Innovative Communication Processors: A Survey

by

Per Gunningberg

SICS R87003
Research Report
ISSN 0283-3638

Innovative Communication Processors: A Survey

Per Gunningberg
Swedish Institute of Computer Science
P O Box 1263
S-163 13 Spånga, Sweden

March 1987

Table of Contents

Innovative Communication Processors: A Survey

Abstract

Some existing innovative processors for execution of multilayer protocols are surveyed in order to identify performance limits and processor architectural trade-offs. The survey is restricted to packet or message handling processors with dedicated software and/or hardware.

The processors are compared with respect to; *performance*, i.e. throughput and delay, to *available protocols*, and to *implementation trade-offs*, i.e. modularity, service access point accessibility, interface to host machine, dependability, implementation language, buffer handling, and context switch with state information. Of special interest are processors for protocols designed according to the OSI-model. One conclusion is that the implementation of buffer handling and context switch is crucial for the performance.

The surveyed processors are divided into two categories; switches for X.25/ARPANET and controllers for Local Area Networks. The switches are often designed as multiprocessor systems, optimizing throughput, while the controllers often are designed with dedicated processors and chips, optimizing not only for throughput, but also for delay and for off-loading the host.

1. Introduction

The ISO:s Open Systems Interconnection reference model for multilayered protocols has gained rapid acceptance for structuring communication functions between communicating computer systems. The seven hierarchically ordered layers in the model relates functions ("services") such as routing, ordering of message, error recovery, synchronization control, information transformation, etc. ISO:s working groups have more or less formally specified the services that each layer should provide, possibly in combination with lower layers. The working groups are also defining standard protocols that provide these services. The model is frequently used for both Wide Area Networks as well as for Local Area Networks, i.e. where heterogeneous environment are likely to be found. In homogeneous systems and over shorter distances many of the functions are not needed and hence simpler or fewer protocols can be used. The reference model has recently got a lot of attention in connection with GM:s effort to standardize a set of protocols for communication between robots, computer controlled assemble machines, management systems and others, in the automated factory of the future. These protocols, called Manufacturing Automation Protocols, MAP, are designed according to the model. In fact, ISO standardized protocols will eventually be used at all layers complemented with unique automation protocols at the Application layer. A similar initiative to standardize the Technical Office Protocols (TOP) has been taken by Boeing. The TO Protocols are almost identical to MAP. These efforts have spurred many of the major American vendors of computing equipment (including IBM) to form a consortium (COS) in order to promote "Open Systems". Similar promotions are taken place within Europe ESPRIT (SPAG) and COSINE and in Japan (POSI).

The ARPANET protocols were the first to be hierarchically organized into layers. The ARPA network was taken into limited operation in the early 70's and today it connects more than 100 networks and more than 1000 computers, covering the whole United States with satellite connections to Europe. The protocols were developed according to the characteristics of long distance leased telephone links over the US continent with several intermediate unreliable switches. Terminal access, file transfer, and electronic mail are the major ARPA high layer protocols and their services became quickly indispensable for the connected users, mostly researchers. During the years, several performance measurements have been done and they demonstrate that under low utilization the response time and throughput are adequate for the distance covered and type of services. Experiments with other types of services were also done, some of them not feasible for this type of networks, like the Networking Operating Systems experiments.

In the late 70's and the beginning of the 80's the concepts of Local Area Networks, LAN:s, were developed. Covering a smaller area(< 5 km) with dedicated cables and no intermediate switches, the LAN:s allow for much higher speed than Wide Area Networks. New lower layer protocols for controlling the access to the cable, taking advantage of the limited area and the speed characteristics were developed and later the subject for standardization. The higher layer protocols remained very much the same as for Wide Area Networks, since the protocols are independent of the media. LAN:s have become attractive for distributed computing since they provide flexibility in the number of processing elements attached, resilient against failing elements, and a homogeneous access scheme. However, distributed computing of type "remote procedure calls" requires much faster response times and throughput compared with terminal access time and file transfer to be feasible. A common experience was that it was only possible to get up to 150 kbit/sec. efficient file transfer on a 10 Mbit/sec. cable, and the turn-around time (to request a data from a remote processing element) was in the order of 10-100 msec. The performance is very much depending on **how** the protocols are implemented. The straightforward implementing strategy is to maintain the layers as separate processes in a regular operating system, gives very poor performance[Wiks83, Jung85, Sjöd87]. In a retrospective view, it is clear that a major source of inefficiency is the copying of messages from one process to another.

During the years, several communication processors have been developed for multilayer protocol execution. Some for Wide Area Networks with many connections and some more recent for Local Area Networks with one connection and with more emphasize on response time.

1.1 Scope of the survey

The objective with this survey is to identify performance limits and architectural trade-offs of communication processors. Therefore it is restricted to:

-existing dedicated innovative communication processors for multilayer protocols,
-processors for packet/message handling, and
-processors with dedicated software and/or hardware.

Some communication processors will be grouped together and described by a representative sample. Traditionally, processors used for front-end processing, remote data concentration, terminal concentrator, channel processor, pabx:s, telephone switches, ISDN-switches, and bus interfaces are also called communication processors. These will **not** be included unless they have some innovative architecturs. Likewise, this is **not** a market survey. Commercial products are only included if they are innovative in some aspect and well documented. A market survey over communication processors in one of its widest definitions can be found in DataCommunication, May 1986[DCom86].

1.2 Comparison criteria

The surveyed processors are discussed and compared with respect to ***Performance, Protocols,*** and ***Implementation trade-offs***:

Performance

•*Throughput.*

Throughput is measured as the number of packets/sec. or bits/sec. that can be processed, see figure 1. Important for the throughput are:

- packet size. Packet sizes vary from 2 bytes to 4096 bytes in this survey.
- number of concurrent connections. The processor might support 1 connection or maybe 500. With many connections it is much easier to get high throughput, especially if they can be hierarchically organized onto a multiprocessor structure, so that all packets do not need to pass a central resource, like a global memory or a common bus. Both throughput numbers for one and several connections are presented when available.
- protocol "complexity". There is no absolute measurement on complexity. A rough judgement on the number and type of functions is used with reference to the ISO protocols. "Low", "medium" and "high" complexity is used and the complexity of ISO protocols are considered to be "high". For example, a protocol with flow control and recovery will usually be slower than a protocol without.
- number of layers traversed. Our experience is that a processor with many layers will be slower since the context switching from layer to layer is one major source of inefficiency.

•*Delay.*

The delay time used here, see figure 1, will be the time required either for the processor to:

a) transmit a packet, measured from when the processor gets aware that a packet is available somewhere, until it is completely transmitted,

b) to receive a message, measured from the time the processor gets aware that a message is on the incoming trunk, until the time the host processor is notified.

c) or in the case of a switch a combined time of a) and b), i.e. the time for the packet to pass the switch.

The delay is considered under ideal conditions, when there is no contention for any of the resources, including the trunk and when there are no partly filled buffers in the transmitter nor in the receiver. The delay also depends on the same factors as affects the throughput, i.e. packet size, protocol complexity, and the number of layers traversed, except maybe for the number of connections. The receiving and transmitting delays are normally very similar and therefore seldom separated.

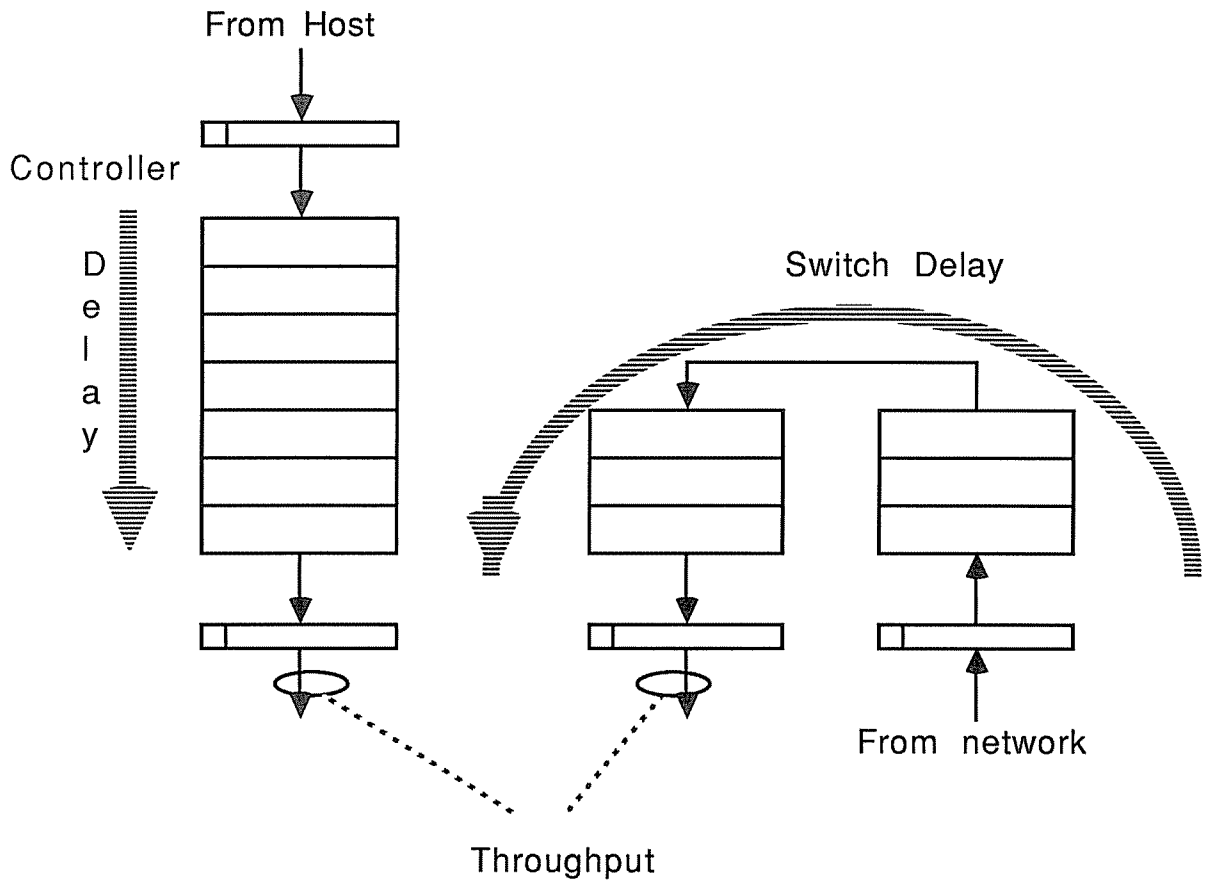


Figure 1. The throughput and delay characteristics of switches and LAN controllers.

Protocols

•*Protocol hierarchies ("stacks") supported and options chosen.*

Of special interest are the stacks ISO/CCITT, MAP/TOP and, ARPANET and their corresponding emerging functional standards (COS, SPAG, COSINE, POSI etc.). Communication processors seldom support whole stacks or complete protocols. Normally, some adjacent layer protocols are supported. It is also common to support only some parts of a protocol, and even to change a standard protocol in order to gain performance, or to exclude options available, e.g. expedited data and connection negotiation.

A typical computer communication system implements a protocol stack with VLSI dedicated chips for the lower layers, with a communication processor for the intermediate layers , and with high layer protocols in the host machine, using kernel and user processes.

Implementation trade-offs

•*Modularity and Service Access Point, SAP, accessibility.*

One of the salient features of the OSI model is the implementation independence of each layer. That is, it should be possible to change one protocol without affecting the other layer protocols. Also, for some applications a designer may want to bypass some layers when he/she does not need the services (as for the MAP Enhanced Performance Architecture [ISO184]). Are these lower layer access points available, or are they embedded together with other layer protocols?

•*Interface to host machine*

One of the crucial points in terms of high performance is the interface to the host, how messages are passed and how control information is exchanged. The messages and the control information must be located in shared memory accessable both by the host and the processor. The access time to them are crucial for the performance. Expensive dual port memory is used for the highest demands and a less expensive DMA interface with a secondary memory is used for the lowest demands. Interrupts and polling techniques are used to inform the host or processor that new control information or commands are available for the other.

•*Dependability.*

A major concern for switch designers is the availability. A malfunctioning switch may block hundreds of computers.

•*Implementation language.*

Does it support high level implementation language? Is there any support system that can translate a protocol written in the language into the software and hardware used in the communication processor? Especially, can it take a protocol specification language?

•*Buffer handling.*

Buffers are used to hold messages or part of messages awaiting to be processed. How to create buffers for messages, how to handle free buffers, how to map buffers into the processors memory system, and how to link buffers have profound effects on performance. Too few buffers will reduce throughput, since it means higher risk for buffer overflow. This will lead to discarded messages followed by resendings. The "window size" is normally adjusted to the number of available buffers and too few buffers means that there will be fewer outstanding messages between a sender and receiver. There is a trade-off between many small buffers linked together and maximum message size buffers, with respect to the time to read messages and memory usage, but an algorithm to get the optimum is in practical terms elusive because such an algorithm will depend on both the actual implementation and the characteristics of the traffic.

•*State information and context switching.*

Each established connection has a context that needs to be saved when the processor switches to another connection. The context contains information about the connection identifier, the sequence number, pointers to buffers holding messages for possible retransmissions, etc.

Context switches will occur when messages should be sent, when messages arrives, and at time-outs. This switching can be considerable when several connections must be maintained. For a 10 Mbit Ethernet a new message may arrive within 10 microseconds in the worst case. The problem is similar to the context switching problem in real time operating systems, though in a communication processor context switches will be more frequent, and the tasks shorter.

1.3 Results and organization

There is no conclusion on "the best buy". Each design has its own trade-offs with respect to delay, throughput, number of connections, cost, etc. Generally, the switches are optimizing for many connections and overall high throughput, sacrificing delay characteristics, while the LAN controllers only have one connection and can neither sacrifice throughput nor delay. In general the processor should off-load the host computer as much as possible since it is easy to swamp the host with the execution of communication tasks. That is independent of layer, protocol, switch and controller.

Two early surveys [Newp72] and [Thei72] of communication processors in the more wider terms was published in 1972. To our knowledge there are no recently published surveys or taxonomies.

The survey is organized into three main sections; first the historical background to the evolution of networks and communication processors is summarized, thereafter the X.25 switches are described, and in the last section Local Area Networks controllers are surveyed. It turned out that the X.25 switches could be be grouped together according to their geographical origin. There are sections for the American, Japanese, and European processors. The survey is concluded with discussions on the design of the surveyed processors with respect to the comparison criteria.

2. The evolution of packet networks and switches

In 1964 Rand Corporation did a study on "how to access data at remote computers and how to provide interactiv terminal traffic to remote computers". The study concluded that a store and forward message network were much more feasible than comparing technologies; fully interconnected leased lines and dial-up services. Based on this study, the Advanced Research Projects Agency(ARPA) of the Department of Defense, 1968 embarked on the implementation of a computer network later known as the ARPA network. The first four sites were operational in late 1969 with a node called The Interface Message Processor or, *IMP*. But ARPANET was not the first successful large commercial message network. It was the airline reservation system SABRE that in 1964 used messages in its *one* hop network, i.e. it was star formed. Indeed, ARPANET was the first to provide *multihop* networks using communication processors which received and stored packets originating from different locations, analyzed them, and finally routed them to one or more destinations hosts. Routing and flow control was hence invented with ARPANET[Lein84].

CCITT had an informal study group on data transmission in 1964, and in 1968 it became Study Group VII (New Networks for Data Transmission). These early studies were only concerned with circuit switching. As late as July 1974 a special Rapporteur was appointed by CCITT for studying message switching. At that time several networks were already in operation besides ARPANET; NPL in the UK, the international reservation airline systems SITA, the public RETD network in Spain and Tymnnet in US, and Transpac in France. In Canada DATAPAC was announced. In England the experimental EPSS by British Post Office was started. Contracts for Euronet and EIN was announced 1975 and 1974, respectively. In Japan NTT did research on a combined circuit/packet network already in 1971. IBMs System Network Architecture, first operational in 1974, is an example on a successful commercial network covering IBMs product line.

The origin of X.25 was from earlier COST activities, and in late 1975 it was quite hastily put together in order to have a standard for the CCITT Plenary meeting in 1976. In spite of many known technical flaws, the X.25 specification was accepted at the meeting. Many of the flaws very solved in 1980, some of them unfortunately, as multiple options[Sirb85].

In connection with the experimental networks, several projects started on the design of dedicated switches. The first designs were based on general purpose uniprocessors but in the mid seventies, several multiprocessor switches were reported on, of which some are included in this survey. Since then, the number of yearly published switches have slightly decreased and in later years there are about 2 or 3 of them. Many of the early switches have successfully been turned into commercial products.

The Local Area Network (LAN) Ethernet started out as a laboratory project in 1974. But not until recently LAN:s have started to gain widespread acceptance and usage, maybe because earlier

there were too many different types of LANs proposed and few of them had the necessary support of vendors. Recent standardization efforts within IEEE and ISO have decided on a few types and this seems to spur the vendors to develop LAN controllers. Many manufactures, including Intel and Motorola, have recently developed controllers or "communication engines" that include protocols from all seven layers of the OSI model, of which some controllers will be reported on here. GM:s MAP effort will also most likely accelerate the development and the acceptance.

3. ARPANET Interface Message Processors.

The ARPA Network has today more than 80 nodes and more than 200 hosts connected. It covers the whole United States and has satellite connections to Europe and is connected to other types of networks over gateways.

During the design of the ARPA network in the late 60's it was decided to use dedicated identical processors at each node, in order to *insulate computer centers* from network tasks, such as routing, buffering, and synchronization and to *insulate the network* from computer center problems and to avoid heterogeneous implementations. The nodes then collectively formed the first multihop packet store and forward subnet. The centers' computers, called hosts, were attached to the network via the nodes. The protocol hierarchy was consequently divided between the nodes and the hosts, such that the nodes implement the physical, data link and network layers, providing node-to-node functions, and the hosts the upper, end-to-end protocols. Initially, the nodes were interconnected with 50 kbit/sec. leased telephone trunks. Later the bandwidth was decreased on some trunks since the top capacity seldom was used.

