北海道大学工学部研究報告
第131号（昭和61年）

Bulletin of the Faculty of Engineering,
Hokkaido University. No. 131 (1986)

LOCAL AREA NETWORKS-A Structured Implementation Example

Drago NOVAK and Yoshinao AOKI

(Received December 27, 1985)

Abstract

The article presents hardware and software aspects of a local area network implementation. First, the Open Systems Interconnection Reference Model, which enables structuring of the communication protocols is described. Taking this model as a basis for the implementation, we present then the intelligent Ethernet controller that serves as an interface between the micro computer and the communication media. The controller is built as a single board computer that supports multiprocessing. For easier communication software structuring and implementation a simple real time kernel was developed. Its primitives and data structures are described next. The primitives enable parallel process execution, process synchronization, and assure mutual exclusion while using common resources. The final part contains the description of the interface data structures and processes implementing the network and transport layer communication protocols. We end thus with the transport layer and a "byte-stream" interface. With this all the work on the network side is done. The remaining communication layers that have to be implemented on the host are not part of this article.

1. Introduction

Local area networks are specific computer networks with high rate data transfer and diameter of few kilometers. Most often they are used in office automation. Currently, two types are most popular: carrier sense and ring networks. These two types differ in the communication media access control mechanism. We will concentrate on the first type with the most representative network called Ethernet known also as the IEEE 802.3 communication standard. It is a baseband packet broadcasting network.

First we will review the layered network architecture model developed by the International Organization for Standardization to take it as a reference in our discussion. Then, the hardware part that implements the lowest layers will be described. The intelligent network controller enables the actual connection of the computer to the communication media and offers enough processing capabilities to free the host or central processor of the communication protocol processing burden.

The software part includes a real time kernel for multiprogramming and various data structures, and processes that implement the upper communication layers. We will take XNS protocol (Xerox Network Systems) as a basis for upper communication protocol. We will try to demonstrate the advantages of an intellidgent network controller that supports

multiprogramming for communication protocol structuring and implementation.

2. Layered Netwotk Architecture

Communication networks are rather complex systems so it is very convenient for implementation and for discussion to divide them into layers, similar to those of operating systems. A model for this layering was developed by the International Organization for Standardization (ISO) and is called Open Systems Interconnection (OSI) Reference Model (Figure 2.1). This model has seven layers starting with hardware and ending with user processes. Each layer performs a set of well defined functions using resources of the lower layer. The final goal is, of course, to offer the user efficient communication services.

The first two layers, physical and data link layer, are responsible for communication media access, data framing, error detection and sending and receiving data. These two layers are usually implemented in hardware using LSI integrated circuits.

The network layer determines how the packets are routed within the network. In this layer the unit of information exchange is a packet. It carries additional information such as final destination necessary for routing. UP to this layer, we were concerned only about each individual unit of data-packet or frame. We did not care about the order of the received packets.

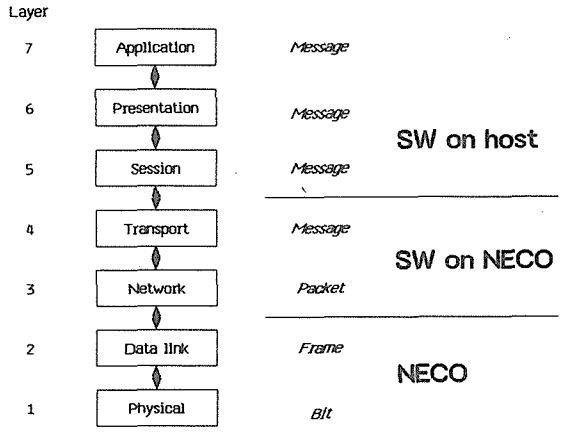


Figure 2.1 ISO OSI Reference Model

The transport layer has to assure a correct transmission of messages, that can, of course, consist of more than one packet. This layer should offer errorfree message delivery. It uses sequence numbers for proper ordering of receiving packets and acknowledgments for lost data identification and retransmission. With this layer we have already a raw network and what we need more are user interface and a set of standard network utility functions similar to utilities in operating systems.

The session layer provides the user interface with functions for setting up and closing connections over network and sending and receiving messages.

The presentation layer performs necessary conversions of messages in order to be understood by application layer that can vary from station to station.

