was turned off.

4.5. Performance Measures

The External Data Link (EDL) is an interface used to access the services of the Data Link Layer directly, allowing users to circumvent the Transport and Network Layers. A user of the EDL accesses iNA960 through the general request block programmatic interface common to all layers in MAPNET2.1 and iNA960. Through request blocks, commands can be issued to establish and terminate Data Link connections, transmit data, post receive buffers for incoming data, configure the Data Link Controller, reassign an individual address for a node, and add and remove multicast addresses to and from a list maintained by the data link node. We measured several aspects of data transfer using this datagram service.

The transport user also accesses iNA960 through this programmatic interface. We used the Transport Class 4 Virtual Circuit Service and measured several aspects of its performance and error recovery mechanisms. We then made comparisons between the datagram service of EDL and the virtual circuit service of Transport, identifying some benefits and drawbacks to using a front-end processor communications board.

4.5.1. Service Accessing Delays

One group of experiments measured two distinct delay times for access to the EDL: the time required for submitting a request block (*CqCommRb* call), and the time required for a mailbox response (*RqReceiveMessage* call).

The delay in submitting a request block is the bound on the least amount of time necessary for using the EDL services. The measure of the *CqCommRb* delay represents the time it takes for a request for service to be placed on a mailbox, the COMMengine to be notified of the request, and for the request to be received and acknowledged by the iNA960 software running on the COMMengine. Packet sizes ranged from 16 to 1024 bytes. One hundred receive buffers were posted using this *CqCommRb* call; each was timed and the delays were averaged. These receive buffers were posted prior to the transmission of any packet to ensure that each packet could be received. The delays for submitting a post buffer request are shown in Figure 4.1 with the vertical bars representing the range of values observed. The one hundred packets were transmitted and each call to *CqCommRb* was timed and the delays averaged. The delays for submitting a transmit request are likewise shown in Figure 4.2.

No values for the *CqCommRb* calls were below 4.0 millisecs, which appeared to be the lower bound on this delay. Since this call must be made for every packet to be transferred, this delay represented the lower bound on the delay suffered for any data transfer using the EDL interface. There is no evidence that the delay was dependent on the size of the packet, which suggested that the processing done for a transmit *CqCommRb* call did no buffer allocation or copying at that time. However, the range of values for the transmit *CqCommRb* call, from 4 to

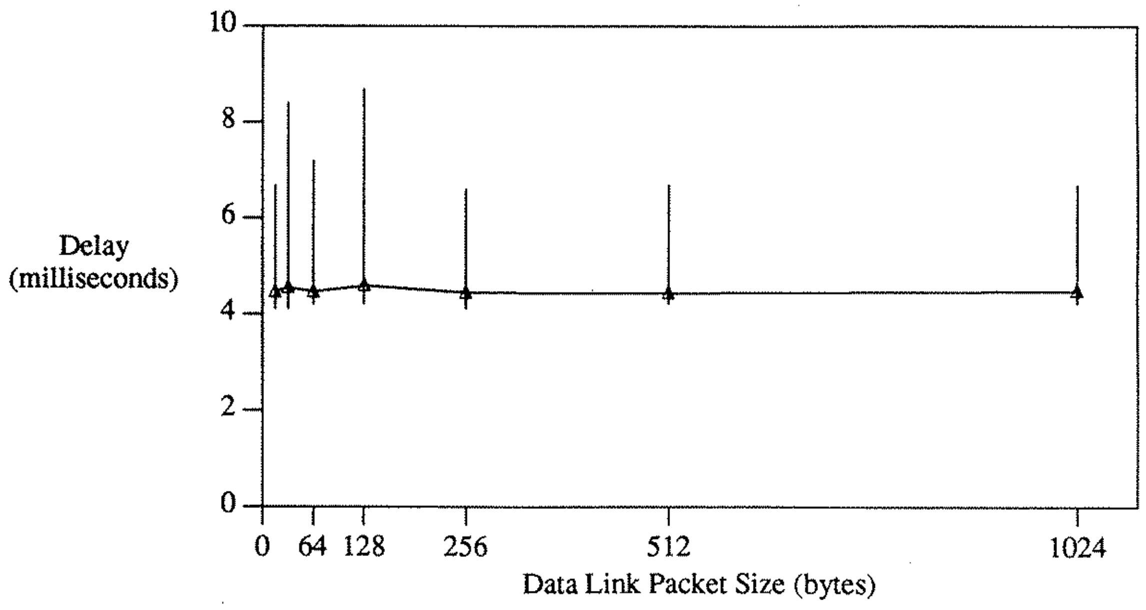


Figure 4.1 — Post Buffer *CqCommRb* Call Delay vs. Data Link Packet Size

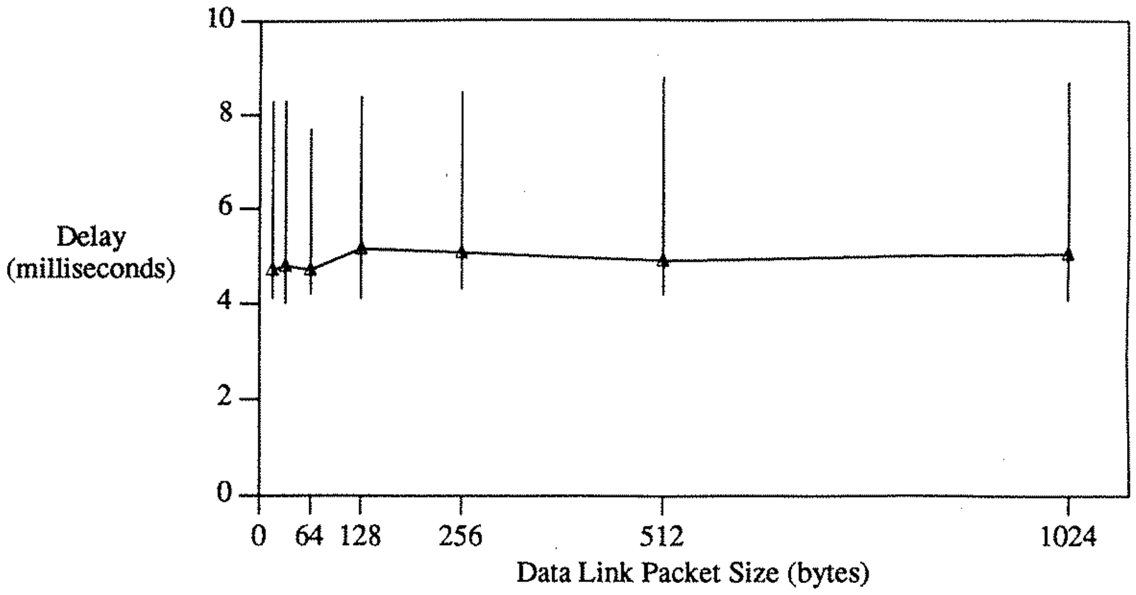


Figure 4.2 — Transmit *CqCommRb* Call Delay vs. Data Link Packet Size

8 millisecs, suggested that this call was a function of the condition of the Multibus or the status of the COMMengine.

After each request for transmit was submitted, the *EDLP* program blocked waiting on a mailbox. This was done by making a *RqReceiveMessage* system call. The delay in waiting for that mailbox to receive the message indicating that the service had been processed was measured for packet sizes ranging from 16 to 1024 bytes. For each of these 100 packets, the delays were averaged and are shown in Figure 4.3.

The delay incurred during a *RqReceiveMessage* call was partially dependent on packet size. Since this call caused the program to block waiting on the processing to be completed, at least one data copy must have occurred. The packet was copied from Multibus memory on the host to the local memory on the COMMengine. The overhead associated with this and other processing is divided into two parts: a constant delay due to submitting a request block and accessing the DMA channel, and a packet size dependent delay for the copy, preparation, and

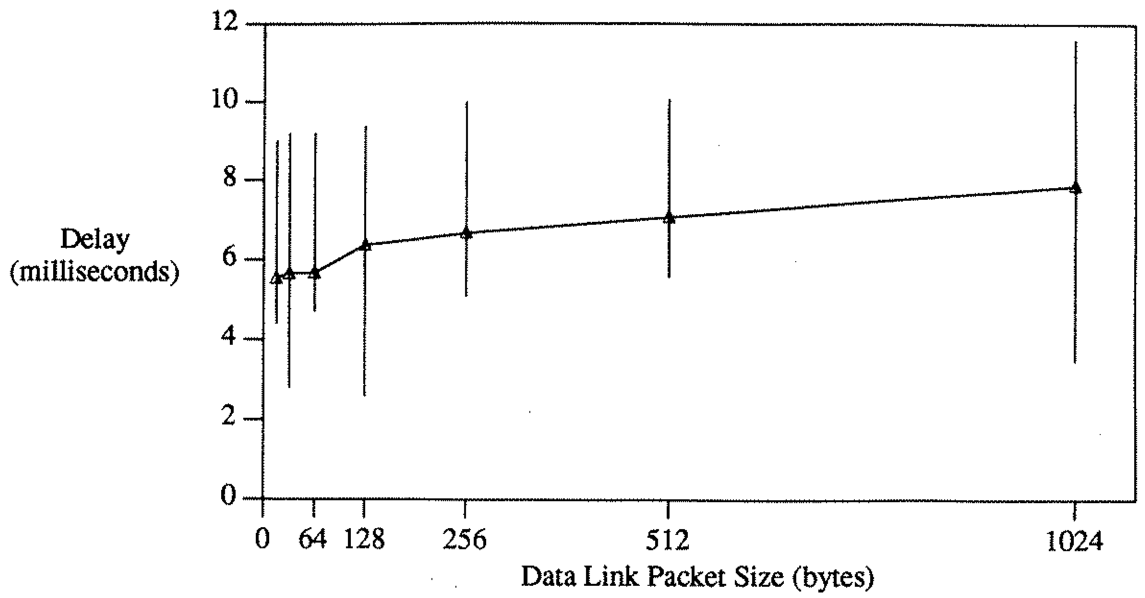


Figure 4.3 — *RqReceiveMessage* Delay vs. Data Link Packet Size

transmission of the packet.