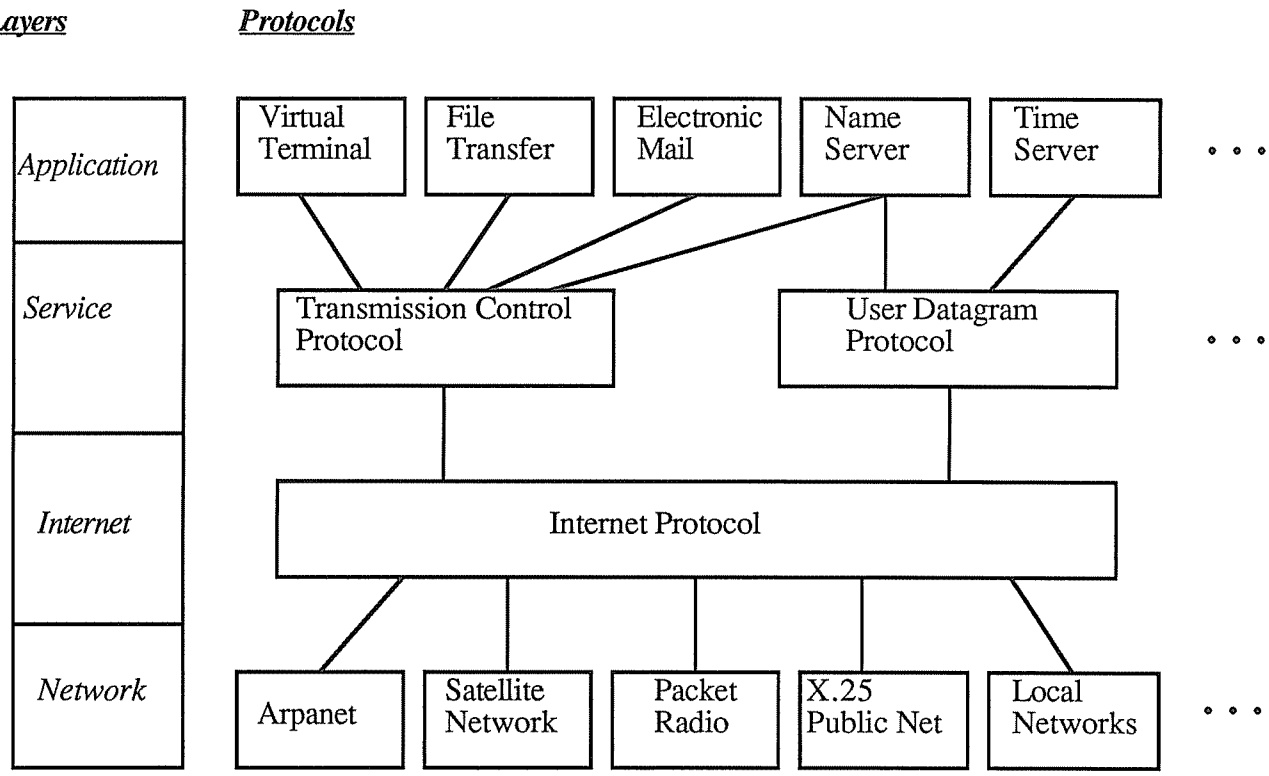


Figure 2. The ARPANET layered protocol hierarchy.

The first node processor, called Interface Message Processor, IMP was based on Honeywells processor DDP-516. In comparison with other minicomputers by that time(1969) it had two advantages that were desirable; a good I/O capability and a "rugged" version. However, already in 1971, it was decided to use the less expensive Honeywell 316 for new nodes. It costed half as much as the 516, was downward compatible, while it still provided two third of the 516's I/O capacity.

By that time,when a terminal user wanted to access a remote host, he needed to connect himself first to the local host which thereafter set up the connection to the distant host. The local host was facing the problem of composing packets out of terminal characters, direct them to the remote host and vice versa (Packet Assemble and Disassemble, PAD). Furthermore, the user was depending on the availability of the local host for his remote access. Burdening the local host with such terminal transit traffic turned out to be unreasonable. Therefore, it was decided to modify the 316 IMP with a PAD function. The modified IMP was called TIP, Terminal IMP and it supported 64 terminals[Orns72].

The two major drawbacks of the 316/516 were their small memory size and that there were interface slots for only four hosts, (less than four for a TIP). Also the fact that Honeywell quitted manufacturering the 316/516 series, stirred some worries about future maintenance and developments. Therefore, Bolt Baranek and Newman Inc. (BBN), that had the responsibility for the network, introduced the Pluribus multiprocessor IMP and TIP in the mid 70's. The objective with Pluribus was to achieve high availability, high throughput, high bandwidth, large amount of host ports, and big memory. It did reach these objectives. Unfortunately, it was not cost-effective for a normal node. For example, a new host interface sometimes required an addition of both a new processor and a new memory bank. BBN was also worried about having two versions of software to maintain, one for Honeywell and one for Pluribus. In the late 70's BBN developed a new switch by using bit-slice technology that had an instruction set emulating Honeywell 316. This new switch, called C/30, is in the process of replacing all older ARPA IMPs including the Pluribus'. The advantages with C/30 are that it can run 316/516 software, it supports more interfaces than 316/516, requires less power, has a 65 kword memory, and has firmware controlled interfaces. The TIP will be replaced by a two C/30, one that acts as an IMP and one that handles the terminal traffic.

The two most interesting designs, the 516/316 IMP and Pluribus, will be described in more detail with respect to the comparison criteria. More details on C/30 can be found in [Haug82].

3.1 The ARPA Honeywell 516/316 IMP

The original Honeywell 516 minicomputer was equipped with a 12 K memory (16 bits/word) and four interfaces for hosts and modems. The operating system was rather conventional and the machine did not have any special hardware to support communication. But from this studys point of view it introduced many of the problems an implementer of protocols will face today when using a traditional machine, so we will look into the software organization with some detail.

The software was squeezed into about 6 K words and organized as interrupt driven tasks. The additional 6 K was used for buffers (5 K) and tables. The program was divided into tasks that handled the node-to-node processing, node-to-host processing and support software, like debugging, statistics, terminal, etc., where the support tasks were run in the background.

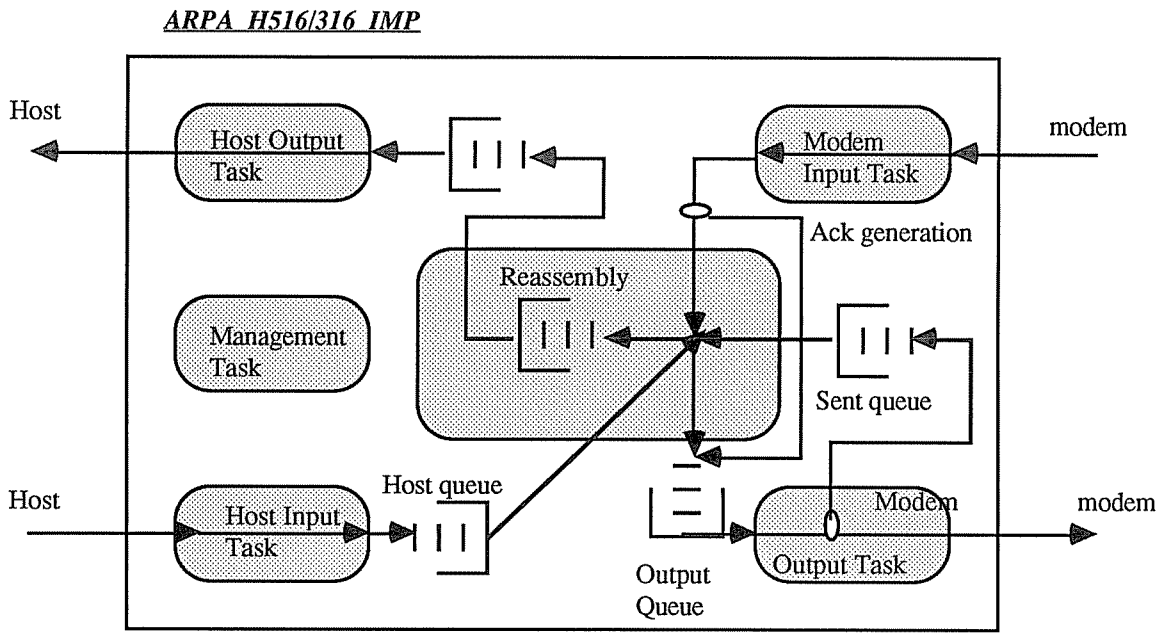


Figure 3. The IMP task structure and packet flow. *Simplified from [Hear72]*

Four principal types of packet queues were maintained (also see figure 3):

Host Tasks Queue Messages that are received on the Host channels are put into this queue, awaiting to be processed by the Main task.

Output Queues: An output queue is organized for each connection to the other nodes. The queues are subdivided into an acknowledgement queue, a priority message queue, a Request For Next Message queue, and a regular message queue. The Send task is serving the subqueues in the same order.

Sent Queues: Contains messages that are awaiting acknowledgements.

Reassembly Queue: Holding messages that are in the process of being reassembled. A message might be fragmented into smaller pieces on the enroute from the source host to the destination host and it is the task of the destination node to reassembly the message before delivering it.

In figure 3, there is also a schematic drawing of the packet flow. One task is not indicated in the figure; the *time-outt* task. It is started every 25 msec by a clock interrupt. The task has three sections: a "fast" time-out section, which wakes up tasks that might be deadlocked, a "middle" time-out section that are resending packets that have been waiting too long for an acknowledgement, and a "slow" section, that updates routing tables according to nodes that are disconnected and that does long term garbage collection of buffers from partly reassembled messages (e.g. from dead nodes). For more details on the other tasks, see [Hear72].

Performance

Packet size: 128 bytes

Number of concurrent connections: four hosts and four nodes

Layers traversed: Two layers, functionally corresponding to Physical through Network layers

Protocol complexity: Ordinary complexity. Reassembly when sending to host.

Throughput

From host through IMP to modem: Estimated maximum is 700 000 bits/sec., provided that there are at least 5 packets a message, i.e. less than 700 packets/sec.

Delay

Store and forward packet to another node: Estimated to .35 msec under ideal conditions.

Protocols

The IMP supports the two lowest layers, Physical and Network, of the ARPA hierarchy. There is full support for HDH, HDLC, VDH, RTP at the Link layer and M/W and BBN 1822. The IMP is obviously also used for SATNET, PRNET, and other networks, but this is unconfirmed information.
Higher layer protocols, e.g. TCP/IP are implemented in the hosts.

Implementation trade-offs

•*Modularity and SAP accessibility* The service is only accessible through the "Host Protocols". It does not seem to be possible to reach lower layers directly. The protocol implementations seem to be totally integrated and also optimized with respect to the physical memory limitation.

•*Interface to host machine* The host can only reach the IMP through the "Host Protocols". One of the reasons for this was to isolate the IMP from host problems.

•*Dependability* A lot of effort was put into the design to get the IMP reliable. Besides using rugged hardware the strategic decision was to isolate the IMP as much as possible from the hosts by physical isolation. Other efforts include automatic restart from power failure and "watchdog timers" that transfers control to a recovery task in protected memory when they are not reset for about a minute. Everything that is unique to a particular IMP also resides in protected memory.

In 1972 it was reported that the Mean Time Between Failure was about one month.

•*Implementation Language* All software was written in assembler code.

•*Buffer Handling* The H316/516 worked with 70 fixed length buffers for storing packets. Each task is guaranteed a minimum number of buffers. Free buffers are chained into a "Free list". A packet, once stored in the buffer, is never moved. When the packet is confirmed to be successfully passed to a host or a node, the buffer is returned to the free list. To avoid buffer lock-up, each connection is guaranteed a minimum number of buffers at both the sender and receiver. Under abnormal conditions, e.g. when a sending node fails, reserved buffers must be released. This is taken care of by a garbage collector that is initiated by a time-out.

•*State information and context switch* The state for each host or node connection is stored in memory. Context switching takes place on demand from an interrupt or trap.

3.2 The Pluribus

The Pluribus was designed to overcome the 316/516 shortcomings while it still was both smaller and less expensive. The most important requirements were formulated as[Hear74]:

◊ expandibility of host, node and terminal interfaces,

◊ modularity, so that a very small IMPs could be designed with the same hardware,

◊ memory expandibility so that the number of buffers could be increased in order to accommodate satellite connections,

◊ higher reliability than H516,

◊ an overall bandwidth of at least 7 Mbit/sec.

With these requirements in mind the team at BBN settled for a multiprocessor design. Highly favorable for such a design was that the team knew the communication algorithms (i.e. the 316/516 protocol implementations) very well, and that the algorithms lend themselves to fragmentation into parallel structures. It is not so much the processing of one message that lend itself into parallel processing as the processing of several messages, especially from different connections. The time to process one message is relatively short so their idea was to have several processors, each working on separate messages. It was then possible to match the number of processors with the number of expected messages. With this arrangement it was believed that the ultimate limit of parallelism was only the organization of the multiprocessor itself.

The Lockheed SUE processor was chosen as the processor to build Pluribus on. It was inexpensive but rather slow and the idea was to compensate its slowness by additional processors. It also had an attractive bus structure that lent itself for multiprocessing.

Based on the total bandwidth requirements the design needed at least fourteen processors. The processors were connected two by two into seven parallel SUE-buses, see figures 4 and 5. Each processor had a local memory (< 16 K) that stored the most frequently used tasks.

A global memory of 32-80K was designed around two dedicated SUE buses. The shared memory was used for buffers, global variables, for storing copies of all tasks, state and management information on the multiprocessor configuration, and others.

A double I/O bus was also designed around the SUE-bus, holding interfaces to modems and hosts, timers, and the so called "Pseudo Interrupt Device" used for assigning tasks to processors.

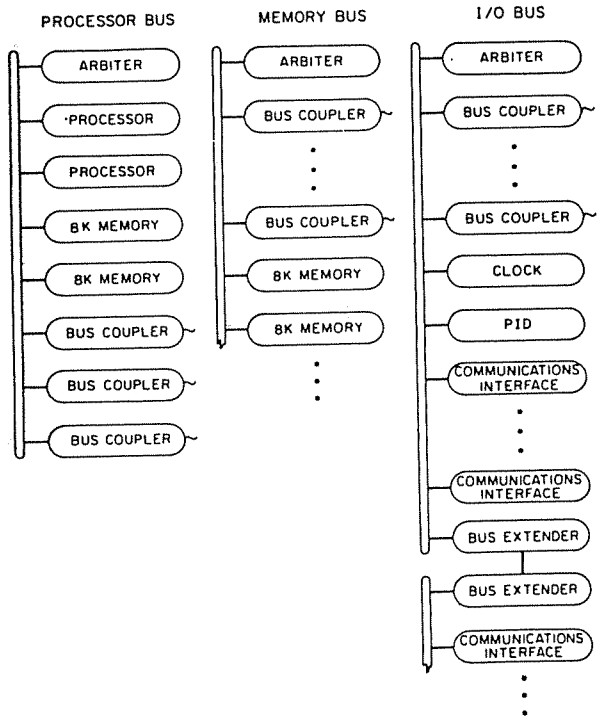


Figure 4. Pluribus different modules, built around the Lockheed minicomputer SUE-bus. *From [Hear73]*

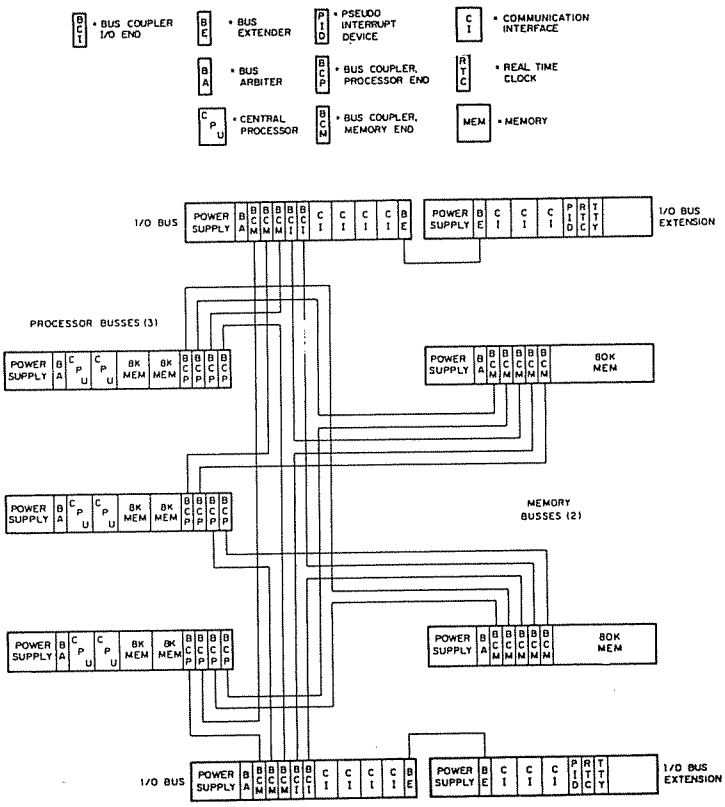


Figure 5. The Pluribus prototype system with all interconnections between modules. *From [Hear73]*

All double processors, global memories, I/O-buses were *fully* interconnected via so called "bus couplers", see figure 5. The bus couplers did the addressmapping, i.e. identified when an address should be directed outside the local bus. It was believed that this interconnection structure gave high flexibility and that it could smoothly vary with system size, in spite of the known risk of contention to the shared global memory.

A lot effort was put into creating the processors as identical as possible so that any processor could do any of the communication tasks. Thus all processors had identical software. Furthermore, the communication software was fragmented into small pieces of code that formed independent tasks(< 300 microsecond). By doing this, high reliability was achieved. A failing processor only affected one small task and thereafter the performance was only slightly degraded since all processors could run all tasks. Also, a new processor could enter the system smoothly.

The fragmentation also simplified the task distribution and scheduling. The key innovative idea was to keep a common queue of pending tasks from which every processor could grab the task with the highest priority and execute it. When a processor is done with a task it looks into the queue for a new task. It identifies the task, removes the identifier from the queue, finds the corresponding unique data in global memory, and executes the task according to the code in its own local memory. In order to get fast access to this queue it is implemented in hardware in the special interface device "Pseudo Interrupt Device" (PID). An interrupt or another task can schedule a new task by inserting an identifier into the queue.

According to Heart et al, the approach has the following advantages:

> ..."By using a passive queue in which the processors check for a new task when they are ready, we avoid some nasty timing problems. Tasks may be entered into the queue at any time, either by a processor or by the hardware I/O devices. This approach is an extremely important departure which avoids the use of conventional interrupts and the associated costs of saving and restoring machine state. Further, this approach neatly sidesteps the problem of routing interrupts to the proper processor".

Performance

Packet size: 128 bytes.

Number of concurrent connections: "unlimited".

Layers traversed: Two layers, functionally corresponding to Physical thru Network layers.

Protocol complexity: Ordinary complexity. Reassembly when sending to host.

Throughput

Estimated maximum 1.3 Mbits on one interface. There are no measured values in our references that confirm the estimated throughput.

Delay

There is no data available in our references.

Protocols

The Pluribus is compatible with H316/516. Hence, it supports the two lowest layers, Physical and Network of the ARPA hierarchy. There is full support for HDH, HDLC, VDH, RTP and BBN 1822. The IMP is obviously also used for SATNET, PRNET, and other nets, but this is unconfirmed information.
Higher layer protocols, e.g. TCP/IP, are implemented in the hosts.

Implementation trade-offs

•*Modularity and SAP accessibility* The service is only accessible through "Host Protocols". It does not seem to be possible to reach lower layers directly. The protocol implementations are totally adjusted to the multiprocessor. The software is divided into strips of code with maximum execution time of 300 microseconds each. All processors have all communication software in their local memory and hence, Pluribus could work with only one processor. A lot of effort has been spent on making the multiprocessor structure modular to be able to increase throughput incrementally.

