A COMPREHENSIVE TAXONOMY AND DESIGNS OF ATM HOST-NETWORK INTERFACES

By

Bo-kyun Na

bkna@top.cis.syr.edu

B.S., Hanyang University (1991)

M.S., Syracuse University (1993)

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in EECS in the Graduate School of Syracuse University

April 2000

Approved ______________________

Dr. Jeoffrey C. Fox

Date ______________________

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Dr. Jeoffrey C. Fox
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Dr. Marek Podgorny

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

... name
(Another Department)

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Preface

This thesis tells you all you need to know about...

Acknowledgements

I would like to thank...

Abstract

This is .....

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations

The last 50 years have been particularly fruitful with regard to electronics innovation, but only in the last 20 years have we seen a significant development in network design. Until the 1980s, the common types of traffic handled by digital networks were interactive data, generally transmitted in short bursts of a few characters, between terminals or between terminals and computers; file transfer, involving the transmission of up to millions of characters or bytes between computers, or between mass storage systems; and, increasingly, digital voice [90].

In the 1990's, with the evolutions from the telecommunication networks and computer-communication networks towards the Broadband Integrated Services Digital Network (BISDN), some important directions and guidelines have been created. The recent directions taken by the BISDN are influenced by the emergence of a large number of teleservices with different requirements, the fast evolution of the semiconductor and optical technology. In this information age, the most significant teleservices are HDTV (High Definition TeleVision), video conferencing, high speed data transfer, videophony, video library, home education, electronic marketing through Internet, and video on demand (VOD). This wide range of network applications introduces the need for one universal network which is flexible enough to provide all the required communication services.

With the recent developments in computer communication technologies, it is now possible to develop high speed computer networks that operate at data rates in Giga bit per second (Gbps) range. These networks have enabled high performance distributed computing and multimedia applications. The requirements of these applications are more diverse and adaptive than those of traditional data applications such as file transfer. The existing implementation of standard communications protocols do not exploit efficiently the high bandwidth offered by high speed networks and consequently, can not provide network applications the required high throughput and low latency communication services [45].

Many researchers have observed that while the link level rates of some networks are now in the gigabit-per-second range, the effective throughput between remote applications is usually several orders of magnitude less. This has intensified the efforts to develop new high-performance computer communication environments that can utilize the enormous bandwidth required by tele-multimedia services and offered by high speed networks.

As can be concluded from the above, the network of today are very specialized and suffer from a large number of disadvantages, the most important being [82]:

Service Dependence Each network is only capable of transporting one specific service for which it was intentionally designed. Only in a limited number of cases and by using additional equipment and with an inefficient use of its resources can it be adapted to other services.

Inflexibility Advances in audio, video, speech coding and compression algorithms, and progress in Very Large Systems Integration (VLSI) technology influence the bit rate generated by a certain service and thus change the service requirements for the network. In the future, new services with unknown requirements will appear. A specialized network has great difficulties in adapting to changing or new service requirements.

Inefficiency The internal available resources are used inefficiently. Resources which are available in one network can not be made available to other networks.

Taking into account all these considerations on flexibility, service dependence and resource usage, it is consequently very important in the future that only a single network exists and that this network of the future is service-independent. This implies a single network capable of transporting all services, sharing all its available resources between the different services.

A single service-independent network will not suffer from the disadvantages described above, but it will have the following main advantages;

Flexible and future-safe Advances in the state-of-the-art of coding algorithms and VLSI technology may reduce the bandwidth of existing applications. A network capable of transporting all type of services will be able to adapt itself to changing or new needs.

Efficient in the use of its available resources All available resources can be shared between all services, so that an optimal statistical sharing of the resources can be obtained.

Less expensive Since only one network needs to be designed, manufactured and maintained, the overall costs of the design, manufacturing, operations and maintenance will be smaller.

1.2 Problems with Current Designs

A host-network interface connects a host to a network for sending and receiving data. It consists of the hardware that connects the network medium with the host I/O bus and the network software that handles an application's communication requests and manages the host-network interface [94]. Most applications communicate over the network via the socket interface and the internet protocols (*e.g.* TCP/IP or UDP/IP).

Before defining problems on current designs of host-network interface, one needs to understand where best to invest design efforts to obtain the maximum performance improvement for the host-network interface, given a particular host architecture and some cost and complexity considerations. An implementation model is introduced as follows [84]:

- The simplest in the range of alternatives available is to build a host-network interface with no DMA capability, no processing on board, and leave all the processing and data movement responsibilities up to the host processor and software.
- The second model is to introduce DMA to assist in the data movement activities,

while still doing very little processing on board. This model is implemented in the current designs for interfacing workstations to the Ethernet.

- The third model is the host-network interface performing most of the protocol processing, with the host software involved in finally delivering the data to the user.

In the same way, implementations of the transport protocol have followed three approaches [45, 8]:

- Improving the implementation of standard transport protocols [17, 20].
- Introducing new protocols [89, 13, 18].
- Implementing protocols on a special hardware [57, 49].

Analyzing the above implementation models, three major problems with current designs can be identified [25, 57]:

- Poor protocol implementations.
- Poor interface between applications and the network interface.
- High overhead associated with operating system functions and context switchings.

First, conventional transport protocols are too complex or awkward for hardware implementation and too slow without it. On a send, the socket layer copies the data from the user's address space into system buffers and invokes transport and network protocols. If the user requests a reliable byte-stream protocol, TCP/IP (Transmission Control Protocol/Internet Protocol) will be used, and the protocol processing will include packetization, error handling, end-to-end flow control, routing, and congestion control. A best effort protocol such as UDP/IP (User Datagram Protocol/Internet Protocol) is simpler and performs only a subset of these tasks. When the protocol processing is finished, one or more packets are handed to the data link layer, which will transmit them over the network [98].

A similar sequence of operations is performed on the receive side, but in the reverse order. Specifically, the data link layer places incoming packets in system buffers, called receive buffers, and after protocol processing has been performed, the data is copied into the user's address space by the socket layer as part of the application's receive call.

The error handling performed by reliable transport protocols has a significant impact on how the data is handled by the protocol stack. Protocols typically use an end-to-end checksum to verify data integrity and time-outs to detect lost packets.

Checksumming requires TCP on the sending and the receiving host to read the data and calculate a checksum; if the checksum the receiver calculates differs from the one inserted in the packet by the sender, the receiver ignores the packet.

To detect lost packets, the sending host starts a timer whenever it sends a packet. If this timer expires before the packet has been acknowledged, the sender assumes the packet was lost or corrupted and retransmits it. The sender keeps a copy of all transmitted data in a system buffer (the retransmit buffer) until the reception is acknowledged by the receiver.

As a part of a send operation, the application writes the message into a buffer in its address space. The socket code copies the data into a system buffer, and the transport protocol reads the data in order to calculate the checksum. Finally, the data link layer copies the data to the host-network interface. In total, the data crosses the memory bus six times, including the first write by the application. The receive operation follows the inverse path of the send operation.

Second, compared to the alternative system where a network interface does programmed I/O transfers and a host performs transport protocol functions, the interfaces that implement transport-level functions have lower performance at the transport level [57]. The primary reason is an inadequate internal memory architecture. Currently, the data transfers into and out of the buffer memory reduces the number of memory cycles available for packet processing. The future system bus technology with a high transfer rate and the burst-mode transfer, and the future networks with a high data rate, will make this problem even more acute.

Finally, most implementations of transport protocols are tied heavily into host operating systems [80]. The reason for large operating system overhead is the structure of the communication process in general and the implementation of network interfaces in particular. In other words, the CPU performs such an important role in the communication process that, consequently, there are too many interrupts, too many context switches, and too large a scheduling overhead.

Unfortunately, the impact of the overhead introduced by the operating systems on the communication process strongly affects the application-to-application communication performance. The major sources of this overhead are [44]:

- Scheduling
- Redundant data transfers in main memory
- Overhead of entities management - timers, buffers, and connection states
- Overhead associated with division of the protocol processing into processes, including interprocess communication

- Interrupts
- Context switching

However, some of these problems may be solved with better implementations of existing protocols. Clark *et al.* [17] suggested that even heavyweight protocols such as the TCP/IP could be extremely efficient if implemented sensibly. It means that while a poor implementation will impede performance, protocols such as TCP/IP are not inherent performance-limiting factors.

Network interfaces were introduced to offload the CPU from the communication process. However, they have been doing only a partial job; interfaces are still being built that interrupt the processor for each received packet, leading to multiple context switches and scheduler invocations.

1.3 Research Approach

When a domain of study or practice contains many distinct objects, the problem of classification arises. Classification means partitioning the objects into a structured set of classes on the basis of meaningful criteria. A taxonomic system is a system of rules whereby objects in a given domain are classified in a particular way. Thus far, in proposed taxonomic systems [92, 26, 72], the motivations are:

- To establish a classification scheme within which the position of architectures relative to one another can be understood.
- As a foundation for constructing an architectural knowledge base.
- As a long-term objective, the construction of a comprehensive, comparative atlas of computer architectures or host-network interfaces within the unified

framework of the taxonomy.

Given a set of objects which we wish to classify, the simplest of taxonomic systems must consist of a single set of taxons among which we can partition the objects. A taxon is a named group of objects that are sufficiently distinct from the objects belonging to some other taxon. The set of properties that determine how objects are assigned to the various taxons are termed taxonomic characters. In more comprehensive classification schemes, several categories can be identified, each with its own set of taxons. These categories can be ranked in a hierarchy. Each object would appear in exactly one taxon in each category.

1.3.1 Taxonomy Features

To refer to the categories and the taxons once a classification scheme has been constructed, a taxonomic system must contain a convenient, methodical, and meaningful system of nomenclature. A good taxonomy should have the following characteristics:

Descriptive It provides a basis for information ordering as needed, for example, for cataloging documents on computer architectures in a library or for organizing architectural descriptions in a textbook. Thus, this property is useful to establish a theoretical framework within which we can meaningfully compare and discriminate between architectures and determine precisely how and where they converge or diverge.

Hierarchical Hierarchical classification systems by their very nature exhibit some particularly attractive properties. They not only provide a basis for comparing and discriminating between objects, but they also make it possible to determine the points of their convergence to a common taxon across category levels. Thus they provide a conceptually more elegant picture of the relationships between objects than do nonhierarchical schemes.

Predictive A classification provides a foundation for predicting certain properties of an architecture. Specifically, given a taxonomic system, and being informed that architecture A belongs to taxon y in category x, we can infer properties or characteristics of A from our knowledge of the taxonomic system.

Explanatory A stronger and more ambitious version of the predictive role is for the classification scheme to provide a basis for explanation. The distinction between predictive and explanatory goals is often subtle and largely a matter of degree. Given a taxonomy, and being informed that architecture A belongs to taxon y in category x, the predictive power of the scheme will allow to infer that A has such and such properties. The explanatory power of the scheme, if at all present, will allow to deduce why A possess these properties.

1.3.2 Research Objectives

The main objective of this thesis is to present a new, comprehensive architectural taxonomic system that appears to possess the desirable characteristics of a good taxonomic scheme. Usually, the discussion of a taxonomy should not be confined to a small scope. Rather, it should encompass the entire architecture. Thus, it is meaningful and often convenient to separate the whole into few levels and construct taxonomic schemes for each level independently. Our taxonomy of host-network interfaces will, hence, consists of two levels: architectural and protocol classifications.

Architectural Classification

We can consider top level as the architectural models of the host-network interface according to the guidlines for classifying architectures.

First, an architectural classification captures the essence of a particular architecture form, without distinguishing different technologies, implementation sizes, or the like. Architectural characteristics collectively constitute an architecture of the host-network interface. Since every host-network interface possesses a physical architecture, the taxonomy should be a system for classifying physical-architecture characteristics. This descriptive feature helps to understand each physical architectural unit in the host-network interface.

Second, the architectural characteristics enable comparisons between physical architectures to determine precisely how and where they converge or diverge. The gaps in the classification can suggest developing other possible physical architectures. While the first feature gives the knowledge of architectural characteristics, this helps to understand taxonomic characteristics within a category (out of several hierarchical categories).

Third, proper architectural characteristics can help to predict the optimized host-network interface. Even if some hardware components enable fast implementation, the architectural structures have to be flexible. Once a hardware component is used, replacing of the unit is costly. Thus, a good taxonomy should support the architectural predictive features on the host-network interface. It helps to understand the performance improvement of a new designed host-network interface.

Following these guidlines, we can approach the architectural classification with the PMS (processor-memory-switch) notation which describes well the relations between processor, memory, and I/O devices [39, 91]. In the PMS notation, a system is described as an interconnected set of components or individual devices, associated with a set of operations. Such a description can be complicated by the amount of detail involved. It may take a whole manual, for instance, to describe the operations of a major host-network interface. Thus, the PMS descriptive system permits very compressed descriptions, and description of only those aspects of the components that are of interest, while ignoring the rest. It also permits primarily the analysis of the amounts of information held in various components, the flow of information between components, and the distribution mechanism that controls the flow.

Note that a physical communication system is built of hosts, host-network interfaces and a network. Thus, the physical characteristics are from these logical structures (*i.e.* hosts, host-network interfaces, and a network) and their connections. There are six types of functional units from which any architecture of the host-network interface can be constructed. These are host CPU, main memory, cache, network buffer, protocol processor, and a switch:

- Host CPU unit executes instructions of the user application which may require a communication, if required, to the protocol processing related to the communication, and it transforms data usually in ways that correspond to basic arithmetic operations.

- Main memory unit is a storage device that passes data to and from the host CPU.

- Cache unit consists of a small fast memory that acts as a buffer between the main memory and the host CPU.

- Network buffer unit is a staging and speed-matching area for data in transit between the host and the network. It consists of a network buffer memory and network FIFOs. Network buffer memory is the storage for transport-layer data (so called message) and information related to the control and management parameters, while network FIFOs consist of two sets of registers for sending and receiving data. It also provides the protocol processor with contention-free memory access to the packet data.

- Protocol processor unit manages packet processing and various bookkeeping functions associated with the protocol.

- Switch unit provides connectivity between other functional units by the one of programmed I/O, DMA (Direct Memory Access), burst transfer by memory controller, or register accessing.

Protocol Classification

At a more detailed level, we can consider protocol implementations. A protocol is a set of communication rules governing the format and semantics of the frames, packets, or messages that are exchanged by the peers. It is used to implement network services, which define what operations are prepared to perform on behalf of its user but do not define how these operations are implemented. Thus, rather than by the network service, a classification by the protocol implementation is more reasonable and understandable.

There are more reasons for having a taxonomical classification of protocols. The one reason is the impact of protocol implementations on the communication system. As the set of rules related to the instructions and data paths has strong influence on the computer structure, protocol implementations can affect organizations of the host-network interfaces.

The other reason is that protocol implementation can support various network-service requirements instead of changing the physical architecture characteristics. Nowadays, with developments of networks, various user applications require sometimes extremely different network services within a network. The host-network interface can support those requirements via different protocol implementations. With how to implement it, therefore, the protocol affects the structure of host-network interfaces.

Another reason is devising a useful model of performances for host-network interfaces. Since the protocol implementation has several common categories, we can figure out a protocol classification to the host-network interface, compare protocol functional units in a category, thus predict useful model. Even in the same physical architecture, the different protocol processing can cause the system to greatly improve the performance.

With these reasons, the taxonomical classification of protocols may approach with functional units and providers of these functional units. In my approach, the main functional units of a protocol implementation are defined as follows:

- Data access in main memory has influence on the performance of host-network interface. For example, for protocol processing such as checksumming, the operating system needs to copy data into the kernel space from the user-application space in a conventional protocol structure.

- Routing by network addresses to find proper destination.

- Flow control to govern the transmission rate in order to avoid overruns at the receiver and/or to avoid congesting the network.

- Error handling in order to detect and correct errors occurred during the data transmission if required by transport service users.

1.4 Organization of Thesis

The organization of this dissertation is as follows. Chapter 1 is the overview of the retionale for my research.

Chapter 2 identifies the driving forces towards better host-network interfaces by surveying the basic architecture, the desirable features, limitations and bottlenecks of current designs.

In chapter 3, we first review the conventional classifications of host-network interfaces. The classifications consider various host-network interfaces for terminals, Ethernets, token-rings, FDDIs, and even for ATM networks. Then, host-network interfaces are analyzed to uncover limitations for the each given classification and propose requirements that a new comprehensive taxonomy has to include. As a result, a comprehensive taxonomy is introduced based on the sublayers which are an architectural and a protocol-processing classifications. We demonstrates the usefulness of the new taxonomic scheme by applying it to several typical host-network interface designs.

Chapter 4 presents new designs of host-network interfaces, which are created according to the design method enabled by the new comprehensive taxonomic scheme proposed in the previous chapter. A simple host-network interface is one example. The interface is designed for cost-effective and minimal hardware-support systems. The major part of protocol processing is implemented on the host processor. These features are described by the new taxonomic scheme. Another new design example, the intelligent host-network interface, is proposed with features of no data copy in main memory, simple and fast protocol processing, and off-the-shelf components related to the network buffer. The host processor can access the network memory as it accesses a cache. Such design has a potential of supporting very high speed networks in the Gbps or even Tera bit per second range.

Chapter 5 analyzes the performances of simple and intelligent host-network interfaces based on the comprehensive taxonomic scheme presented on previous chapter. The effective application-to-application bandwidth of both transmitting and receiving communications is analyzed for the simple and intelligent host-network interfaces, respectively.

Chapter 6 summarizes and concludes this dissertation with a concise presentation and effects of taxonomic scheme on the design of host-network interfaces, which is the main concepts of this research.

Chapter 2

Review of Host Network Interfaces

2.1 Introduction

The network I/O architectures become a bottleneck in the communication subsystems, since the current processors have been optimized for computational tasks but not for data moving tasks. In current local area networks (LANs), the effictive application bandwidth is often an order of magnitude lower than a nominal bandwidth provided at the network medium [14]. For example, out of the physical bandwidth of 10 Mbps at the medium level of the Ethernet, only about 1.2 Mbps is available to each application [14].

The host-network interface is the network I/O which becomes the bottleneck in communication subsystems. As mentioned in [16], the recognition of this resulted in an increasing interest on designing high performance host-network interfaces [4, 19, 27, 8, 84]. The reason that the network I/O becomes the bottleneck lies on the implementation of communication protocols and redundant data copying, as discussed on Chapter 1.

Therefore, to understand why the interfaces become the bottleneck on communication systems, it is necessary to first consider how communication protocols are implemented in most of the existing machines.

2.2 Overview of Host-Network-Interface Architectures

The main functions of a host-network interface include sending data from the host to the network and receiving data from the network to the host. These functions can be described with a simple host-network interface shown in Figure 2.1 [94].

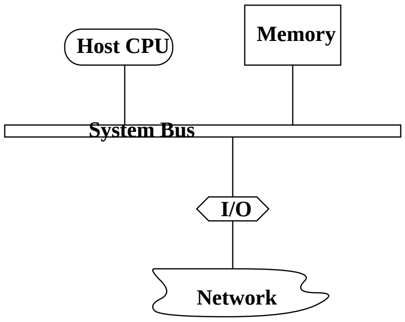


Figure 2.1: Diagram for the architecture of a host-network interface

In the sending path, which is taken by data through a conventional protocol stack (TCP/IP), the application writes data into its application buffer and then invokes a system call to send data. The socket layer copies the application data into a socket buffer in the kernel space. The transport protocol reads the data in the socket buffer to compute a checksum and, finally, the data is copied out to the network by the host CPU with the routing information. Thus, the memory system is accessed five times for each portion of data sent. Figure 2.2 shows the diagram with 5 data copying steps. In the receive path, data is copied first from the network into a socket buffer in kernel memory by the host CPU. The transport layer checksum is verified and, as soon as the application is ready, the socket layer copies data from the socket buffer to the application buffer in user memory. Finally, the application reads the data. Thus, the memory system is also accessed five times for each portion of data received.