The application layer depends on application. Usually, it implements functions like electronic mail, virtual terminal, file transfer, etc.

This short review will give us a better insight in the hardware and software architecture for local area network implementation described in the following paragraphs.

3. Architecture of the Local Area NEtwork COntroller-NECO

Local area network controller is the basic element for local area network construction.

It provides an interface between the computer and the transmission media. Controller can be as simple as RS232 interface or as complex as micro computer. The complexity depends on the low level protocol for the access to the transmission media, transmission speed and system requirements. Here, we will describe a complex intelligent interface controller to the Ethernet local area network that was developed for experiments. Before we start with the description let us review the design goals. These are the connection of VME bus (VME is a commercial name for the IEEE-P1024 bus) based micro computers to the Ethernet, connection of passive devices to the Ethernet, multiprocessing capability and stand alone VME bus master capability.

The NECO is built as a single board computer with a high speed 68000 processor. Local memory is 256 kbyte wide and can be accessed by the local CPU, Ethernet controller and VME host (main CPU)(See Figure 3.1). Access conflicts resolve local bus arbiter which gives higher priority to requests from Ethernet controller and from the VME bus. Memory interface was designed with a special care in order to allow access without wait states. This is namely the key point for higher performance. The actual interface to the Ethernet is implemented by using Am7990 two chip set from Advanced Micro Devices. The first is LANCE (Local Area Network Controller for IEEE 802.3) that performs the data link level protocol. The second is SIA (Serial Interface Adapter) which provides Manchester encoding and decoding of serial bit stream.

Ethernet is a baseband, packet broadcasting local area network. The transmission media is called "ether" and most commoly this is a coax cable. All stations listen to the traffic on the cable and receive only packets that are addressed to them. A distributed control policy called CSMA/CD (Carrier Sense Multiple Access with Collision Detection) is used for the shared communication channel access. The station starts transmission when there is nothing going on in the channel (no carrier). During transmission, the transmitting device listens for a collision. A collision occurs when more than one device start transmitting simultaneously. Ethernet protocol defines the minimum length of the collision that assures the detection by all devices. After collision detecting, all devices will defer transmission and start retransmission after random delay period.

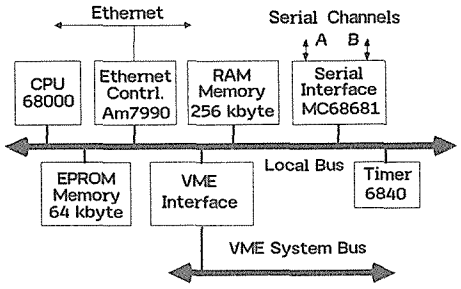


Figure 3.1 Block Diageram of the Network Controller NECO

The LANCE integrated circuit implements the described CSMA/CD policy, picks up packets with the right address from the communication channel and stores them in local memory of the NECO and transmits packets it gets from the local memory. The interface between local CPU and LANCE are common buffers in the local memory. During LANCE initialization, two buffer rings are established, receive buffer ring and transmit buffer ring. Each buffer has its own descriptor with status information. There is a bit called OWN that enables buffer passing. LANCE owns the buffer when the OWN bit in buffer descriptor is set. It passes the buffer to the CPU by clearing this bit. The LANCE scans the two buffer

rings in order to get free buffers for the receiver and to get some work (full buffers) for the transmitter.

Besides the Ethernet interface, there is the VME interface. It enables NECO to become VME bus master in multiprocessor environment, to send and to handle interrupts, etc. These and other facilities like EPROM memory, programmable timers, etc enable easy and efficient implementation of higher communication layers within the NECO.

4. Kernel for Efficient Implementation of Communication Protocols

Implementation of communication protocols is a typical real time programming example. The easiest way to structure and implement the protocol is to break it into more or less independent entities-processes that can run parallel and can communicate and synchronize their execution. The classical way is to implement a kernel with a scheduler and a set of elementary system functions or primitives that support synchronization and mutual exclusion. Let us describe the organization of the kernel implemented on our NECO controller.