4.5.2. One Way Delay

The total delay incurred for the delivery of one EDL message required timestamping that message as it was transmitted and noting the time as it was received. To eliminate any queuing delays, one buffer was posted and one packet sent for each run of the experiment. This ensured that all resources were ready and waiting to service that one packet as it traversed the network. Packet sizes ranged from 16 to 1024 bytes by powers of 2. Four runs of the experiment were performed for each packet size; these delays were averaged and the results are shown in Figure 4.4. There was a linear relationship between the packet size and the delay suffered during transfer. For packets of length L, the end-to-end delay was approximately (8.5 + L/125) milliseconds.

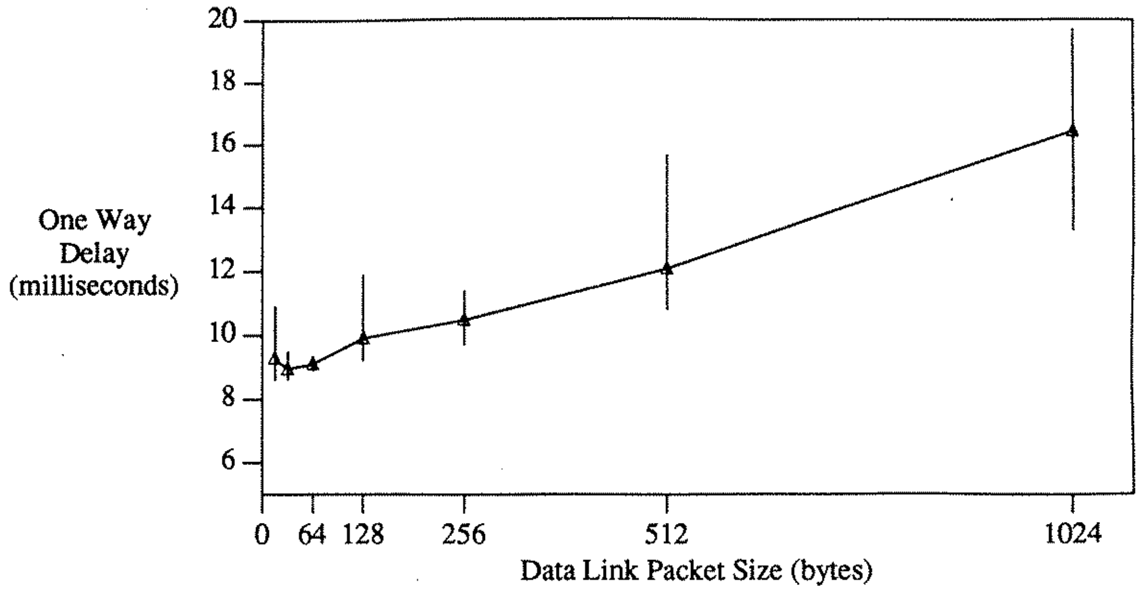


Figure 4.4 — One Way Data Link Delay vs. Data Link Packet Size

The delay which a Transport Service Data Unit (TSDU) message suffered as it traversed the network was also measured. The TSDU message sizes ranged from 16 to 1024 bytes by powers of 2 as in the EDL experiments and then from 928 bytes to 5569 bytes by multiples of 928 and those multiples plus one. The TSDU message size of 928 represents the largest amount of data that could be transferred using a single 1024-byte Transport Protocol Data Unit (TPDU) (the remaining bytes are reserved for upper layer headers). One transmit buffer and one receive buffer were used. A delay of 100 milliseconds between the transmission of each TSDU message ensured that all resources were ready and waiting to process that message, and that there would be no queueing delay.

As Figure 4.5 shows, delays increased linearly with TSDU message size until the TSDU size exceeded 928 bytes. At 929 bytes, a penalty was assessed due to the overhead required to segment the TSDU into two TPDUs. This penalty was repeated each time the TSDU size exceeded a multiple of 928 bytes. The first division of a TSDU into two TPDUs cost about 20

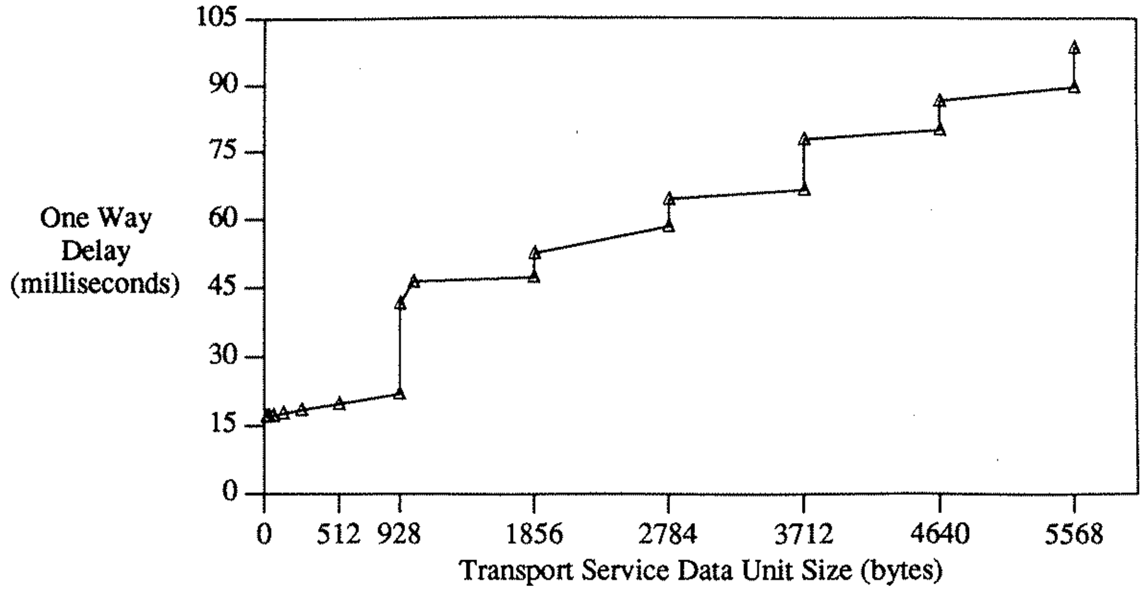


Figure 4.5 — One Way Transport Delay vs. Transport Service Data Unit Sizes

milliseconds; further TSDU segmentation cost 5 to 15 milliseconds per additional TPDU required.

4.5.3. Throughput

The EDL throughput was measured for packet sizes ranging from 16 to 1024 bytes. One hundred buffers were posted on the receive node and 100 packets were sent sequentially from the transmit node. This throughput, therefore, was a measure of network efficiency with respect to packet size, and did not necessarily represent the best possible throughput at the Data Link Layer.

The timestamp of the first packet received was subtracted from the time the last packet was received to obtain a total time to process the 100 packets. The total amount of data sent (100 times the packet size) was divided by this total time to arrive at the throughput, shown in Figure 4.6. There was a very strong dependency between the packet size and the throughput.

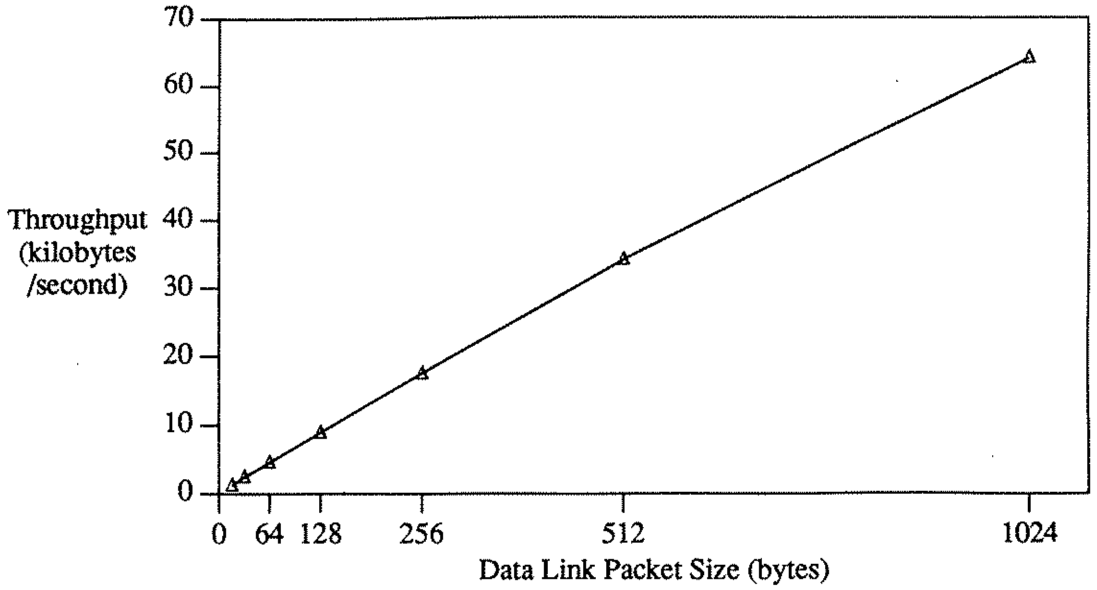


Figure 4.6 — Data Link Throughput vs. Data Link Packet Size

For packets of length L, throughput was approximately 65L bytes/second.