2.2.1 Design Techniques

The architecture of a host-network interface depends on the required performance and cost. If we implement all the functions in software, we end up with a very simple host-network interface that consists of just network buffer (FIFO). It is, of course, going to be very slow since the host CPU must perform the normal computing tasks as well as the communication tasks.

One can reduce the load of the processing from the host CPU by adding processing capabilities to the host-network interface. They are of two types: simple DMA (Direct Memory Access) based and intelligent host network interfaces.

DMA based Design

A network adapter board for a LAN can be based on DMA with on-board controller [86]. Programs running on the host CPU can exchange frames with on-board controller through shared areas in main memory. The on-board controller performs all operations required by the network medium-access scheme automatically and independently from the host CPU. The host CPU attention is required only to handle error-free data frames. The controller is able to perform a network reconfiguration algorithm when joining a network, that is network jamming, polling, and establishment of the NID (Network Identification). It is also able to detect a token-loss situation and perform the token-recovery algorithm. All these operations are performed without

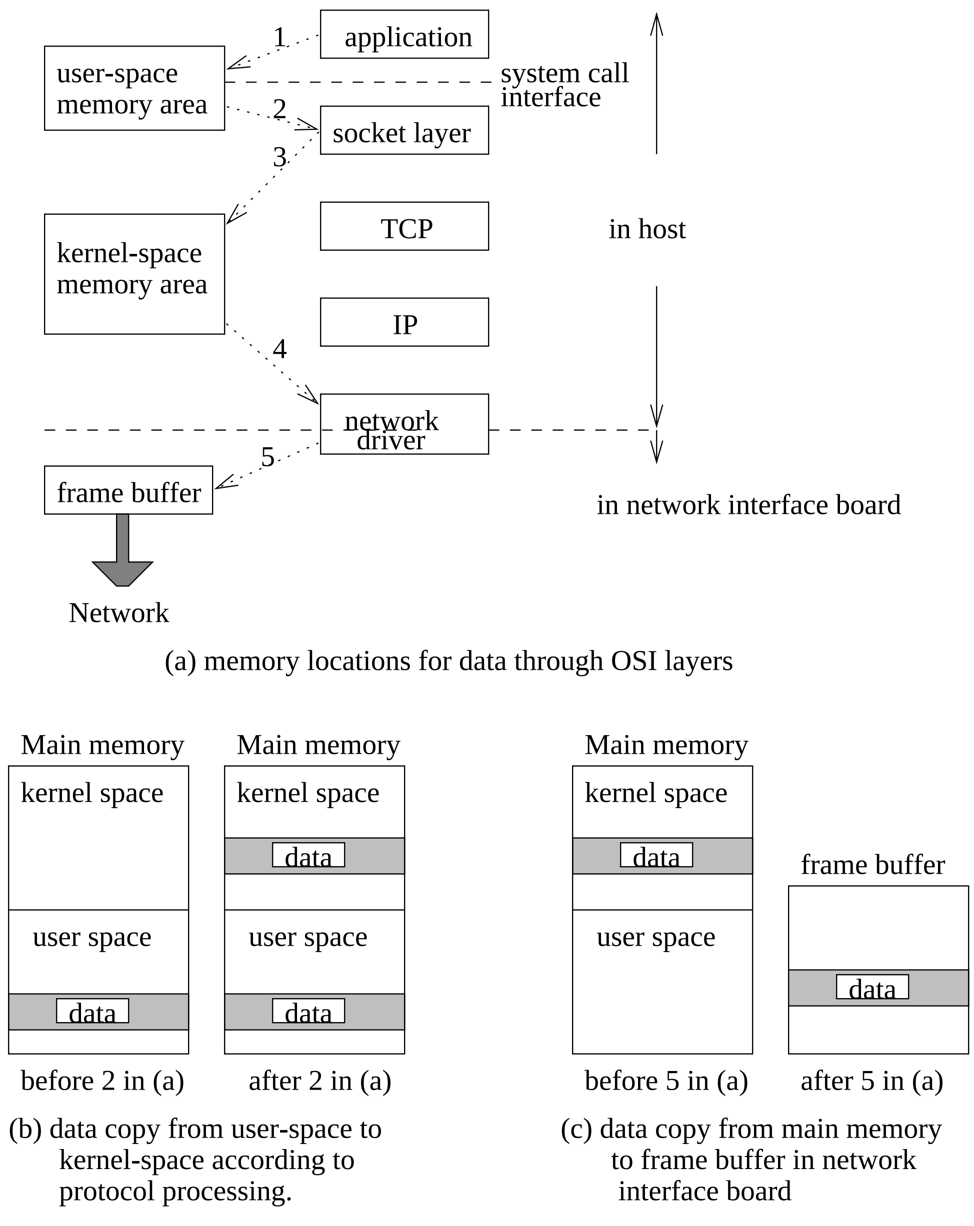


Figure 2.2: The diagram of data copying on a simple host-network interface

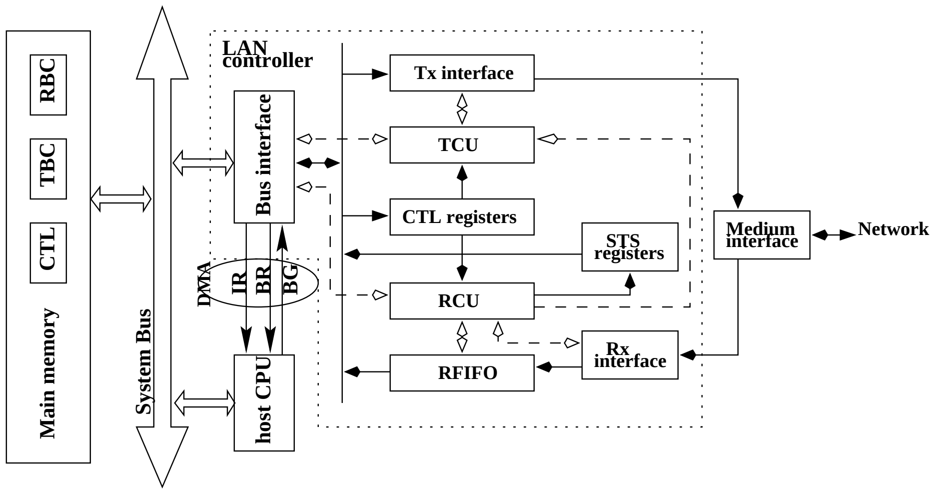


Figure 2.3: The connection between a host and a simple DMA-based host-network interface

the host CPU intervention, and the CPU is informed only about the ultimate failure or success. During normal operation, the controller monitors all frames arriving from the network medium. It accepts only the error-free frames with destination addresses matching MIDs (Medium-layer Identification) stored registers in the controller during its initialization. The token and acknowledge frames are handled by the controller and do not require the host CPU's attention.

Figure 2.3 represents a model of a host-network interface equipped with WD-2840 LAN controller. The host CPU can either write to or read from the controller's internal registers (CTL registers). The controller can draw the host CPU's attention - for example, after frame reception - via the interrupt (IR line). A program running on the host CPU can prepare data frames to be transmitted by the controller by placing them in the shared memory area. The host CPU programs can affect the controller's operation. For example, they can request frame transmission by writing order codes to the controller's internal registers. The controller can prepare received frames for the host programs by placing them in shared memory (RBC). Status information

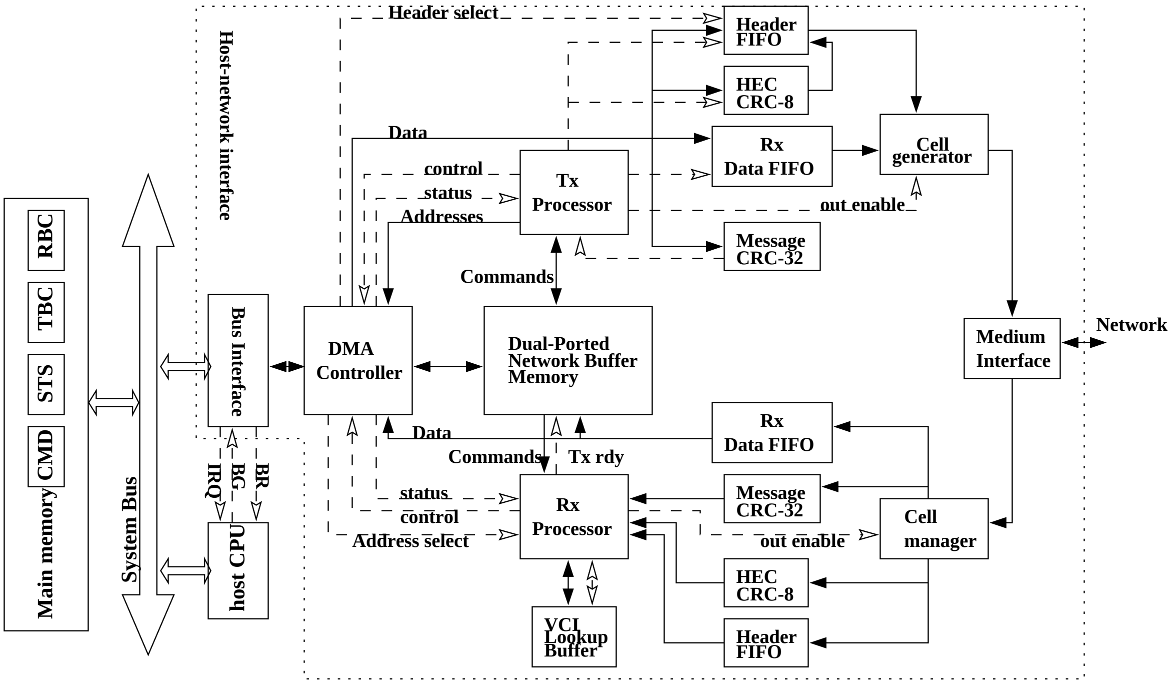


Figure 2.4: An intelligent host-network interface

reflecting the controller's operation is made available for host programs partially in the internal status registers of the controller and partially in the shared memory (CTL). All the controller's accesses to shared memory are performed as DMA read or write operations. The DMA access is controlled by the bus request (BR) and bus grant (BG) lines.

Intelligent Design

The most complicated design, but the one likely to yield the highest performance for real-time traffic, is a host-network interface with an on-board protocol processor, DMA capability, and CRC-checksum error handling during transferring data, as shown in Figure 2.4.

The AURORA OSIRIS host-network interface [31] implements the most critical, high-speed functions in hardware and consists of two independent halves - send and receive - each controlled by an Intel 80960 microprocessor.

The host-network interface board attaches to a TURBOchannel slot provided by DEC workstations. From the host's prospective, the interface board looks like a region of memory. A combination of host software and code running in the on-board protocol processors determine the detailed structure of this memory. In general, the memory is used to pass buffer descriptors between the host and the host-network interface such as header information, flow control parameters, and commands between the host and the protocol processor. Network data is not buffered in the dual-port network buffer memory; it is transferred directly from/to main memory buffers using DMA.

In the transmission, the general paradigm is that the host passes buffer descriptors to the protocol processor through the dual-port network buffer memory, and the protocol processor executes a segmentation algorithm to determine the order in which cells are sent. The program running on the protocol processor writes commands to a DMA controller and a cell generator. Thus, the DMA controller with base address and length of the intended data transfers directly data from the user space of main memory to the transmission FIFO. Simultaneously to the data transfer, CRC-32 is partially checked by a hardware for the intended data, then the protocol processor sums the partially computed CRC values to put the CRC value on the trailer of AAL (ATM Adaptation Layer). On the other hand, the header error checksum is computed by a CRC-8 hardware component.

In the reception, the protocol processor reads from a FIFO the VCI (Virtual Channel Interface) and AAL information that is stripped from cells as they are received. The VCI lookup buffer keeps VCIs to refer VCI at header formatting of the transmitting direction and to protect cell-misordering of the receiving direction. CRC-32 hardware computes partially the AAL CRC-checksum during transferring of data into the Rx FIFO. Then, the protocol processor sums partial CRC values. By examining this information, and using other information from the host (a list of reassembly buffer), the protocol processor determines the appropriate host memory address at which the payload (T-PDU) of each received cell is to be stored [28, 7]. It then issues commands to a DMA controller; typically one command is issued for each ATM cell received. As part of the reassembly algorithm, the protocol processor decides when it is necessary to interrupt the host.

2.2.2 Parallel Architectural Designs

Other techniques to reduce latency in sending/receiving of data introduce parallel protocol processing [44, 107, 75]. Applying parallelism in designing communication subsystems is an important approach to achieve the high-performance needed in today's distributed computing environment.

The main idea behind horizontally oriented protocol [44] is the division of the protocol into functions instead of layers based on ISO/OSI reference model. The functions, in general, are mutually independent in the sense that the execution of one function can be performed without knowing the results of the execution of another. Thus, intercommunication between the functions is substantially reduced and some functions can be executed in parallel.

Zitterbart [107] discussed the different levels and types of parallelism that are typically applied to communication subsystems design. They adopt a hybrid parallelism approach, which is based on layer and packet levels. In layer parallelism, different layers of the hierarchical protocol layers are executed in parallel, while in packet parallelism, a pool of processing units is used to process incoming (and outgoing) packets

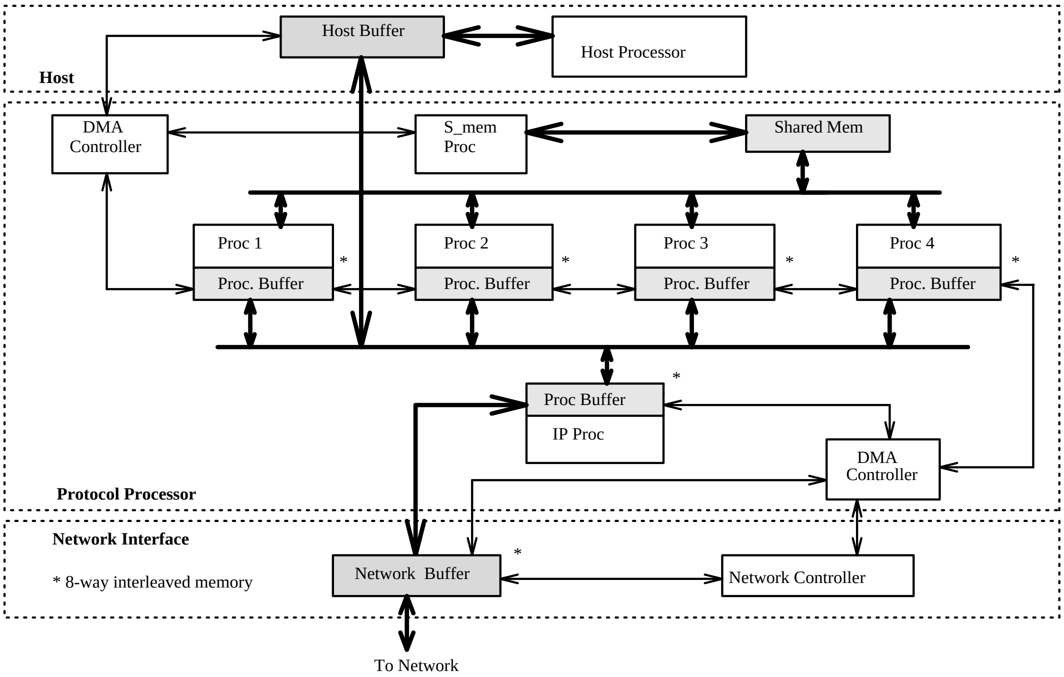


Figure 2.5: Parallel Communication Subsystem

concurrently.

In the design shown in Figure 2.5, a processor is used (IP proc) to handle the IP processing, and four transport processors (proc 1, proc 2, proc 3, and proc 4) are used to handle the TCP processing. On the arrival of a segment, IP proc executes the IP. Then, one of the transport processors is selected according to a round robin scheduling policy, to run the TCP for the arrived segment. Therefore, multiple segments can be processed concurrently using different transport processors. G. Neufeld also introduced these techniques to develop a parallel host interface [75].

2.3 Conventional Host-Network Interfaces

Several conventional host-network interfaces are described as examples. All interfaces are reviewed based on the degree of intelligence of the interface, the way to partition protocol processing between host and interface, the host interfacing, the data copying level, and how to perform checksumming functions.

2.3.1 Medusa FDDI Interface

The Medusa FDDI interface, loosely based on WITLESS, is a research prototype that was designed for the HP Apollo Series 700 workstation at Hewlett-Packard Labs in Bristol [4].

The most demanding aspects of the host-network interface and protocol stack design is viewed as the provision of high throughput all the way up the protocol stack to the application.

Therefore, as shown on the Figure 2.6, the interface contains the retransmission buffer memory that is required to support a single-copy protocol stack, and appears to the host as a block of memory in I/O space. All network, transport, and socket layer processing is performed by the host. And the network buffer memory is used as an mbuf (memory buffer) in conventional protocol processing of TCP/IP, resulting in a single copy architecture. The network buffer memory is placed in the host-network interface and a number of fixed-size blocks. The only protocol-specific part of the host-network interface design is hardware support for the transport-layer checksum.

A user process presents data to the interface buffer by means of socket layer procedures. By allowing the socket layer to perform the copy operation, the data is packetized before passing to TCP. And a checksum is computed on the fly during the data copy operation. After the packet data is copied to the network buffer memory, then the device driver copies the headers into the space reserved at the front of the buffer and causes the packet to be transmitted.