The kernel contains data for process presentation, scheduler, primitives and system clock interrupt service routine. Processes are represented by data structure called the process descriptor. They are created at compile time and their number is fixed. Although the process descriptors are already created, only the processes that were explicitly started by START primitive can run. The process descriptor (Figure 4.1) contains pointers for chaining, current context of the process, pointer to the signal waiting for, pointer to the next process descriptor that is in the same queue on the signal, timeout counter, timeout context pointer, process identifier and process status.

```
typedef     struct      proc_d{
            struct      proc_d      *next_pro;
            struct      proc_d      *prev_pro;
            struct      cntx        p_cntx;
            struct      proc_d      *next_s;
            struct      sig         *s;
            int         tout;
            struct      cntx        *t_cntx;
            int         pr_id;
            int         status;
}PROC;
```

Figure 4.1 Process descriptor

All process descriptors are collected in a vector and are chained during initialization. The scheduler scans the process descriptor chain and selects on robin round basis the next process. The processes can be in two basic states: ready to run or waiting for a signal.

One of the timers that have the highest interrupt priority was selected for the system clock. The system clock periodically interrupts the CPU activity-current process execution. The scheduler selects a new, ready to run process and activates it. This allows time sharing,

that is parallel execution of independent processes. The processes have to implement communication protocol. So they are not independent. They have to use common resources and synchronize their activities. For this purpose a set of basic functions, primitives, was implemented. The processes have their own stacks and run in 68000 processor's user mode. Calls to the primitives are implemented using the TRAP instruction. Like an interrupt, a trap is an exception and pushes current status and next address into supervisor stack. Kernel primitives run on the highest interrupt level in supervisor mode. Implemented are SEND, WAIT, P, V, START, WDOGON and WDOGOFF primitives.

The first two primitives, SEND and WAIT, are used for synchronization. They implement the concept of signal that is similar to the one defined in Modula. The signal is a pointer to the process descriptor queue.

```
typedef     PROC*SIG;
```

The WAIT(&s) primitive puts the calling process in the queue on the siglal s and sets its status to "not ready to run". The SEND(&s) primitive removes the first process from the waiting queue and changes its status to "ready to run". The WAIT results always in context switch, while the process calling the SEND primitve continues the execution. The signal that is not awaited gets lost. This property has many advantages but can also be very dangerous and causes deadlocks. The process that was too slow to come to its WAIT point can wait forever if the signal had been already sent.

P and V operations deal with semaphors that are similar to signals except that they have a counter that counts how many times P and V operations were performed.

```
typedef     struct      sem{
            int sval;
            struct      proc_d      *qptr;
}SEM;
```

The P primitive is similar as the WAIT primitive. It decrements the semaphor value and if the value becomes negative, it puts the process into the waiting queue. The V primitive simply increments semaphor value if it is nonnegative, else it also takes one process from the waiting queue. Semaphors are very convenient for mutual exclusion when accessing common resources and for common buffer management. When used for mutual exclusion purpose, the semaphor value is initialized to 1 and we get a binary semaphor. The code within the brackets P(&mutex), V($mutex) represents a critical region that can be entered only one process at a time.

As already mentioned, the START(func, id) primitive starts function "func" as process with "id" identification number. The next pair of primitives implements the watchdog principle. The process starts its watchdog with the WDOGON primitive. This primitive saves the current context as a point where to continue when time out occurs. It initializes also the watchdog counter. Typical calling sequence is:

```
int cnt;            /*watchdog counter value*/
```

```
CNTX alpha;      /*context save area*/
. . .
if(wdogon(&alpha, cnt)){
    /*time out processing*/
}
else{
    /*activity that is controlled by watchdog*/
    wdogoff( );
}
```

In the example, the context is saved in variable "alpha" and the pointer to "alpha" is written into process descriptor. The return value of WDOGON function is zero. The watchdog timer interrupt routine decrements all watchdog counters (one per process). If the counter reaches zero, the time out condition occurs and the context in the process descriptor is replaced by the time out context. In our example with "alpha" and the return value is changed to nonzero. The described mechanism protects processes from getting cought in endless loops.

The described elementary kernel functions fascilitate structuring and implementing of the communication protocol.