A transport user buffer was a dedicated segment of user memory which held a message as it was prepared for transmission. When the measurement program submitted a transmit request, the buffer holding the packet to be transmitted was not used again until iNA960 released it. The performance program used the multiple transmit user buffers for multiple transmit requests. Once that message had been acknowledged by the receiver, the user buffer was returned to the circulating queue of free buffers and reused.

As with the Data Link experiments, the concept of a user buffer refers to the use of a dedicated amount of memory for storing a message (TSDU) to be transmitted or received. Once that message has been processed, the user buffer may be returned to the pool of buffers and reused. The user buffers available are kept on a circulating queue. The receiver posted all of the user buffers it had and waited for messages to arrive. As they arrived, the message was returned to the user in a user buffer, then the user buffer was posted again. The transmitter submitted all the messages stored in its user buffers, then waited for them to be processed. A user buffer was not returned to the transmitter until all of the data contained in it was acknowledged by the receiver.

By varying the number of user buffers allocated by the transmitter and the receiver, the number of transmit and receive buffers needed to optimize throughput was determined. The number of receive buffers was set to a high number (10) and the number of transmit buffers was increased from 1 to 10. The throughput was measured at each buffer configuration. Seven transmit buffers resulted in the best throughput, as shown in Figure 4.7. Then the number of receive buffers was decreased to one. Again, the throughput was measured at each buffer configuration. The number of receive buffers which resulted in the best throughput was 5, as shown in Figure 4.8. This buffer configuration was used in all experiments measuring throughput against another variable.

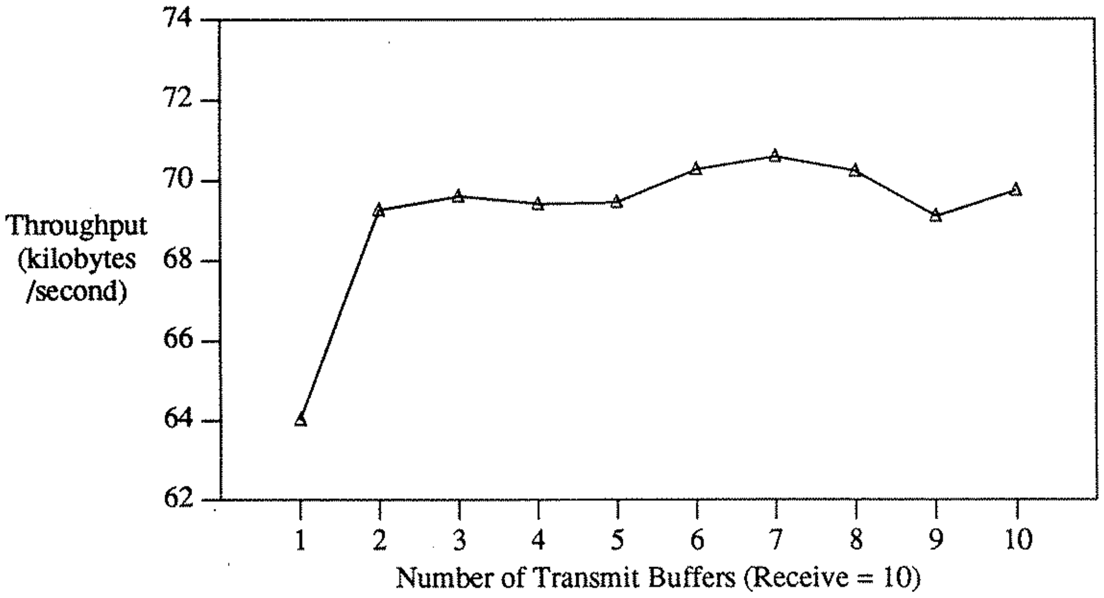


Figure 4.7 — Finding the Optimal Number of Transport Transmit Buffers

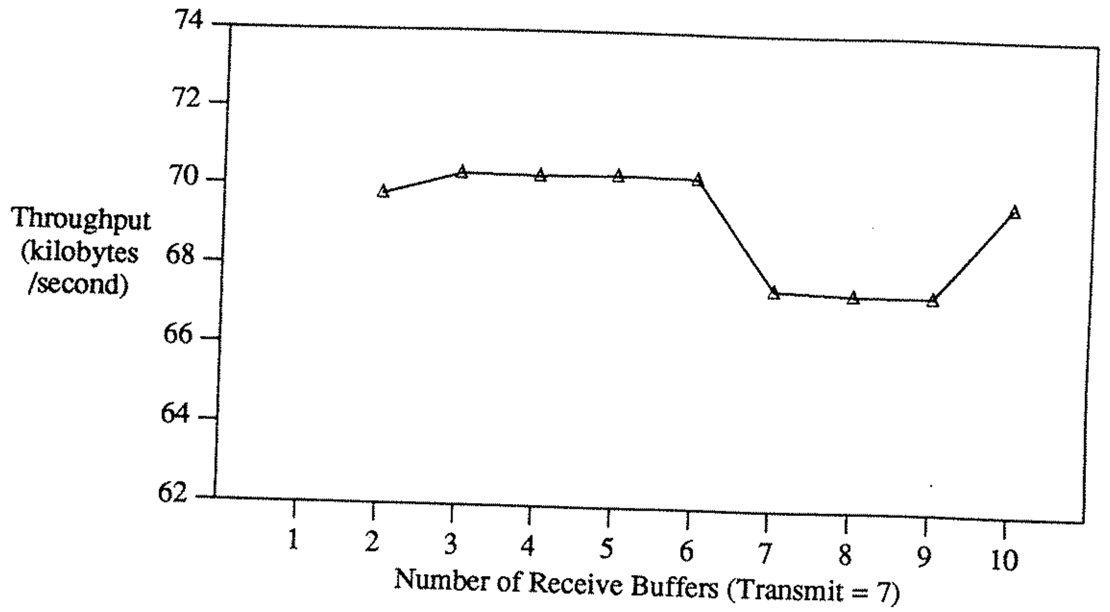


Figure 4.8 — Finding the Optimal Number of Transport Receive Buffers

As seen in Figure 4.9, throughput increased linearly for TSDU message sizes that were equal to or smaller than 928 bytes. At a TSDU size of 929 bytes, throughput suffered due to the overhead associated with segmenting one TSDU into two TPDUs. A decrease in throughput was likewise observed whenever the TSDU size exceeded a multiple of 928 bytes. The curves of the throughputs for each multiple of 928 (the peaks in the graph) and the multiples plus one (the dips in the graph) were quadratic, with the two curves converging asymptotically to approximately 72 and 55 kilobytes/second, respectively. The difference between these two values reflected the penalty for segmentation.

4.5.4. User Buffers

The user of EDL is solely responsible for the maintenance of user buffers. A user buffer was a dedicated segment of user memory which held a packet as it was prepared for transmission. When the *EDLP* submitted a request to transmit, the buffer holding the packet to

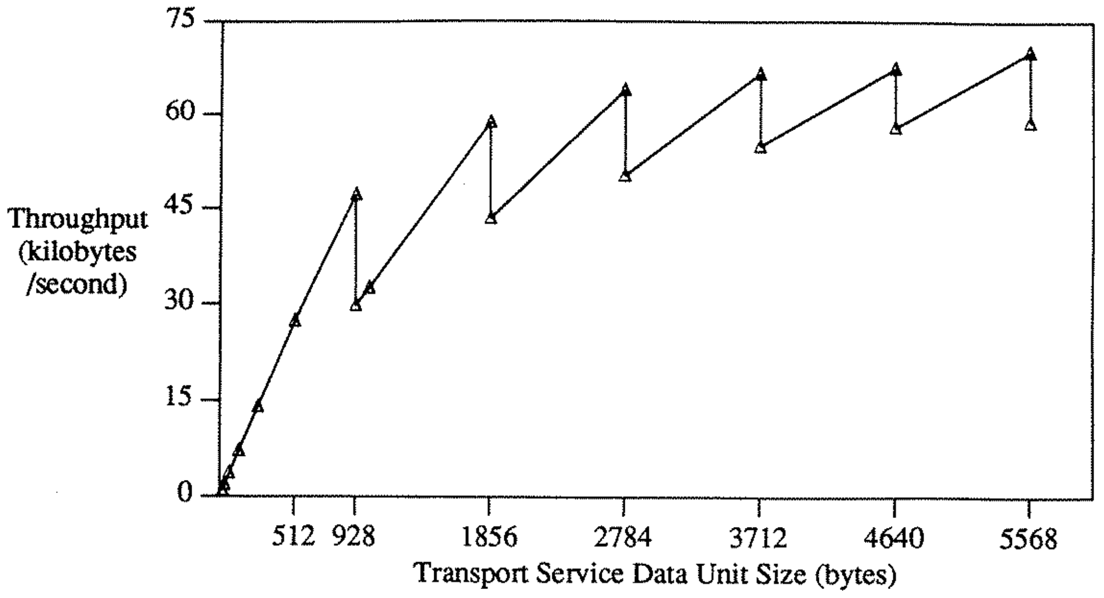


Figure 4.9 — Transport Throughput vs. Transport Service Data Unit Size

be transmitted was not used until the program returned from the *RqReceiveMessage* call. The *EDLP* performance program represented multiple user buffers as multiple consecutive *CqCommRb* calls submitting transmit requests. As *RqReceiveMessage* began to receive the processed requests, the block of memory used for the transmit packet was returned to the pool of free user buffers.