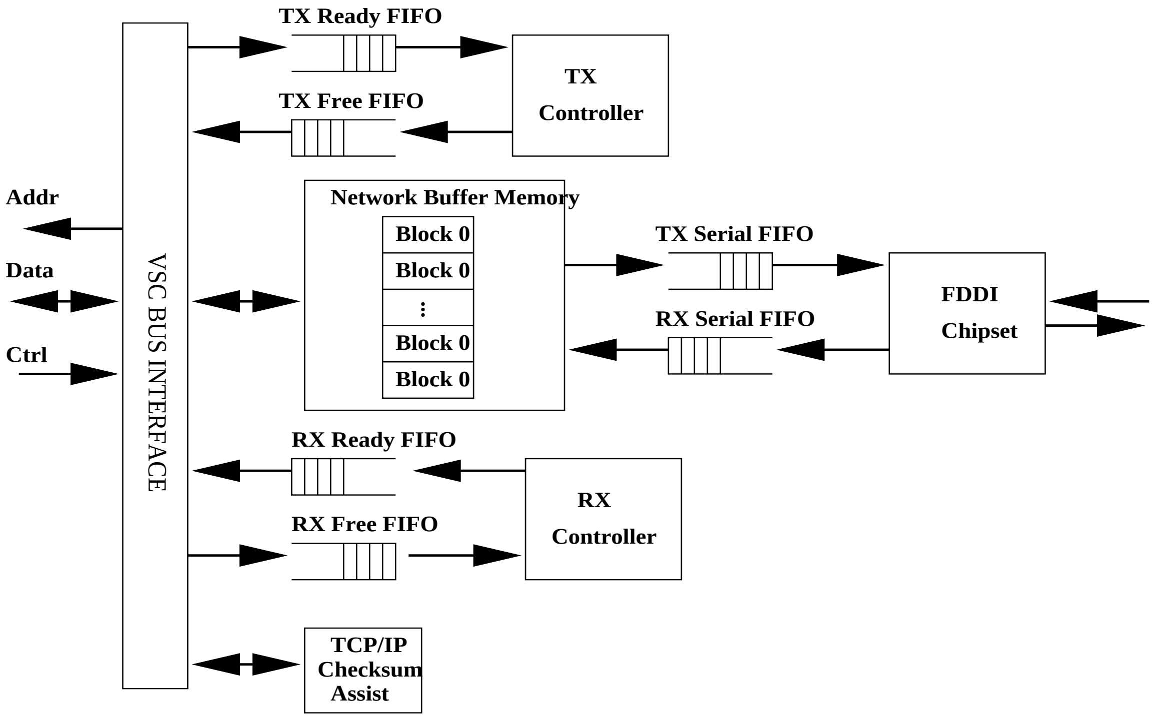


Figure 2.6: The Medusa FDDI interface

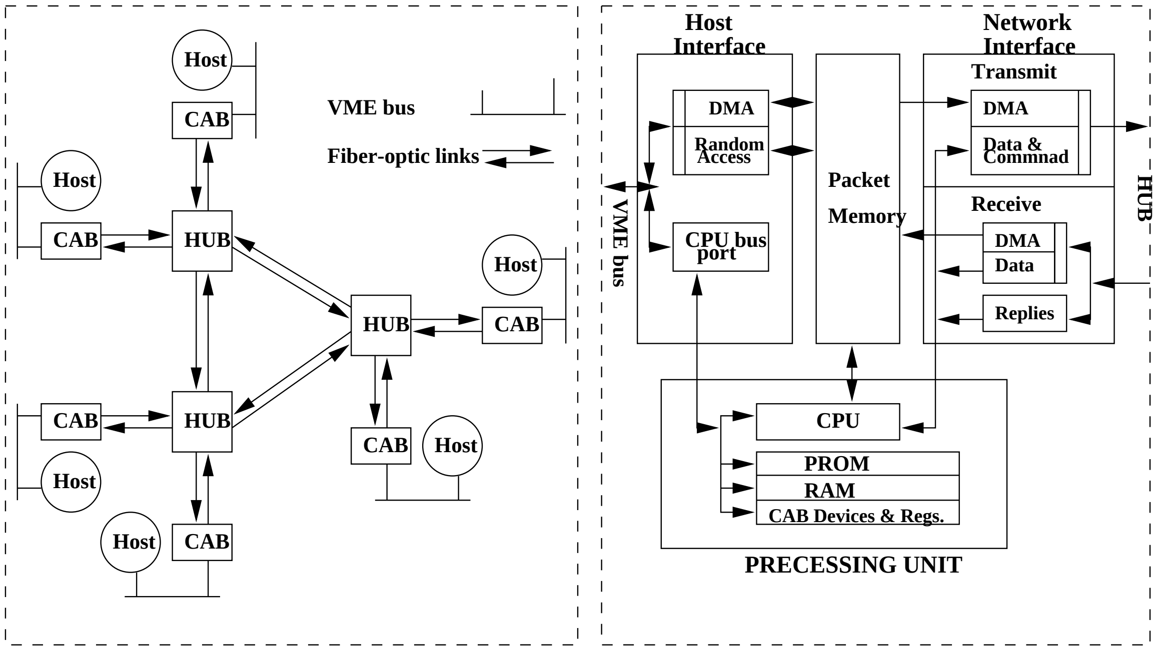
This architecture has advantages: Single data copy operation between main memory and the network buffer memory; Hardware support for transport-layer checksum. But it also has a disadvantage: Since the HNI does not support protocol processing on board, data transfers between the network buffer memory and the main memory have overhead (for example, packet header bits).

2.3.2 Communication Accelerator Board (CAB)

The interface is used in the context of the Gigabit Nectar testbed at Carnegie Mellon University. The CAB design [69] provides an architecture that interfaces high speed networks to different types of hosts. In contrast to most out-board protocol engines, the Nectar CAB has a flexible architecture, where all interactions between the network and the host are programmable. This structure allows arbitrary protocols to be implemented. One of the implementations is for DEC workstation using the TurboChannel bus.

In the CAB for DEC workstation, protocol processing is performed on the host. Data is transferred between user space to CAB memory and by using system DMA, resulting in a single copy scheme. Checksum is calculated when the data flows into network memory from the host main memory or from the network. Media access control (MAC) is performed by hardware on the CAB, under control of the host. And the CAB is designed to concentrate on MAC support for switch-based networks, specifically HIPPI networks. The upper layer protocol processing is performed at host and its headers and trailers are copied to the reserved memory space on the interface to cause the packet to be transmitted.

Figure 2.7 (b) shows the block diagram of CAB architecture. There are three major blocks of the CAB architecture: processing unit, host interface, and network



(a) Nectar system overview (b) CAB block diagram

Figure 2.7: Communication Acceleration Board (CAB)

interface. The processing unit consists of a SPARC processor, program memory, and support logic including counters, timers and a serial port. The host interface is designed for VME bus, and includes slave ports for the host to access the CAB, DMA controller and bus master logic for the CAB to access VME bus devices, and interrupt logic. Network-buffer memory, called packet memory, may be considered as a component of host interfacing. The network interface consists of fiber optic data links, queues (called FIFO's) for buffering data streams, DMA channels for transmission and reception, and associated control and status logic. The host communicates in programmed I/O with the protocol processor on CAB for control information. But data is transfered through direct memory access (DMA).

The Nectar CAB implements the source routing. Thus, it generates HUB identifier which selects the proper switch port (see Figure 2.7 (a)). The data-copying is the one-copying from user space of main memory to the network buffer memory. Then, processing unit generates CRC checksums and other header information. Flow control between CAB and HUB at the packet level is maintained by the use of start of packet (SOP) acknowledge commands. When the CAB sends an SOP to the HUB, a flag is set in the CAB network interface. The mechanism allows only one unacknowledged packet at any time. The error-handling is processed with CRC checking, length indicator of data, and sequence number on each packet.

This architecture has advantages of transport layer protocol processing on the host-network interface, supporting a variable-size packet format, and no data copying in main memory and in network buffer memory by the help of mailbox and upcall procedure. But low-level flow control between CAB and HUB may be unnecessary, which is also not supported in ATM architectures.

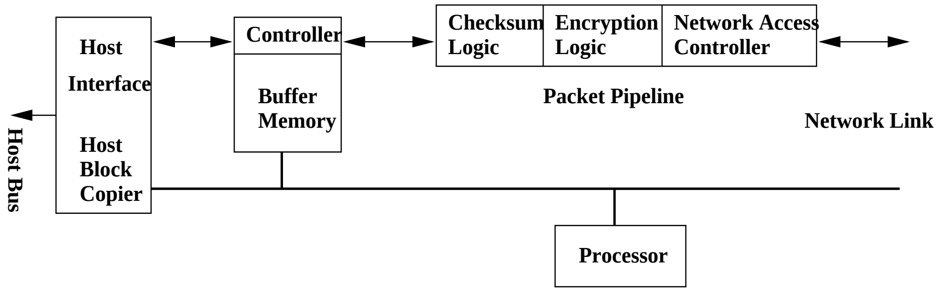


Figure 2.8: Network Adaptor Board (NAB) Internal Architecture

2.3.3 VMTP-NAB

The Network Adapter Board (NAB) [57] is an intelligent interface designed to couple with a transport protocol called Versatile Message Transaction Protocol (VMTP). VMTP is a request-response transport protocol specifically designed to facilitate implementation by a high-performance network adaptor.

The host interfacing architecture is designed for minimal latency, minimal interrupt processing overhead and minimal data transfer on the system bus. The NAB uses an internal memory and pipelined processing architecture that implements some performance-critical transport layer functions in hardware.

The network adapter board for VMP multiprocessor system has been designed using an on-board processor and network buffer memory. Besides a general-purpose processor on the NAB, there are other hardware elements to assist protocol processing, such as a host block copier for moving data between the host and the interface buffer using a burst-transfer bus protocol, and a packet processing pipelined for checksumming, encryption and decryption. Figure 2.8 describes the block diagram.

Host Interfacing: The host-to-HNI treats small and large data transfers differently, handling the small transfers using a programmed I/O interface and the large data transfers with DMA. The HNI-to-host interrupts host on a data segment boundary and not for each packet transferred. For multi-packet transfers, this significantly reduces interrupt processing overhead.

NAB protocol-processing architecture: On-board processing of checksumming, encryption, and packetization of data minimizes bus transfers. This also avoids having to transfer data to the host cache, which may improve the cache performance. A packet pipeline, executing some key performance critical functions is used to increase throughput, particularly for large data transfers. The pipeline latency for short packet transfers is reduced by using few stages and a small unit for data transfers between the stages. Connectionless memory accessing provided by a memory architecture based on dual-ported memory reduces buffering latency and increase the packet processing rate. It allows processing of a packet by the on-board processor to proceed in parallel with the transfer of subsequent packets from the host to the buffer memory and from the network to the buffer memory. Block copier hardware is used to transfer data at full blast between host memory and the HNI memory, thus reducing bus occupancy and buffering latency.

VMTP NAB architecture is an intelligent host interfacing: that is, for short message programmed I/O and for large message DMA data transfer. Other advantages are that host interrupts for the packet reception is reduced and a hardware support for CRC, encryption. But NAB board is designed only for VMTP protocol processing, not for TCP/IP or others. NAB architecture reduced data copying but still has unnecessary data copying into the network buffer memory.

In protocol implementation, VMP NAB has an advanced packet format in which the checksum field is located in the trailer of the packet rather than in the header. Data flows with one-copying from user space of main memory to network buffer memory. Then, the network adaptor board adds 32-bit CRC checksum. The packet processing on the network adaptor board is pipelined to perform checksumming, encryption and transmitting of packets. To support the flow control on transmission of packets, a timer is used in implementing a fine-grained rate-based policy.

2.3.4 OSIRIS/ORBIT ATM Interface

AURORA is an experimental wide area network testbed whose main objective is the exploration and evaluation of technologies that will support operation at or near giga-bit per second bandwidths. To support such network speed and convert data between the network format (ATM cells) and a format useful to the host (transport level message - TPDU), two host-network interfaces, OSIRIS [31] and ORBIT [7] are designed to support TURBOchannel bus on DECstation 5000 workstations and MICROchannel bus on the IBM RS/6000 workstations, respectively. But they have an architectural design concept.

ORBIT [7, 8], the host interface designed at University of Pennsylvania has been centered on developing a high-performance and intelligent host interface for IBM RISC System/6000 workstation host in the Aurora Gigabit testbed environment.

The design philosophy for the architecture is based on providing a "common denominator" set of services in dedicated hardware. All per-cell activities, such as ATM header and adaption layer creation and processing (including segmentation and reassembly) are performed by the host interface in hardware. The host is responsible for all high-level activities to achieve flexibility in protocol implementations.

The MicroChannel architecture bus on the RISC System/6000 has been chosen as the host interface's point of attachment for its high bandwidth. DMA scheme is used in transferring data between the memory in host and the buffer on interface.

It has on-board hardware elements for cell related activities and DMA device for

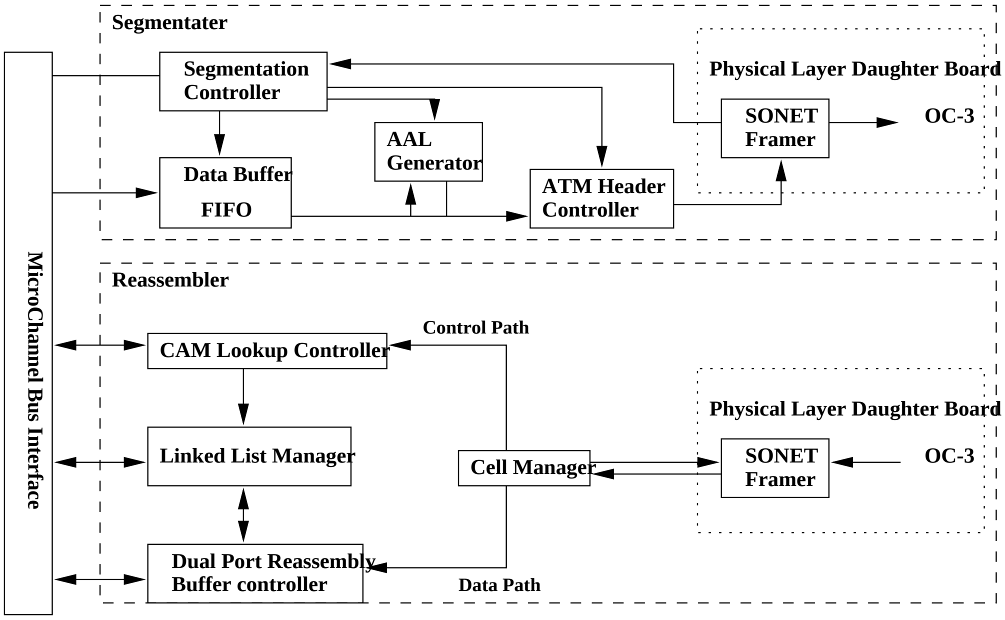


Figure 2.9: Operating environment for AURORA Host-network Interface

data transfer. Zero data copying is involved. The network interface is connected to the host main memory through I/O bus.

The OSIRIS ATM interface is built for the AURORA Gigabit Testbed at Bellcore Communications Research [27, 31]. There are two Intel 80960 RISC microprocessors, one for transmitting and one for receiving, to perform the ATM protocol processing and flow control for a trunk group of four STS-3c lines (622 Mb/s). The communications between the host and the microprocessors is through shared memory. In addition to DMA, there are other hardware elements designed to assist data transfer. As a result, the OSIRIS yield highest throughput so far. It is an intelligent interface.

Figure 2.9 shows that the host does not have a direct connection to the protocol processor. But the host communicates with a protocol processor in the network adaptor through control and network management information buffer in terms of programmed I/O and polling schemes. The most important architectural feature is direct memory access between user-space main memory and a space of transmitting FIFO, which is a register buffer, at sending data to the intended network channel. With the start address and the length of data on the control and communication management information by the host, the protocol processor segments data which is on user-space main memory. After segmentation, the protocol processor invokes DMA controller to transfer segmented data into the transmitting FIFO on the network adaptor. Conventional taxonomic schemes, however, can not explain this unique feature of AURORA project.

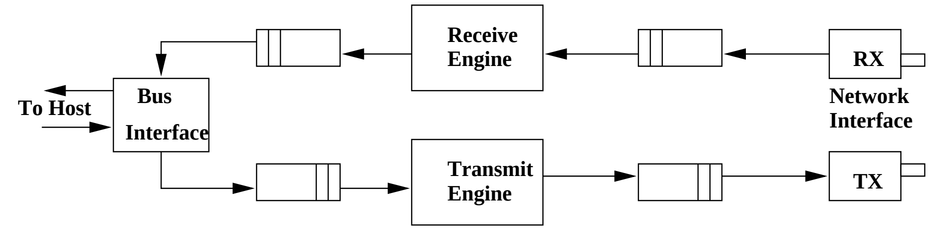
In protocol implementation, AURORA supports ATM network with statistical switching. In particular, data-copying functional unit is very efficient and distinguishable. In OSIRIS or ORBIT the intended data is copied to transmitting FIFO directly, while in VMP NAB data is copied to network memory and then to transmitting FIFO. The flow control is also based on data arrival rates. According to the error-handling specification of ATM network, error handling is supported only for transport layer. For data-link layer, no error reporting and correction processes are supported, but only header error bit checking is supported.

2.3.5 Fore's Systems ATM adapters

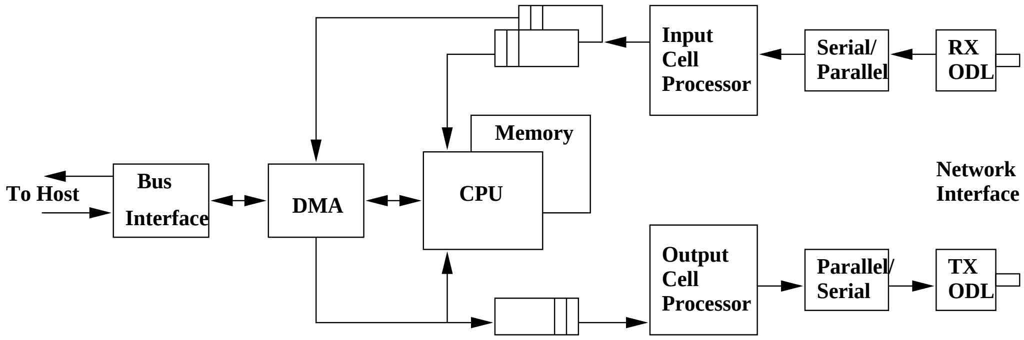
Two types of ATM interfaces were developed at Fore's Systems [19, 20]: simple slave, and intelligent host-network interfaces as shown in the (a) and (b) of Figure 2.10.

The simple slave interface implements protocol processing in the slave mode of the I/O bus. Thus, data is read and written to the transmit FIFO and incoming cells are read from the receive FIFO. Since interrupting the host is costly, a number of cells are queued before interrupting the host to reduce the interrupt overhead.

High layer protocol processing, ATM Adaptation Layer and ATM layer protocol



(a) Simple ATM host-network interface



(b) Intelligent ATM host-network interface

Figure 2.10: Fore's systems ATM host-network interfaces

processing are performed at the host. The transmitted data is copied to interface through programmed I/O in the conventional manner.

The transmit engine handles CRC generation, cell formatting and delineation. The receive engine handles cell synchronization, CRC checking for the cell header and VBR (Variable Bit Rate) segmentation and reassembly payload, and word alignment.

In the contrast, the intelligent interface has an on-board processor with local memory and DMA capability as well as CRC checksum hardware. In the receive path, the on-board processor polls the status of the receive FIFO. When a cell is received, it reads the header information and determines where the cell payload should be transferred over the system bus. It then instructs the DMA controller to carry out the actual transfer. On the transmit path, the networked data is first copied to the kernel space for high layer protocol processing in host then DMAed to the interface memory for ATM layer protocol processing using the RISC CPU on board. The host writes a descriptor to the network buffer memory indicating the address and length of a message to be transmitted. The on-board processor generates the cell header and instructs the DMA controller to transfer the payload over the host system bus to the transmit FIFO. The process is repeated until the entire message is transmitted. The segmentation and reassembly (SAR) processing code can interleave the reception and transmission of cells, and take into account cells with different priorities. The network buffer memory stores the tables and the code for SAR processing, the header information (VPI, VCI, MID, and service type), and the transfer address on the host memory.

Since the simple interface has low cost to build but the significant host interrupt latency for time-critical services, it is applicable for data communication in which the latency may not be significant. The host processor should spend processing time on protocol processing such as SAR and header information. As other disadvantage, network data can not access directly to the other peripheral devices. The network data can reach the peripheral devices via the host processor or main memory.

Since the on-board processors allow direct data transfer between the network and audio or video devices, it can eliminates the latency and overhead of going through the host memory system. It is useful for real-time multimedia applications. But the interface is not able to implement other transport protocols except TCP/IP. For example, To implement XTP, the CRC should be included on the trailer. But the HNI puts the CRC value on the header by a hardware component.

2.4 Summary

In this chapter, we reviewed several host-network interfaces based on their fundamental protocol functions, architectural features, some example conventional interfaces, classifications, limitations and overheads, and desirable features.