5. Layered Protocol and its Impact on Implementation

In this section we will chose one of the most common protocol called XNS (Xerox Network Systems) to see how it is structured and how it can be implemented. We have already described the lowest layer of our local area network. It is an Ethernet with a coax cable and an intelligent controller NECO. The hardware implements a communication channel access control mechanism and data framing. This corresponds to the first two layers of the OSI model and to the level zero of the XNS protocol. The XNS Internet Transport Protocol has three levels that implement the functions up the transport layer of the OSI model. Since this seems to be the natural boundary between the network and the network station, we will implement this part of the protocol on the controller and upper layers on the host computer.

5.1. Network Layer protocol

The XNS Internet Transport Protocol (ITP), level one, performs the routing function. This is also called the Internet Datagram Protocol. The object of the routing functions is a datagram, also called the internet packet. The datagram contains all information necessary for its delivery to the destination. Although it carries related data with other datagrams, it is treated by the router as a completely independent entity. At this level we are concerned only about how to deliver the packet. If it is impossible, i.e. the destination is not visible, we can even throw the packet away. On the levels below we had only host addresses that are always unique. On this level, we have additionally a network address and a socket address. The first is a unique Ethernet local area network address, that helps forwarding packets

through different connected networks. The latter is an object inside the host to which the packets can be delivered and from which packets may be transmitted. For the routing purposes there is a routing table that contains the addresses of the networks in which the router is interested, and addresses of the stations-internet routers that have access to them. The internet router, this is a router at the junction of two or more networks, forwards packets between networks, so it must know the addresses of all networks connected to. Some sockets within a station called "well known sockets", have a predefined address and offer various services. As we will see later, there is a routing protocol on the next level implemented as a client process that updates the routing table and periodically broadcasts its own address.

Protocol layers introduce a control structure-header into data communication enitites. The data part of the frame that is an object of the data link layer is the internet packet with ITP level header and data. This header contains the destination and source address and gives the packet the datagram property. It includes also a type field, representing thus an interface to the next upper level.

To implement the described protocol we need processes for receiving and transmitting internet packets, we need a pool of buffers and mechanisms to assure mutual exclusion and synchronization. We have also to define the bidirectional structure-a socket to which it is possible to pass the incoming packets and that will constitute interface to the next level processes. Everything except sockets was already implemented in the kernel. Figure 5.1. 1 shows the rough implementation scheme of the Internet Datagram Protocol.

Sockets are data structures with an identification field, receive and transmit buffer queues and receive and transmit semaphors.

```
typedef     struct      socket{
            short       soc_id;
            SEM         soc_rec;
            SEM         soc_tx;
            int         *soc_rbp;
            int         *soc_tpb;
}SOCKET;
```

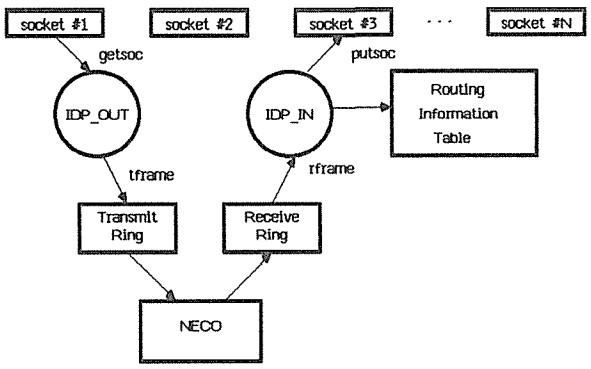


Figure 5.1.1 Network Layer Implementation scheme

The two processes IDP_OUT and IDP_IN, communicate with Ethernet controller within NECO through transmit and receive buffer rings by calling functions "tframe" and "rframe" for transmitting and receiving, resp. Both functions are protected against reentrancy with mutual exclusion semaphors, and we use semaphors for synchronization of access to the buffer rings. Both function have a similar structure:

```
tframe()
{
```

```
    p(&tsem);     /*mutual exclusion*/
    . . .
        p(&frame_sent);     /*waiting for buffer*/
    . . .
    v(&tsem);
}
```

The LANCE interrupt service routine as the partner that uses the same buffers is therefore:

```
lan_isr()
{
    . . .
    if((st = lan_stat)==RERAME)
        v(&frame_received);
    if(st==TFRAME)
        v(&frame_sent);
    . . .
}
```