An experiment was performed to observe the effect of varying the number of user buffers available to the transmitter. The receiving node posted 30 buffers before the start of the experiment to ensure that all packets would be received. Then the transmit node had to recycle a varying number of user buffers. As more buffers were used, more consecutive calls *CqCommRb* were made and thus more packets could be enqueued on the mailbox waiting to be sent. The packet size used was 1024 bytes. The results of various numbers of user buffers is shown in Figure 4.10.

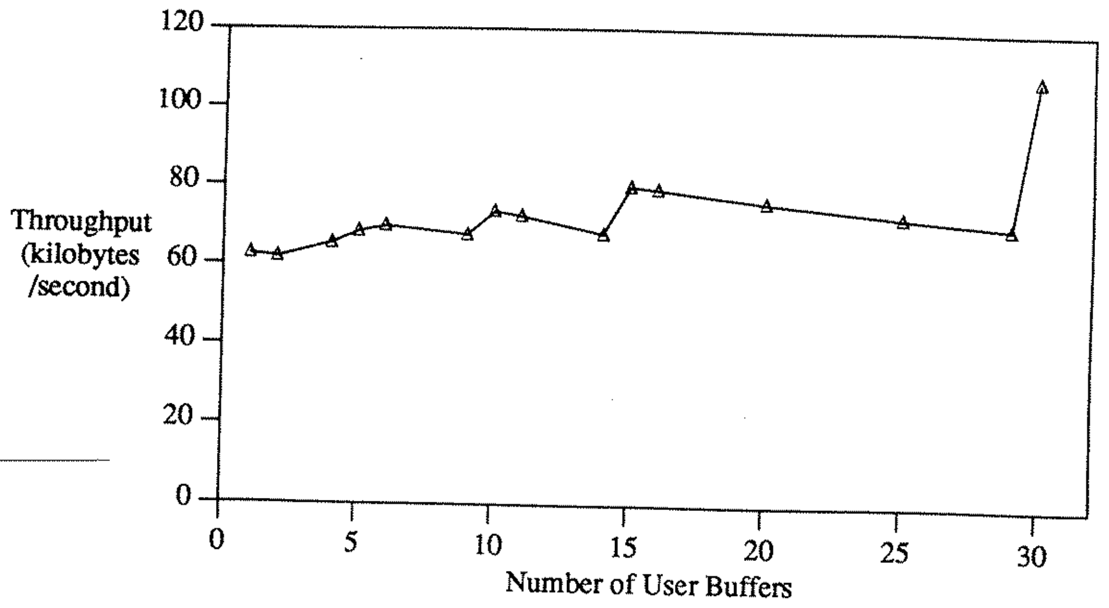


Figure 4.10 — Data Link Throughput vs. Number of User Buffers

By varying the number of user buffers, the peaks of throughput were observed at even divisors of 30 (i.e. 2, 3, 5, 6, 10, and 15). Between these divisors there was a steady decrease in throughput as the number of user buffers increased. This indicated that there was a small throughput penalty imposed where the number of user buffers was not an integer divisor of 30.

Likewise, the user of Transport may vary the number of user buffers employed. To observe the effect of the maintenance of many user buffers, the number of transmit and receive user buffer was increased symmetrically from 1 to 100, and the throughput was measured. Figure 4.11 shows the trade-off between increasing the resources and increasing the overhead to manage these resources. There was a significant increase in throughput with increasing the number of transmit and receive buffers from 1 to 5. However, additional user buffers did not appear to provide faster service. This is consistent with the results of the optimal buffer configuration experiment, which found that 7 transmit buffers and 5 receive buffers provided as good or better service as any other configuration.

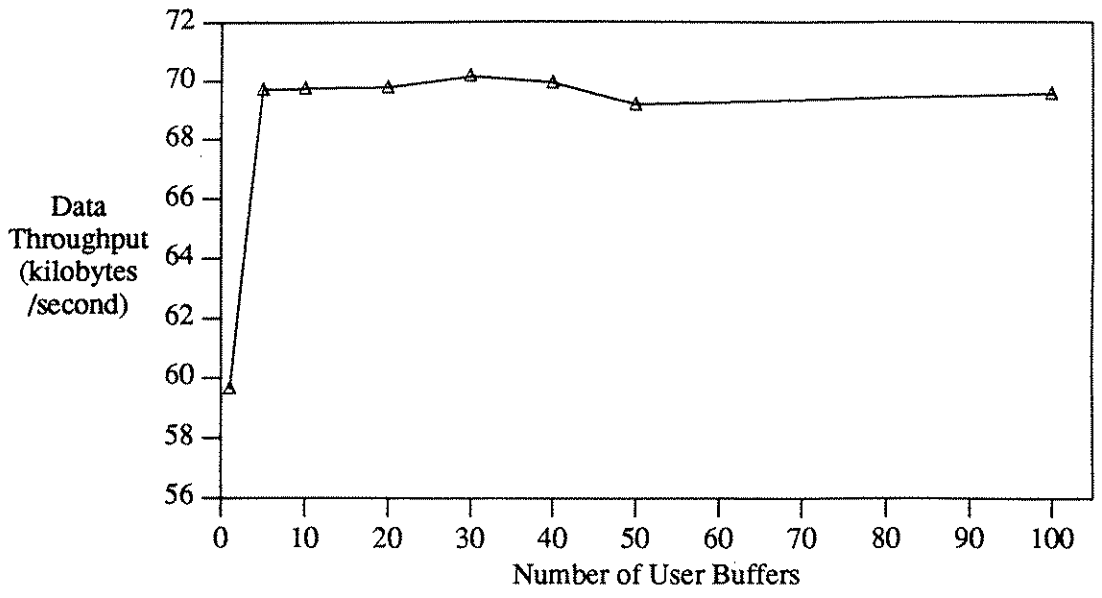


Figure 4.11 — Transport Throughput vs. Number of User Buffers

4.5.5. Number of Messages Per Second

The number of EDL messages per second was measured using 100 transmit and 100 receive EDL user buffers for maximum enqueueing. These EDL user buffers were represented similarly to the transport user buffers. One hundred EDL user buffers were used to enqueue 100 messages so that all of the messages could be transmitted at the peak rate. Figure 4.12 shows the number of messages transmitted per second for various packet sizes. The peak EDL transmission rate observed was 156 128-byte messages per second.

Because the optimal configuration for the transport user buffers was found to be 7 transmit and 5 receive user buffers, the number of messages per second for Transport Layer was a function of the throughput. When throughput was optimized the number of messages per second was optimized as well. Figure 4.13 shows the number of messages transmitted per second for various TSDU sizes. The peak transmission rate observed for Transport was 56 16-byte messages per second.