The fundamental functions which are common in conventional designs can be grouped as protocol processing and data manipulation functions. The protocol processing functions are typically packetization, error handling, flow control, and routing functions. Typical data manipulation functions involve data copying, error detection for data, buffer management, encryption, and packet formatting functions.

Classifications of conventional host-network interfaces have been described in several literatures and many papers, as a formalized review (we will refer these in next chapter). However, conventional classifications are focused on interface designs. They do not cover all host-network interfaces but only few small categories. They do not support the whole systematic view. Some classifications are based only on hardware, some others are based on the protocol processing (It will be refered in next chapter too). Thus, they do not provide sufficient help to support a new design work based on the classification.

As the network I/O system, host-network interface seems to cause bottlenecks in communication systems. The functional limitations of current host-network interfaces can be related to the memory access, operating system, or protocol processing. While memory-accessing overheads are related to features of the hardware components, operating system and protocol-processing overheads stem from software shortcomings. Thus, those overheads should be removed to improve cost/performance, reliability, and flexibility features.

In addition to the network specifications, the host-network interfaces are designed according to the flexibility, efficiency, high-speed data transport, cost/performance effect, and operation reliability. However, existing taxonomic schemes do not address these issues.

According to those classification limitations, a new comprehensive classification scheme is needed to support a new design of cost effective, flexible, reliable, and high-speed host-network interface. In the following chapter, a taxonomy will be introduced fullfiling these requirements.

Chapter 3

Comprehensive Taxonomy of Host-Network Interfaces

Whenever a domain of research for a system contains many different objects or features, a classification scheme can help to understand the system better. Objects are partitioned into a structured set of classes on the basis of meaningful set of criteria. The classification scheme, so called taxonomic system, is a system of rules whereby objects in a given domain are classified in a particular way.

Beyond that, a useful role of developing a classification scheme is to establish a theoretical framework within which we can meaningfully compare and discriminate between architectures and precisely determine how and where they converge or diverge. Sometimes, a good classification scheme can provide a foundation for predicting certain properties of an architecture.

3.1 Introduction

Skillicorn [92] has mentioned three reasons for classifying architectures: The first reason is to understand what has already been achieved. Until the past two decades, almost all computer systems used von Neumann architecture. However, since that time, the growth in computer systems with different kinds of parallel and distributed features has been explosive, and it is not at all clear which architectures have the best prospects for the future. Within a computer system, the communication system is regarded as an important component.

The second reason for having a classification of architectures is that it reveals possible configurations that might not have otherwise occurred to a system designer. Once existing systems have been classified, the gaps in the classification can suggest other possibilities. Of course, not all such possibilities will result in improvements.

The third reason for a classification scheme is that it allows useful models of performance to be built and used. As already mentioned, a drive for greater performance lies behind almost all new architectural ventures. A good classification scheme should reveal why a particular architecture is likely to provide a performance improvement. It can also serve as a model for performance analysis.

Over the past 10 years, there has been a rapid growth in the number of proposed and constructed host-network-interface architectures. Despite of the bred deployment of the host-network-interface architectures, there is no comprehensive taxonomy. The only existing classifications are often used to argue that the proposed architecture will praise better performances than other architectures. For these reasons, therefore, a comprehensive taxonomy will be introduced in this chapter without any prejudice.

The development of taxonomic system for host-network interfaces serves several points of immediate interest. First, the taxonomy helps to understand the host-network interfaces. Since the architectures of host-network interfaces have been developed according to the requirements of network topology, performance, and efficiency, the structural distinctions can issue the points of classification scheme. For example, programmed I/O and DMA data accesses in main memory are the different techniques on these requirements. Further, taxonomic system introduces flexible and optimized designs for special applications. The best host-network interface can be designed based on the comparison between classes of taxons, the elements of classification in taxonomic system. Optimizations such as block copy or special store buffers can help improve the performance of unnecessary accesses by transferring data in chunks.

Before exploring the classification of host-network interface, because limitations of host architectures related to the communication paths have caused the host-network interfaces to be developed in several ways, the understanding of these limitations can help to get the reason why the classification is important and how it is focused on. Thus, it can contribute to understand why a good taxonomy is useful in studying the host-network interface architectures and how new approach applies to a number of modern architectures of host-network interfaces.

Thus, after showing the limitation of current connections between a host and a host-network interface, the new taxonomy will be presented.

3.2 Examples of Taxonomy

Good taxonomy should group together those objects that are strongly related in an important way. For example, computer engineers classify computers based on their functional views and on information flow between units. Flynn [35] classifies computer architecture as four categories by the number of instruction and data streams that they can process simultaneously. The categories are:

SISD single instruction stream with single data stream: Traditional sequential computers are based on this model introduced by John von Neumann.

SIMD single instruction stream with multiple data stream: A single control unit dispatches instructions to each processing unit. In an SIMD computer, the same instruction is executed synchronously by all processing units. Processing units can be selectively switched off during an instruction cycle. Examples of SIMD computers include the Illiac IV, MPP, DAP, CM-2, MasPar MP-1, and MasPar MP-2.

MISD multiple instruction stream with single data stream: The same data stream flows through a linear array of processors executing different instruction streams. This architecture is also known as systolic arrays by H.T. Kung for pipelined execution of specific algorithms.

MIMD multiple instruction stream with multiple data stream: Each processor is capable of executing a different program independent of the other processors. Examples of MIMD computers include the Cosmic Cube, nCUBE 2, iPSC, Symmetry, FX-8, FX-2800, TC-2000, CM-5, KSR-1, and Paragon XP/S.

More descriptive classification of the computer architectures has been proposed by D. V. Skillicorn [92]. It extends Flynn's taxonomy, especially in the multiprocessor category. It is also a two-level hierarchy in which the upper level classifies computer architectures based on the numbers of processors for data and for instructions and the interconnections between them. A lower level can be used to distinguish variants even more precisely and based on the state machine level of processors. The scheme can be summarized as follows: The first level describes an architecture by specifying:

- the number of instruction processors, denoted by nIP.
- the number of instruction memories, denoted by nIM.
- the type of switch connecting IPs to IMs.
- the number of data processors, denoted by nDP.
- the number of data memories, denoted by nDM.
- the type of switch connecting DPs to DMs.
- the type of switch connecting IPs and DPs.
- the type of switch connecting DPs to DPs.

The second level refines the first level taxonomy describing whether or not the processors can be pipelined and to what degree, and by giving the state diagram behavior of the processors.

Because there are multiple functional units, connections between functional units are made using abstract switches that can be implemented in four different ways:

- 1-1, a single functional unit of one type connects to a single functional unit of another.
- n-n, the ith unit of one set of functional units connects to the ith unit of another.
- 1-n, one functional unit connects to all n units of another set of functional units.
- n×n, each units of one set of functional units can communicate with any unit of another set of functional units and vice versa.

The main functional units in this scheme are the instruction processors and the data processors. Skillicorn precisely defines these concepts in terms of the set of functions each element performs.

Thus, the functions of the instruction processor are to

- determine the address of the next instruction to be executed on the basis of local state information and the state information passed to it by the DP,
- access the IM to fetch the instruction,
- receive and decodes the fetched instruction,
- inform the DP of the operation to be performed,
- determine the operand addresses and passes them to the DP, and
- receive the state information from the DP after the latter has executed the operation.

The data processor carries out the following steps:

- receive the operation to be performed from the IP,
- receive operand addresses from the IP,
- fetch operands from the DM,
- execute the operation,
- store results in the DM, and
- return state information to the IP.

Table 3.1: A fragment of Skillicorn's taxonomy

class	nIP	nDP	IP-DP	IP-IM	DP-DM	DP-DP	Name
3	0	n	-	-	$n-n$	$n \times n$	loosely coupled dataflow
4	0	n	-	-	$n \times n$	-	tightly coupled dataflow
6	1	n	$1-1$	$1-1$	$1-1$	-	von Neumann uniprocessor
8	1	n	$1-n$	$1-1$	$n-n$	$n \times n$	type 1 array processor
9	1	n	$1-n$	$1-1$	$n \times n$	-	type 2 array processor
13	n	n	$n-n$	$n-n$	$n-n$	-	separate von Neumann uniprocessors
14	n	n	$n-n$	$n-n$	$n-n$	$n \times n$	loosely coupled von Neumann
15	n	n	$n-n$	$n-n$	$n-n$	-	tightly coupled von Neumann

Using this taxonomic scheme, Skillicorn established a single category of 28 classes. Table 3.1 reproduces a small fragment of his taxonomy.

According to the nature of the instruction interpretation and execution cycle, the internal organization of, and the relationships between, storage components and functional units, and the means by which instruction processing is controlled, S. Dasgupta [26] proposed a hierarchical taxonomic scheme using the chemical metaphor.

The taxonomy is built from seven primitive concepts referred to as atoms. Atoms of the same type can be combined into more complex entities called atomic radicals, which in turn can be combined into still more complex concepts called nonatomic radicals. Finally, nonatomic radicals are combined into molecules, which denote entire architectural entities.

The symbols iM, sM, C, sI, pI, sX, and pX identify the atoms as follows:

- M stands for a main memory module for instructions or data. By the prefix, sM denotes a simple memory and represents a potential for a unit of information to be accessed per memory cycle, while iM denotes an interleaved memory and represents a potential for multiple units of information to be accessed per memory cycle.

- C stands for cache.

- I stands for an instruction preparation unit. The symbol sI represents a single-stage instruction preparation unit/processor and pI represents the pipelined instruction preparation unit/processor. The function of an instruction preparation unit is defined by the following set of operations:
  - determine the next instruction to be executed,
  - fetch the instruction from instruction M or instruction C,
  - decode the instruction,
  - compute the effective address of the operands,
  - transfer the operand addresses and the operation to the instruction execution unit, and
  - receive the state information from the execution unit.
- X stands for an instruction execution unit/processor. The symbol sX represents an X that can execute only a single instruction at a time, while the symbol pX represents an X with the potential for executing several instructions at a time. The function of X is defined as following:
  - receive operand addresses and operation from I,
  - fetch operands from data M or data C,
  - carry out the operation,
  - store the result in M or C, and
  - return state information to I.

Except that operands can be fetched from C and results stored in C, X is functionally identical to the DP in Skillicorn's scheme. Similarly, except that instructions can be fetched from C, I is functionally identical to the IP in Skillicorn's scheme.

A replicated atom is termed an atomic radical. The subscript is termed the radical number, for example, pI_m denotes m instances of an instruction preparation unit.

A combination of a C radical or a M radical with an I radical, an X radical, or another combination constitutes a nonatomic radical. By convention, the C or M radical must be to left of the nonatomic radical. When the number of replicated atomic radicals is more than one, the combination is enclosed in parentheses, for example, $(iM)_m.(C.pI)_n$.

An I (X) molecule is a single or a replicated combination of an MCI (MCX) radical that represents a complete instruction preparation (execution) subsystem within a computer at abstraction level (or called endoarchitectural level). A macromolecule is a single or a replicated combination of an I molecule and an X molecule that represents a complete computer at the endoarchitectural level. By notational convention, the I molecule appears to the left of the X molecule.

The structure of a radical or molecule is determined by using replication and link operation. Let head(R) and tail(R) denote, respectively, the largest leftmost radical and the largest rightmost radical in a radical R. For example, $R = iM_m.(C.pI)_n$, then $head(R) = iM_m,\ tail(R) = (C.pI)_n$. By definition of replication, the rep(R) means multiple replicated radicals of the R radical. On the other hand, the link operation has several ways to link between radicals or molecules as following: with radicals of R1 and R2, link(R1, R2) cause

- the simple link if tail(R1) and head(R2) are both nonreplicated,
- the right divergent link if tail(R1) is nonreplicated but head(R2) is replicated,
- the left divergent link if tail(R1) is replicated but head(R2) is nonreplicated, and

- the bidivergent link if both tail(R1) and head(R2) are replicated.

where the simple link is similar to the $1-1$ or $n-n$ connections, the left (right) divergent link is the 1-n (n-1), and the bidivergent link is the $n \times n$ in Skillicorn's switch connections.

Table 3.2 shows the molecular formulas for a variety of well-known computer architectures.

Table 3.2: Molecular formulas for some well-known computers

Name	Formula
Illiac IV	$(sM_{64}.sI)(sM.sX)_{64}$
Cray-1	$(iM.C.pI)(iM.C_n.pX_9)$
Cray X-MP	$(iM_m.(C.pI)_n)(iM_m.(C_r.pX_s)_q)$
CM-2	$(iM.C.pI)(sM.sX)_{64k}$
IBM 3838	$(sM.pI)(sM.pX_7)$
IBM RP3	$(iM.(sM.C.sI)_n)(iM.(sM.C.sX_2)_n)$

3.3 Conventional Taxonomy for Host-Network Interfaces

In this subsection, we review the main taxonomic features of host-network interfaces and identify their limitations. Classification schemes for host-network interfaces have been described in several articles [45, 95, 50, 19]. Even though there are a lot of classification schemes, a formalized taxonomy which can include and classify all of them in terms of proved methods does not seem to exist. The conventional classifications are focused on the aspects described in the following section.

3.3.1 The number of Copies for Data Transfer

Steenkiste *et al* [94] classifies the interface architectures by the number of copies to transfer data from memory into network. He describes several overhead sources such as data copying, buffer management, protocol processing, interrupt handling and system calls, and some other depending on the system. In particular, as networks get faster, data copying and checksumming dominate the other overheads because those operations make heavy use of a critical resource such as memory bus. Therefore, host-network interfaces are classified as one of traditional 5-copy data flow, outboard buffering, and DMA based host-network interface architectures. Similarly, Dalton *et al* [25] analyzes TCP/IP protocol implementation in terms of minimizing data movement in memory.

Figure 3.1 (a) shows the dataflow when sending a message using simple host-network interface. Application data in user space of main memory are copied to one place of a system buffer on kernel space of main memory. The transport and network protocols processes data on the system buffer. The shadow line is the checksum calculation which comes after data copying into system buffer. Then, data is copied into the transmit FIFO on network interface board. Therefore, there are a total of five bus transfers for every word sent.

Moving the system buffer into the network interface board, the number of data transfers can be reduced. Figure 3.1 (b) shows the data flow with outboard buffering. While data is being copied into the system buffer in the network interface, a partial checksum is calculated and after summing the partial checksums the complete checksum is sent to the system buffer. The number of data transfers through bus has been reduced to three. Besides using the bus more efficiently, outboard buffering also allows packets to be sent over the network at the full network-medium rate,

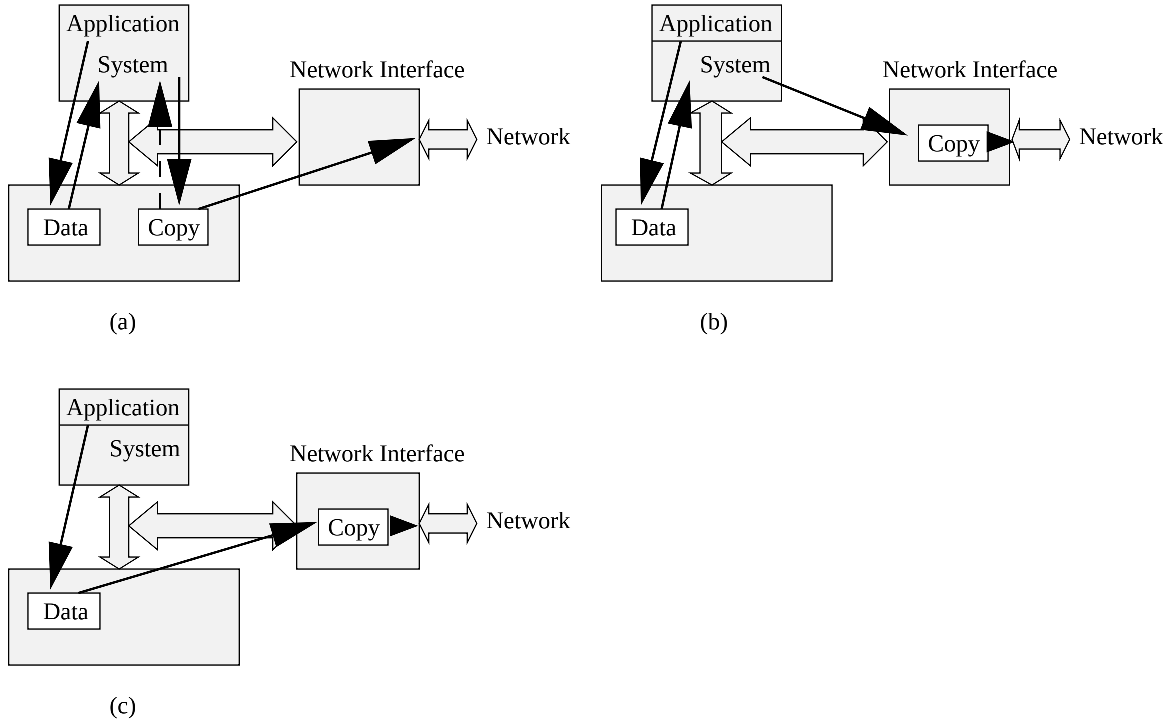


Figure 3.1: Data flows in host-network interfaces

independent of the speed of the internal host bus.

Using DMA (Direct Memory Access) and user-level protocol processing, the number of data transfers can be further reduced. Transport layer and network layer protocols usually have been processed at the kernel-level for data protection or other reasons. This kernel-level protocol processing causes data to be copied from user space to kernel space of main memory. On the other hand, this unnecessary data copying can be removed with protocol processing at the user-level. For example, a daemon owned by root can work as a transport protocol and can solve the data protection problems between user spaces on the memory. Thus, data-copying protocol at user level sends to the DMA controller the start address and the length of intended data on the user-space of main memory. Then, the DMA controller copies data directly from an application space of main memory to the outboard buffer. Moreover, while data is being copied, the checksumming is done with a special hardware, which results in less load on the host. Besides reducing the load on the bus, DMA has the advantage that it allows the use of burst transfers.

3.3.2 Access Control Buffer Size

When a processor receives data from external devices such as network through DMA and device registers for access control, the processor becomes aware of an external event (e.g. message arrival) via interrupts or by polling status registers. Both notification mechanisms are costly because interrupts have high latency and polling wastes processor cycles and other system resources. In Typhoon systems [85], access control logic with cache snooping protocol reduces delay such as zero-cycle access to all protocol state information. Only network-queue and memory-bus interfacing delays are occured. In other words, the cache snooping protocol saves time from the delays for

Table 3.3: Summary of network interface devices

NI/CNI	Accessible Queue Size	Queue Pointers	Home	Name
NI_{2w}	2 words	-	-	CM-5
NI_{16w}	16 words	-	-	Alewife
$NI_{128}Q$	64Byte cache blocks	-	-	StarT-NG
CNI_4	4 cache blocks	-	network interface	Typhoon
CNI_{512}	512 cache blocks	explicit	network interface	Typhoon
$CNI_{16}Q_m$	16 cache blocks	explicit	main memory	Typhoon

frequently-using-control informations between host processor and protocol processor on the network adaptor.

Mukherjee [72] classifies host-network interfaces based on the cache coherent protocol, the size of control information queue, and where the queue is. He denotes the traditional host-network-interface devices as NI_iX an coherent host-network-interface devices as CNI_iX. The subscript i specifies the portion of a network buffer queue accessible to the processor. The default unit of i is memory/cache block unit, but can also be specified in 4-byte words by adding the suffix, w. The place holder X is either empty, Q, or Q_m. Empty for X field represents the simple case without explicit head or tail pointers to manage the network buffer queue, Q represents that the accessible queue can be mapped in memory address space, and Q_m denotes that the queue is main memory. Table 3.3 shows the taxonomic scheme of coherent host-network interfaces.