•*Interface to host machine* The host can only reach the IMP through the "Host Protocols".

•*Dependability* One of the most important design requirements was reliability. It was obtained both by the modular design and by using fault-tolerance. The modular design allowed processors to fail without affecting other processors. The fault-tolerant features included a large array of techniques; watchdog timers on buses and I/O, redundant buses and I/O interfaces, regular test programs, redundant software structures, and "boot-strap" approaches to reconfigure the multiprocessor after a failure.

Measurement figures taken over some years demonstrate that 92% of rated throughput is achieved 99.76% of the total time. The fault-tolerant techniques are described in detail in [Kats78].

•*Implementation Language* Unknown.

•*Buffer Handling* Scarcely described in available references. However, it seems to be similar to H316/516.

Redundant software is used to detect missing buffers due to failures. Each buffer is tagged and every process needs to set the tag when using the buffer and resetting them when deallocating in order to avoid cross-referenced buffers. There are also watchdog timers associated with each buffer to ensure that they circulate properly.

•*State information and context switch* There is no context switch in the usual meaning, since each task is independently executed by any processor. However, there is a fast degenerated switch in order to check if a high priority task has arrived to the PID. State information for each connection and message is stored in global memory.

4. Japanese Packet Switches

Several packet switches aimed at X.25 networks have been developed in Japan since the late 70:s, both at universities and industry research institutes. Most active (at least most frequently published) has NTT Telecommunications Networks Laboratories been and we will concentrate on their effort. Also University of Tokyo, Kokusai Denshin Denwa Co. (KDD), and Mitsubishi Co. have produced prototypes that are unique in some aspects. Significant for them all is that they are designed for high overall throughput based on many low speed X.25 connections.

KDDs design is aiming at the concentration of many terrestrial connections into a satellite link[Ono83]. These connections include X.25, X.75, Digital Voice, and CSMA/CD. The design is organized into one processor for each type of connection, including one for the satellite link. An experimental set-up was planned to be in operation late 1983.

Mitsubishis approach is rather conventional by using a minicomputer and multiplexers. One interesting aspect is that they have optimized the operating system so that it will handle interrupts and make context switches faster. It is claimed that it made a considerable impact on performance, but not how much[Tana83].

4.1 NTT:s packet switch D51

An X.25/X.21 chip was developed at NTT in the early 80's[Tona83]. The chip was thereafter used in the design of the high throughput switch D51. Commercial versions of D51 (it is unclear if it is still called D51) was installed in major cities in Japan in 1985[Yash85]. In 1986 NTT published a design study of a new X.25 VLSI chip that is exploiting the inherent parallelism in X.25[Aoki86]. We will here discuss the initial VLSI chip, the D51, and the new chip.

The X.25 chip ECL 1604

The functions of X.25 level 2 and the control part of X.21 were split into 9 "fundamental" tasks by the design team of this chip. Three alternative distributions of these tasks over hardware (sequential logic) and firmware (a processor) were considered in a trade-off discussion with respect to the desirable rate of 64 kbit and chip limitation area. All of them could be implemented into either hardware or in firmware. Alternative C was completely in hardware while A and B had a mixture of both, see figure 6. The alternative B was chosen because A was too slow and C was breaking the chip limitation area. The chip is therefore logically divided into a Line Control Unit, a Processor Unit, and a Memory Unit, see figure 6.

The interface to the host is typical for communication chips. The Host activates the chip by setting command words in registers or in predefined memory locations. In receiving mode, the chip sets pointers to the data buffers and informs the host that data is available by interrupting it. At transmitting mode, the host sets a pointer to the data to be sent. The chip starts sending the data and interrupts the host when it is done it.

Functions \ Methods		A	B	C
Line Control	Flag Generate/Test	○	○	○
	'0' Insert/Delete	○	○	○
	Aborting an Invalid Frame	○	○	○
	FCS Generate/Test	◎	○	○
	ABORT Generate/Test	◎	○	○
	IDLE Generate/Test	◎	○	○
Link Status Control	Frame Supervision/Control	◎	◎	○
	Retransmission Control	◎	◎	○
	Timer Control	◎	◎	○

○ : Hardware ◎ : Firmware

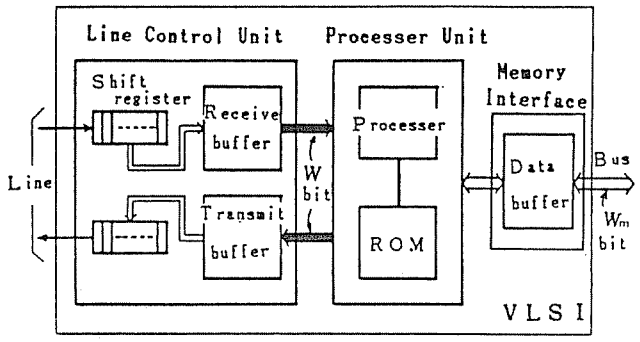


Figure 6. The functional blocks of X.25 and the VLSI structure. *From [Tona83]*

The D51 Packet Switch

D50 was NTT:s first multiprocessor switch, built around IC boards[Naga84]. D51 is their second generation switch, also a multiprocessor design. It is better than D50 in many aspect of which the ability of incrementally add new connections and the much smaller physical design, due to the use of X.25 chips, are considered to be the most important.

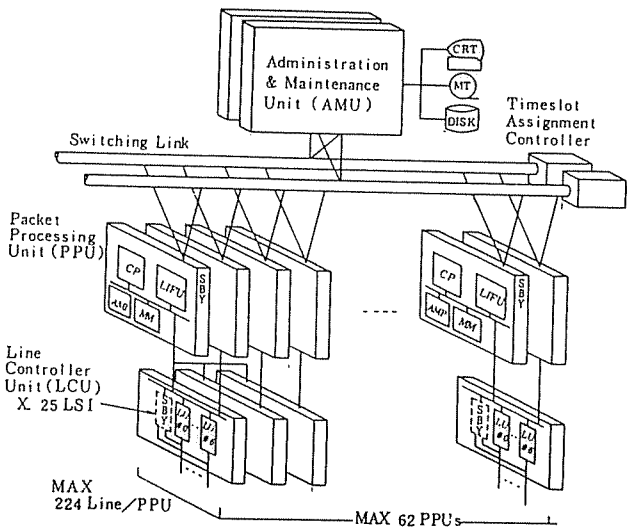


Figure 7. The D51 packet switch organization. *From [Yash85]*

The D51 consists of at most 62 "Packet Processing Units" (PPU), a "Switching Link", and an "Administration & Maintenance Unit" (AMU)[Naga84, Yash85, Niwa86]. The PPU controls several Line Controller Units that are using the above ECL 1604 X.25 chips. See figure 7.

One interesting point is how the X.25 and management functions are distributed over the units. Level 1 and 2 of X.25 are implemented in the chip, while level 3 is implemented in software. This means that all packets must pass through the PPU processor, even if the destination connection is in the same Line Control Unit as the source connection. The PPU:s themselves communicate over the switching link, according to a token bus scheme. The whole packet is sent over the link (6 Mbyte/sec.), from one local memory to the other, when the source PPU gets access to the token. The sending PPU first needs to reserve buffers at the receiving PPU.

It is worth to note that the AMU has nothing to do with the actual transmission of packets.

Performance

Packet size: Varies from 32 to 4096 bytes with an estimated average of 346 bytes.
Number of concurrent connections: It is unclear! According to the figure and one place in the text [Yash85] the maximum number is 62 times 224, and another place in the text the number is 62 times 4!
Protocol complexity: X.25 level 1-3

Throughput

Overall throughput is estimated to be more than 10 000 packets/sec., i.e. about 27 Mbit/sec., independent of number of connections. An individual X.25 connection is restricted to 48 kbits/sec.

Delay

Estimated to 200 microseconds. For maximum load it is estimated to 400 microseconds.

Protocols

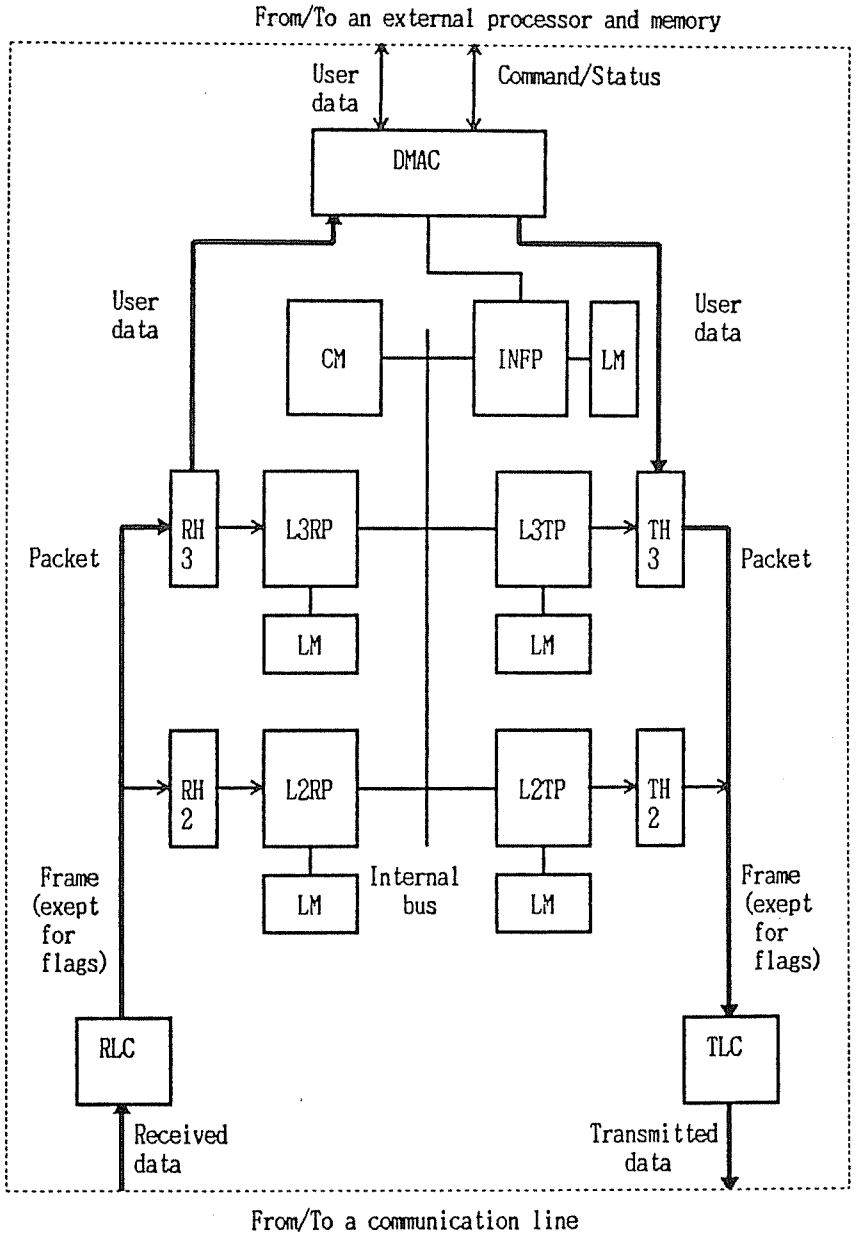
X.25 level 1-3, most likely 1980 version. It is not stated what options that are supported.

Implementation trade-offs

•*Modularity and SAP accessibility* Level 1 and 2 are implemented in a chip, while 3 is implemented in software in a dedicated processor. 1 and 2 are not directly accessible.

•*Interface to host machine* This is a pure switch. There are no host connections.

•*Dependability* A lot of effort has been put into the design in order to make it reliable. Some of the more important are; the switching link and AMU are duplicated, the PPUs are identical so that surviving PPU:s can take over the task from a failing one, error correcting codes are used for memory reads and on data transfers.



CM: Common memory
DMAC: Direct memory access control
INFP: Interface processor
L2RP: Layer 2 receiving processor
L2TP: Layer 2 transmitting processor
L3RP: Layer 3 receiving processor
L3TP: Layer 3 transmitting processor
LM: Local memory (ROM/RAM)
RH2, 3: Receiving header buffer for layer 2, 3
TH2, 3: Transmitting header buffer for layer 2, 3
RLC: Receiving line controller
TLC: Transmitting line controller

Figure 8. Block diagram of the parallel X.25 chip. *From [Aoki86]*

•*Implementation language* CHILL is used.

•*Buffer handling* Buffer assignment is critical at the interface to the switching link, therefore the sending PPU reserves buffers at receiving PPU before transmission. There is no information on PPU internal buffer handling

A new parallel X.25 chip

A new chip is under the design at NTT that will utilize the inherent parallelism in X.25 in order to gain performance[Aoki86]. Basically, the X.25 processing is divided into four parallel tasks; the senders and receiveres of both level 1 and 2. Obviously, the senders and receiveres are not completely independent, since they share many state variables, like sequence numbers and P and F bits. In order to ensure correct ordering of events it is identified how these shared variables are accessed and taken into consideration in the design. The two levels, however, are completely independent with respect to header processing.

The chip is organized into a processor with a local memory for each of the four tasks, plus one processor for the "upward" interface, see figure 8. A common memory holds the shared variables.

The performance has been estimated by calculating circuit delays and number of instructions needed for each packet. For 100 bytes packets the maximum throughput is estimated to about 20 Mbit/sec. and the limit is the processing time for a packet. For packets over 400 bytes the throughput is instead limited by the sequential bit sending circuit, to about 50 Mbit/sec.

4.2 University of Tokyo's X.25 Switch

A prototype of a multiprocessor switch designed at University of Tokyo is reported in [Sait82]. It is a typical set-up with dedicated communication processors controlling X.25 connections while the processors themselves communicate over a common memory. The problems with these set-up are the contention and arbitration to memory. The contention problem is reduced by using a local memory for each processor, that stores the whole X.25 software. The arbitration and contention problem is also attacked by dividing the common memory into partitions, one for each processor. A processor can write and read in its own partition and only read the others. A sending processor writes the packet in its own partition while a receiving processor only can read this partition.

One innovative idea is to separate the virtual circuit handling procedure from the communication processors. Therefore a main processor is used to set-up and close the connections, to establish routes, etc., so that the communication processors are off-loaded these tasks. The communication processor only needs to handle data packets and to sort out all non-data packets to the main processor. This means that the throughput might be limited by a congested main processor if there is an excessive amount of connection establishments per second. It is estimated that there are at

least 10 data packets for each administrative packet.

Some measurements were taken from an experimental set-up, based on Z80, that has a performance far from the initial requirements. The Z80 was able to handle 270 kbit/sec. on 1000 bit packets.

5. European Packet Switches

5.1 The French SCIPION switch

The objective with the SCIPION project is to design a common switching node for packet, circuit, and satellite communications. The idea is that the nodes should form a backbone of high bandwidth lines. A node is not only a "super multiplexer" to another node because X.25 connections can also be switched internally, see figure 9. Here, only the Packet Switch Module (PSM) is reported on[Put78].

The PSM has the capacity of 64 connections, each with a throughput of 48 kbit/sec., and can set up and clear up to 1000 virtual circuits/sec.

The designers divided the X.25 "functions" into two categories: *"repetitive treatments"* and *"supervision treatments"*. Basically, the supervision category include the tasks of connection set up, clear, and switch management, and the repetitive category the data transfer tasks. This division of tasks seems to be very similar to NTT:s D51 switch, but with other names. Also the implementation is similar. Both use special hardware for the data transfer phases and both use supervision processors for the more complicated connection management algorithms.

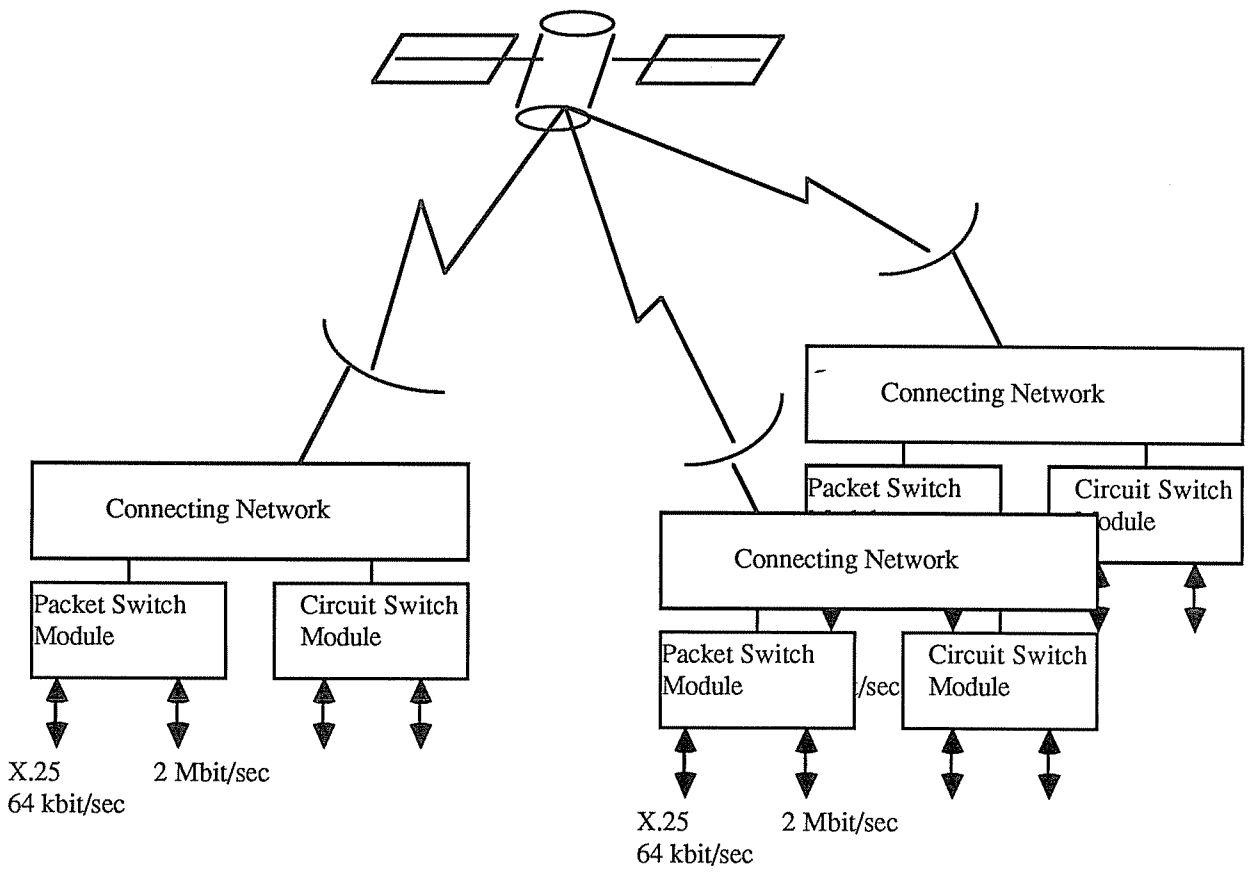


Figure 9. The SCIPION functional block description.