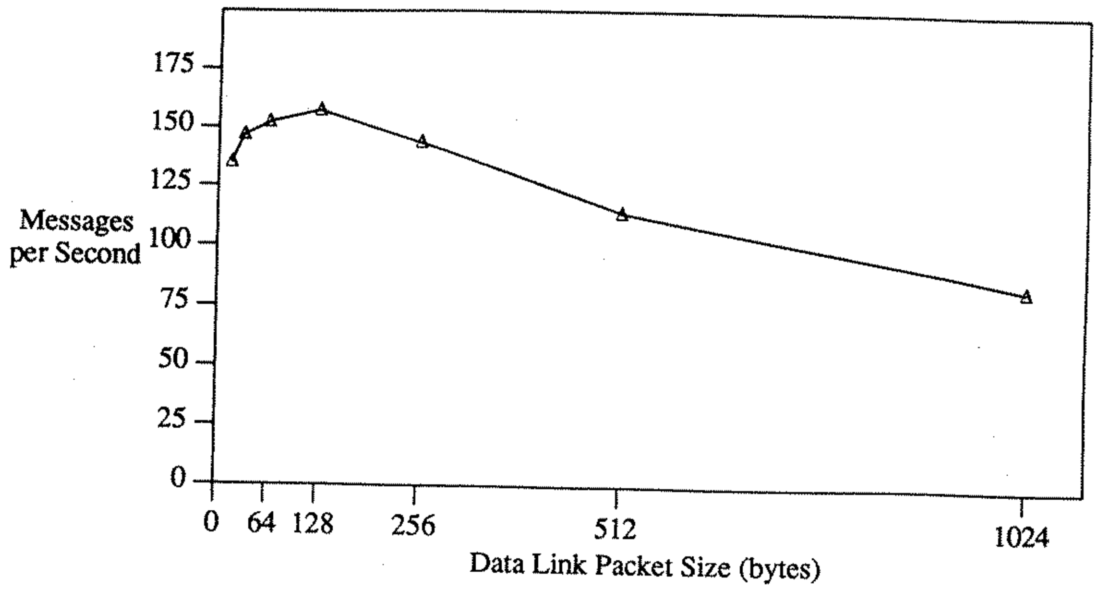


Figure 4.12 — Number of Messages per Second at Data Link Layer

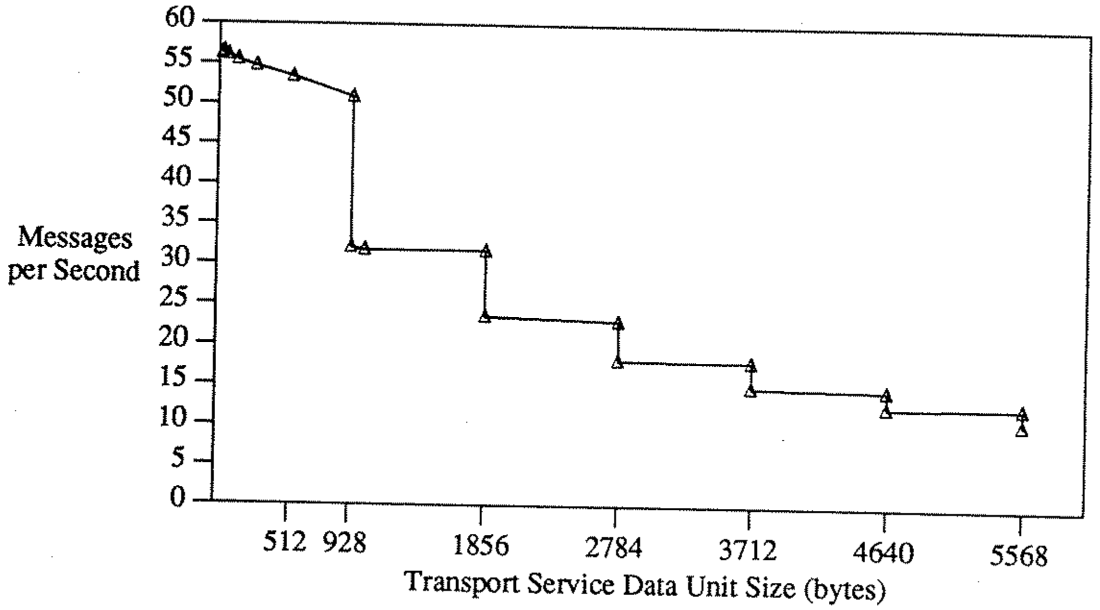


Figure 4.13 — Number of Messages per Second at Transport Layer

4.5.6. Retransmission Timer

The Network Management Facility (NMF) interface provided access to intra-layer configuration parameters such as the retransmission timeout interval. The actual retransmission timer uses an adaptive algorithm, adjusting itself to the characteristics of the network. The user of iNA960 cannot set the actual retransmission timer value. Instead, one supplies the Minimum Retransmission Timer setting which is used as the minimum for the actual retransmission timer setting at any point in time. Intel limits the Minimum Retransmission Timer value to no lower than 100 milliseconds during configuration of iNA960. However, we used the NMF functionality to reset this value lower than 100 milliseconds.

The values of these related timer settings ranged from 5 milliseconds to 500 milliseconds. The values at the high end of the range were chosen to show what would happen if the floor (minimum) value of the retransmission timer was set reasonably large. The values at the low end of the range likewise show what would happen if the value was unreasonably small. Ten user buffers were used by both the transmitter and the receiver to ensure that the hardware and software tasks remained busy.

As the Minimum Retransmission Timer value was decreased, as shown in Figure 4.14, the throughput remained mostly unaffected until the value was set at 50 milliseconds. For values above 50 milliseconds there were no retransmissions due to network error, so reducing the minimum value of the retransmission timer did not affect throughput. Throughput peaked at 50 milliseconds and then dropped dramatically for lower values. At these points duplicate TPDUs were being sent and rejected because the retransmission timer was expiring before the original TPDUs could be acknowledged. Thus we observed that 50 milliseconds was the lowest usable value that could be supplied to the Minimum Retransmission Timer for an unloaded network. We concur with Intel documentation [INTE86a] that this value should never be set below 100

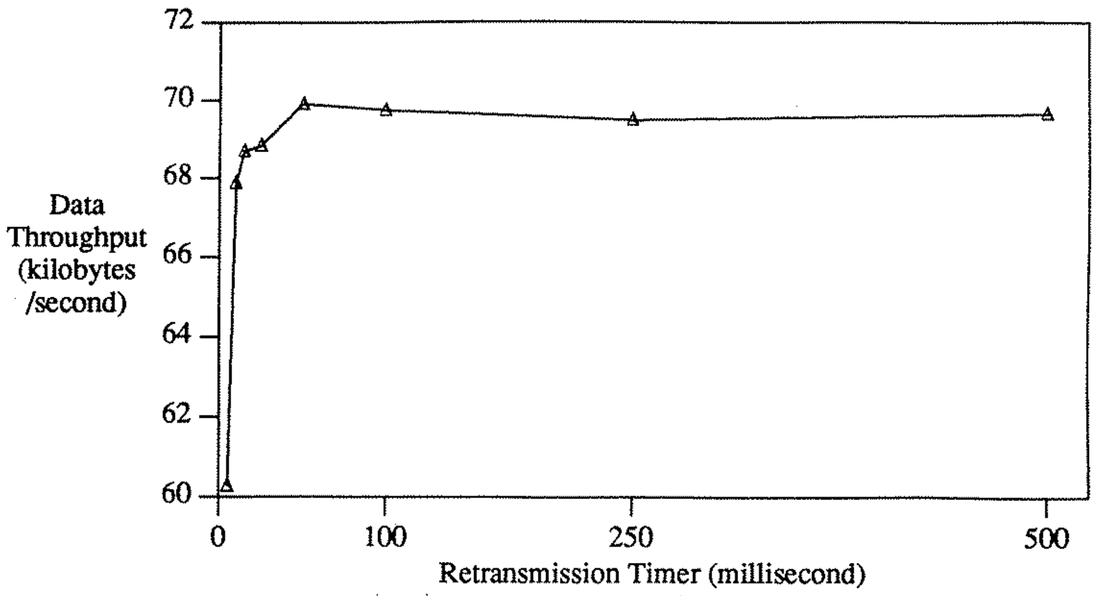


Figure 4.14 — Transport Throughput vs. Setting of the Retransmission Timer

milliseconds for a general purpose network. However, the implication of setting the Minimum Retransmission Timer to a value no less than 100 milliseconds is that lost TPDUs are not detected for at least one-tenth of a second. The default configured Minimum Retransmission Timer value is 500 milliseconds.

4.5.7. Segmentation

The TPDU is the packet frame and consists of both data and headers. The Network Management Facility allowed the user to specify a maximum TPDU size. iNA960, however, uses the minimum of 1024 bytes (the maximum Data Link packet size allowed in our configuration) and this maximum TPDU size as its actual TPDU size.

Figure 4.15 shows the relationship between throughput and TPDU size. Throughput is defined as the rate of data delivery in bytes per second and excludes the bytes required for framing. As TPDU size decreases, throughput decreases because the ratio of header bytes to

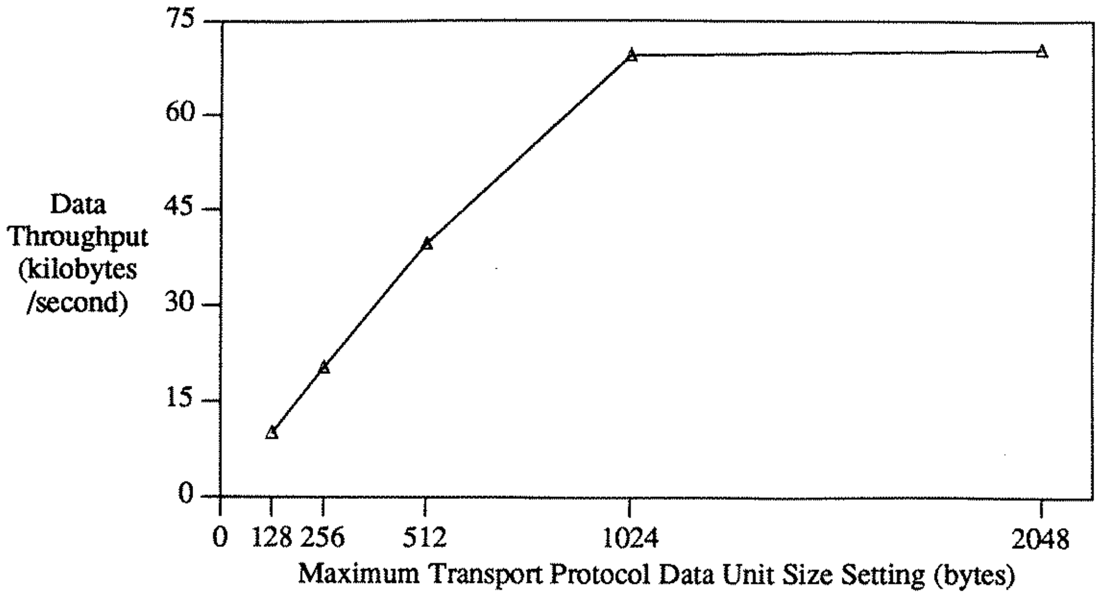


Figure 4.15 — Transport Throughput vs. Maximum Transport Protocol Data Unit Size

data bytes increases.