He has also examined host-network interfaces with how to connect to the host,

- by memory [70] or I/O buses [27, 7]: It is popular host interfacing in conventional host-network interfaces. The host processor is connected to the interface by a port of dual-ported main memory or system bus.

- by processor registers [50]: By adding new registers on the processor, the host CPU store and load network data through those registers. Those registers are connected to the host-network interface.

- by level-1 cache controller [50]: The cache controller control data transfers. At transmitting, the host CPU give command to the controller to send data to the network. At receiving, cache controller store data to the main memory without host instructions.
- by level-2 cache bus [6]: The host interfacing is through cache bus. Data transfer can be fast but can cause cache bus to be overloaded and cache coherence problem.

Examples of this host-network interface taxonomy are listed in table 3.4.

Table 3.4: Taxonomy of host-network interfaces with connectivity

Placing	Name
Memory or I/O bus	OSIRIS
Processor Registers	J-machine
L-1 cache controller	MIT Alewife
L-2 cache bus	StarT-ng

3.3.3 Protocol Co-processing between host and protocol processor

Cooper *et al* [19] classified high-speed host-network interfaces as simple and intelligent network interfaces due to the significant difference in performance and implementation complexity such as protocol processor, memory accessing way, and DMA capability.

To perform the protocol processing on interface board, some kind of controller or CPU must be used. The onboard processor can access the physical network interface with significantly higher bandwidth than the host processor, and it can respond to individual packets more quickly than the host processor because the host interrupt overhead is avoided.

The connection between the host and the host interface is either a FIFO that is accessed via a single address in the host's address space or a network-buffer that is mapped into a potentially large range of the host's address space with random accessibility.

The actual bandwidth of transfers to and from host memory over the bus depends much on the transfer mode, single-word versus burst mode. While the host executes some processes, DMA controller can access the main memory with burst-mode data transfer without host interrupts.

In simple interfaces, the receive architecture handles packet synchronization, CRC checking for the packet header and the payload of VBR (Variable Bit Rate) SAR (Segmentation and Reassemble), and word alignment. The transmitter architecture similarly handles CRC generation, packet formatting and delineation. Together with those features, data is read and written directly by the host processor. On the other hand, intelligent interface likely to yield the highest performance for real-time traffic can perform the SAR processing more efficiently than the host processor, thus the host processor can assign more processing power to application layers. With DMA capability for burst mode data transfer, more bandwidth can be achieved and host processor uses less interrupts.

3.3.4 Memory Access

In [31], the comparison of the DMA performance with programmed I/O is determined by how fast an application program can access the data in each case. Even though reading data into the cache causes a dramatic decrease in throughput from the pure

DMA results, the throughput of the DMA remains above what can be achieved by programmed I/O simply because of the high performance penalty for word-sized reads across the bus.

In [4], memory access is described with address-mapping spaces and data accessing mechanisms to classify host-network interfaces. Thus, there are network interface boards in memory space and in I/O space with programmed I/O, DMA, and burst mode by the memory controller.

The network buffer memory on host-network interface board can be mapped as a part of main memory. In this class, data can be moved between network buffer memory and main memory by the processor using load and store instructions.

With programmed I/O, the processor may read or write from I/O space using single-word load and store instructions, just like the case of memory space. The only difference is the uncached access to the I/O space.

With DMA, an I/O device can read or write main memory directly without involving the host processor. Cache coherence during DMA is maintained by the operating system. When each transfer-intended data is smaller than the transfer unit of a DMA data, DMA can cause data to be delayed to compose bulk of data during transfer.

Finally, an I/O device can use the burst mode data transfer by the memory controller between memory and I/O space. With pipelining, the burst mode by the memory controller can boost up the speed of data transfer.

3.3.5 Host Interfacing

Typical implementations of network interfaces using the I/O bus suffer from bandwidth and latency limitations imposed by DMA start-up times, low I/O cycle times compared to the CPU cycle times, limited I/O address space, multiple data copying, and CPU bus contention. Placing the network interface on the other side of dual-ported main memory can alleviate these problems and it is compatible with new wide-bandwidth memory techniques such as Rambus technique [101]. Using VLSI technology, [50] designs a network interface chip for very high-speed network. Henry and Joerg [50] propose four categories for existing host-network-interface architectures: operating system based DMA; user-level memory mapped; user-level register mapped; and hardwired.

OS-level DMA-based Interfaces The host relegates message handling to the DMA under the operating system's control. At the hardware level, both of send- and receive- machines send and receive messages by initiating a DMA transfer between main memory and the network channel. At the software level, the sending of a message is accomplished by writing the message into the memory and executing a send system call which initiates the DMA transfer from the memory to the channel. Receiving messages also involves the operating system, and requires the program on the receiving host to explicitly perform a receive operation.

User-level Memory-mapped Interfaces Sending and receiving messages can be processed by user level operations and by memory mapping. The important feature of these interfaces is not that the hardware is actually memory mapped, but that the bandwidth and latency of accessing the network adaptor is similar to that of accessing memory. Typically, messages are sent by the user's process composing the message and executing a send command. The host finds out that a message has arrived either by polling to check if a message has arrived or by an interrupt which is generated upon message arrival.

User-level Register-mapped Interfaces While the memory mapped interfaces store or load messages into memory to send or receive, a processor with an on-chip network adaptor can eliminate these loads and stores by mapping the interface into the processor's register file rather than its memory. An incoming message can implicitly appear in a predetermined set of general registers. Words of an outgoing message can be computed directly into other predetermined general registers. By mapping the host-network interface into the register file, network systems at sending and receiving sides allow for low-overhead, high-bandwidth communication of data.

Hardwired Interfaces To execute a function in many communication environments, a hardwired unit can implement the function much faster than a software. The hardwired interfaces completely support the sending, receiving, and the interpretation of arrived messages in hardware. Since the messages are controlled without software intervention, they can be handled very efficiently. These interfaces, however, do not implement the general message passing model or explicit user-level model of the network because they take control away from the programmer and the compiler. This model is appropriate for a shared memory or dataflow machines.

Examples for each of classification categories are shown in the table 3.5. Since the classification is for host-network interfaces of the tightly-coupled processor systems, all examples are related to the parallel computer systems.

Table 3.5: Examples of the classification for tightly-coupled processor-network interfaces

Class	Name
OS-level DMA based	nCUBEs, iPSC/2
User-level memory mapped	CM-5, MDP
User-level register mapped	CM-2, iWARP
Hardwired	Alewife, Monsoon

3.4 Limitations of Conventional Schemes

Although a number of classifications have been proposed for taxonomic features of the host-network interfaces, it seems that these classifications are often designed to support only the argument about superiority of the proposed solution. The conventional taxonomic schemes have some important limitations.

The first limitation is lack of the predictive power. With a given information from the taxonomic scheme, our knowledge of the scheme would then allow us to predict the values of the taxonomic characters corresponding to this class. However, neither the schemes nor their nomenclatures allow us to infer to what extent or at what level two or more architectures belonging to distinct classes are similar or the extent to which different architectures from a single host-network-interface family are similar, without actually comparing the values of the respective taxonomic characters. This is because conventional taxonomic schemes have only one category and, consequently, are non-hierarchical.

The second limitation is the comprehensive power. So far, the communication networks have been developed in local area, metropolitan area, wide area, and inter-processor communication networks, respectively. Thus, the host-network interfaces have deployed adaptively to their network systems. Even though a taxonomic scheme has the predictive power in a narrow scope, it should have the taxonomic characteristics for all kinds of host-network interfaces. However, each taxonomic scheme has selected taxonomic characteristics to provide the superior of its architecture to other architectures. For example, some articles, [50, 72], classify host-network interfaces in tightly-coupled systems, while the others, [19, 4, 94], classify them in loosely-coupled systems.

The third limitation is the systematic power. A computer network system implements proposed protocols based on hardware or software functional units. Architectural taxonomy must consider both aspects of the hardware and the software. However, conventional taxonomic schemes only consider taxonomic characteristics implicitly, called hardware functional units. They might be considered explicitly with separated levels such as an abstract machine and a protocol processing levels.

The fourth limitation is the explanatory power. When the notion of explanation is considered explicitly as a desirable system characteristic, the significance of the various taxonomic characters and their possible values should probably have been more carefully delineated. For example, Steenkiste *et al*'s [94] scheme has little overt explanatory power. However, if it is given the precise characterization of data-copying place with the number of data copying, this scheme has considerable potential, or latent explanatory capabilities.

3.5 Proposed Taxonomy for Host-Network Interfaces

Our approach is based on two classification techniques that have been used to classify and characterize computer systems [92, 91].

Our objective in this research is to develop a taxonomy with the following features:

- Good descriptive capability that can clearly describe the architecture of a host-network interface in short time period and price manner.

- With predictive power, the taxonomy can be used to extrapolate the behavior of a host-network interface and thus can be used to design future host-network interfaces that can efficiently and cost-effectively meet the requirements of a given class of applications.

- The hierarchical feature allows us to introduce more detailed and different levels of abstraction in order to achieve accurate description of the host-network interface architecture and thus, gives better foundation to extrapolate the host-network behavior.

Figure 3.2 shows a block diagram of our taxonomy that consists of two main levels of classification: Architectural and Protocol classifications.

In the architectural classification, we delineate the main components required to transfer data from the host to the network and vice versa, *i.e.* receiving data from the network and delivering it to the host. Furthermore, we delineate at this level how these components are connected as well as how they interact to perform their tasks.

Architectural Level	Protocol Level	
	Transmission	Reception
Host - Network processor	Packetization	Depacketization
Host - Network memory	Data copying	Data copying
Cache - Network processor	Flow control	Flow control
Cache - Network memory	Error handling	Error handling
Main memory - Network memory	Routing	Routing

Figure 3.2: A block diagram of hierarchical taxonomy by architecture and protocol for host-network interfaces

In the protocol classification, we identify all the software functions, required to transfer data from the host to the network and vice versa, *i.e.* the tasks required to receive data from the network and deliver them to the host. In addition, this level shows how each task is mapped into the component identified by the architectural classification. The protocol classification can be further refined by identifying all possible techniques to implement each protocol task. This refinement is useful to guide designers to develop a host-network interface architecture that meets a certain performance and cost objectives.

These two levels of classification provide us with all the information required to describe and thus analyze the performance of a host-network interface architecture.

3.5.1 Architectural Classification

We will approach the architectural classification with the PMS (processor-memory-switch) notation, which describes well the relations between processor, memory, and I/O devices [39, 91]. In PMS notation, a system is described as an interconnected set of components or individual devices, associated with a set of operations. Such a description can be complicated by the amount of detail involved. Thus, the PMS descriptive system permits very compressed descriptions, that is, describes only those aspects of the components that are of interest, while ignoring the rest. It also permits the analysis of the amounts of information held in various components, the flow of information between components, and the distribution of the control that accomplishes the flow.

Note that the physical communication system consists of hosts, host-network interfaces and a network. Thus, the physical characteristics result from these logical structures and their connections. There are six types of functional units from which any architecture of the host-network interface can be constructed. These are host CPU, main memory, cache, network buffer, protocol processor, and a switch:

- Host CPU unit is to execute instructions of the user application which may require a communication to, if required, the protocol processing related to the communication, and to transform data usually in ways that correspond to basic arithmetic operations.

- Main memory unit is an intelligent storage device that passes data to and from the host CPU.

- Cache unit consists of a small fast memory that acts as a buffer between the main memory and the host CPU.

- Network buffer unit is a staging and speed-matching area for data in transit between the host and the network. It consists of a network buffer memory and network FIFOs. Network buffer memory is the storage for transport-layer data (so called message) and information related to the control and management parameters, while network FIFOs are two set of registers for sending and receiving. It also provides the protocol processor with contention-free memory access to the packet data.

- Protocol processor unit manages packet processing and various bookkeeping functions associated the protocol.

- Switch unit provides connectivity between other functional units in one way of programmed I/O, DMA (Direct Memory Access), burst transfer by memory controller, or register accessing.

3.5.2 Protocol Implementation Classification

The main protocol tasks are packetization/depacketization, error handling, flow control, and routing. In what follows, we describe each of these tasks and the different mechanisms or techniques to implement each tasks.

Packetization/Depacketization Unit

Functionality: Since received data might be different from those sent to the network, data is bundled to catch such errors and it is transmitted as packets of bits with some overhead bits.

A packet consists of a sequence of bytes with a header, an intended data (payload), and with or without a trailer. According to the routing unit and buffer-size limits in the sender and receiver, the intended data size can be fixed or variable. If an application program sends a message larger than the packet data size, then the operating system or a protocol processing unit fragments the message into a series of packet data and reassembles the packets into the message in the receiving process.

Packets may arrive for several receiving processes in one host, and it would be desirable to receive data directly in the receiving process' address space. Most medium interfaces put received packets in the first available buffer on the queue (*i.e.* network buffer FIFO), without interpreting the packet contents. The depacketization, also called data composing, of the incoming packet stream, therefore, has to be done by a protocol engine and copying message to the appropriate location in the receiving application process's address space is usually unavoidable.

Classification: The packetization functional unit can be classified by the processing unit and by packet length as follows:

1. A variable-sized data unit packetized by the host.
2. A variable-sized data unit packetized by the interface.
3. A fixed-sized data unit packetized by the host.
4. A fixed-sized data unit packetized by the interface.

Data-Copying Unit

Functionality: The path taken by the data as it progresses through a conventional protocol stack is illustrated in [4, 94]. The application writes data into its buffer and then invokes a system call to send the data. A typical interface between the application layer and the transport layer is the socket function. The socket copies the application data into a buffer in kernel space called the socket buffer. Then, the transport layer reads the data in the socket buffer to compute a checksum and finally the data is copied out to the network interface using programmed I/O, Direct Memory Access (DMA), burst transfer by memory controller, or register file on host processor. Thus, the memory system is accessed five times for each word of intended data. Similarly on the receive path, data is copied first from the network interface (FIFO) into kernel memory using programmed I/O, DMA, burst transfer by memory controller, or register file on host processor. The transport layer checksum is verified and when the application layer is ready, the socket function copies data from the socket buffer in kernel memory to the application buffer in user memory space. Finally, the application process reads the data. Thus the memory system is also accessed five times for each word of intended data.

There are several ways to reduce the number of data copies between main memory in the host and the network adaptor. Few of these techniques are illustrated in Figure 3.3 through Figure 3.6 showing the path of data in pictorial.

Classification: 1. 0-copy from user space on main memory to network:

- The host processor informs the intended data address and length to the network protocol processor.
- The network protocol processor initiates DMA controller to transfer data from user space of main memory to network.
- Data flow is described as arrows on Figure 3.3.

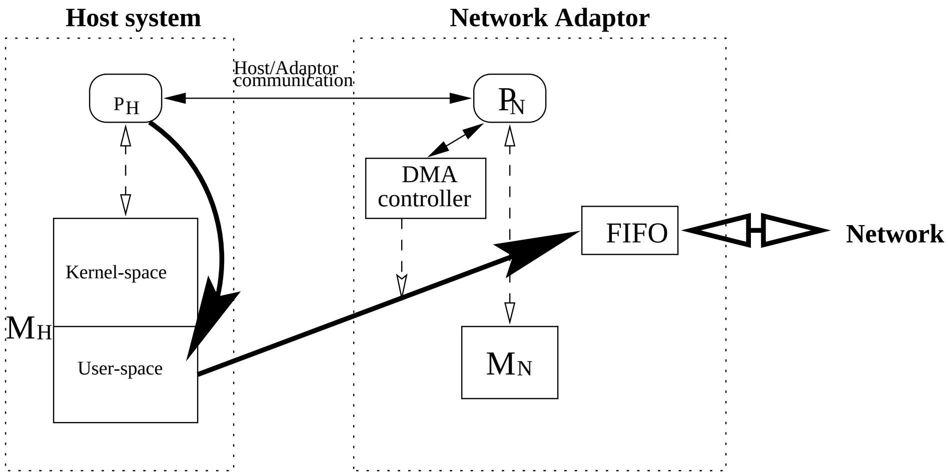


Figure 3.3: Dataflow in network interface with zero copying

2. 1-copy from user space of main memory to network buffer memory, then to network:

   - After data is copied to network buffer memory, network interface processes some other protocol processing such as CRC and composing of header fields.

- This is showed on Figure 3.4.

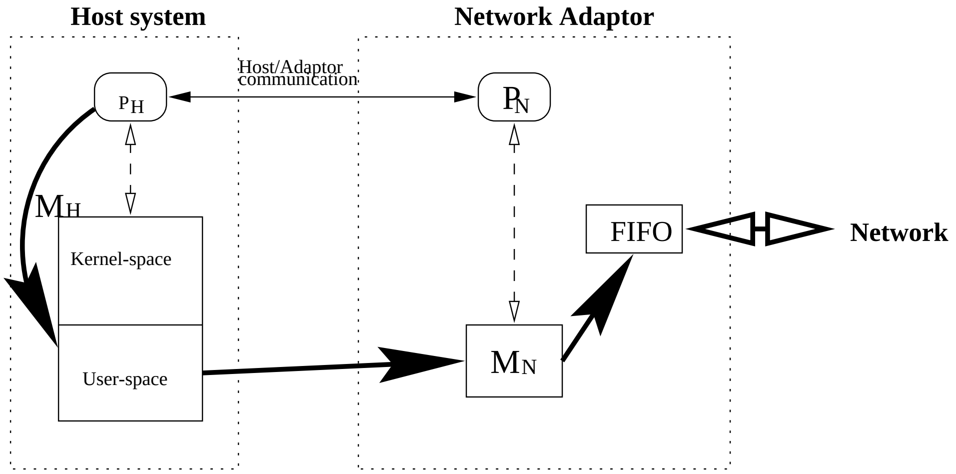


Figure 3.4: Dataflow in network interface with 1-copying from user space in memory to network buffer memory

3. 1-copy from user space of main memory to kernel space of main memory:
   - To process protocol, data is copied to kernel space.
   - Good security and protection of data between users.
   - Figure 3.5 shows the pictorial scheme of this category.
4. 2-copy from user space to kernel space of main memory then to network buffer memory:
   - Transport protocol processing on kernel space except CRC checking which is supported on host-network interface.
   - Figure 3.6 shows the data flow of this scheme.