The PSM is organized around; two buses, a supervision processor, a shared memory and the following dedicated modules(also see figure 10):

- "Couplers" to X.25 lines and "connecting network"

- "Procedure handler",
- "Logical Channel Switch",

The Procedure handler is divided into three submodules, one that provides the Link data transfer, one that provides the Network data transfer, and one that is controlling the other two. The first two are implemented as Finite State Machines with Programmed Logic Arrayes while the third is "a fast microprocessor". All three share an "identifier storage". The "Logic Channel Switch" controls the internal connection between two X.25 lines.

There exists a laboratory prototype, but there are no performance measurement figures are available in our references.

Performance

•*Throughput*

Only specified as 64 kbit/sec. each for at most 48 connections.

•*Delay* Not specified.

Protocols

ISO 1980 X.25 with minor (unclear) modifications. There are also some node-to-node backbone internal protocols.

Implementation trade-offs

•*Modularity and SAP accessibility* It does allow the supervision processor to add functions. Lower layers are not accessable.

•*Interface to host machine* None. This is a switch.

•*Dependability* Not mentioned.

•*Implementation language* Unknown

•*Buffer handling* Unknown

•*State information and context switch* There is special hardware for both. The context is stored in a "Context memory" which holds the most frequently accessed parts of the state. The memory is located in the Procedure Handler and it has a direct input of the X.25 line identifier which points to the current context. A special "identifier storage" is also used to speed up context switch.

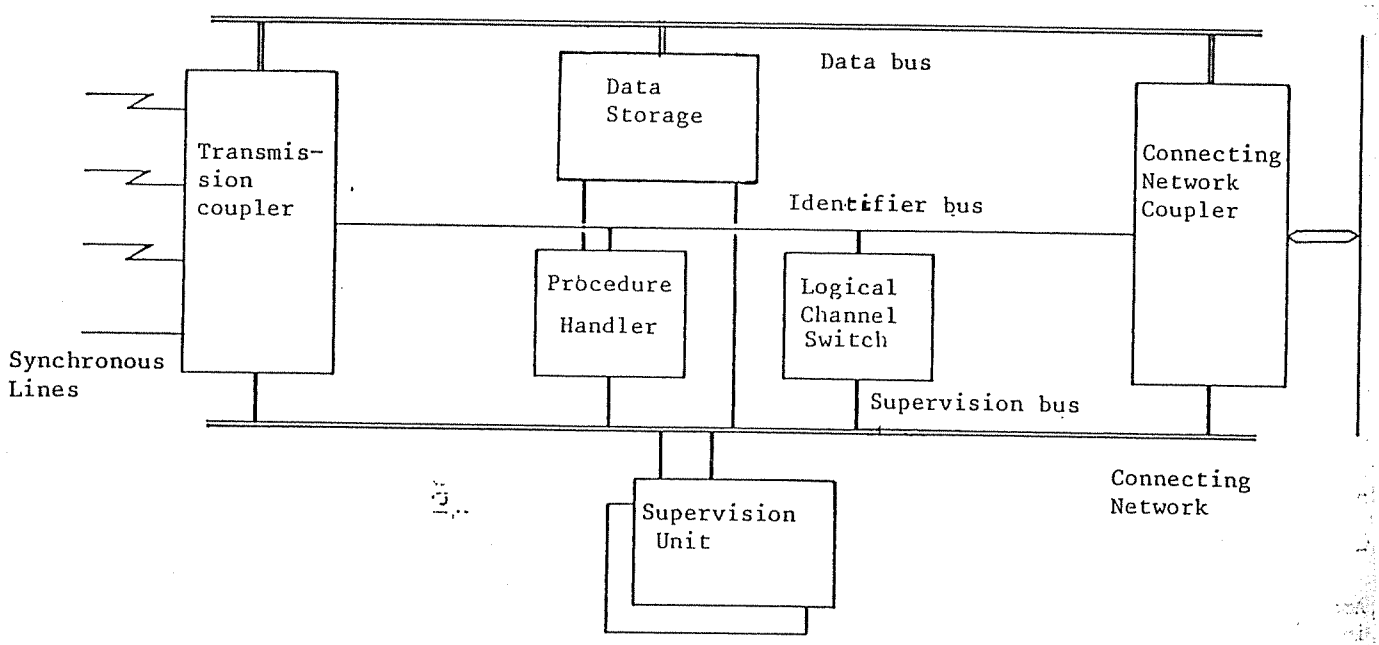


Figure 10. Functional block diagram of the of the Packet Switch Module *From [Put78]*

5.2 Siemens EWSP switch

Already in 1980, Siemens in Germany had a commercial X.25 packet switch available, called EDX[Hube80]. It is a part of a more general switch for circuits, packets, and messages. It is a hierarchically organized multiprocessor system with a duplicated bus, see figure 11. A processor, a memory, and X.25 hardware comprises a "Line Controller" which are "...largely responsible for protocol handling" while a "...Central Processor mainly performs the tasks of establishing and clearing connections and of coordinating the Line Terminators responsible for data transfer". All Line Controllers are connected to both buses and communicate over a shared memory. The benefit of this hierarchical organization is incremental growth, since "Network Nodes" can be added to control several Central processor and so on.

At 1980 there were more than 80 systems in operations.

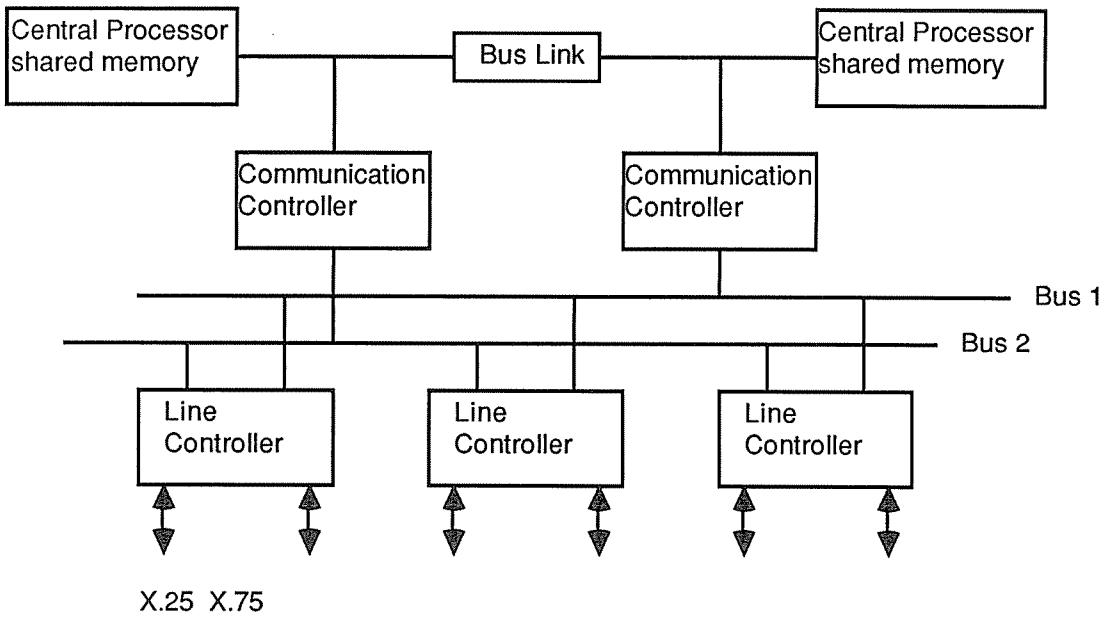


Figure 11. The EDX switch.

EWSP - A high performance switch

Recently a new packet switch design from Siemens was disclosed[Mair86]. The objective is to make it economical for the whole range of 100 to 10 000 lines. It is a multiprocessor design similar to the old 1980 switch organisation. Compared with the old switch, it is operating with more advanced Line Controllers (called Terminator Units instead) than before. Furthermore, the shared memory and the Control Processor of the 1980 EDX is replaced with several "Switching Units". The previously shared memory is now physically subdivided, and together with a dedicated processors they comprise these Switching Units. Instead of using one central processor to control all X.25 lines as in EDX, the control is distributed over all Switching Units. Still, the Switching Units are responsible for call set up and close down. Both sending and receiving units are interacting in the set

up of calls. Thereafter, when the data transfer phase is started only one units shared memory is used for the interaction while the other unit is on standby in case of problems, see figure 12.

A major difference compared to EDX is that a double "fast" ring is used instead of the double bus, but it is not mentioned why a ring is used. The ring connects all the units and access is granted according to a protocol similar to standard ring protocols.

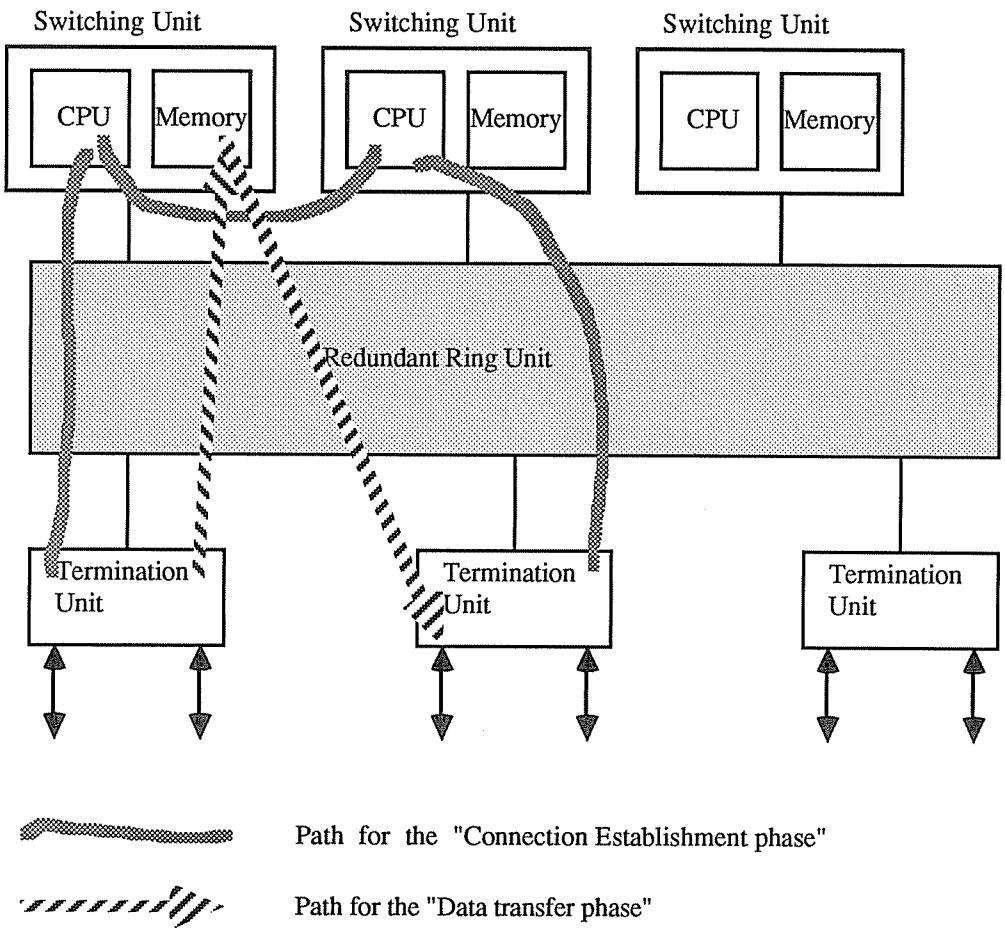


Figure 12. Functional Block diagram of Siemens EWSP switch.

Performance

Packet size: According to X.25, X.28, i.e. max 128 bytes
Number of concurrent connections: max 10000/switch
Protocol complexity: Ordinary complexity, X.25, X.75, X.28

Throughput

For one line: specified to max 128 kbit/sec. For one TU: specified to max 128 kbits/sec. A faster TU for 2 MBit/sec. will be developed.

Delay

There is no specification.

Protocols

CCITTs X.25, X.75, and X.28. There is no discussion on protocol options.

Implementation trade-offs

•*Modularity and SAP accessibility* Only X.25 is reachable.

•*Interface to host machine* This is a pure switch. There is no host connection.

•*Dependability* In order to increase dependability the following measures are used:

- A redundant ring with crosschecking and reverse transmission, so that some failure modes can be tolerated.
- Error correcting codes in memories,
- Alternative ring connections in the units,
- A failing Switching Unit tasks can be distributed to the other units.

•*Implementation language* This is a design study, any plans for implementation language are not available.

•*Buffer handling* A buffer reservation scheme is used for communication between units. Before the transmission of data takes place between the Switching and Termination units, the sending unit is reserves buffers at the receiving unit. The buffers are ".... organized on the wraparound principle".

5.3 Great Britain switches

A design study of a packet switch from Plessey, aiming at a total throughput of up to 1Mbit/sec., is reported in [Rest80]. It is a multiprocessor design consisting of a double bus and up to 16 identical "Link Modules", see figure 13. A Link Module is a rather conventional design built around an X.25 level 1 and 2 chip, a CPU with programs in ROM/RAM and a shared memory, called "Packet memory". The level 3 processing is done by software in the CPU and the organization is reminiscent of the one in ARPAs IMP. A Link Module can process up to 64 kbit/sec. The modules communicate over one of the buses by copying packets from one Packet Memory to the other. The other bus is used to acknowledge the reception of the packet, in order to avoid congestion and buffer overflows at the receiving module.

The performance has been studied by simulation. A single module with 10 links is compared to a system with 10 modules with one link for each of them. An interesting observation is that the system of 10 modules for low load have *longer delay* per link compared to the single module, and shorter delay for higher load. This is explained with that for low load there is no contention for resources neither in the single module nor in the 10 module system, and hence the copying of a packet over the bus is the dominating factor. While for high load, contention in the single node will dominate over the bus transfer. The minimum delay for a single node is about 300 microseconds and for multinode it is about 600 microseconds.

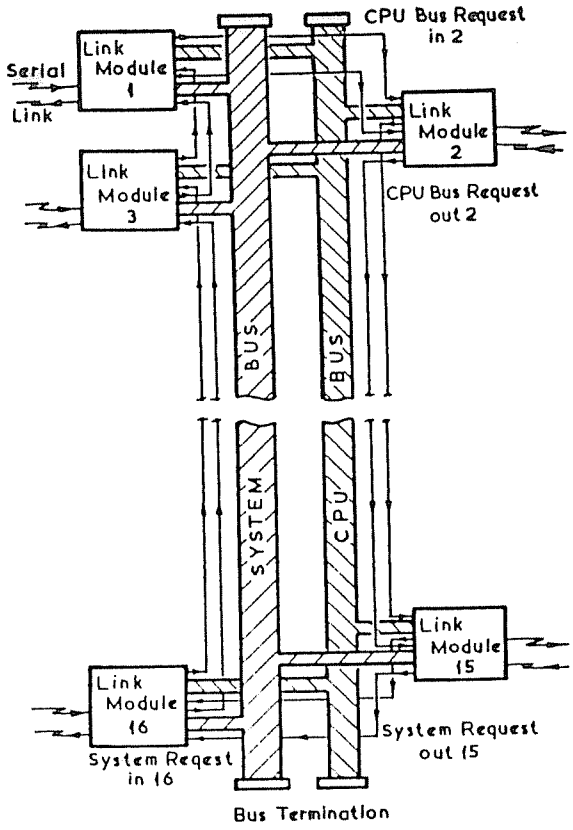


Figure 13. Plessey's node processor. *From [Chil84]*

A design similar to Plesseys is reported on from University of Manchester[Chil84]. The basic organizational differences are a shared memory as opposed to local memories and a triplicated bus that is controlled by a token protocol as opposed to the time multiplexed Plessey bus.

One interesting design is the buffer memory. The shared memory is physically divided into several banks, each one corresponding to an individual X.25 line. Each bank is of the multiport type allowing parallel write and read. This organization will reduce contention, because it allows for parallel access to the banks and parallel read and write to one bank when there is no conflict. Furthermore, a bank is logically divided into "pages" which obviously are corresponding to buffers.

5.3.1 The MININET switch

A switch for very short messages has been developed for the Local Area Network MININET. This non-standard LAN is aiming at high-speed instrumentation applications. Typical for the majority of this type of applications are that the exchanged information is normally less than 16 bits and that short transportation delays are more important than high throughput. Therefore, MININET is operating with only 32 bit messages, including headers, and the switch has been optimized for short delays. The goal is to get a switching delay that is less than 10 microseconds. The switch connects "host connections", to one or several channels, i.e. to one or several MININET LAN:s.

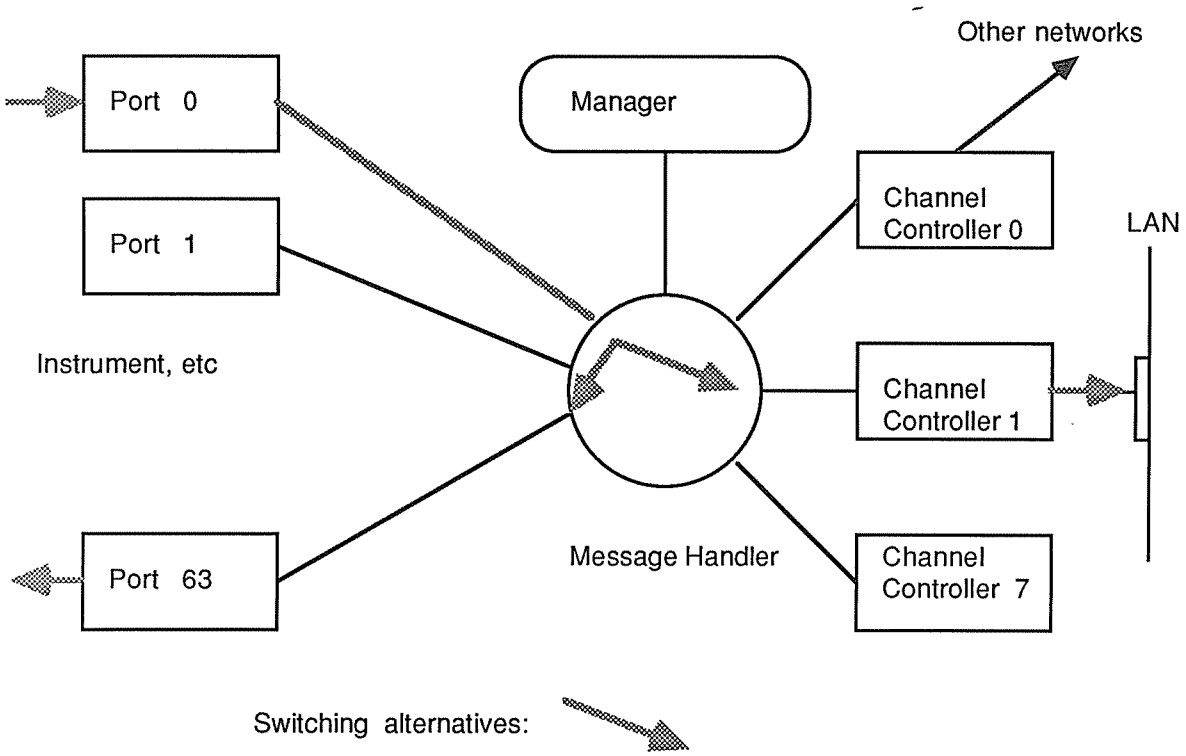


Figure 14. Functional division of the MININET switch. *Adapted from [Morl83]*

It should be noted that the switch only supports the specific MININET protocols, although the service is similar to X.25. The design and implementation were carried out in cooperation between Polytechnic in London, England and University of Bologna, Italy and is mainly reported in [Morl83].