4.5.8. Virtual Circuits

We varied the number of virtual circuits, or connections, from 1 to 16 to observe its effect on throughput. As seen in Figure 4.16, one virtual circuit provided the best throughput because it required the least internal overhead. Even though multiple virtual circuits between two stations provided additional avenues for transfer of data, all virtual circuits used the same physical connection and thus overall throughput was not enhanced. In fact, there was the penalty of maintenance overhead levied on the user of multiple virtual circuits (connections), reducing the overall throughput.

The dramatic reduction in throughput for 3 virtual circuits was an anomaly. The experimental evidence strongly suggested that this was a worst case occurrence. The asynchronous characteristics of the communications between the host and the COMMengine,

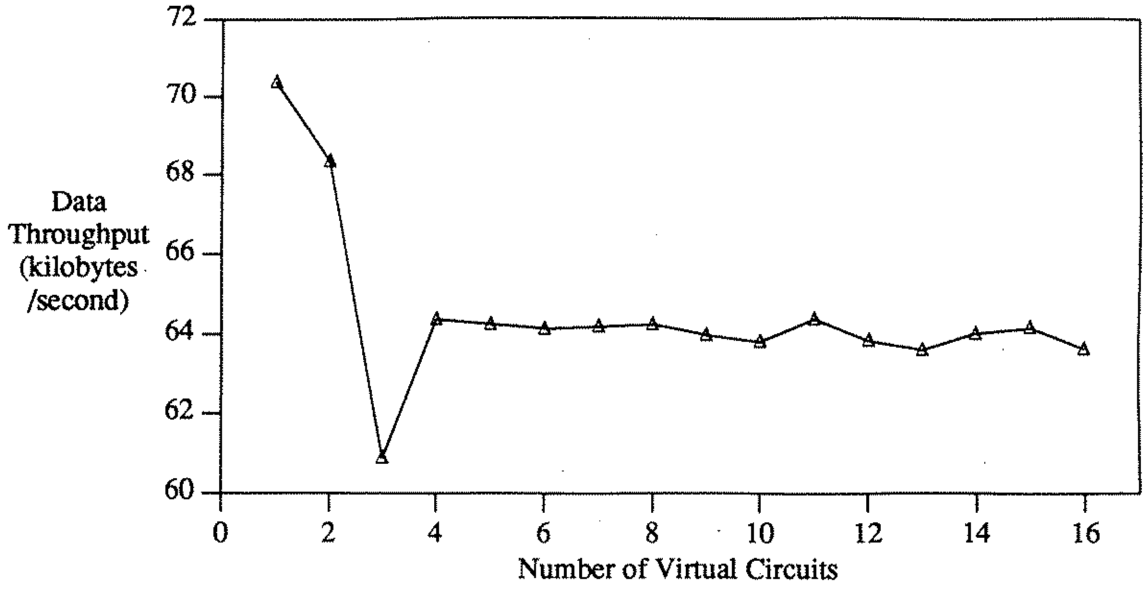


Figure 4.16 — Transport Throughput vs. Number of Virtual Circuits

the number and use of the internal transmit and receive buffers, and the token bus access all contribute to delays imposed by the system. We hypothesize that the poorer performance observed for 3 virtual circuits was due to a worst-case alignment of these factors, although we were unable to verify the hypothesis due to proprietary restrictions on the software.

4.5.9. Window Size

In Transport Class 4, a *window* is an ordering of *sequence numbers* that are termed active. The sequence number identifies and orders a particular TPDU so that the receiver may reassemble multiple TPDUs into one TSDU. The window *slides* to incorporate new sequence numbers as the TPDUs are acknowledged and their sequence numbers become inactive. Thus, the size of the window dictates how many unacknowledged TPDUs a receiver is willing to buffer. The receiver communicates this information via a *credit* field. The receiver can control the flow of data by varying its window size and thus throttle the transmitter by reducing its credit. A window size of 0 is called a *closed window*, and effectively shuts off the transmitter until the window is reopened by the receiver.

The maximum window size was set using the Network Management Facility and throughput was measured to show the impact of lowering the maximum window size. The range of maximum window sizes was 1 to 15, where 15 was the default setting from the iNA960 configuration. This configuration also prevented the window from closing, even when this was appropriate. The decision not to allow the window to close was based on Intel's empirical data which showed better performance at the risk of losing messages due to lack of resources [INTE86a].

As the receiver's buffer space was filled with incoming TPDUs, the number of TPDUs that it had room to receive decreased. To keep from being overrun, the receiver sent a "credit" with the acknowledgements. This credit told the transmitter how many more TPDUs the

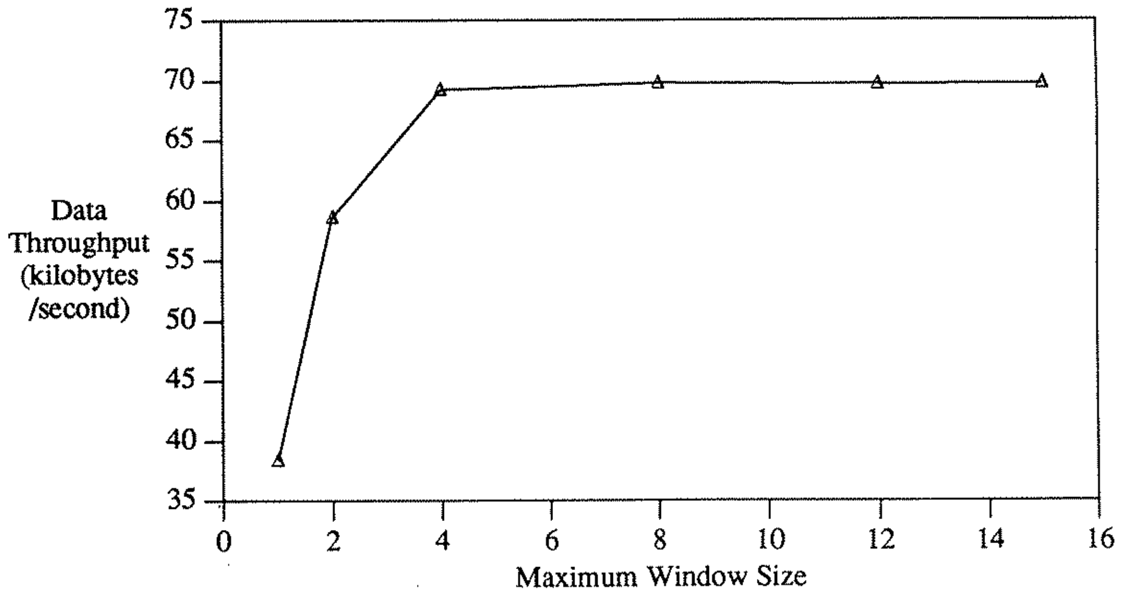


Figure 4.17 — Transport Throughput vs. Maximum Window Size

receiver was prepared to handle; it was in the range of 0 to the maximum window size. As the maximum window size was decreased, the credit was likewise decreased. This caused fewer TPDUs to be in the pipeline to the receiver and thus decreased the throughput. As Figure 4.17 shows, there appeared to be no effect when decreasing the maximum window size from 15 to 4, but decreasing it from 4 to 1 caused the throughput to suffer dramatically. This indicated that the receiver could not handle TPDUs sufficiently fast for the additional credit greater than 4 to matter. However, credit less than four caused the transmitter to throttle itself to having only one or two TPDUs in the pipeline.

4.6. Layer Comparisons

There were benefits and drawbacks of using a front-end processor like the iSXM554 COMMengine. Having the communications services provided by a front-end processor allowed concurrency. The COMMengine and the host processor ran in parallel and interacted via the Multibus. The host processor was not concerned with servicing the messages as they arrived asynchronously from the physical network. More computing time could be dedicated to the user application since the use of the network did not require CPU cycles.

However, the only means of access to the COMMengine and its software was through the MIP interface across the Multibus, which was a bottleneck. By accessing iNA960 through the External Data Link interface, restrictions were placed on the size of the messages that could be transmitted by the user application. Large messages had to be buffered in the host's memory and delivered to the EDL in smaller (1024 byte) segments. In this instance many small messages were sent through the MIP interface, across the Multibus and to the COMMengine, subsequently causing the throughput to suffer.

By accessing iNA960 through the Transport Layer, large messages were delivered to the front-end processor and stored there until they could be processed. iNA960 and the underlying data link hardware worked most efficiently while there was data in buffers on the COMMengine. When iNA960 was finished processing one large message, it could be sent another to be stored in the on-board buffers. This increased throughput by decreasing the number of interactions across the MIP interface.