Flow-Control Unit

Functionality: The objective of the flow control is to ensure the sender the flow of offered-data load just enough to achieve a throughput that is very close to that

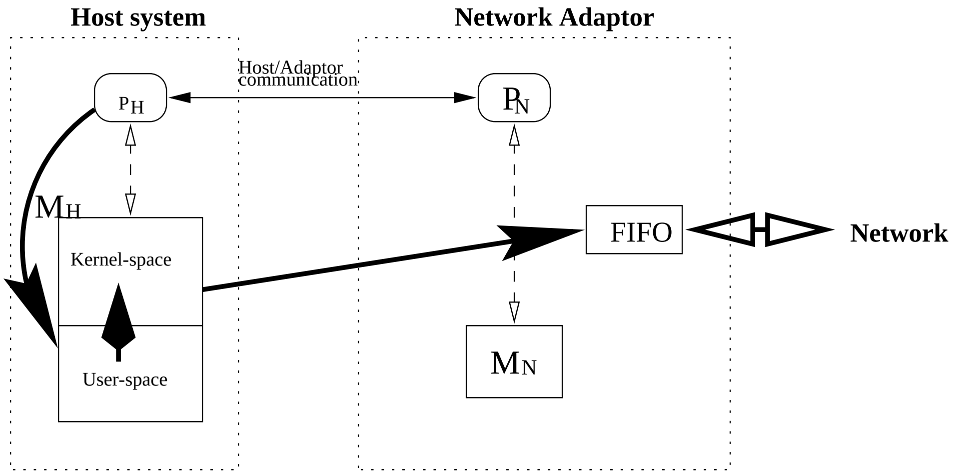


Figure 3.5: Dataflow in network interface with 1-copying from user space to kernel space in main memory

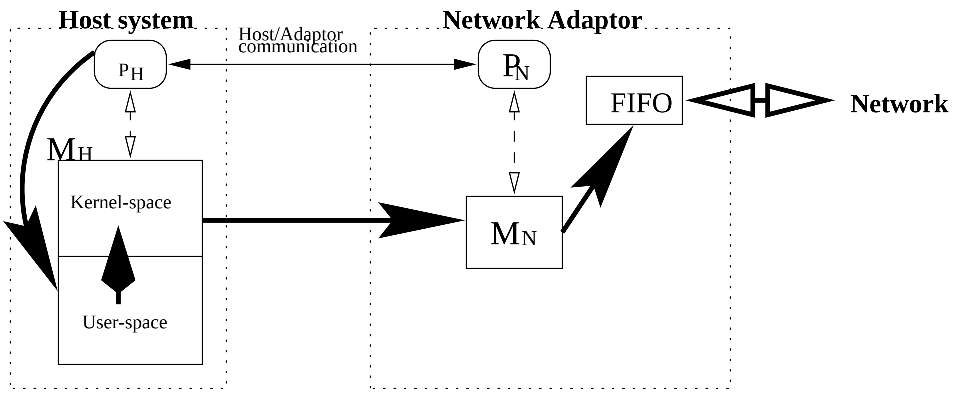


Figure 3.6: Dataflow in network interface with 2-copying

of the resources' capacity at which the receiver can receive, process, and forward the data to its user, with very low loss. This requires cooperation between the users and the network. The network notifies congested users in a timely fashion, and then, the user's application reduces the flow accordingly.

At a more detailed level, the receiver decides the optimal capacity of packets by the number of connected channels, protocol processing, buffer availability and so on.

The host-network interface on the receiver side usually detects the time, burst size, and acknowledgements of packets, while the host of receiving side reports the flow-control states to the transmitter. The state-reporting is a mix of acknowledgements, packet arrival rate, credit signal, or control symbols. Then, the transmitter extracts the state-reporting parameters of flow control from the opposite link and applies usually to a hardware on the host-network interface to control the packet flow of an intended channel.

There are, generally, three methods of flow control: window-based, rate-based, and credit-based. Window-based flow control, [56, 3], limits the amount of source transmit data that can be sent, called the window, and dynamically adjusts the window size based upon feedback. Rate-based flow control, [76], dynamically adapts the source transmit rate in response to feedback. The transmitter needs to know the transmission rate and the burst size of packets. But VMP NAB uses the interpacket time instead of burst size. Credit-based flow control, [64], is a scheme which returns permission (or credit) to send data to a source end or intermediate node. In addition, Myrinet and MINI (Memory-Integrated Network Interface) uses flow-control symbols such as START and STOP.

Classification: Receiver flow control schemes (RFC):

1. Maintaining a receiving window based on acknowledgements and sequence numbers.
2. Packet counting by software on the host processor.
3. Packet counting by hardware on the host-network interface.
4. Inter-packet time by hardware on the host-network interface.

5. Packet accepting/discarding based on buffer availability.

State-reporting schemes (SR):

1. Acknowledgements based on window.
2. Packet arrival rate based on inter-packet time or burst size of packets.
3. Start/stop control symbol.

Transmitter flow control (TFC):

1. Sending packets based on window.
2. Control the rate of transmitting packets.
3. Detecting control-symbols/controlling the flow of sending packets.

Error Handling Unit

Functionality: The error handling of high-level protocol layer must perform error detection, reporting, and correction if the network is not reliable. however, the error handling of low-level protocol layer is up to the protocol implementation. For example, ATM network handles errors only for header fields, not for payload part. On the other hand, Ethernet handles any errors in the low-level protocol layer (data-link layer level).

Error detection is performed by means of sequence numbers, length fields, and checksums. Sequence number can be used to detect lost and miss-routed delivery, and to protect against duplicated informations. Even in case of complete delivery of data through the network, the data can be verified by means of length fields in packet headers or trailers to detect errors. The basic method enabling the receiver to check for corrupted delivery is the transmission of redundant information on the CRC field along with the data to be protected. To support high-speed networks, some hardware components complemented by software algorithms help to check CRC fields because conventional high-speed networks carry small fixed-length packets and require fast routing (Cut-through routing).

Error reporting mechanisms serve to explicitly inform the sender about errors detected by the receiver, then the sender retransmits the erred data to the receiver. Error reporting by the receiver is desirable to expedite error correction. The error reporting should indicate the receiver's complete state with respect to received data such as gaps if out-of-sequence data is buffered, negative acknowledgements if out-of-sequence packet is received.

Error correction method used by almost all protocols to recover from errors is retransmission of the corrupted or missing data. According to the error reporting, there are two methods: Forward error correction (FEC); and backward error correction, which is so called automatic repeat request (ARQ). With FEC, the receiver itself can correct some errors in terms of polynomial coding technique such as Hamming code. The CRC checking unit can correct single-bit error(s) according to the number of digits in the polynomial code without any retransmission from transmitter. It might reduce the overhead resulted from retransmission. But this technique requires more processing power from the receiver processor. Even if this technique could serve to recover from any type of one-bit errors, it did not work by itself. With ARQ, data are retransmitted based on control information. This control information is obtained from the receiver side.

Classification: Error-detection functions (D):

1. CRC
   - Type: CRC-8, CRC-10, CRC-12, CRC-16, CRC-32, or CRC-CCITT.
   - Domain: A control part, A data part, or A control + data part.
   - Implementor: Hardware or software.
2. Length indicator of data.
3. Sequence number.
4. Multiplexing identifier (MID).

In the domain part of CRC, a control part means the control information in a packet such as header. A data part also means user information part such as cell payload in ATM network.

Error-correction functions (C):

1. Forward Error Correction (FEC).
2. Automatic Repeat Request (ARQ).
3. FEC + ARQ.

Routing/switching Unit

Functionality: When an application process tries to setup a connection to another application process, it must specify which process and destination process to connect to. The address generating (or address resolution) functions work to generate the source address, destination address(es) on network and intermediate-node addresses if needed.

The transport protocol address is the application port number on which the application process is running. Also network protocol address consists of host number and network number as well as transport protocol address. On the other hand, data link protocol address comes from frame numbers. Therefore, all elements to compose addresses upon establishing a connection and on sending packets are, at least, port numbers, host numbers, network numbers, and frame numbers [99].

In the Internet as an example, TCP addresses are a pair of IP addresses and port numbers each for source and destination applications. An IP address is also a pair of a network number and a local host number, and each host should have at least one unique network address. On the other hand, a data link protocol address consists of a pair of a local frame number and a neighbor frame number, not the one of destination host.

In the classical approach of the circuit switching transfer mode, a physical circuit is occupied for the complete duration of the connection. In packet switching transfer mode, user data is encapsulated in packets which are containing additional data in header of packets to be used inside the network for routing, error handling, flow control, and so on. The physical circuit is shared with other connections and is occupied only short time for each connection in circulating order, no matter when the connection use the time slot. These packets have a variable length and thus require a rather complex buffer management. According to the routing information on the header of each packet, the source-end can generate the destination address as well as intermediate node port addresses. This is called source routing information scheme. On the other hand, the virtual circuit switching multiplexes statistically data from each connection for better network utilization. Thus the physical circuit is only occupied at a time for connections which require data transfers. Also, the virtual circuit switching reduces the protocol processing such as error detection and correction in data link layer.

Classification: Therefore, the routing/switching functional unit can be classified as the following:

1. Circuit switching
2. Source routing
3. Packet switching
4. Virtual circuit switching (statistical switching)

3.6 Illustrative Examples

In this section, we show that our taxonomy can be applied to a representative set of host-network interface architectures. We also discuss its novel features to describe the host-network interface architecture at protocol implementation as well to predict the performance of the host-network interface architecture under consideration.

AURORA

Figure 3.7 shows the architectural classification of the AURORA OSIRIS or ORBIT host-network interface [8]. Even if this is one of the most complicated host-network interfaces, the PMS notation can simplify it's description with only architectural characteristics, capturing all selected features.

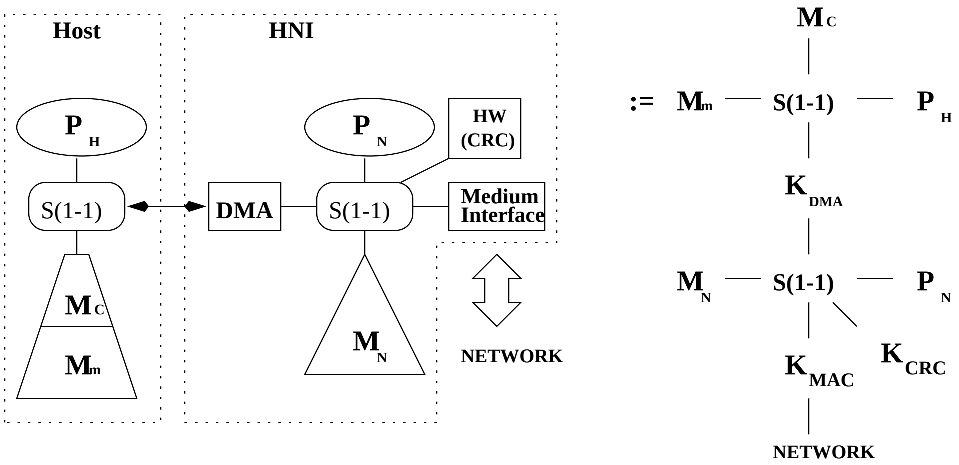
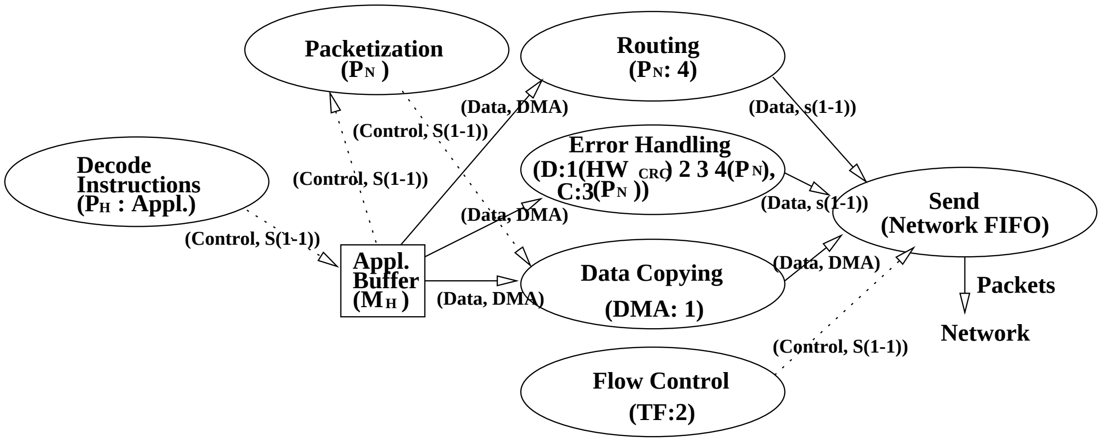


Figure 3.7: The architectural classification of AURORA host-network interface

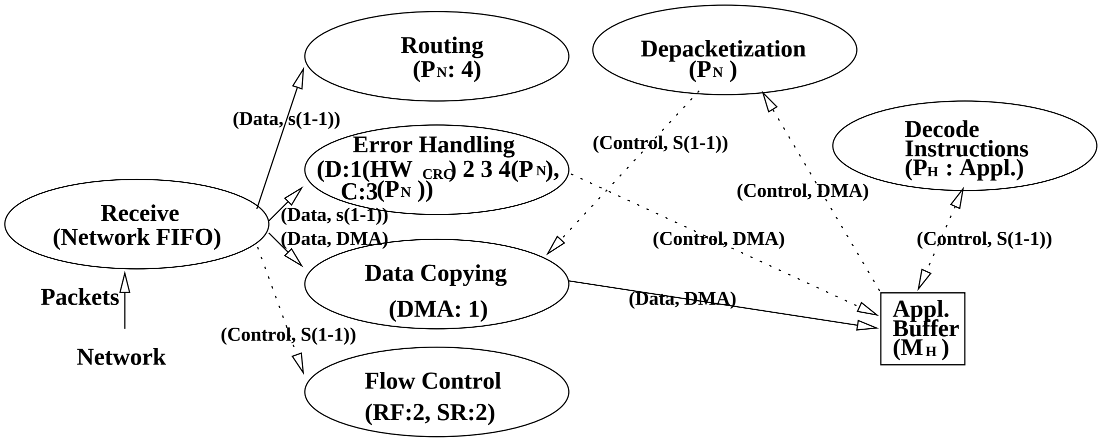
The only direct communication between a host (P_H) and a protocol processor (P_N) is performed via buffer and status/control registers in order to maintain control information and network parameters. A program running on the host can prepare commands and data to be transmitted for the protocol processor by placing them in the buffer space of the main memory. The protocol processor can also prepare status information and received data for the control programs by placing them in the buffer space of the main memory.

In contrast to the connection between P_H and P_N, the connection between M_H and M_N on AURORA provides the ability to transfer a block of data between M_H and M_N without P_H intervention. This requires that the DMA controller should be capable of generating memory addresses and transferring data through a system bus, *i.e.* it must be a bus master. The P_H is still responsible for initiating each block transfer. The DMA controller can then carry out the transfer without further program execution by the P_H. P_H and DMA controller interact only when P_H must yield control of the system bus to the DMA controller in response to requests.

As shown in the architectural classification, an example has captured the behavior



(a) Transmission



(b) Reception

Figure 3.8: The protocol classification of AURORA host-network interface

of protocol functional units, using AURORA host-network interface.

Figure 3.8 (a) shows the functions of protocol classification at transmitting data into a network:

- Initialization: Data transmission is initialized from the host's instructions. It causes M_N to store network parameters, and status and control information.

- Packetization: P_N generates the address and length of data in the main memory which are computed from the information given by the host. These address and length of data are informed to the DMA controller.

- Data copying: P_H gives to P_N the information which is the memory address and offsets for ATM payloads, (ATM protocol data units). After P_N segments data, DMA controller transfers data on the user space into the send FIFO which is a part of M_N. This transfers data directly from user application space of main memory into the network.

- Flow control: Cell count signals from the receiving Connection Management Table and cell arrival rate from the P_H is used by P_N to control the flow of ATM cells.

- Error handling: During data transfer from M_H to M_N, checksumming is executed for data and cell header error is checked with the information on M_N by the CRC-8.

- Routing: Network addresses include application port numbers, host numbers, network numbers for both source and destination. these information is stored in the Connection Management Table at the M_N after connection establishment.

- Sending: Data link level protocol send ATM cells into the synchronous optical network.

Figure 3.8 (b) shows the state diagram of receiving protocol functions. That is, at receiving data from network, the functions of protocol classification are:

- Receiving: Data link layer protocol processes ATM cell receiving from ATM switch. The received cell is stored in a receiving FIFO.

- Error handling: The CRC checksum unit compares a header CRC value from the cell header field with the cell header. Non-consecutive 1-bit errors can be corrected by the CRC algorithm in a limited number. CRC-32 unit executes the checksumming for the cell payload but the comparisons are executed on the P_H.

- Flow control: P_N counts the cell arrival rate from the state information for the threshold of the receiving FIFO.

- Routing: P_N has the routing information from received cell header. Thus, P_N extracts VCIs (virtual channel identifiers) and sequence numbers. If the cell does not have correct routing information, the cell is discarded. Also, P_N uses application port numbers to decide the receiving application space during data transfer.

- Data copying: According to the user port number and sequence number given from the P_N, DMA controller transfers data into the user space of main memory.

- Depacketization: Each decomposed data (cell payload) are by the P_N reassembled with each other according to the header and AAL information. P_N generates the proper address in main memory for reassembed data. The addresses for reassembled data is informed to the DMA controller to transfer into the user space of main memory directly.

- Interrupt: After transferring of a whole message, DMA controller interrupts P_H to alarm that data is safely stored in main memory.

As we discussed previously, these two classifications can clearly describe how the host-network interface can perform the tasks required to transmit and receive data.

Furthermore, our taxonomy can provide hints for the performance and for limitations of the architecture (Figures 3.7, 3.8):

- Even though this architecture is the one of the most complicated host-network interfaces, the PMS notation provides comprehensive descriptions of architectural characteristics.

- The protocol classification state diagram shows the parallel protocol processing between error handling and data copying.

- The state diagram also clearly shows how many number of data copying and where/when happen.

- The provider of the protocol processing is shown on figure.

We can also use syntax notation to classify this host-network interface (Figure 3.9). As shown in the previous section, RFC means receiver flow control, SR for state report, TFC for transmitter flow control, and D/C in error handling means for detection/correction, respectively. In architectural classification, "reg" denotes register.

Protocol Level				
Packetization	Data copying	Flow control RFC/SR/TFC	Error handling D/C	Routing
4	1	2/2/2	1,2,3,4/3	4

Architectural Level				
$P_H - P_N$	$P_H - M_N$	$M_C - P_N$	$M_C - M_N$	$M_H - M_N$
reg				DMA

Figure 3.9: The syntax notation of AURORA host-network interface

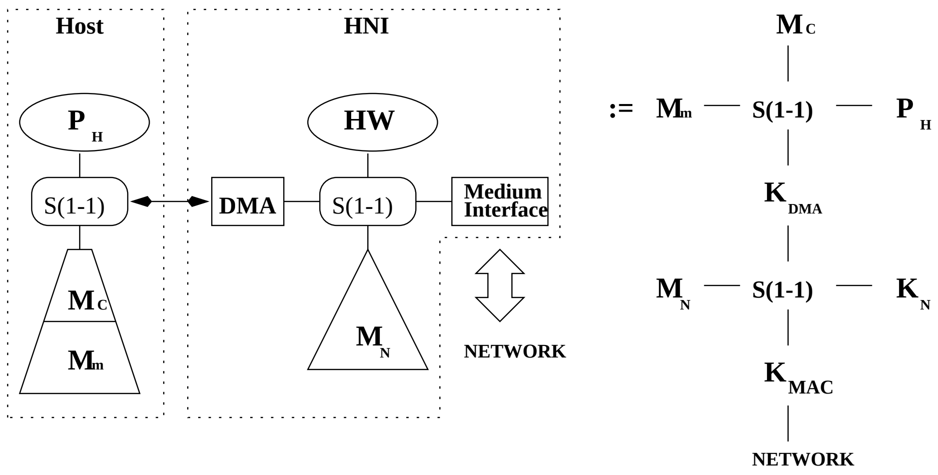


Figure 3.10: The architectural classification of Western Digital WD2840

WD2840

Following in analogous, we can classify other architectures. For example, Figure 3.10 shows a block diagram for architectural host-network interface proposed with Western Digital's WD 2840 Ethernet chip [86]. Figure L2-DMA-uC shows the protocol classification of this design.