The switch is organized into the following modules; "Ports", e.g. interfaces connecting instruments, a "Message Handler", a "Manager", and "Channels" that interfaces the LANs, see figure 14. The Ports are carrying out functions corresponding to level 1 and 2 of X.25, while the Message Handler is only routing messages from port to port, port to channel or channel to channel. Virtual circuits are set up and cleared by the Manager, which also updates routing tables and monitors the whole switch. The interconnection between the modules consists of a set of four dedicated buses, a Port bus, a Packet Bus, a Channel Bus, and a Management Bus. The path for a normal message from a port to a channel, starts from the Port Bus, over the "Port Bus Controller" to the Packet Bus, over the Channel Bus Transfer Controller, into the Channel Bus, and eventually to the Channel, see figure 15. The Management bus is not involved after that the connection is set up and when everything runs smooth. The access to these buses is controlled by a "Master Arbiter" which is one of the more interesting designs of this switch.

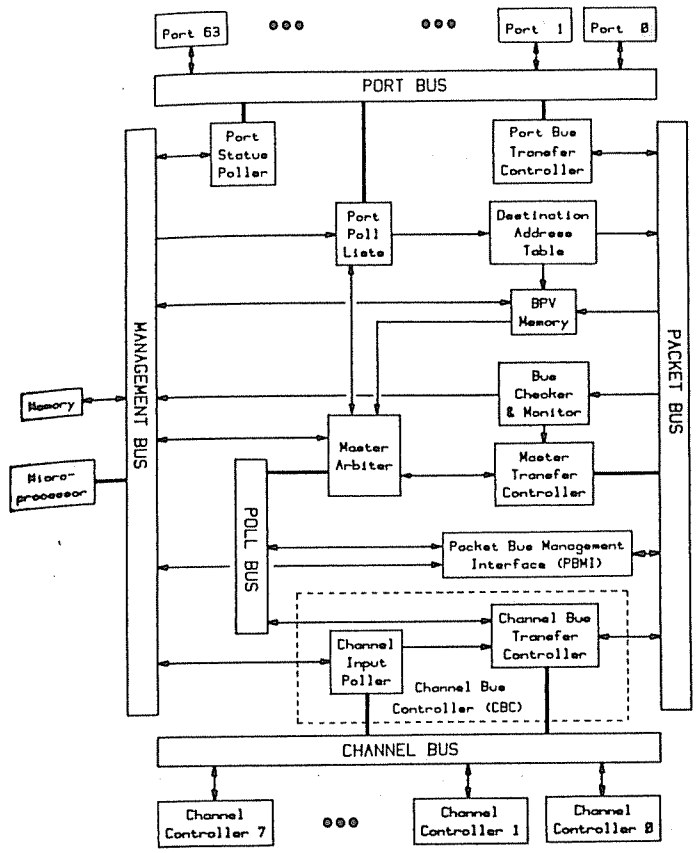


Figure 15. Block structure of a MININET node. *From [Morl83]*

The Master Arbiter is operating in a polling mode. It checks if a port, channel or the manager has some messages to be send. When it detects a message, the Arbiter first checks if the receiving part has enough buffers and it also checks routing and flow control information to determine if the message should be forwarded. These checkings are done in order to avoid buffer overflows. There are two main problems with polling schemes: to ensure that all sources can be polled within the required time limit, and to ensure fairness between the sources. The elaborate polling scheme is claimed to consider these problems[Morl83].

The Arbiter is implemented as a microprogrammed Finite State Machine and the poll time for 10 ports and 2 channels is estimated to 3.5 microseconds. Note, for the estimated peak load, 100 000 packets/second on 10 channels, the packet processing time must be less than 1 microsecond!

In order to minimize delay, the Message handler controllers are implemented either with microprogrammed Finite State Machines(i.e. Port Status Controller, PBMI queue controller Channel Bus Controller) or with hardwired synchronous techniques(i.e. Channel input poller, MTC, Port Bus transfer controller). Also, very few and short queues are used in order to gain performance.

Another interesting design is the dedicated operating system in the Manager unit. The task communication of the operating system is implemented with "event queues" that hold messages, time-outs, and other structured communication items. A task can be scheduled to wait for an message to arrive to an empty queue.

Performance

Packet size: 32 bits, 16 bits user data field
Number of concurrent connections: At most 10 channels
Protocol complexity: MININET protocols, considered less complex than e.g. X.25. Provides services similar to X.25.

•*Throughput*

Estimated maximum is 100 000 packets a channel/sec., i.e. more than 25 Mbits/sec. Measured is 700 000 packets/sec., locally from port to port.

•*Delay*

From port to channel, estimated average delay is less than 10 microseconds.

Protocols

Three layers of the MININET architecture.

Implementation trade-offs

•*Modularity and service access point accessibility* The highest layer is only accessable. The number of Ports and Channels can vary.

•*Interface to host machine* The connection is through an "Port", e.g. IEC-625 port and speech ports. (It is unclear whether the latter one is under implementation or not.)

•*Dependability* Reliability is not a major concern for this switch. Parity is used and tests are run during initialization. Some "watchdog" timers are used to avoid buffer

"hugging".

•*Implementation language* The Message handler is implemented with sequential circuits and microprogrammed controllers. The operating system kernel is written in assembler language while the rest is written in Pascal.

•*Buffer handling* In the Message handler there are very few buffers in order to decrease delays. The few that exists are implemented in hardware. The Master arbiter ensures in its polling sequence that buffers are available at a receiving port, channel, or manager.

•*State information and context switch* State information about port and channel connections, i.e. routing path as well as available buffers at the receiving switch, is stored in dedicated registers accessable to the Master arbiter. The polling cycle of the Arbiter identifies the port or channel and does the context switch by setting a register pointer to the information of the current connection. All other state information, like routing tables, are stored in the Manager Memory. The context switch in the operating system of the Manager seems to be a regular task switch.

5.4 PHILIPs Development of Bells 1PSS

In cooperation with AT&T Bell, PHILIPs has further developed the 1PSS switch for the European market, aiming at the medium to large X.25/X.75 packet networks" and for a future integration into a ISDN switch[Deus86]. Bells latest ring based version of 1PSS, aimed for the US market, is used for the development. The ring version is a development of the old star shaped 1PSS switch, capable to handle many more connections.

This European version is adjusted to X.25/X.75 protocols, but these protocols are *not* proposed for communication between the switches themselves. Instead AT&Ts internal protocol is supposed to be used, i.e. dedicated lines are needed. Also a network control center is needed for their operation.

The switch is organized into three basic building blocks, see figure 16:

- the "Packet Administrative Module" (PAM): An AT&T 3B20D computer.
- the "Packet Switching Module" (PSM): a fast duplex token ring(32 Mbit/sec.) with "Ring Peripheral Controllers" (RPC) as interfaces to the PAM and with "Packet Switching Units" (PSU) as dedicated front-ends for X.25, X.75, Internal Protocol, and possible other protocols,

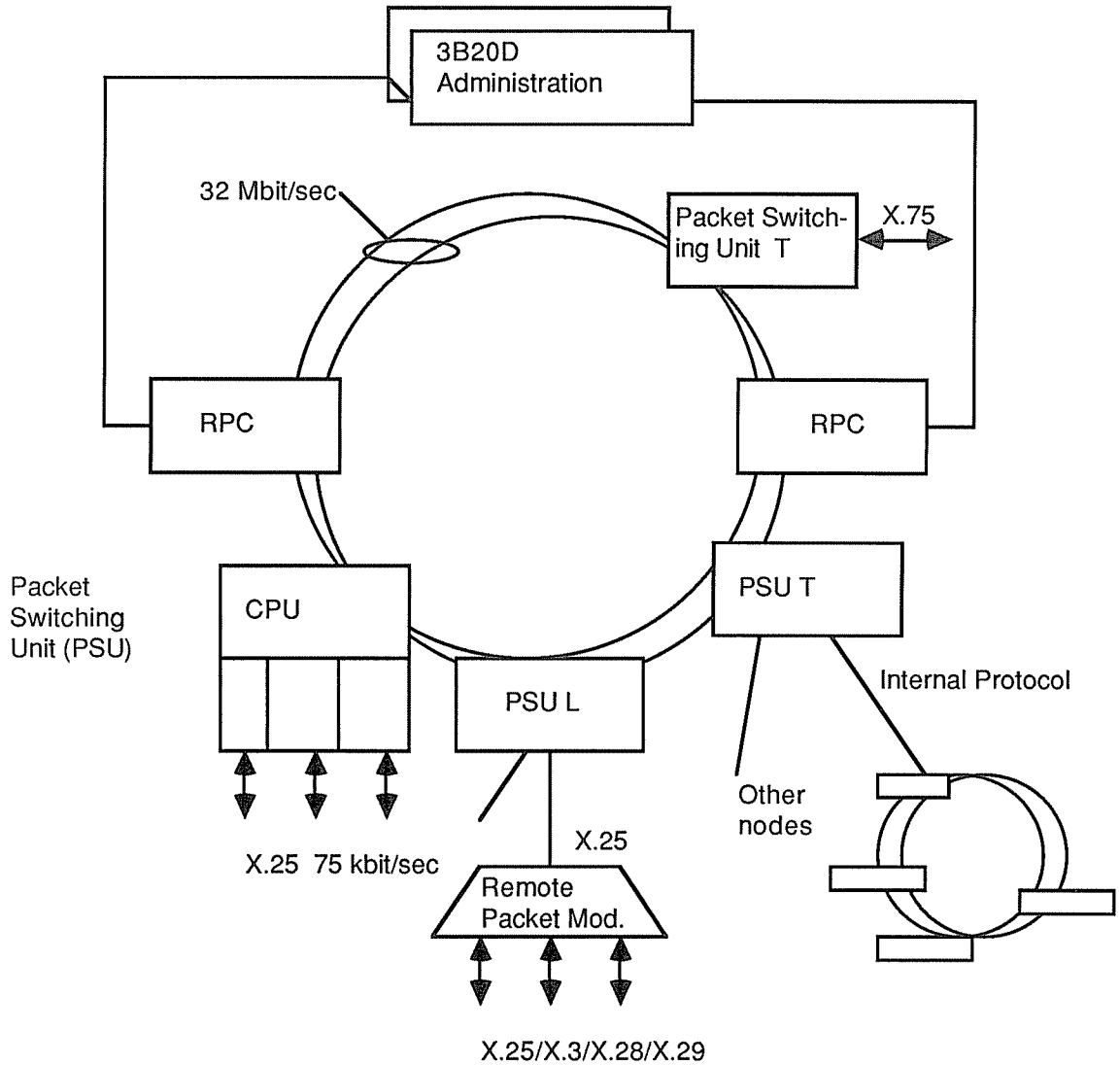


Figure 16. Hardware structure of PHILIPs/Bells switch.

- the "Remote Packet Module" (RPM): for protocol conversion between X.25 and IBM synchronous protocols and concentration of low speed X.25 connections.

An interesting approach is that all modules run the same real-time version of the operating system UNIX™. All modules have the UNIX RTR/RTX kernel and interprocess message system. The message system is providing an inter*processor* communication function, simplifying communication within the switch. Depending on module type, additional different system software is used, like the "Operations and Administrations Software" for The Packet Adminstrative Module, and the "Transport Software" for the Packet Switching Units.

The X.25 software is resident in the Packet Switching Units. Level 1 and 2 are implemented in hardware and level 3 in software, i.e. in the Transport Software. The PSU themselves communicate over the ring via the interprocess system, i.e. with memory-to-memory communication.

Performance

Packet size: Sizes are according to X.25 and X.28. Estimates are based on an 64 byte packet average.
Number of concurrent connections: With 500 PSUs and 200 RPM up to 10000 "subscribers" are estimated to be able to connect.
Protocol complexity: Ordinary complexity, X.25, X.75, X.28, and Internal protocol.

•*Throughput*

For one Packet Switching Unit: 150 packets/sec., i.e. 76 kbit/sec. and 50 call set-up/sec. For 500 PSU and 200 RPM the switch is estimated to be able to handle 40 000 packets/sec., i.e. over 20 Mbit/sec. 600 to 800 call set-up will then be supported.

•*Delay*

There is no estimate on delay.

Protocols

CCITTs X.25, X.75, X.28, X.32 and AT&Ts Internal Protocol are supported. X.25 level 3 provides SVC and PVC, normal and restricted fast select, CUG facilities, flow control and throughput parameter negotiation and others, see [Deus86].

Implementation trade-offs

•*Interface to host machine* This is a switch. There is no host connection.

•*Dependability* There is a strong emphasize on reliability. At the bottom line, AT&Ts fault-tolerant series 3B20D is used, with duplicated processor, disks, and links. Furthermore, the ring is duplicated for redundancy. The switch is designed to be maintained during normal

operation. However, there does not seem to be any measures taken to recover from a PSU failure.

•*Implementation language* This is a design study, any plans for choice of implementation language is not available.

•*Buffer handling* Buffer handling seems to be of no concern. It can maybe be speculated that the common interprocessor software might take care of the buffer handling.

•*State information and context switch* These problems are not mentioned. It can also be speculated that these issues are implemented as a part of the operating system and hence not given any special attention.

6. Local Area Network Controllers

6.1 Ethernet Controllers

The Ethernet controller is probably the most common Local Area Network controller. Many manufactures have controllers available and when comparing them it is clear how different the same protocols can be implemented. Some of them are implemented with dedicated chips, some with dedicated "off-the-shelf" microprocessors, while some use regular TTL interfaces to specific hosts. We will not here list specific products, instead we will describe and discuss the different trade-offs between some implementations, using some representative samples. We hope to demonstrate where the bottlenecks are located in the implementations, and what kind of performance that can be expected.

Taylor, Oster, and Green[Tayl83] compared in 1983 four controllers of varying sophistication; LANCE from AMD/Mostek Motorola, i82586 from Intel, the EDLC from SEEQ, and the MB611301 from Ungermann-Bass/Fujitsu. Basically, all of them use temporary FIFO buffers in the controller to hold both packets awaiting to be send as well as received packets. All of them share some some kind of buffer memory with the host, and all of them have registers for pointers to buffers. All of them provide the Physical layer functions within the controller. This is just about what is common. The more advanced, i82586 and LANCE have Link layer functions resident in the controller, e.g. the back-off algorithm, while the other rely on the host to provide these functions.

The memory management design is crucial for high performance. It should not only feed or read the controller with the maximum Ethernet bitrate, but also to ensure that neither the CPU nor the system bus will get completely busy, effectively blocking all other processing. The above mentioned controllers use different memory managements which are classified below and it is clear that the overall performance is heavily depending on whether *the memory* is:

- *local to the controller,* Then it is likely to match the Ethernet speed requirements (at least 1.25 Mbyte/sec.). A dual port memory, with one port to the host and one to the controller, is a straightforward approach. Still, the controller may generate many interrupts that might flood the host if it generates interrupts for each message (to check the address), for calculating the back-off algorithm, to indicate messages available, to reject runt messages, and others. Possible frequency of interrupts is illustrated with that a new message could arrive within 10 microseconds from the previous one. Of course, many of these interrupt generating tasks are handled in the more sophisticated controllers.

- *shared with a dedicated LAN processor,* The memory is sharing an I/O-bus with a dedicated processor, a controller, and other devices. Even if the bus could match the speed requirements, it must be designed so that it will not block other processor activities. Therefore it is not likely that the controller will own the bus for a whole long message and it is therefore necessary to buffer packets in the controller. The size and number of these buffers will depend mostly on message burst size, frequency, and characteristics of other activities. But with big buffer space follows that the message delay will increase. On the positive side, a dedicated

processor will off-load the host many of the real-time tasks. Furthermore, the processor can serve the host with data adjusted to the hosts normal memory management system and without requiring immediate response from host when a message arrives. The buffer management scheme of the memory can thus be tailored to the controller since the host does not need to access it. And at last, with a dedicated processor many high level protocols can be moved from the host into the processor, off-loading the host even more.

- *accessed over a shared bus.* This is the simpliest design. The controller works as a regular DMA-device. Even if the host memory can match the speed there is a high risk of bus hugging when a long message has arrived, blocking other devices, including the host. The risk of flooding the host also occurs here. The advantages are the simple controller design and that there is a lower frequency of data copying. However, the Ethernet protocol complexity is moved to the host instead, probably off-setting the gain of using a simple controller.

The buffer management also has a profound effect on performance. Buffers are normally chained together and several of them are needed for a message. Crucial is the number of memory accesses needed to find the next buffer. Normally, each buffer has a descriptor, indicating where the buffer is, its length, status, etc. Some controllers, like LANCE, use linked lists of buffer descriptors while others directly use chained buffers. Important for the performance is if the host operating system needs to update, insert, and remove buffers. Another important consideration is the size of a buffer. Big *fixed sized* buffers have less overhead, but might waste more memory space than dynamic small buffers. At last, the number of buffers must match the speed characteristics, the burst distribution, host occupancy, and other dynamic factors.

6.1.1 Performance Evaluation of two controllers

A comparison of two controllers is reported in [Minn83]. The report is interesting because it compares the performance the user perceives from both a *process-to-process* and a *file transfer* connection using these two controllers. The measurements are done on identical PDP-11 11 machines with the same high level protocols, i.e. the ARPA TCP/IP and FTP. It is possible to draw conclusions about the two different memory management schemes with respect to user performance.

The two controller candidates are 3Com Corporation's 3C300 UNIBUS Ethernet Controller and Interlan Inc's NI1010 Ethernet Communications Controller.

The Interlan controller uses DMA to access the PDP-11:s main memory over the UNIBUS. Interrupts are generated on packet reception and packet transmission completion and at the end of each DMA. The back-off algorithm is performed in the controller and hence will not cause any interrupts to be generated. The controller accepts chained main memory buffers and the PDP sets the controller registers for buffer address and length. The controller also has some hardware FIFO buffers for temporary storage, see figure 17.

The 3Com controller uses a local dual port memory in the controller instead. The memory is organized into 16 fixed sized buffers of 2 K bytes each, see figure 18. The controller has registers that are used to point at buffers in the memory. At sending, the PDP sets these registers, pointing at

the buffers to be sent and in receiving mode they are set by the controller and read by the PDP. The FIFO buffer is used to hold the pointers to packets awaiting transmission completion. Interrupts are generated on packet reception and packet transmission completion, and on detection of Ethernet jam (collision). Thus, the back-off algorithm is processed in the host.