Figure 4.18 shows that time-dependent messages should employ the EDL rather than the Transport Layer. There was a constant cost of approximately 7.5 milliseconds associated with using the additional functionality provided with the Transport layer, which may be too expensive for short command/response or status/request messages. The bound on the message size was 1024 bytes for the Data Link Layer. At Transport, the bound was 928 bytes before the TSDU message was segmented into two TPDUs. For messages of length L and smaller than

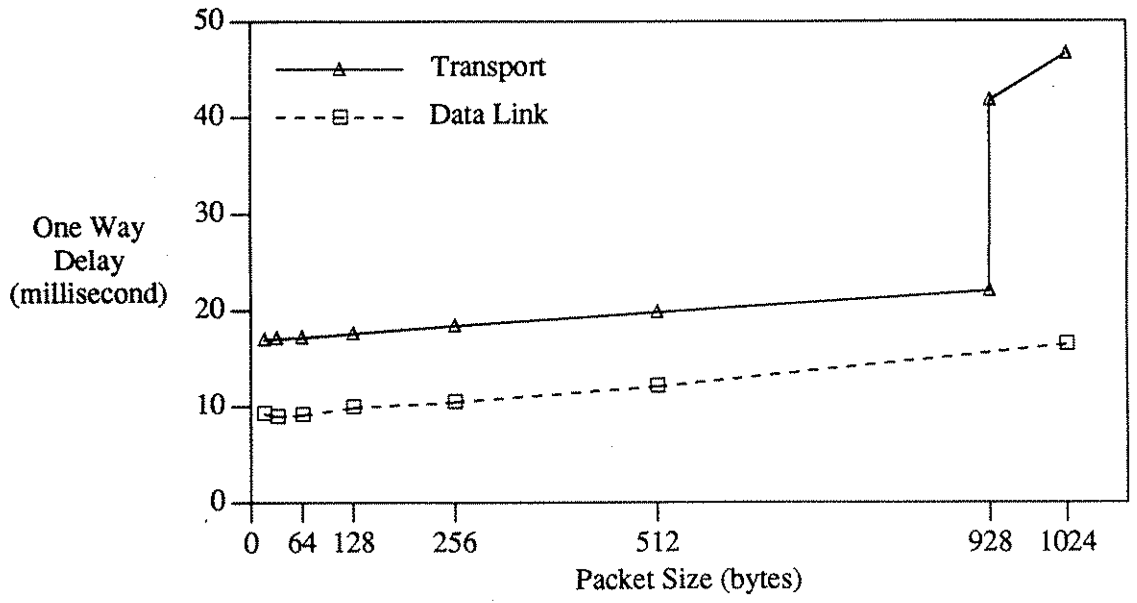


Figure 4.18 — One Way Transport and Data Link Delay vs. TSDU and Data Link Packet Sizes

928 bytes, the one way delay at transport was approximately (16.5+L/250) milliseconds; using the EDL interface it was approximately (9+L/250) milliseconds.

Data Link provided better throughput for packets of the same size, as seen in Figure 4.19. This was because the TSDU messages were all small enough to fit into one TPDU, so segmentation was not an issue. For messages of length 928 bytes or shorter, the difference between the throughputs reflected the difference between the overhead associated with a relatively simple Data Link data transfer service and the overhead associated with a complicated but reliable Transport data transfer service. Data Link, however, is restricted to packets of size 1024 bytes or smaller. When messages are larger than 1024 bytes, the message must be segmented.

When an application process uses Transport, it moves entire TSDUs onto the front-end processor, then Transport segments the TSDUs into TPDUs as required. If the application

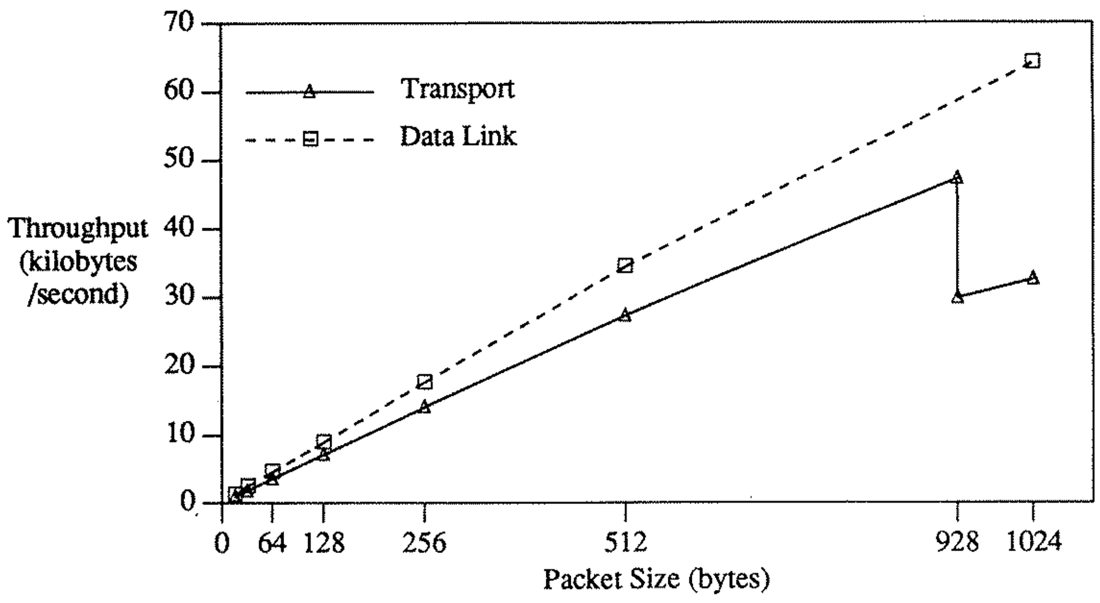


Figure 4.19 — Transport and Data Link Throughput vs. TSDU and Data Link Packet Sizes

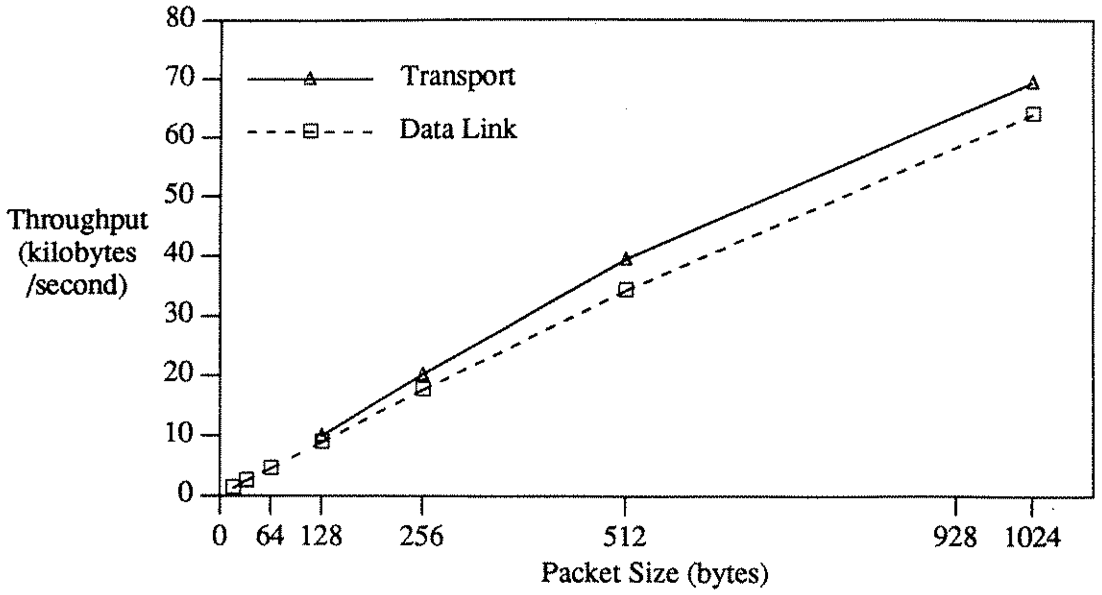


Figure 4.20 — Transport and Data Link Throughput vs. TPDU and Data Link Packet Sizes

process uses EDL, the host must perform the segmentation and deliver packets no larger than EDL is configured to accept. By forcing the maximum TPDU to be the same size as the Data Link packet, we observed the difference between segmentation on the front-end processor by Transport and segmentation on the host by the application process. In Figure 4.20, the throughput for Transport is slightly greater than throughput for Data Link. This shows that an application process cannot segment messages on the host as well as Transport can segment the messages on the front-end processor. Thus, in terms of performance, segmentation on the front-end processor was more efficient.

Chapter 5

CONCLUSIONS AND FUTURE WORK

5.1. Observations

The iSXM554 COMMengine provided a front-end processor which successfully off-loaded the task of communications from the host. Its primary advantage was that it provided transparent segmentation (the reduction of an arbitrarily large TSDU into multiple TPDUs). Even so, segmentation was expensive — the first segmentation increased end-to-end delay by approximately 20 milliseconds and each additional segmentation added another 5 to 15 milliseconds. When using the largest TSDU not requiring segmentation (928 bytes), throughput averaged 47 kilobytes/second; when using the smallest TSDU requiring segmentation (929 bytes), throughput dropped to 30 kilobytes/second.