In Figure 3.10, a DMA data transfer is controlled by a controller which is not designed to execute commands fetched from the main memory and it can not operate as a processor on the host's system bus. The processing orders for the controller from programs running on the host are passed by writing order codes at fixed addresses to control registers located on host-network interface. Data transfer is between the host

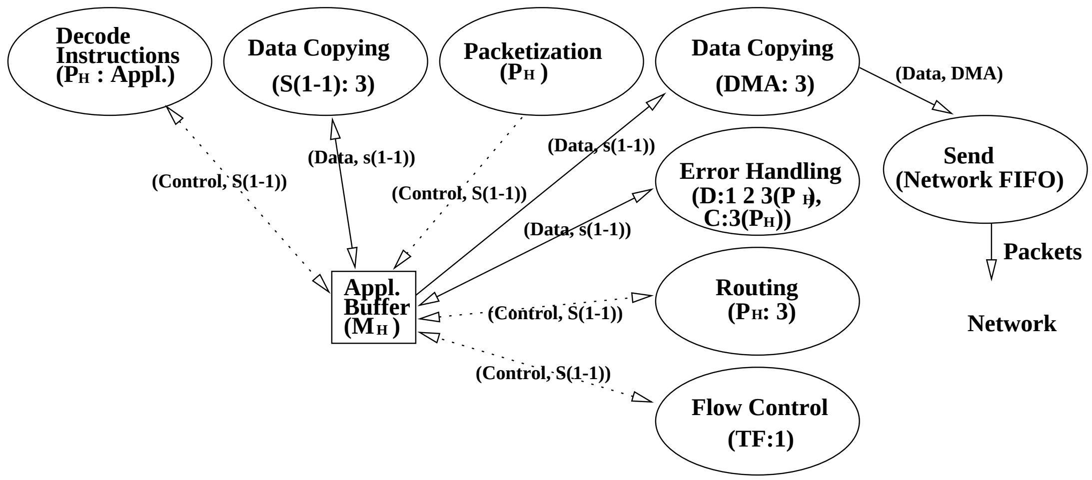


Figure 3.11: The protocol classification of Western Digital WD2840

main memory (M_H) and network FIFOs using DMA.

Figure 3.11 shows that the host-network interface implements CRC-checksumming, data copying from kernel space of main memory to network FIFO, routing, and medium interfacing (in data-link layer). It shows the reduced data copying as three memory accesses during protocol processing to send each data unit since both CRC-checksum and data copying works simultaneously.

Jaguar

Matt Welsh *et al* [102] at University of California at Berkeley have presented a new mechanism, Jaguar. It provides Java applications with efficient access to system resources such as network interfaces while retaining the protection of the Java environment. In fact, implementing efficient communication in Java requires both fast access to low-level system resources and direct manipulation of memory regions external to the Java heap such as communication buffers.

The direct and protected access to system resources is accomplished through compile-time code transformation which maps certain Java bytecodes to short, inlined machine code segments. It is called a Jaguar code mapping concept.

Jaguar also allows Java applications to directly manipulate memory outside of the Java heap such as specially-allocated buffers for communication. This eliminates the expense of copying data between Java and external memory which is outside of the Java heap.

An example of Jaguar, JaguarVIA, is implemented as a Java interface to the Berkeley Virtual Interface Architecture (VIA) communication layer [10, 66, 1].

Berkeley VIA is implemented over the Myrinet system area network. Thus, the architectural context in our taxonomy is the same as Myrinet's host-network interface.

- Host processors are dual intel PentiumII processors with Linux operating system.
- Protocol processor is the Myrinet's LanAI.
- Switching components are programmed I/O for control and status information (transmit and receive descriptors and doorbells), DMA for data transfers.

Figure 3.12 shows the architectural context of JaguarVIA. A user application in Host makes transmit descriptor which has data location and size in main memory. And it sends a transmit doorbell, the pointer of the transmit desscriptor, to the protocol processor in the host-network interface board. Thus, the protocol processor can directly access the main memory without copying data.

In the protocol implementation context of our taxonomy, JaguarVIA has the following features:

- Packtization: A packet has a variable sized data unit and is composed by the host.

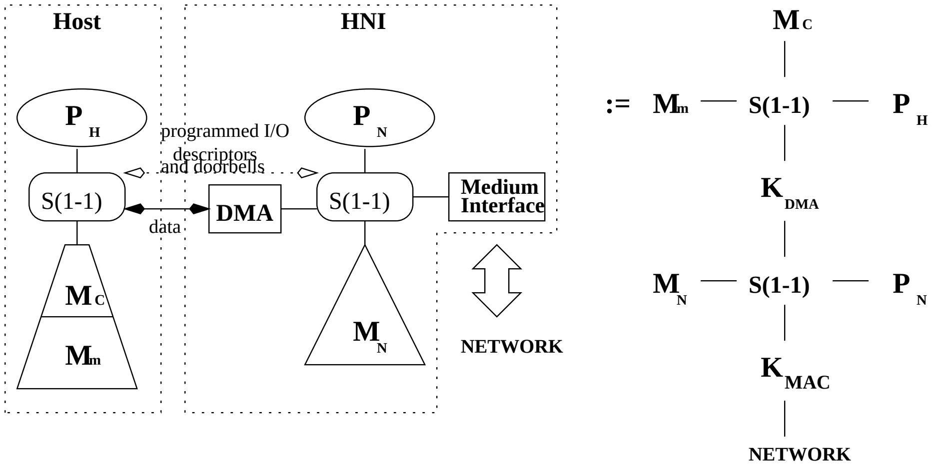


Figure 3.12: The Archtectural Classfication of JaguarVIA

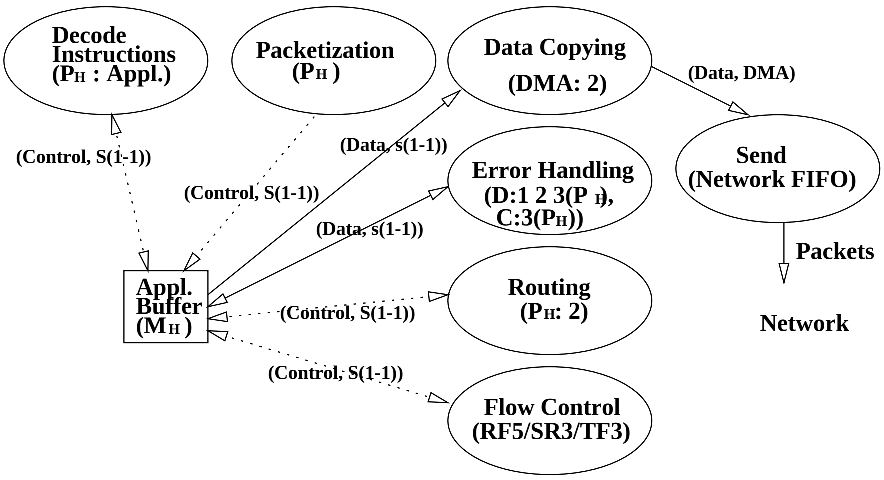


Figure 3.13: The protocol classification of JaguarVIA

- Data copying: In user application, Jaguar implements pre-serialization of Java object codes. Also, by External objects mechanism, a user application of Java does not need to copy data to kernel space. Data is transfered by DMA directly between network FIFOs and user space in main memory.
- Error handling: The 8-bit CRC is checked on the entire packet, including the header, and is carried in the packet trailer. This CRC is computed by hardware in the host-network interface board. The user application includes length indicator of data and sequence number in the purpose of error handling.
- Flow control: JaguarVIA does not support flow control mechanism. In Myrinet system area network, the data flow is controlled on every communication link. At the receiver, the packet is accepted or discarded according to buffer availability and the state reporting is by start or stop control symbol. When the transmitter get the start or stop control symbol from the receiver, the flow of packets are controlled.
- Routing: With cut-through routing, the packet is advanced into the required outgoing channel as soon as the header is received and decoded.

JaguarVIA accesses low-level of machine resources without system calls which are issued by the operating system, while Java native methods have the communication system that copies data from a user space into a kernel space in main memory. As examplified in Jaguar, direct accessing machine resources, which is developed from Java native methods by inlining of Java codes, our taxonomy shows that a communication system can be improved in performance, associated with only high-level protocol processing. This leads us to an understanding that performance of the communication system can be achieved only by optimizing application programming interface (API).

Table 3.6: The syntax notation of conventional host-network interfaces

	Architectural Level					Protocol Level				
Name	$P_H - P_N$	$P_H - M_N$	$M_C - P_N$	$M_C - M_N$	$M_H - M_N$	Packetization	Data copying	Flow control RFC/SR/TFC	Error handling D/C	Routing
WD 2840		PIO			DMA	1	3	1/1/1	1,2,3,4/3	3
Medusa	PIO	B_{mc}	PIO		B_{mc}	1	4	1/1/1	1,2,3/3	3
VuNet					DMA	3	3	1/1/1	1,2,3/3	4
Yes V2	reg	DMA			DMA	4	3	3/2/2	1,2,3,4/3	4
PHNI	PIO	PIO			PIO	4	2	1/1/1	1,2,3,4/3	4
MINI		PIO		B_{cc}	PIO	4	3	5,1/3,1/3,1	1,2,3,4/3	4
Myrinet	PIO	PIO			DMA	1	2	5/3/3	1,2,3/3	2
Nectar CAB	PIO	DMA			DMA	4	2	1/1/1	1,2,3/3	2
VMP NAB	reg	PIO			B_{mc}	2	2	4/2/2	1,2,3/3	3
SHRIMP				B_{cc}	DMA	2	3	1/1/1	1,2,3/3	3
ORBIT	reg				DMA	4	1	2/2/2	1,2,3,4/3	4

Syntax Notations for Host-Network Interfaces

Table 3.6 shows our syntax notation for host-network interface designs that have been reported in the literature. As shown in the table, in the architectural classification, "PIO" stands for programmed I/O, "DMA" for direct memory access, "reg" for register, and "B_x" for burst mode transfer by either of memory controller or cache controller. In the protocol classification, RFC under flow control stands for receiver flow control, SR for state report, TFC for transmitter flow control, and D/C in error handling means for detection/correction, respectively.

3.7 Summary

I have described a classification for host-network interfaces that is considerably more comprehensive and discriminating than those proposed in the past. My taxonomy extends conventional schemes by better discriminating between architectures and between protocol processing, as proposed in this thesis.

The taxonomic scheme includes hierarchical structure in architectural classification. It results in understanding similar interface families by showing underlying relationships. With it's predictive power, the taxonomy also suggests a number of unexplored architectural versions that might be outstanding to look for new, innovative designs. It also classifies almost all of the common host-network interfaces that support local area networks, telecommunication networks, and networks for parallel and distributed systems. The taxonomy is divided into two levels, and each taxonomic characteristics are carefully selected to reduce the implicit concepts of software protocol implementations as possible.

At the architectural classification level, architectures are distinguished by the host processor, the protocol processor, main memory, cache, network buffer memory, FIFOs, and their connections. At this level, the connection is denoted by programmed I/O, DMA, burst transfer by memory or cache controller, or register. At the protocol implementation classification level, further discriminations can be made by describing whether each of the protocols is processed by hardware or software, each of the protocol functional units is in the host or in the network adaptor board. In this classification, the functional units are routing/switching, data-copying, flow-control, and error-handling. With this scheme, transport protocols can be fully classified.

To be useful, a taxonomy must shed light on what is already known and help in assimilating new understanding. I believe that the proposed taxonomy is a natural extension of conventional classifications and that it is straightforward enough to be used as an intellectual tool for understanding, and as an engineering tool for the design of future host-network interfaces.

Chapter 4

Taxonomy-based Designs of Network Adaptors

In the chapter 3, we have described the new approach to designing host-network interfaces. In the proposed approach, we base network design on the new detailed taxonomy we have developed.

In this chapter, we apply this methodology to design host-network interfaces. To demonstrate effectiveness of our approach we have designed two host-network interfaces: a simple and economical interface; an intelligent interface with protocol processors and the burst mode data transfer capability. Both interfaces support asynchronous transfer mode (ATM) network and ATM adaptation layer 5 (AAL-5).

4.1 Simple Host-Network Interface Design

The objective in designing the simple network interface is to design a host network interface which supports a cost-effective system that require minimal hardware support or be used privately. There is a couple of points to be made regarding this consideration.

First, a host and a network should be connected by a fast-but-cost-effective component through the system I/O peripheral bus such a DMA (Direct Memory Access). In the architectural classification, this simple host-network interface is advantageous because it does not need an on-board protocol processor that is the most costly component in an interface design.

Second, the simple host-network interface should be easily adjusted to the changes of environment when there is a need. This flexibility of the host-network interface can be obtained by supporting transport protocols without modifying the host-network interface architecture. When an application requires two different transport protocols, the host executes those protocols. Thus, the host-network interface does not need to be modified.

With the advantage of the low cost and minimal protocol processing on the host-network interface, this simple interface can be used for low speed network and multiple protocol support environments such as personal computers or inexpensive workstations. The interface will be described according to the two-layer model used in our taxonomy.

4.1.1 Design Requirements

In chapter 3, our taxonomy has two layers, architectural and protocol implementation classification layers. Thus, we will describe the design requirements in two taxonomical contexts.

Design Requirements in Architectural Context

This design requires a connection component at the host-network interface by the architectural classification of the proposed taxonomy. This connection component affects mostly the performance of the host-network interface system. It can be the bottle-neck of the whole network system compared to fast-enough network medium.

On typical system peripheral buses of workstations or PC's, peak bandwidth can be achieved by some form of burst mode transfer. Because the interface should share the system peripheral bus with local peripheral devices, the connection component is required to have the comparative bandwidth to that of the system bus. Thus the connection component should be designed very carefully.

The error checking on the cell header requires fast processing. If a cell arrives before error checking on the previously arrived cell is completed, the newly arrived cell may be lost. Therefore, the interface is required to support at least the CRC checking for header part on each cell. Because the header is only five-bytes long, this component can be implemented by a low-cost hardware. Also, it is possible to implement the error detection and correction of a message data by the host since the payload is 48-bytes long and CRC checking on transport layer depends on transport protocol. For example, on AAL3/4 each cell has the CRC field for payload but on AAL5 only the last cell has the CRC field for the transport protocol data unit.

The simple interface requires neither on-board processor nor state information which is necessary to be maintained between cells. Such a host-network interface presents individual cells to the host and the ATM protocol processing is done by the host.

Design Requirements in Protocol-implementation Context

This design, also, has the following requirements on protocol processing based on our taxonomy:

Routing/switching In ATM network, the virtual circuit switching (asynchronous transfer mode) is required. Thus, the host needs to generate only the sender's and receiver's addresses. The addresses should be stored in main memory as cell header parameters.

Data copying Data copying should be reduced to zero with transport protocol processing in user space of main memory because it causes the system performance to be degraded.

Error handling The host is required to detect, report, and correct errors for each cell. But CRC checking for cell header should be faster than CRC checking for a message. Also, it is not efficient to load the host because the host consumes so much processing power on interrupting system and computing every CRC checking for cell headers. Thus, it is better to implement CRC checking for cell headers by a hardware component on the host-network interface.

Flow control Since this design is not for high performance system, the host can implement the flow controls of incoming and outgoing data. For this design, the rate-based and window-based flow control can work well.

Packetization The host is required to segment and reassemble data in fixed size since there is no need of on-board protocol processor as well as since the status and control information is stored in main memory.

4.1.2 Taxonomy-based Design Approach

According to a new taxonomy given in the previous chapter, the simple-interface design supports the following taxonomic schemes:

- In the architectural context,
  - the only connection is between main memory and network FIFO in terms of direct memory access: DMA for $M_H - FIFO$.
- In protocol implementation context,
  - Virtual circuit switching.
  - Data copying between user space of main memory and network FIFO.
  - CRC-8 by hardware for header error check, CRC-10 by software for data error check, sequence number, MID, and length indicator of data.
  - Window-based and rate-based flow control

On the above paragraph, we showed the concised taxonomical contexts. In next subsections, we will see them more in deatil.

Architectural Classification

In the context of architectural classification layer, the simple interface is structured with transmit FIFO, receive FIFO, CRC-8 for header error check, and a simple control logic to directly access main memory. Figure 4.1 shows the simple ATM host-network interface architecture.

Thus, in this simple interface architecture, the host executes any protocol processing in software. The host network interface works only for CRC checking of header

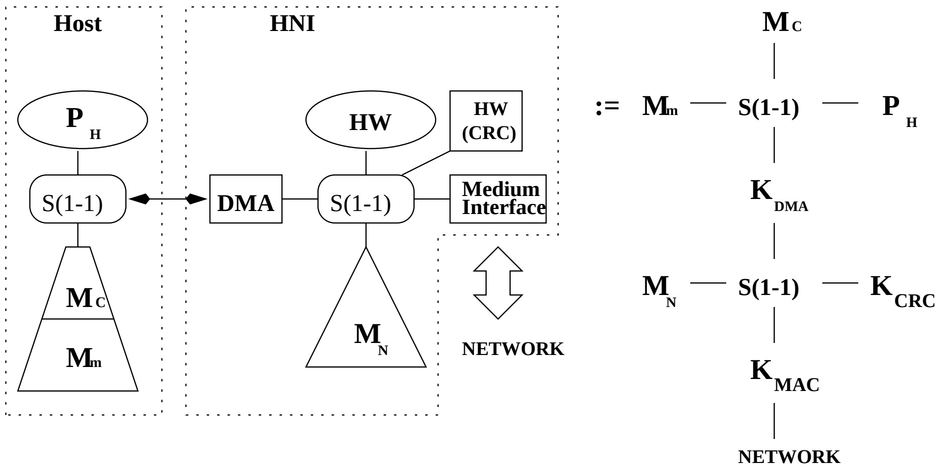


Figure 4.1: The architectural classification of simple ATM host network interface

error check and for correctness of destination address. Then, DMA controller transfers data from/to FIFO to/from main memory. Since the host executes almost all of protocol processing, the transport protocol data units (TPDU) are stored on the main memory by the host at transmitting time and by the DMA at receiving time. As a result, the host network interface does not need to have expensive network buffer memory.

CRC checking by hardware on the interface supports only header part for both receiving and transmitting paths. For cell payloads, the host processor calculates CRC in terms of CRC checking software.

Indeed, real performance improvements come from relieving the host processor which has to transfer packets to and from main memory. This can be achieved if the transfers are handled directly by the host-network interface instead of the host processor, assuming the host can assign more cycles to the applications while the transfers are served by the host-network interface. Also, the actual bandwidth of transfers to/from host main memory over the system bus depends more on the mode of transfer and single-word versus burst mode, than whether the transfer is done by

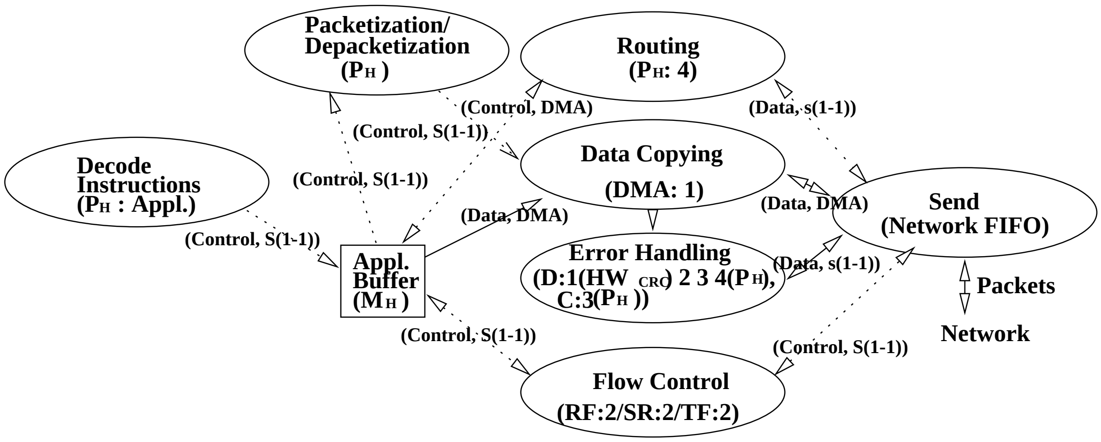


Figure 4.2: The protocol implementation of the simple ATM host-network interface

the host processor or the host-network interface. With excluding the host on the data transfer, the host can, at least, save clock cycles by not serving transfers of packets.