The measurements taken on these two controllers include throughput on the bare controller, when data is in main memory, throughput using TCP/IP and FTP on otherwise empty machines, and the response time, i.e. the time it takes to send from controller to controller and back again. The measurements were done on a 10 Mbit Ethernet.

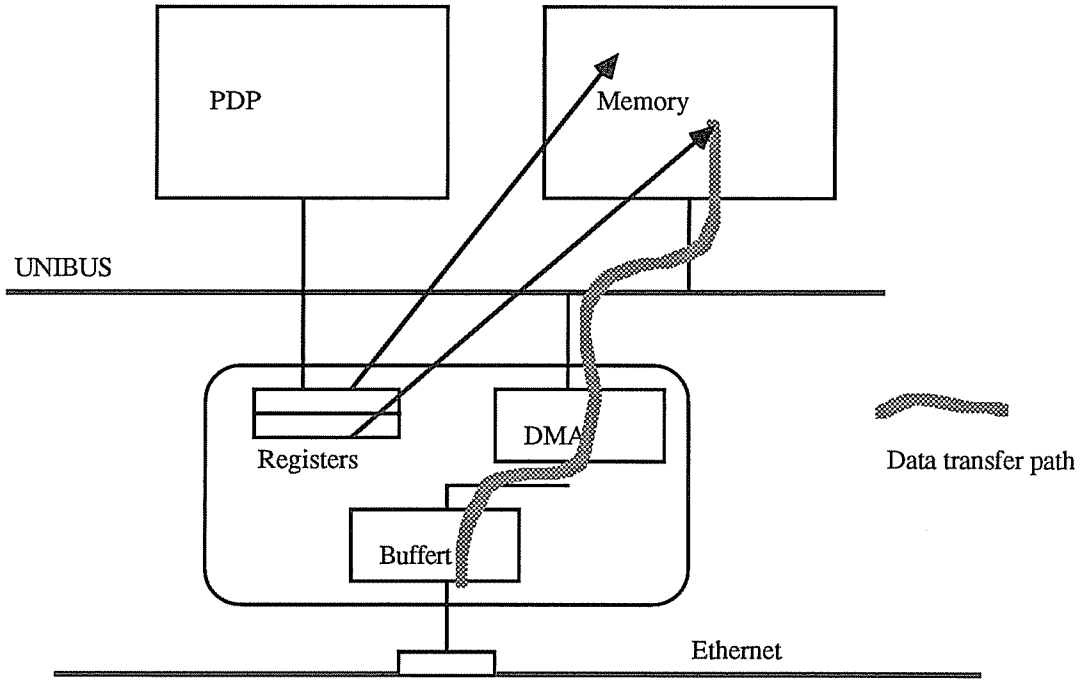


Figure 17. The Interlan Ethernet Communications Controller.

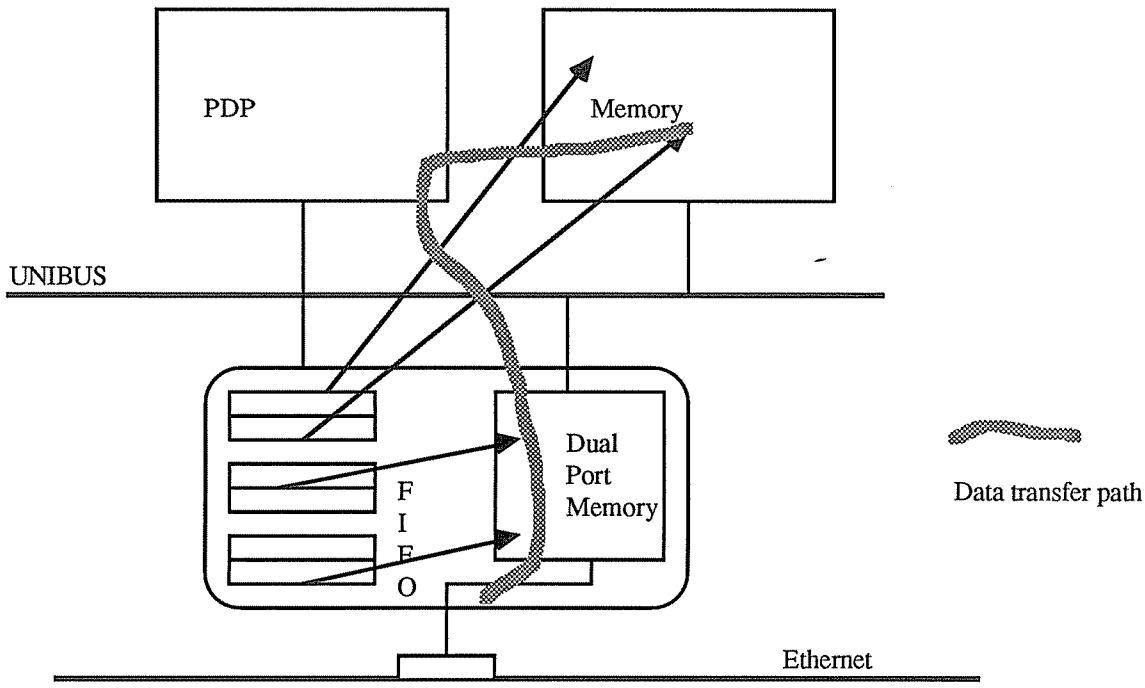


Figure 18. The 3Com UNIBUS Ethernet Controller.

Performance

Throughput on bare machines:

Packet size: Several sizes were used and the result is presented as a diagram, see figure 19.

Number of concurrent connections: One

Protocol complexity: Ethernet

Throughput: For data *stored in the main memory,* see figure 19. For 1000 bytes length the performance is about 200 packets/sec., i.e. 1.6 Mbit/sec., for both controllers. This should be compared to earlier results when the data was placed directly into the 3Com controllers memory and then the throughput was almost 10 Mbit /sec, which shows that copying data over the UNIBUS indeed is a limiting factor. Interesting is that 3Com has better performance for small packets and Interlan better for longer packets. This is explained with that Interlan has significantly longer start-up time for the DMA process compared to the 3Com transfer, but when Interlan is started up it has shorter copy time per byte(2.5 microsecond compared to 6.4 microsecond).

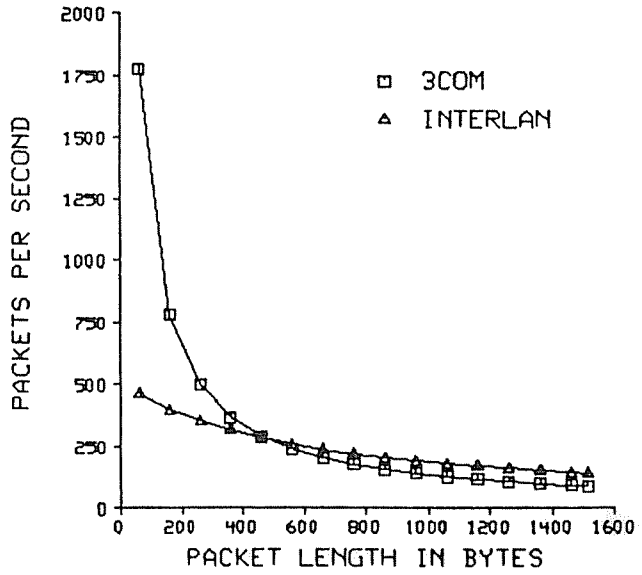


Figure 19. (Modified) Packet Transmit Test. *From [Minn83].*

Throughput over TCP/IP

Packet size: The stream oriented TCP was used for the transfer of one megabyte.

Number of concurrent connections: The measurements is done from a writer to a reader.

Protocol complexity: TCP/IP using Ethernet. High protocol complexity.

Throughput: The throughput is summarized in the table below. From the table it is clear that both controllers perform just about the same. From a user point of view the throughput is only 3.5% of Ethernet capacity while the communication is occupying between 55 and 90 % of its total capacity of the PDP-11 CPU. It is claimed that the controllers achieved the same (poor?) performance *due to different reasons*. The performance loss for the Interlan is believed to be caused by excessive interrupt overhead while the loss for the 3Com is believed to be caused

by the copying from memory to the dual port shared memory in the controller. In contrast to the Interlan, the 3Com controller requires that the CPU does this copying. The many interrupts of the Interlan controller are due to the internals of TCP that uses a 128 byte buffering scheme. For each buffer a new DMA operation must be started. Another observation is that the testprogram is neither negligible nor dominating.

		3Com		Interlan	
Performance Metric	Units	Writer	Reader	Writer	Reader
Throughput	Kbits/sec	355	355	365	365
Interrupt Load	int/sec	100	95	550	545
TCP/IP CPU usage	percent	70	45	60	30
Test prog. CPU usage	percent	13	25	22	25
Total CPU usage	percent	90	75	90	55

Table. Process-to-process test results. *From [Minn83]*

Throughput over FTP

Packet size: One megabyte is transferred.

Number of concurrent connections: One connection for transfer of a file from disk to a disk at another computer.

Protocol complexity: FTP using TCP/IP/Ethernet. Very high complexity.

Throughput: The throughput is summarized in the table below.

		3Com		Interlan	
Performance Metric	Units	Writer	Reader	Writer	Reader
Throughput	Kbits/sec	124	124	145	145
Interrupt Load	int/sec	90	75	240	220
TCP/IP CPU usage	percent	10	10	10	10
Test prog. CPU usage	percent	50	70	65	75
Total CPU usage	percent	98	95	95	92

Table. FTP test results. *From [Minn83]*

One direct observation is that performance has dropped to less than 1.5% of the Ethernet rate and that the CPU usage is close to 100%. The increased CPU usage is due to the more computational intensive file tasks. It is clear that contention occur to some resources, in this case to the UNIBUS, since the disk controller, the Ethernet controller, and the processor compete for the bus. By later reconfiguring the PDP with a MASSBUS instead, the throughput was increased with 77% and by writing optimized assemble code for copying from main memory into the 3Com controllers shared memory an additional 32% speed up was achieved.

•*Delay*

Packet size: Variable size packets, see figure 20 below.

Number of concurrent connections: One connection back and forth.

Protocol complexity: Ethernet.

Delay: Delay is measured as the response time, i.e. the time from a packet is sent and until it is received back from the other controller. At the start, the packet is placed into the main memory, for both controllers. For a 1000 bytes packet the response time is about 12 msec. Not more than 2 msec was spend on the cable. The slightly longer times for Interlan is explained with that the controllers must do two memory reads (one at each end) while it is enough with one for the 3Com. See figure 19.

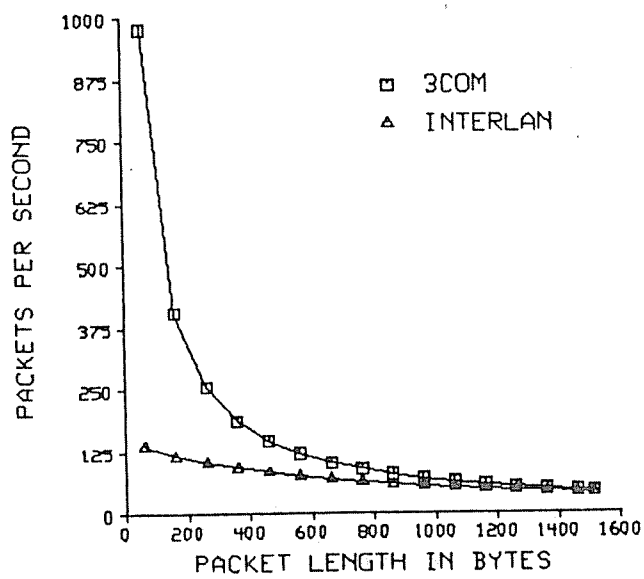


Figure 20. Packet Echo test. *From [Minn83]*

Protocols

•*Protocol hierarchies supported*Both controllers support Ethernet. The Interlan controller also provides the back-off algorithm.

•*Protocol options supported* See above.

•*Modularity and SAP accessibility* Only the Ethernet service interfaces seems to be accessible.

•*Interface to host machine* 3Com uses a dual port shared memory in the controller and registers for control and addresses. Interlan uses DMA to the main memory and registers for control and addresses.

•*Dependability* Not discussed at all.

•*Buffer handling* Buffer chaining is used in the main memory for Interlan. For 3Com 16 dedicated buffers are used together with a table indicating used and free buffers.

•*State information and context switch* Both support only one cable connection. Both rely on fast host context switch in order to handle frequent interrupts.

6.2 Intel Innovative LAN products

During the last years Intel has announced several Local Area Network controllers. The announced products are ranging from complete multiboard computer systems, to firmware for processors, and to dedicated chips. In terms of chips we will discuss two of them; i82586 and i82588[586-84,588-84]. Both chips are complete Local Area Network controllers. i82586 is the more advanced chip aiming for IEEE 802.3/Ethernet with up to 10 Mbit/sec., while i82588 is a more "cost sensitive" chip aiming at "personal computer networks" with a speed up to 2 Mbit/sec. The i82586 also can support these lower speed networks.

Intel has also announced the "iNA 960/961 Network Software"[iNA-84] package that provides ISO Internet and Transport, Class 4, Protocols. The software runs on an Intel 16 bit microprocessor and it is adjusted to the controller i82586. When this package is combined with a Token Bus controller, Token Bus software (802.4) , and MAP software for layers 5 thru 7, the whole system is called 554 MAP Communications Engine[554-85]. The 554 system is not yet released, only advanced information is available.

What is interesting from our point of view is how Intel has distributed the different layer implementations, and what the trade-offs are.

6.2.1 The i82586 Ethernet Controller

Although the chip supports other CSMA/CD protocols we will only consider the Ethernet functions. It is clear that the designers of the chip tried hard to off-load the host as many communication tasks as possible. Therefore, the chip is supporting a shared dual port memory organization. It has on the chip memory management, the back-off algorithm, as well as extensive collection of statistics. All these functions are controlled by a "Command Unit" that executes commands. The shared memory is reached over a system bus, not necessary an Intel Multibus, using a built in DMA controller. The organisation is depicted in figure 21.

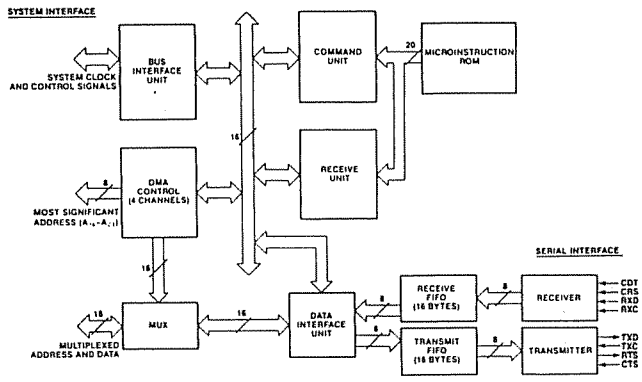


Figure 21. 82586 Functional block diagram. *From [586-84]*

The most innovative feature is maybe the buffer management which is controlled by the chip. The host does not even need to keep track of free buffers, it only needs to; read pointers to free buffers, to recently received buffers, when completely out of buffers, and to initialize buffer areas.

When the host wants to send a frame it generates an "attention signal" to the controller. The controller respond with reading the "command word" in a predefined memory location that points to the packet buffer. Similarly, when a packet is received by the controller, it generates an interrupt to the host, which responds by reading the "status word" in memory, holding a pointer to the packet buffer. Of course, the command and status words also contain other information, since this is a convenient way to inform each other about operational status, statistics, dead receiveres, etc.

The buffer size is set by the user. Many buffers are likely to be used for long frames (maximum 1518 bytes) and these buffers are ordered into linked lists. For each buffer it is associated a "Buffer Descriptor" holding pointers to the actual buffer and to the next linked descriptor. For each *received frame* there is an additional descriptor that has information about the address and type.

The frames to be sent are also organized into linked list of descriptors of buffers. But there is no frame descriptor. Instead, for each frame there is associated a set of *commands*. These commands are interpreted by the "Command Unit" in the controller, and such a command can be for example, "Transmit", or "Set-up multicast address", etc. This is an elegant way to tailor the code for each frame, thereby off-loading the host and providing flexibility. On the other hand, this organisation is causing another indirect step in the code that might prolong the transmission delay. In figure 22 the buffers and the "commands" are called "Command Lists".

Since this is based on a preliminary information there is no performance data available.

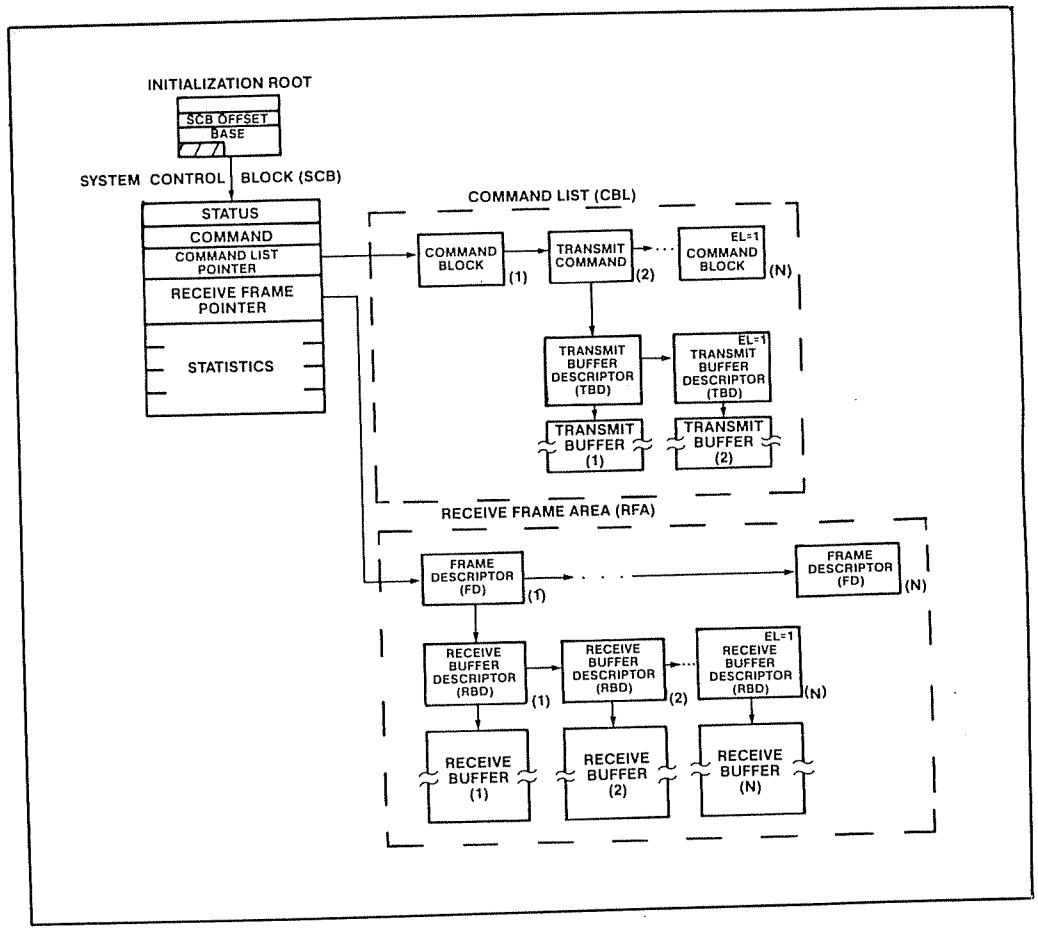


Figure 22. i82586 Shared memory structure. *From [586-84]*

6.2.2 The 554 MAP Communication Engine

The 554 board is aiming at the MAP market as a front-end, and it will eventually (since this information is only advance information) support all layers of MAP. There are no time plans, but for layers 1 thru 4 the existing iNA 960 software will be modified and a Token Bus Handler will be used. For layers 5 thru 7, "Intel will also provide implementation of the MAP layers 5 through 7 as a product" to be run on the same board. The envisioned different product implementations of all 7 layers are illustrated in figure 23.