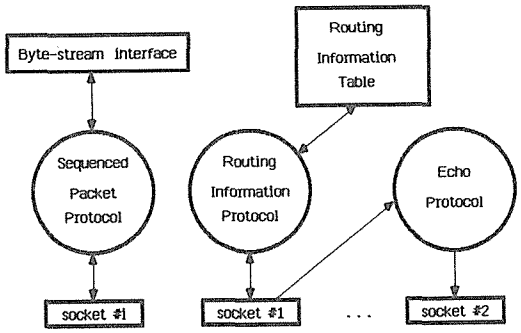
The IDP_OUT process scans sockets and puts packets into the transmit ring. The IDP IN gets packets from the lower level, interprets the address field, performs packet forwarding if necessary or passes them to the appropriate socket. Forwarding is necessary only if the host is connected to more than one network. One way to implement forwarding when using intelligent controller is to assign special socket for this purpose or to let the processor on the controller execute the "tframe" function on the other controller. With our NECO controller this is possible because the processor on one controller can become VME bus master and can access the local memory of the other.

5.2. Transport Layer Protocol

This layer provides the error free transmission of messages that can be split into more packets. This level is called the Level Two Internetwork Protocol. At this level, we deal with retransmissions, acknowledgements, etc in order to get data error free and in right sequence through the communication channel. We have mentioned already that the protocols below are not reliable. So it is possible that a packet will get lost and since many paths between communicating stations may exsist, the packets we receive may be out of order. Again, we need a special header for the control information. It carries sequence numbers, acknowledgements, buffer allocation information, etc. This level provides also connection establishment capability. Upper layers or client processes communicate with each other by sending messages over established connections. This level consists of many protocols: the Echo Protocol, Error Protocol, Sequenced Packet Protocol, Packet Exchange Protocol, Routing Information Protocol, etc. The Sequenced Packet Protocol implements the connection establishment, termination and reliable data transfer. It offers to the upper level "byte-

stream" or "block-stream" interface similar to usual devices like terminals or disks. The Routing Information Protocol keeps the routing information table used by router up to date. Other protocols offer some additional services that are useful but not absolutely necessary.

We see that this level has very clear interfaces to the adjacent levels, the sockets downwards and for instance "byte-stream" upwards. The latter has data and control part for "data received" or "data transmitted" type of information. The functions of this layer can be implemented by separate processes (Figure 5.2.1)



Fibure 5.2.1 Transport Layer Implementation Scheme

Up to this point, all interfaces were shared between processes on the same processor -that of the intelligent controller. With the "byte-stream" and/or "block-stream" interface we came to the border between the host and the local processor. This requires a different kind of mutual exclusion and synchronization implementation since the cooperating or competing processes reside on different processors. The controller NECO provides some facilities for this purpose. First is, of course, the TAS instruction (test and set) that enables mutual exclusion implementation. And second, there is an interrupt structure for synchronization. Interrupts can be sent from the controller to the host and in the opposite direction. The host can interrupt the activity of the controller by using VME interrupt structure or by reference to the specific address on the controller. Interrupts are used as notify signals and can be converted into true signals as defined by kernel by puting the SEND calls into the interrupt service routines.

6. Conclusion

We presented the complexity of the hardware and software that is necessary for a usual micro computer to become a station on the local area network. The problem has been solved by layering the communication protocol and by using a powerful intelligent network controller that supports multiprogramming and multiprocessing. The implementation is based on the definition of the data interfaces between the communication layers. The communication protocol algorithms are implemented as a set of cooperating processes using common interfaces.

7. References

a) XNS Internet Transport Protocols, Fuji Xerox, December 1981
b) The Ethernet-A Local Area Network, Data Link and Physical Layer Specifications, Xerox, Intel, Digital, Novemeber 1982
c) Xerox Network Systems Architecture, Fuji Xerox, April 1985
d) Douglas Comer : Operating System Design, The XINU Approach, Prentice Hall 1984
e) Andrew S. Tanenbaum : Computer Networks, Prentice Hall

f) James Martin: Computer Networks and Distibuted Processing, Prentice Hall 1981

g) D. Novak, Y. Abe, Y. Aoki: Universal Network Controller as a Basic Element in Local Area Network Implementation, Proceedings of the Conference of the Institute of Electrical Engineering of Japan, Hokkaido Branch, Sapporo, September 1985, p. 308