Data Link supported a connectionless-mode (datagram) service for packets up to 1024 bytes. Transport used that underlying connectionless-mode service and added segmentation, sequencing, acknowledgment, and reassembly to provide a connection-oriented (virtual circuit) service. As expected, direct access to the Data Link Layer provided shorter one-way delays and higher throughput than did direct access to the Transport Layer. For large messages, however, sending one large TSDU to Transport and allowing it to perform segmentation was more efficient that having the host perform segmentation and pass multiple smaller packets directly to Data Link.

In terms of absolute performance, a two-station system with no other computational tasks using the Data Link interface supported continuous transmission rates ranging from 156 128-byte messages per second to 84 1024-byte messages per second. At Transport, it supported continuous transmission rates ranging from 56 16-byte messages per second to 13 5000-byte messages per second. When Transport was supplied continuously with large messages, segmentation overhead limited the maximum throughput to approximately 72 kilobytes/second. We confirmed that Transport's retransmission timer should not be set to less than 100 milliseconds; the implication is that lost packets will not be detected for at least 0.1 second which could be very significant in real-time control systems. We observed that a window size of 4 was sufficient to achieve maximum throughput, and that using multiple virtual circuits decreased data throughput due to circuit maintenance overhead.

5.2. Future Work

There are several extensions to this research that we plan for the near future. Some of the work will be further investigation into MAP network, such as upper layer performance analysis and the methods provided by MAP to join several subnetworks or networks. Other work is directed toward possibly improving the Intel implementation of the Transport retransmission algorithm. The impact of the speed of the operating system will also be studied. Finally, the Physical and MAC Layers will be replaced with IEEE 802.3 CSMA/CD network and an analysis will be performed of the Technical Office Protocol (TOP).

5.2.1. Upper Layer Data Transfer Analysis

The Session and Case Layers in MAPNET2.1 use the data transfer services of Transport, mapping their data transfer functions directly onto those of Transport. Therefore the evaluation of Transport data transfer was performed and generalized to represent the data transfer performance of the upper layers. Another set of performance experiments will be done on the actual Session and Case data transfer services to verify this generalization.

Experiments will also be designed to measure certain aspects of File Transfer, Access and Management (FTAM) and Directory Services. It is expected that delays in file access and transfer with FTAM will be insignificant when compared to the delays caused by disk access. Since CASE and FTAM use names to identify the remote process, the Directory Services must be invoked to make the name-to-address translation at least once. Measures of the access time for the data base, along with the data transfer analyses, will provide more insight into the costs of using Application Layer functionality.

5.2.2. Bridges, Routers and Gateways

A bridge as described by MAP can be constructed using an iSXM186/51 CSMA/CD board with a 586 Multimodule placed onto it to provide two segments of CSMA/CD networks sharing one node. By using multiple tasks to capture Data Link packets, those intended for the other segment can be redirected to that segment. This would make the two segments transparent to the Network Layer and above. It is expected that there would be a penalty for subnetwork isolation, which can be measured using the External Data Link Performance (EDLP) tools we developed here. Although the Physical and MAC Layers would be IEEE 802.3 CSMA/CD rather than IEEE 802.4 Token Bus, the cost of including a bridge in a network can still be observed and generalized.

Similarly, a router may be constructed by placing a 586 Multimodule onto the iSXM554 Token Bus board, allowing an Token Buss segment to be joined with a CSMA/CD segment. Again, EDLP will be employed to provide the data transfer across the two heterogeneous segments and measurements of the cost of joining two segments in this manner.

A gateway may be constructed by placing two COMMengines into the Multibus backplane and running the full MAPNET2.1 on each. A task would be monitoring messages received by one COMMengine. If a message requires translation and transmission on the other COMMengine, a dedicated task would reroute it. This would simulate a gateway from a MAP network to some other network. It is expected that the cost of using a gateway is severe because the message needs to be propagated through two full seven layer stacks.

5.2.3. Retransmission Timer Algorithm

The adaptive retransmission timer employed by iNA960 Transport does not handle degraded service due to lost or damaged packets in a timely and efficient manner. An abbreviated Transport can be constructed using the External Data Link services. This transport will be able to transfer and acknowledge data and retransmit packets that are lost or damaged. Lost or damaged packets can be artificially induced. By studying the distribution of the frequency of lost or damaged packets, an error-prone system can be simulated. This will be the testbed for developing an algorithm that minimizes the cost of lost or damaged packets by adapting according to recent conditions.

5.2.4. Upgrading the Central Processing Unit Board

The new iSXM80386/20 CPU boards are reported to improve the speed of the operating system by 5 times, according to Intel. By replacing the current iSXM80286-10A CPU board with this newer, faster, more advanced processor, the impact of the operating system on communications may be seen. Furthermore, this increase in speed will provide a better indication of the actual speed of the front-end processor for communications.

5.2.5. Technical Office Protocol TOP

Technical Office Protocol is essentially MAP functionality on all Layers of the ISO stack except the Physical and MAC Layers, where IEEE 802.4 Token Bus is replaced by IEEE 802.3 CSMA/CD. A similar evaluation of performance is foreseen, with emphasis on those characteristics whose behavior we have studied here, and characteristics which many be Physical Layer dependent.

The IEEE 802.3 standard is an older, more established Physical and MAC Layer protocol. Intel has developed several COMMengines designed for CSMA/CD, the most recent of which is the iSXM552A board, which employs a special Ethernet controller called a 82586 co-processor. This co-processor relieves some of the MAC Layer responsibilities from the board's CPU. Also, this protocol is less complicated than Token Bus, requiring no modem or additional hardware such as a headend remodulator.

We expect to see improved results in all respects. This may not be so much the result of a superior Physical Layer protocol as it may be the maturity of the product.

BIBLIOGRAPHY

BYER86	Byers, T.J., "MAPing the Islands of Automation", *PC World*, December, 1986, pp. 269-277.
COLV86	Colvin, M.A., and Weaver, A.C., "Performance of Single Access Classes on the IEEE 802.4 Token Bus", *IEEE Transactions on Communications*, December 1986, pp. 1253-1256.
ELEC86	Electronic Industries Association IE-31 SP 1393A Working Group, *RS-511 Manufacturing Message Service for Bidirectional Transfer of Digitally Encoded Information*, Draft 5, June, 1986.
FRAN86	Franx, C., "The Synchronous Clock System", Unpublished, July, 1986.
GENE86	General Motors Manufacturing Automation Protocol Committee, *GM MAP Specification*, version 2.2, 1986.
IEEE84a	The Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard 802.1B Station Management*, 1984.
IEEE84b	The Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard 802.2 Logical Link Control*, 1984.
IEEE85	The Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard 802.4 Token-Passing Bus Access Method and Physical Layer Specifications*, 1985.
INDU85a	Industrial Networking, Inc., "MAP/One Cable Starter Kit *Installation Guide*", 1985.
INDU85b	Industrial Networking, Inc., "MHR-40 Head End Remodulator *Installation Guide*", 1985.
INTE85	Intel, "iRMX86 Reference Manuals", 1985.
INTE86a	Intel, "iNA960 R2.0 Configuration Guide", Final Draft, December 1986.
INTE86b	Intel, "iNA960 R2.0 Installation Guide", Pre-Release, August 1986.
INTE86c	Intel, "iNA960 R2.0 Programmer's Reference Manual", Final Draft, November 1986.
INTE87a	Intel, "MAPNET2.1 User's Guide", July 1987.

INTE87b	Intel, "The Big MAP Attack", Unpublished, March 1987.
ISO7498	International Organization for Standardization, *Draft International Standard 7498*, "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", October 1984.
ISO8072	International Organization for Standardization, *Draft International Standard 8072*, "Information Processing Systems - Open Systems Interconnection - Transport Service Definition", June 1986.
ISO8073	International Organization for Standardization, *Draft International Standard 8073*, "Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification", July 1986.
ISO8326	International Organization for Standardization, *Draft International Standard 8326*, "Information Processing Systems - Open Systems Interconnection - Basic Connection Oriented Session Service Definition", September 1986.
ISO8327	International Organization for Standardization, *Draft International Standard 8327*, "Information Processing Systems - Open Systems Interconnection - Basic Connection Oriented Session Protocol Specification", September 1986.
ISO8473	International Organization for Standardization, *Draft International Standard 8473*, "Information Processing Systems - Open Systems Interconnection - Data Communications Protocol for Providing the Connectionless-Mode Network Service", March 1986.
ISO8571	International Organization for Standardization, *Draft International Standard 8571*, "Information Processing Systems - Open Systems Interconnection - File Transfer, Access, and Management", August 1986.
ISO8602	International Organization for Standardization, *Draft International Standard 8602*, "Information Processing Systems - Open Systems Interconnection - Protocol to Provide the Connectionless-Mode Transport Service Utilizing the Connectionless-Mode or Connection Oriented Network Service", February 1986.
ISO8649	International Organization for Standardization, *Draft International Standard 8649*, "Information Processing Systems - Open Systems Interconnection - Service Definition for Common Application Service Elements", June 1986.
ISO8822	International Organization for Standardization, *Draft International Standard 8822*, "Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service Definition", July 1986.

TANE81 Tanenbaum, A.S., **Computer Networks**, Prentice-Hall, Inc., 1981.

WEAV87 Weaver, A.C., "Local Area Networks and Busses: An Analysis", *NASA--Johnson Space Center*, January 1987.