The board consists of an Intel 80186 processor, a set of Token Bus Handler chips, Multibus interfaces, and Memories, and others see figure 24. This means that the host must accept the Multibus interface and to communicate over a "flag port" in the interface. Both the host and the i80186 can interrupt each other over the port. Furthermore, the host must follow the "Multibus Interface Protocol" to be able to reach the 554 protocols.

There are no estimates about performance for this set up. A comparable set up, with iNA 960, 80186 and the LAN controller 82586 has been estimated to have a "typical" throughput over the Transport layer ranging from 100 K to 300 K bytes/sec., depending on options[554-85].

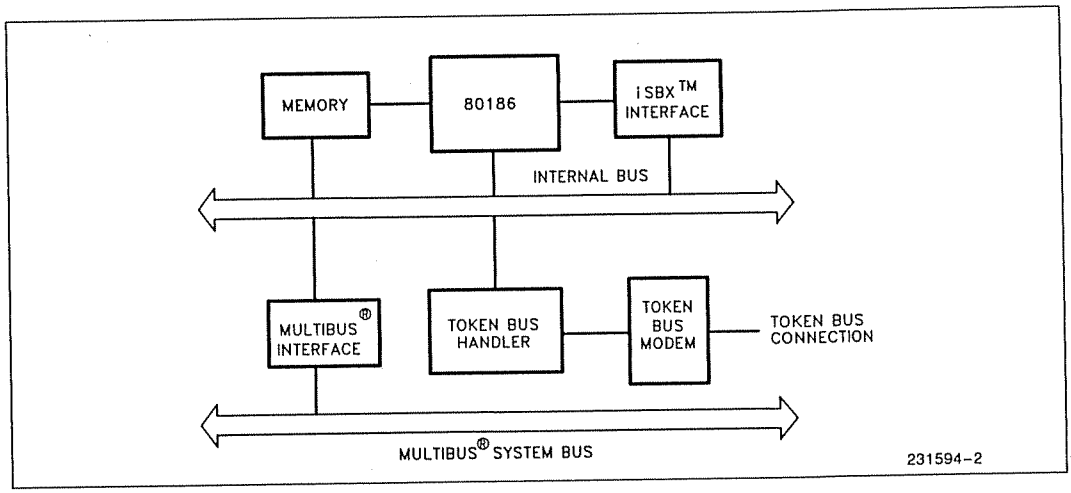


Figure 23. The iSXM 554 hardware structure *From [554-85]*

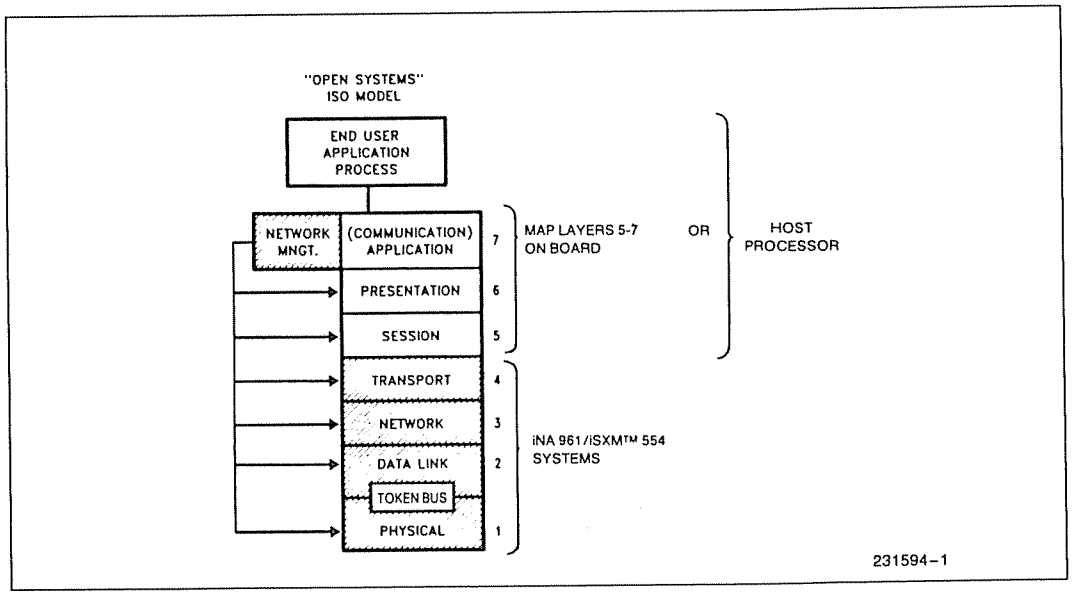


Figure 24. The iNA 961 software. *From [554-85]*

6.3 Motorola innovative LAN products

Motorola has announced communication products during the last years, including an X.25 controller and a complete MAP 2.1 front-end board[MCMA85]. We will here focus on this board called the "Advanced MAP Interface Controller", (MVME372) and its components (see figure 25). It implements all seven layers and it is interesting to analyse how the layers are distributed over the components. The major components are a Token Bus Controller, a MC68020 MPU, a memory, and MAP software. The Token Bus Controller chip implements the Physical Layer and the full Media Access Control functions of the Data Link Layer, while the MC68020 handles the remaining Logical Link Control functions of the Data Link Layer and the layers 3 through 7, see figure 26. Since it is not feasible or sometimes not possible to implement all parts of the layer 7 protocols outside the host, due to interaction with the host's filesystem, some protocol functions in Motorolas design resides in the host. This means that an implementer must adjust his host to what the board provides. In order to ease this adjustment Motorola provides its own "Buffered Pipe Protocol" and "Buffered Pipe Control Interface Routines" which interfaces the layers on the board. If a Motorola host is used, these protocols are already available as a part of the operating system. If another host is used these protocols must be implemented.

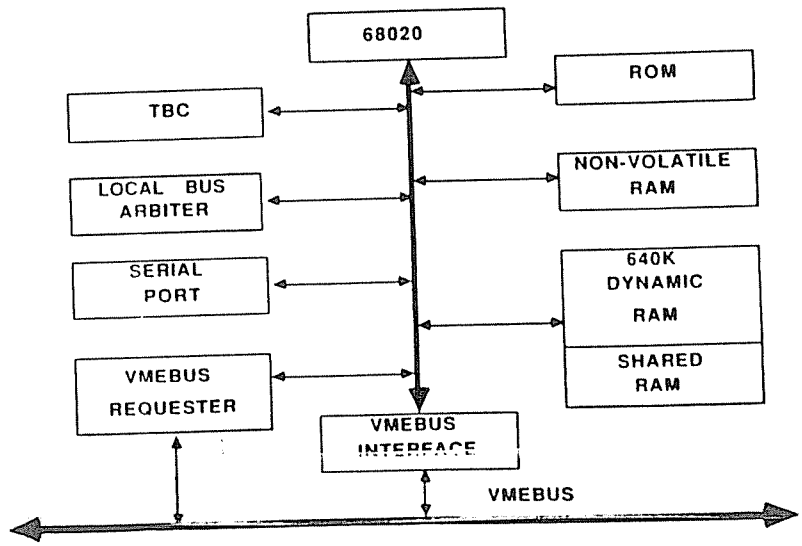


Figure 25. Functional Block diagram of MVME 372. *From [MCMA85]*

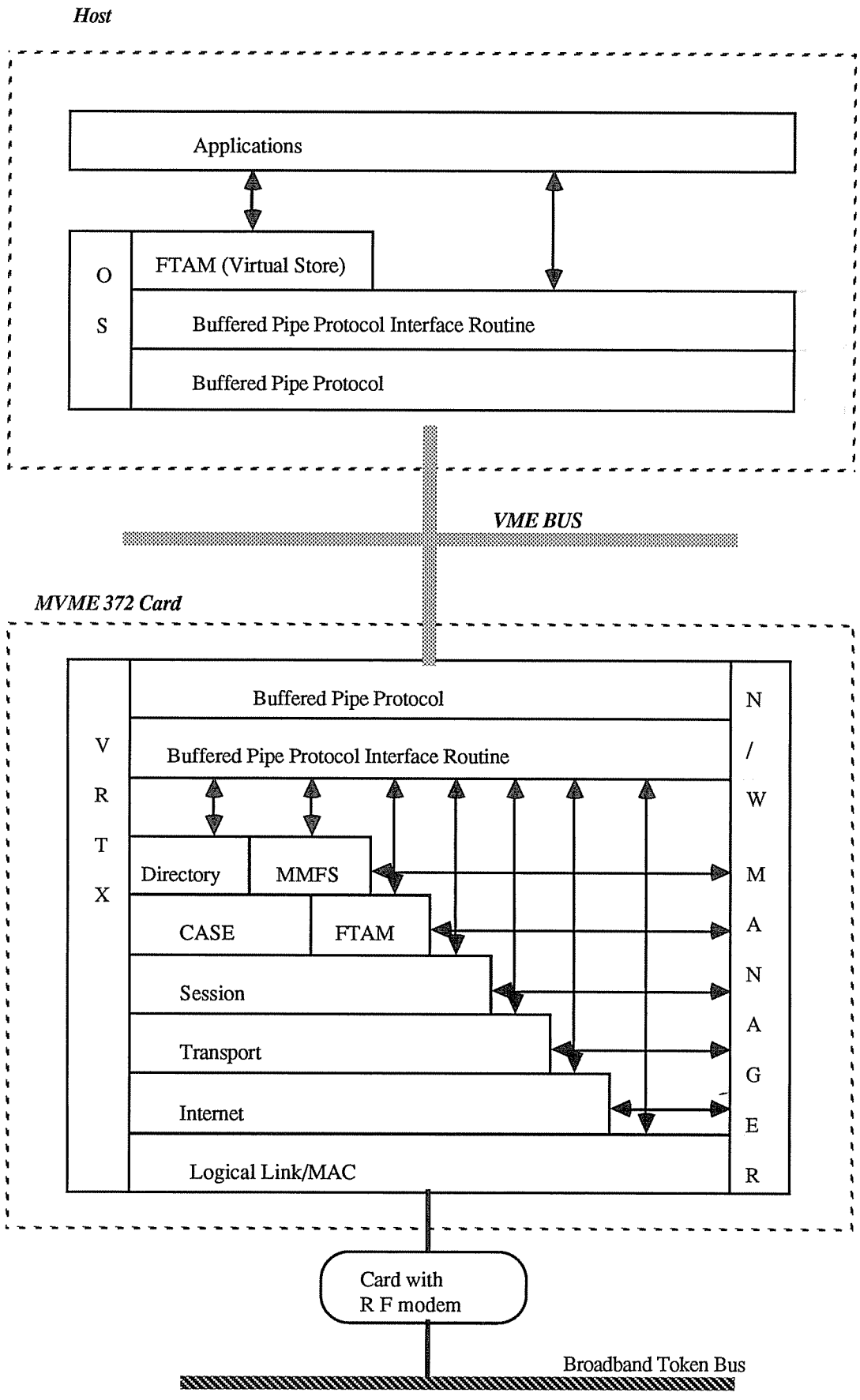


Figure 26. Motorolas implementation of MAP 2.1

6.3.1 MC 68824 Token Bus Controller

The Token Bus Controller is a new VLSI chip released late 1985 from Motorola. It handles the media access control portion of IEEE 802.4. Internally the chip's design is based on a microcontroller that manages a four channel DMA controller, a 40 byte FIFO, an ALU, registers, and other support circuits, see figure 27. As with Intels controllers, 68824 communicate with the local processor (MC68020) with command registers, frame descriptors, and an interrupt vector. The FIFO is used to allow overlap between local bus accesses and the transmission or receiving of frames. The local microprocessor sends commands direct to the controller, such as initialization commands, transmit, and clear interrupt status, as opposed to Intels philosophy of using commands stored in the frame descriptor. Otherwise, Motorola and Intel use similar buffer structures, with frame and buffer descriptors, see figure 28. Intels controller is handling the buffer management. This is not the case for Motorolas controller where the buffer handling is based on a cooperation with the host.

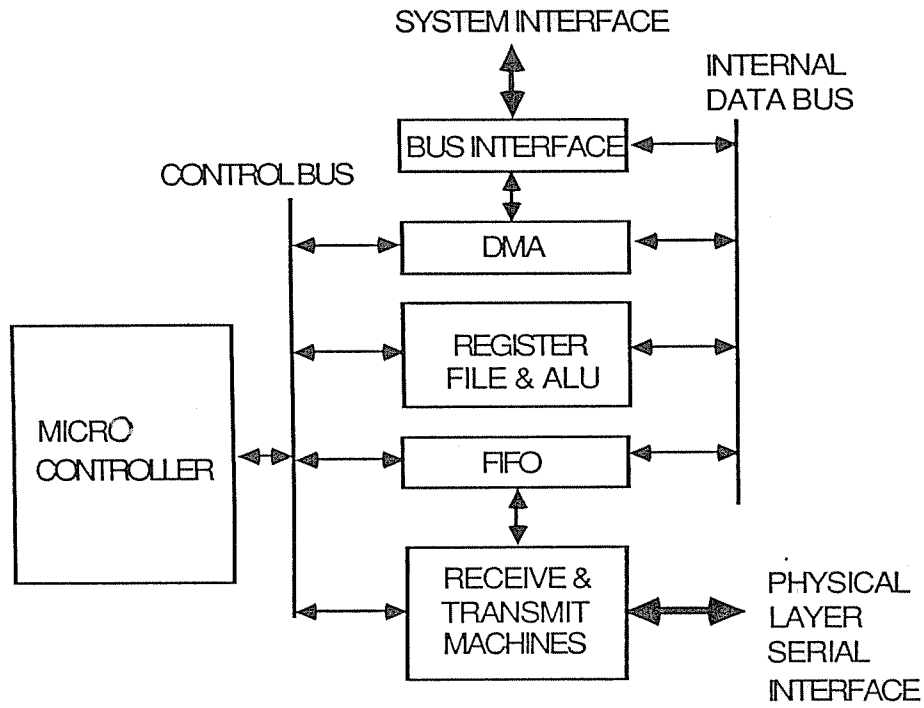


Figure 25. Functional Block description of the Token Bus Controller. *From [TOKE85]*

6.3.2 The MicroMAP Software

This Motorola software implements GM:s MAP 2.1 specification and is announced to be available third quarter of 1986. It complements the Token Bus controller but can also be used independently because it will be available in source code, i.e. in "C". The layers 3 thru 7 are implemented as tasks under VRTX, the real time, multi-tasking kernel for MC680xx.

Communication is accomplish by a VRTX "post and pend", a mailbox type of message service. The layer interfaces are accessable with two blocks; the "Event Parameter Block" and the "Buffer Descriptor Block". When two layers pass messages to each other, an Event Parameter Block is used for primitives and parameters while the Buffer Descriptor Block is used for pointers to the user

data.

All protocols are implemented with a task for each layer in order to promote independent implementations. Furthermore, within a task the protocols are implemented as "finite state machines", that also "should allow changes and additions". For multiple connections, e.g. in the Transport Protocol, there is a finite state machine instance for each of them.

The FTAM implementation highlights an interesting implementation restriction when providing application protocols outside the host. FTAM is divided into the "FTAM Protocol Machine" and "FTAM Virtual Filestore". Motorola has chosen to incorporate only the FTAM protocol machine within the MicroMAP because the FTAM Virtual Filestore maps the hosts real file system to the onto the virtual filestore. Therefore, they are providing the FTAM Virtual Filestore separately, written for a UNIX host(SYSTEM V/68), see figure 26.

Since this is based on a preliminary information there is no performance data available.

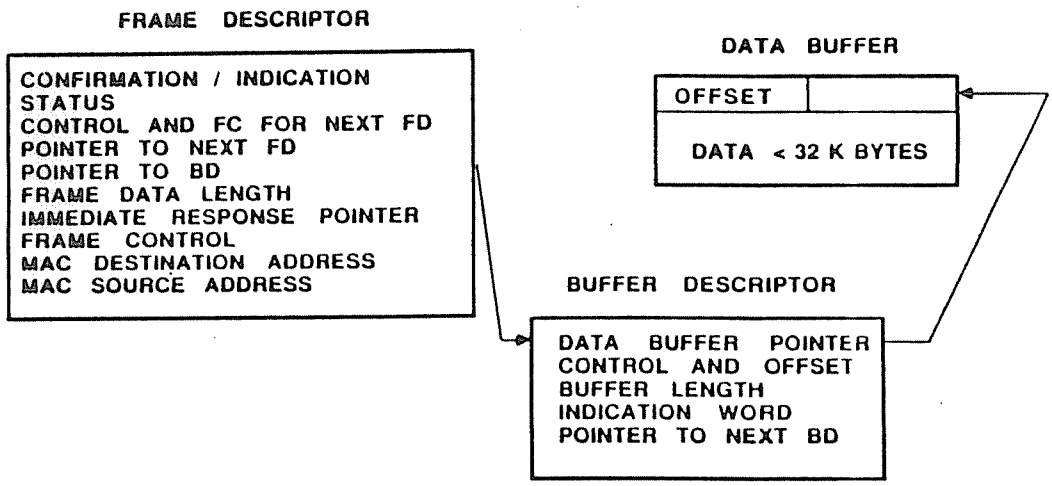


Figure 28. The linked buffer structure of MC 68824. *From [MCMA885]*

7. Discussion and Conclusions

The surveyed processors have been divided into two categories, switches for Wide Area Networks and controllers for Local Area Networks. Both are designed to execute layered protocols but with somehow different performance goals and completely different organisations. Typical for the switches are that they support many network connections and that they provide a packet switching service up to and including the Network Layer, while the controllers normally only serve one connection to the network, providing one or all seven layers of the OSI-model. Furthermore, the surveyed switches optimizes overall throughput by switching many low speed connections, typically around 1000 with less than 48 kbit/sec. each, while the controllers have one fast connection, typically 10 Mbit/sec., that requires fast receiving and handling of packets.

Switches

The survey switches are multiprocessor systems with one exception, the original ARPA IMP. Several connections are normally assigned to each processor according to a static partitioning, also with one exception, the ARPA Pluribus, that *dynamically* assigns processors to the connections. Typical for such a normal system is that it uses X.25 with level 1 and 2 implemented as chips while level 3 is executed with software in the processors. The weak point with these systems is the packet transfers between processors. Two approaches are used; a shared global memory or local memories with "memory-to-memory copy" of packets. More specifically, the weak point is that contention occurs either to the shared memory or to the interconnection structure. Of course, contention could be reduced if some "connection locality" could be achieved during the partitioning of connections to the processors. Another weak point is that it is necessary to maintain some flow control between the processors since a processor might be very busy when the others want to send to it.

LAN controllers

The Local Area Network controllers has recently evolved very fast, maybe due to the LAN standardization. The evolution tendency is to include more and more layers into the controller and thereby off-loading the host. That this is necessary, is clear from the comparison of the two Ethernet controllers that demonstrated that a file transfer was consuming more than 95% of the available host CPU-time, independent of controller. One difficult controller design issue is how to transfer data between the host and the controller, i.e. where to locate the shared memory, how to access data in buffers, and how to organize context switches between several connections.

Both Intel and Motorolas "LAN controllers" represent very much the state of the art. Both are built around a local bus, with dedicated chips for the Physical and Media Access part of the Link Layer and with a local microprocessor that executes the other layers. Communication with the host is done via a double port memory, interrupts, status and command registers.

Comparison criteria

When possible, the processors were compared with respect to performance, protocol and implementation trade-offs. It is clear from the comparison that it is difficult to conclude that one design is superior to another since they optimize different aspects and use different protocols with different traffic characteristics. However, some observations has been done that are summarized below.

Performance

•*Throughput.*Number of connections, packet sizes, and protocol complexity is crucial for throughput, e.g. large packets and many connections will increase it. All the high performance switches are estimated to handle about 20-30 Mbit/sec. Except for MININET, these switches are expected to handle as many as 10 000 low speed connections. MININET, on the other hand, is designed to handle "at most 10 channels" with very short packets but with protocols less complex than X.25. A conclusion we have drawn is that MININET demonstrates that it is possible to switch 100 000 packets/sec between a few network connections, while still using hierarchically organized protocols. The short message length, 32 bits, is not the only reason. Similar message header processing exists for much longer messages and other protocols hierarachies. In fact, the data part of a message do not need to be processed at all. It only needs to be copied once, in a careful design.

The physical interfaces of the controllers are designed to handle the high speed of LANs, typically 10 Mbit/sec. However, when adding more protocol and layers, the throughput quickly degrades. Minnich and Cotton report that for ARPANETs protocol suite of IP/TCP/FTP the throughput degraded to less than 150 kbit/sec. over a 10 Mbit/sec. Ethernet. The bottleneck is obviously both in the interaction between host and controller as well as in the implementation of the higher layer protocols. The newer controllers, that Intel and Motorola, represent, are likely to increase the performance, but not dramatically. But they will most likely off-load the host many of the communication tasks which also is very important.

•*Delay*. The switches do not optimize delay characteristics. The switch delay is sometimes unimportant in long distance networks where other factors are dominating. Also, for low speed switch connections there is no need to serve a connection until the next packet will arrive or until the local buffers have filled up. Therefore, many of them do not state any figures. For NTT:s switch the delay is estimated to max 400 microseconds.

For the controllers, the media access protocol specifies the maximum packet handling time. E.g. Ethernet specifies that two packets may arrive within 64 microseconds. As with the throughput, the delay will quickly degrade with higher layer protocols. The Minnich/Cotton study reports that it takes about 10 msec for a packet to be send back and forth when using the bare Ethernet, while the Murray/Enslow study has measured delays over 50 msec when using a Virtual Terminal Protocol over a connection oriented protocol and Ethernet. Since the higher layers are so much slower than the Physical and Data Link layers, flow control must be

performed at the higher layers so that the lower layers will not be flooded with packets that the higher layer can not keep up with. If lower layer buffer space is flooded, packet must be discarded.

Protocols

For the switches, protocols up to and including the Network layer are used. Normally they support X.25 in one or another version, except for the ARPA IMPs and MININET that use their own protocols. It is seldom reported on which version of X.25 they have implemented or what options are available or if it is complete or not.

The dominating protocols for the LAN controllers are CSMA/CD(especially Ethernet) and more recently the Token Bus protocols (ISO 8802.3 respectively ISO 8802.4). The earliest controllers only supported the Physical Layer and the Media Access Part of the Link Layer. Today, both Intel, Motorola, and Texas Instrument will or already have released "LAN controllers" that provide protocols for all seven OSI layers, aiming at the MAP and TOP protocol stacks. It is likely that we will see more of these multilayer controllers in the near future.

Implementation trade-offs

• *Modularity and Service Access Point accessibility* The switches do not normally provide access to the lower layers. In terms of flexibility, most of them are emphasizing incremental growth of the number of connections. Pluribus unique way of job management provides hardware flexibility, since the connections are not directly coupled to any particular processor. The appearent drawback with the approach is the division of the code into tiny tasks of 300 microseconds each.

Both Intel and Motorola provides access points to the lower layers with propriety interface protocols, which be implemented in the hosts.

• *Interface to host Machine* The switches are not normally directly attached to the hosts. The LAN controller are usually using buffers, interrupts, and command and status registers to interface to the host. The tendency is to design more advanced interfaces, e.g. in terms of off-loading the host the buffer management and by using command interpretations in the controller.

•*Dependability* Since a switch may connect thousands of computers it will affect many users if it fails. Therefore, most switch designers have been concerned about dependability. The concern has been concentrated very much on the interconnection structure since it will affect all connections. Double buses or rings are used in order to tolerate a failed link. Also the "management processor" is quite often duplicated for the same reasons, while the peripheral processors normally do not have any protection. It is obviously acceptable that a

few connections get disconnected with a failed processor. Pluribus, with its organisation can accept that processors fail since the connections are not "hardwired" to any specific processor. Only the performance will degrade.

For the LAN controllers, the dependability is of no major concern.

• *Implementation language* This is not a big issue for the switches since the users never will get into the switch.

For the controllers, we have found that Motorola boosts that its MicroMAP software is written in C in a portable form. It remains to see if it is possible to port it to another operating system, or to replace one layer in the controller.

• *Buffer Handling* The buffer management scheme is the single most important factor for performance. It also demonstrates the most variety. However, it is clear that unnecessary copying from one buffer to another will rapidly decrease performance. Garbage handling of buffers, and to retrieve buffers is complicated in multiprocessor systems, especially when connections and processors fail. Fixed or variable size buffers is another issue that different implementation use either one. The argument for variable size is that it does not waste memory space while the argument for fixed size is that it is faster. Buffers could also be organized into chains of small fixed sized buffers. But then there are more memory accesses necessary which will slow down the controller.

It is also clear that LAN controllers need many buffers to be able to receive bursts of packets because higher layers protocols can not consume packets in the same rate as may arrive on the network.

The multiprocessor switches operate with small dedicated pools of buffers due to their lower speeds. On the other hand, when two processors in a switch exchanges packets, the sending processor needs to ensure that there are buffers available at the receiving processor, which will complicate buffer handling.

• *State information and context switch* The processors may need to switch from one connection to another when a new packet arrives. Then it needs to reach the state of the new connection and to restore the old one. How fast this can be done, depends on how closely in time the packets may arrive. For Ethernet they may arrive within 10 microseconds which make it clear that a hardware switch must be used at the lowest level.

Some switches explicitly report on special hardware to provide the switch, i.e. the MININET and SCIPION, that identifies the packet and directly points to the corresponding state. Pluribus, with its unique organization does not need to have a context switch for its processors, since context is stored in global memory and any processor can serve any packet.

8. References

[Ahuj82] Ahuja, S. and Asthana, A., "A Multi-Microprocessor Architecture with hardware Support for Communication and Scheduling", *Proceedings of Symposium on Architectural Support for Programmning Languages and Operating Systems.* Palo Alto, California, 1982, pp. 205-209.

[APN167] APN 167 System Description, LM Ericsson, Dept for Standard Systems, S-126 25 Stockholm, Swden, 1986

[Aoki86] Aoki, M., Uchiyama, T., Tonami, S., Hayakawa, A., and Ichikawa, H., "Protocol Processing for High-Speed Packet Swiching Systems",*1986 Int. Zürich Seminar on Digital Communication.* D2.1-D2.6.

[Balp85] Balph, T., Dirvin, R. and Shvager, Y., "Token bus controller uses monolithic structure to chip away MAP costs", *Electronic Design,* November 11, 1985.

[Cara85] Caraballo, M., Hendershot, D., Millas, R., and Weakley, T., "Programmable Communications Processor", *IBM Disclosure Bulletin,* Vol. 27, No 10B, March 1985, pp 6323-6324.

[Ches81] Chesley, H. and Hunt, B., "Squire A Communications-Oriented Operating System", *Computer Networks,* No5, 1981, pp. 333-339.

[Chil84] Chilton, P. and Depledge, P., "MEUNET A High Performance, High Availability Packet Switch", *Digest of Papers, The 14:th Int. Conference on Fault-Tolerant Computing,* Kissimmee, Florida, 1984, pp. 164-169.

[DCom86] Data Communications: Guide to Datacommunications Equipment 86, *Data Communications,* Mid-May 1986, pp 154-155.

[Deus86] Deussing, P. and Eckhardt, B., "The Distributed 1PSS Architecture: A High Reliable Switch for High Performance Packet Switching Network", *Proceedings of Eight Int. Conference on Computer Communication,* München 1986, pp. 353-358.

[Ensl84] Enslow, D and Murray, P. "An Experimental Study of Performance of a Local Area Network", *IEEE Communications Magazine",* Nov 1984, Vol 22, No. 11, pp 48-53.

[Dyna86] Dynapac Systems, "The PSX family", Dynatech Communications SARL, Paris, 1986.

[Hann85] Hannaford, S., "The challenge for today's operating system designer", *Data Communications,* October 1985, pp.197-202.

[Haug82] Haughney, J., "Anatomy of a packet-switching overhaul", *Data Communications,* June 1982, pp. 85-97.

[Hear72] Heart, F., Kahn, S., Ornstein, S., Crowther, W. and Walden, D., "The Interface Message Processor for the ARPA computer network", *1972 Spring Joint Computer Conference, AFIPS Conference Proceedings,* Vol.40, 1972, pp. 551-567.

[Hear73] Heart, F., Ornstein, S., Crowther, W. and Barker, W., "A new minicomputer/multiprocessor for the ARPA network", *1973 National Computer Conference, AFIPS Conference Proceedings,* 1973, pp. 529-537.

[Hube80] Huber, J., "EDX A Uniform System Architecture for Circuit, Packet, and Message Switching", *Proceedings of fifth Int. Conference on Computer Communication,* Atlanta, 1980, pp. 195-201.

[iNA-84] Intel "iNA 960 Network Software", Preliminary, Intel Corporations, Santa Clara, California, 1984.

[ISO 84] ISO/TC 184 N81E, "Real-time discrete parts manufacturing communications architecture", ISO TC 184/SC 5, 1986.

[Jung85] Jungefeldt, T. and Linell, J. "Connecting a Minicomputer to an X.25 Network", The Royal Institute of Technology, Dept of Telecommunication Systems-Computer Systems, Stockholm, September 1985.

[Kats78] Katsuki, D., Elsam, E., Mann, W., Roberts, E., Robinson, J., Skowronski, S., and Wolf, E., "Pluribus-An Operational Fault-Tolerant Multiprocessor", *Proceedings of IEEE,* Vol. 66, No. 10, October 1978, pp. 1146-1159.

[LAN86] LAN/II Overview, Interactiv Systems/3M, 1986

[Lein84] Leiner, B., Cole, R., Postel, J. and Mills, D. "The DARPA Internet Protocol Suite", *IEEE Communications Magazine"*, March 1985, Vol. 23, No 3, pp 29-34.

[Mair86] Mair, E., Hausmann, H., and Naessl, R., "EWSP A High Performance Packet Switching System", *Proceedings of Eight Int. Conference on Computer Communication,* München 1986, pp 359-364.

[MCMA85] Motorola "VMEmodule Advanced MAP Network Interface", Product Preview, Motorola GMBH, München, Germany.

[Mill72] Mills, D., "Communication Software", *Proceedings of the IEEE*, Vol. 60, 1972 pp. 1333-1341.

[Minn83] Minnich, M. and Cotton, C., "An Evalution of two UNIBUS Ethernet Controllers", *Proceedings of 8:th Int, Conference on Local Computer Networks,* Minneapolis, Minnesota, 1983, pp. 29-36.

[MMAP86] Motorola "Preliminary MicroMAP Software Product Description", Motorola LTD, Milton Keynes, England.

[Morl83] Morling, R., Cain, G., Neri, G., Longhi-Gelati, M. and Natali, P., "Design of a High-Speed Word-Switched Transport Station", *IEEE Journal on selected Areas in Communications,* Vol. SAC-1, No. 5, November 1983, pp.740-750.

[Naga84] Nagasawa, M., Aoki, M., Yoshida, Y., and Kato, M., "Role of Packet Switching Networks in Future telecommunications Network -- A Proposal of High Speed Packet Switching Networks", Musashino Electrical Communication Laboratory, NTT, Musashino-shi, Tokoy, Japan.

[NeRy72] Newport, C.B., and Ryzlak, J., "Communication Processors", *Proceedings of the IEEE,* Vol. 60, Nov. 1972, pp 1321-1332.

[Niwa86] Niwa, A., and Yamada, T., "A 32-Bit Custom VLSI Processor for Communications Network Nodes", *IEEE Journal on Selected Areas in Communications,* Vol. SAC-4, No. 1, January 1986, pp. 192-199.

[Newp77] Newport, C. and Kaul, P. "Communications Processors for TELENET's Third Generation Packet Swtiching Network", *Proceedings EASCON 77*, pp 8-2A to 8-2L

[Ono83] Ono, K., Matsuo, K., Takizuka, T., Iisako, S., Teshigawara, Shimizu, Y., and Kojima, H., "Design of a Multiprocessor Based Satellite Packet Communication Processor for Integrated Services Networks", *Satellite and Computer Communications* Elsivier Science Publisher, 1983.

[Orns72] Ornstein, S., Heart, F., Kahn, S., Crowther, W., Rising, H., Brussell, S., and Michel, A., "The Terminal IMP for the ARPA computer network", *1972 Spring Joint Computer Conference, AFIPS Conference Proceedings,* Vol.40, 1972, pp.243-254.

[Put78] Put, P. and R. Renoulin, "Architecture of a High Throughput Packet Switching Node", *Proceedings of Fourth Int. Conference on Computer Communication,* 1978, pp. 208-215.

[Redd83] Reddi, A., "Pipeline and Parallel Architectures for Computer Communication Systems", *Proceedings of 1983 Int. Conference on Parallel Processing,* August 1983, Columbus, Ohio, USA, pp. 148-150.

[Rest82] Restorick, F. and Pardoe, B., "A Multi-Microprocessor Design for use in Packet Switched Networks", *Proceedings of Sixth Int. Conference on Computer Communication,* London 1982, pp. 811-816.

[Rind76] Rinde, J., "Tymnet I: An alternative to packet switching technology",*Proceedings of Third Int. Conference on Computer Communication,* Toronto 1976.

[Rous86] Rouse, D., Spires, R., and Wallace, R., "The Number 2 Signal Transfer Point: An Overview of the AT & T Common Channel Signaling Packet Switch", *Proceedings of Eight Int. Conference on Computer Communication,* München 1986, pp 370-374.

[Rudi86] Rudin, H. "Trends in Computer Communications", *IEEE Computer,* Vol 19, No 10, November 1986, pp 25-33

[Sait82] Saito, T., Inose, H., Wada, M., and Shibagaki, H., "High Capacity Packet Switching System by means fo Multi-microprocessors", *Proceedings of Sixth Int. Conference on Computer Communication,* London 1982, pp. 817-822.

[Schm82] Schmidtke, F., "A Communication Oriented Operating System Kernel for a Fully Distributed Architecture", *Proceedings of Sixth Int. Conference on Computer Communication*, London 1982, pp. 757-761.

[Seit83] Seitz, N., Wortendyke, D., and Spies, K., "User-Oriented Performance Measurements on the ARPANET", *IEEE Communications Magazine,* Vol. 21, No. 5, August 1983, pp. 28-44.

[Sirb84] Sirbu, M. and Zwimpler, L. "Standards Setting for Computer Communication: The Case of X.25", *IEEE Communications Magazine"*, March 1985, Vol. 23, No 3, pp 35-45.

[Sjöd87] Sjödin, P. "Performance Aspects in a Local Area Network Implementation ", Licentiate thesis in preparation, Swedish Institute of Computer Science, Box 1263, 163 13, Stockholm, Sweden, 1987.

[Tana78] Tanaka, Y., Matsunaga, H., Matsuda, C., and Yamamoto, "Communication Control Processor for Computer Networks: Performance Evaluation", *Proceedings of Fourth Int Conference on Computer Communication,* 1978, pp. 71-76.

[Tayl83] Taylor, D., Oster, D. and Green, L., "VLSI Node Processor Architecture for Ethernet", *IEEE Journal on Selected Areas in Communications,* Vol. SAC-1, No. 5, November 1983, pp. 733-739.

[Thei72] Theis, D., "Communications Processors", *Datamation,* August 1972, pp 31-43.

[TOKE85] Motorola "MC68824 Token Bus Controller", Preliminary Data sheet, Motorola LTD, Milton Keynes, England, 1985.

[TOKE/] "TOKEN/NET, TIM-200, TIM-220", Concord Data Systems, 303 Bear Hill Road, Waltham, MA 02154, USA, 1985

[Tona83] Tonami, S., Aoki, M., Ishitani, T., and Fukuda, H., "A VLSI for Communication Control", *IEEE Int. Conference on Communications,* Boston 1983, pp. B3.7.1 B3.7.5.

[Wiks83] Wikström, R. "Design of Network and File Transfer Services in SUNET, the Swedish University Network", Licentiate Thesis, Institute of Technology, Uppsala University, Uppsala Sweden, UPTEC 83110 R, Nov 1983.

[Wile86] Wile, G. and Gowan, D., "The Architecture of the DPN Data Networking System", *Proceedings of Eight Int. Conference on Computer Communication,* München 1986, pp. 365-369.

[X.25-86] Motorola "X.25 Protocol Controller (XPC)", Technical Summary, Motorola LTD, Glasgow, Scotland.

[Yash85] Yashiro, Z., Tonami, S., Nishiwaki, M., and Ishino, F., "A Distributed High-Throughput Packet Switching System", *Proceedings of IEEE Int. Conference on Communications*, Chicago, 1985, pp. 27.1.1-27.1.5.

[554-85] Intel "iSXM 554 MAP Communications Engine", Advance Information, Intel Corporations, Santa Clara, California, 1985.

[586-84] Intel "82586 Local Area Network Coprocessor", Preliminary, Intel Corporations, Santa Clara, California, 1984.

[588-84] Intel "82588 Single Chip LAN Controller 82588: High Integration Mode 82588-5: High Speed Mode", Advance Information, Intel Corporations, Santa Clara, California, 1984.