Under the above facts, through direct memory access (DMA), the host-network interface as an I/O device can read or write main memory directly with burst mode data transfer, but without involving the host processor. Cache coherence during DMA is maintained by the operating system, with the aid of purge cache and flush cache instructions.

When the cells are received, the host-network interface interrupts the host. Since interrupting the host is costly, it is not feasible to interrupt the host on a per-cell basis. The DMA controller interrupts the host when the last cell of the intended channel is received. For the real time service, the DMA controller should interrupt more frequently based on a timer or a threshold of receiving FIFO fulfillment.

Protocol-Implementation Classification

Figure 4.2 shows the protocol implementation of the simple host-network interface as the other classification context of architectural classification.

Routing/switching To support ATM network, the simple host-network interface has the virtual circuit switching (or asynchronous transfer mode). It means that one application occupies the network medium only when it has data needed to send. No matter when it is sending data, it is not necessary to occupy the network medium at a short moment (which is called a time slot) if the application program does not have data to send at the moment. Thus, the network is utilized with maximal bandwidth by applications. The host generates the source-end and the destination-end addresses. The intermediate-switch-port addresses are generated and routed by the routing server on the network. Addresses, routing information, and control information are stored in the main memory since protocol processing is done by the host.

Data copying Instead of copying data from user application memory space to the kernel space to process transport protocol, data for communication is taken care of in the user space memory area by a daemon process which is owned by root authority. With the shared memory technique, the daemon process can access different user spaces occupied by each process without any data copying overheads. It occurs between the user space of the main memory and the transmitting/receiving FIFO's. Thus, this data copying scheme reduces several copying between the host and the host network interface.

Error handling To handle errors on networks, the simple interface uses CRC, sequence number, multiplexing identification number (MID), and length indicator of data as the error detection parameters, and forward error correction with automatic repeat request (FEC + ARQ) as the error correction schemes. Since the AAL-5 puts the CRC for data at the last cell of the message and the host computes the CRC for data, the CRC for data is stored on the main memory and attached on the last packet with the length indicator of data. The CRC for header error check is required to be computed for every cell. Sequence number and MID are also put on each cell. CRC-8 is used for header error check and computed by the hardware on the host-network interface. After segmentation of data, the host puts sequence number on the payload of the ATM cell to re-compose the complete message of a user application on the destination-end. When the same channel is shared by several messages, the MID's are used to distinguish them. For example, a voice and a picture can have the same channel with different MID's, then, this multiplexing may solve the synchronization problem. A single-bit error can be corrected by CRC, and concatenated single-bit errors can be corrected by ARQ. The error correction algorithm converts the mode from FEC to ARQ when any single-bit error is detected from no error state and it converts from ARQ to FEC when no error is detected from any error state [82].

Flow control To control the data flow, the host uses basically acknowledgements. When the receiver acknowledges a cell, the receiver means receiving cells until the cell owned the sequence number in a window size. Thus, the transmitter retransmits the rest of cells. When the transmitter receives the acknowledgement for the last cell of the window the transmitter steps up to the next window. Moreover, the host counts the number of received cells at a time for an intended channel to control of data flow accurately.

Packetization The host brings the message data and segments it with a cell payload size. When the cell header parameters are storing into the network FIFO, the CRC for the cell header is checked and the value is stored in the FIFO. Since the cell size is fixed, the transmit and receive FIFOs can reduce the FIFO fragmentation problem. For example, each block of the FIFO can be fixed to 53 bytes for ATM network. Then, the FIFO does not have any un-using FIFO spaces. It means that this system does not have internal fragmentation problem on the FIFOs.

4.2 Intelligent Host Network Interface

The most complicated design, but the one likely to yield the highest performance for real-time traffic, is the ATM (Asynchronous Transfer Mode) host-network interface which implements some protocol services by the on-board protocol processor and by other components.

The network interface should use the resources of the host as efficient as possible in high speed network because it is not desirable to take up most of the host's CPU and memory resources to network related activities.

This design is mainly focused on network-related processing by the host-network interface, not by the host. The objective of this design is reducing the heavy loads of the host processor on protocol processing and the system I/O peripheral bus for more usage by multimedia peripherals. Thus, the host can support more-processing-required jobs such as real-time video compressions or a huge database searching.

To accomplish our objectives, the host-network interface should meet some stringent requirements to provide an efficient network connection.

4.2.1 Design Requirements

We will show the design requirements according to the taxonomical two-layerd contexts.

Design Requirements in Architectural Context

In the architectural context, this host-network interface is an intelligent and high-speed I/O peripheral connected to a network. To support all transport protocol implementations at high speed, the interface is required to have hardware components that are high costly. The on-board protocol processor reduces the heavy load of protocol processing on the host processor and it can adjust better to fast changes of high-speed network such as a sudden increase of incoming data from network, delay and jitters, or even on error handling.

CRC checking by a hardware component can also be implemented simultaneously while data is being transferred between main memory and network FIFO. This parallelized processing can support the better performance than host's software CRC checking for the high speed network.

As a connection component in our taxonomy, the system peripheral bus is popular but it can cause other I/O peripherals to have more competition on the bus. For example, video/audio-related or massive data storage peripherals need to occupy the system peripheral bus prior to other I/O peripherals. The conventional host-network interface design in ATM network system has been focused on the Direct Memory Access (DMA) technique between the host and the network interface. However, if the format of data in memory is not compatible with the format of DMA data path, DMA-based host-network interfaces have to access the host with time delays which are occurred on the DMA. For example, on an Alpha processor machine, a transfer on 64 bit DMA has one delay time unit to the first 32 bit data until the later 32 bit arrives. Also, in DMA techniques, on-board protocol processor gives the information for a needed data to the on-board DMA controller. Then, the controller takes the intended data from main memory. If the host has just created the intended data and saved it into the cache, the data transfer through DMA should wait until data is stored in main memory. Therefore, a better technique is required, instead of DMA on the system I/O peripheral bus, as the connection component between the host system and the host-network interface.

Design Requirements in Protocol-implementation Context

In the protocol implementation context, each protocol component can be implemented in a pipelined or parallelized way by a protocol processor or hardware components. Thus, each protocol component has the following requirements:

Routing/Switching Only the sender's and receiver's addresses are required during connection. The addresses should be accessed by both the host and the protocol processor because the host refers addresses at sending and the protocol processor refers those at receiving.

Data copying In kernel mode, protocol processing of the conventional TCP/IP, the data should be copied from an application process in user level memory area into a kernel level memory area to process TCP by the "Socket functions." Alternatively, the data can be transferred from the user space via network buffer memory using multiple threads or shared memory without copying to the kernel space. Thus, a user-mode thread or daemon is required to transfer data between the main memory and the network buffer memory.

Error handling Since ATM network can be used for multiple applications simultaneously, an identification is required to distinguish each applications. Also, CRC-8 error checking is required for ATM cell headers and CRC-32 error checking is required for ATM cell payloads. Since the network is considered as a high speed, the network buffer should be managed without any blocking of data transfers. Every incoming data should be transfered to main memory or other peripherals before next incoming data become lost because of no space to store.

Flow control The network buffer memory is required to store the flow control parameters. The on-board protocol processor counts the sending and receiving cells, informs the rate to the host, and sets the optimized flow control parameters.

Packetization The protocol processor is required to packetize a payload with a cell header parameters which are stored in network buffer memory. If the message data is segmented and reassembled by the protocol processor the host is likely to process data from the ATM network as a local data.

4.2.2 Taxonomy-based Design Approach

A complicated design, however the one likely to yield the highest performance for real-time traffic, is an ATM host-network interface based on the pipelined dual-port cache memory and an on-board protocol processor, as shown in Figure 4.3.

By the taxonomy, the cache-based host-network interface has the following taxonomical features:

1. Figure 4.4 shows the architectural classification of the intelligence host-network interface in architectural level.

   - $P_H - M_N$: Burst transfer by cache controller (B_{cc}) for status and control information, start address and length of data, and connection establishment parameters.

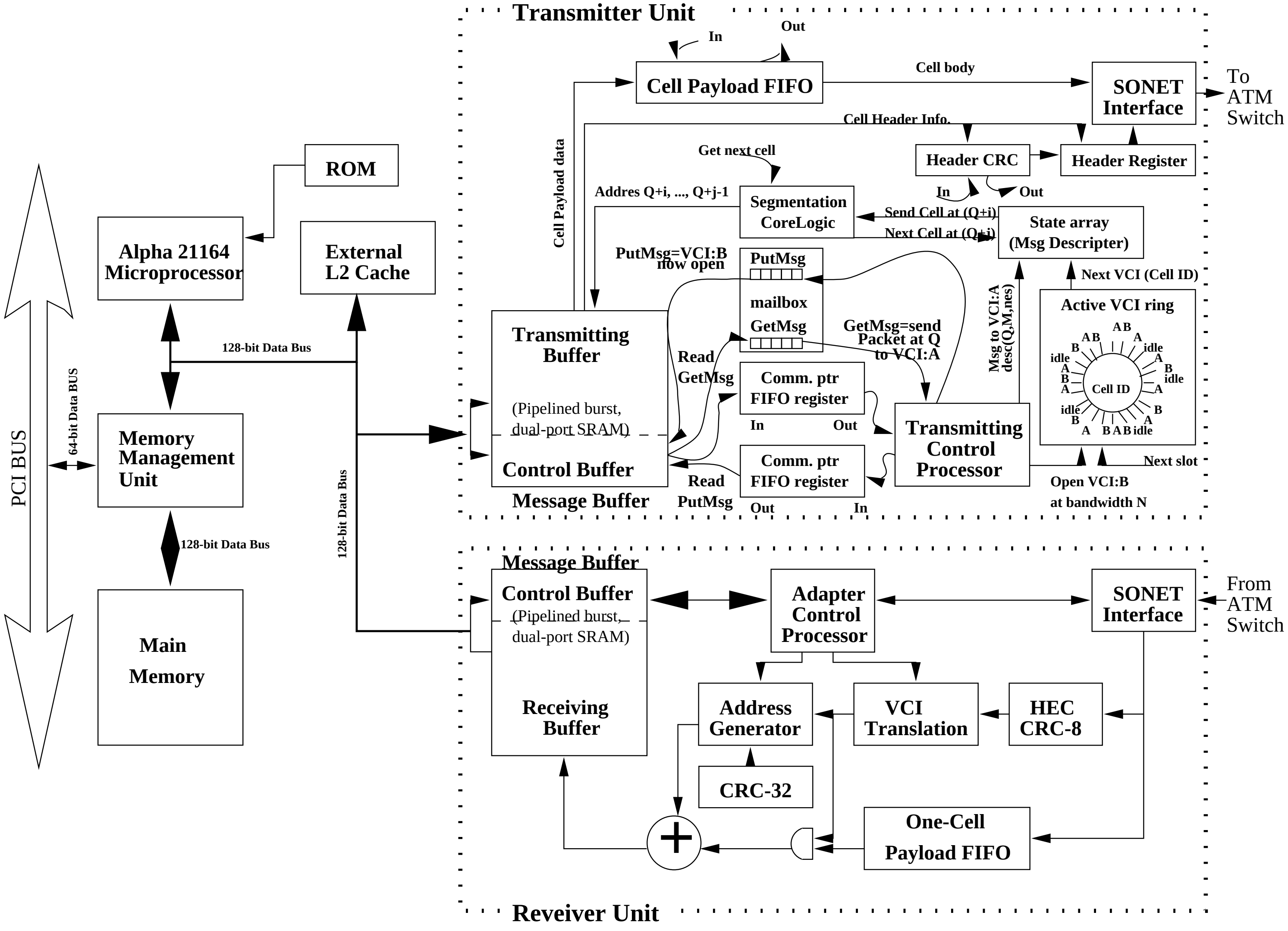


Figure 4.3: The relationship of Host-Network Interface to host processor.

- $C_H - M_N$: B_{cc} for data transfer.
- $M_H - M_N$: B_{cc} for data transfer.

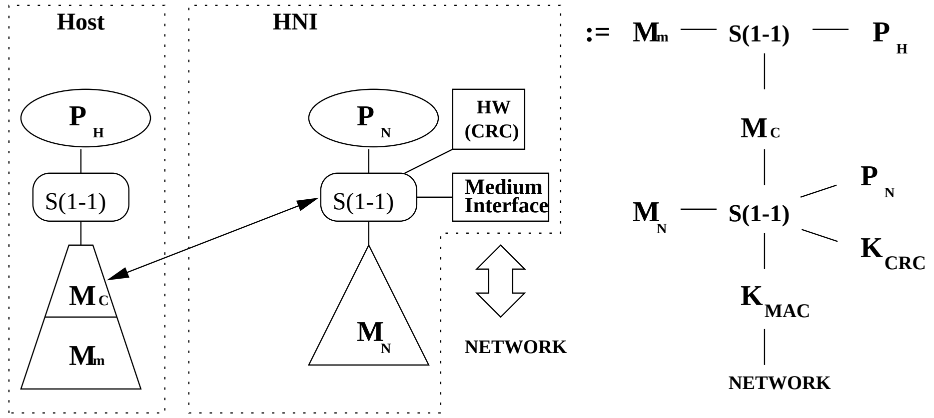


Figure 4.4: The architectural classification of the intelligence ATM host network interface

2. Figure 4.5 shows the protocol-implementation classification of the intelligence host-network interface in protocol implementation level.

   - The protocol processor packetizes in the fixed size.
   - Virtual circuit switching.
   - UNF (user space in memory-network buffer-FIFO) data copying scheme.
   - Error-handling,
     - Type: CRC-8, CRC-32, sequence number, MID, length indicator of data.
     - Domain: control part, data part.
     - Implementor: Hardware.
   - Rate-based flow control.

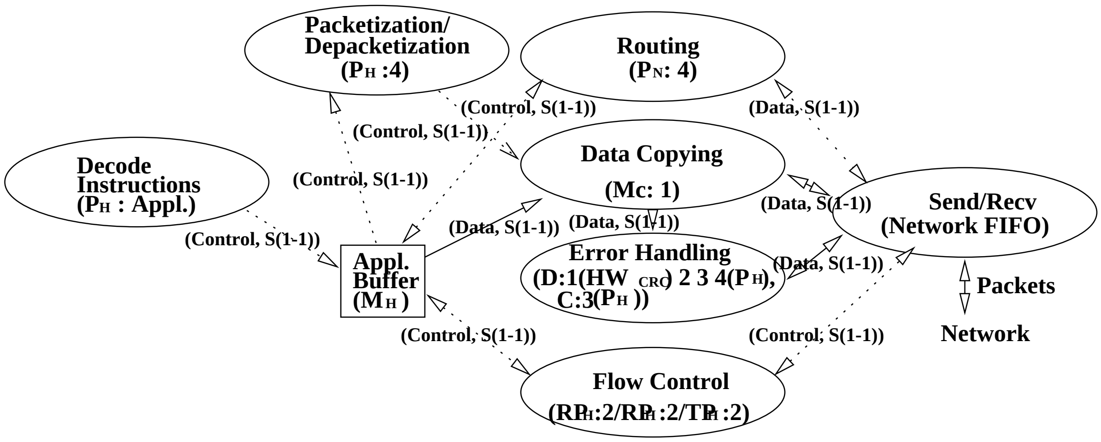


Figure 4.5: The protocol implementation of the intelligent ATM host-network interface

Architectural Classification

A processor on the ATM host interface can perform the segmentation and reassemble (SAR) processing more efficiently than the host processor as it is dedicated to this task and it has low-latency, high-bandwidth access to the network interface, [19]. The on-board processor would allow direct data transfer between the network and audio and video devices, eliminating the latency and overhead of going through the host.

Moreover, since the network buffer memory is a dual-ported cache, the data access can be faster than the data access from main memory through a system bus. During transmission of the data into the network, the host can write data directly into the network buffer or the cache controller can transfer data in burst mode into the network buffer from main memory or other I/O devices. Additionally, the network buffer can be mapped into memory space with high bandwidth and lower latency to/from main memory via pipelined burst data transfer mode. Also, the network buffer has less cache-coherency problem since data is accessed from/to cache.

The host has a connection with the network buffer memory to execute instructions for status and control information, start addresses and length of data, and connection establishment parameters. The connection is the bursted transfer mode by the cache controller.

If the host need to move data from cache memory to network buffer memory, it can be moved in pipelined burst transfer by cache controller. As even worst, when the host need to move data from main memory into the network buffer memory, data can be transferred in pipelined burst mode by the cache controller because the buffer is dual-ported memory in which one way is accessed by the host and the other one is by network protocol processor. The transfer is the exact same as cache miss of data in cache system.

Protocol-Implementation Classification

Lightweight protocols and protocol offload to programmable adapters, each based on a single protocol processor, are two approaches that are proposed to cope with the bottleneck in communications. However, programmable adapters do not seem to have good cost-performance rates since they use expensive hardware components.

Therefore, we will look at mechanisms of protocol processing in the host and in the on-board processor with the lightweight protocol concepts, simple and high speed processing.

Routing/Switching For the routing of ATM network, the host manages the information to route the destination(s). However, the on-board protocol processor can access the routing table in the network buffer and packetize/depacketize to destinations by this table. By the ATM switching technique, the ATM switch assorts packets based on the addresses given in the header of the packet itself.

Especially, in our connection scheme, it is accomplished in a deterministic manner, VCI ring [75]. According to a round-robin processing policy, VCI ring can process the preemptive policy for a special channel. The VCI ring also serves as a pointer into the packet descriptor. When a connection comes up for service in the VCI ring, the packet descriptor is pointed, read, and then the network addresses are generated for the connection, and other control information is prepared for the header of the cell. The VCI ring allows continuous control of bandwidth. Managing changes in bandwidth assigned to connections is done by the on-board processor, simply by modifying the number of ring elements in use by each connection. The packet descriptor, called connection management table, is shared and updated by the host and the network protocol processors.

Data-Copying in user space The data in a user space is copied through the network buffer which is a kind of cache memory. Thus, it can reduce the number of copying and even result in the very fast data accessing (just like from main memory to the cache).

Even in the category of the user-space data copying, there may be several ways according to using the shared-memory processes or multithreads. We use shared memory (*e.g.* *shmem*() in UNIX) between the daemon process and application processes in terms of buffer managers on interface part in Figure 4.6.

Figure 4.6 shows a general model for protocols derived under the user-level multiprocesses. A client process sends requests to the daemon via an IPC message queue (for example, *ipcs*() in UNIX) built into the interface object. The daemon returns the result of the request via the same IPC facility. User data, however, is written to and read from two buffers that are managed by a buffer manager in the interface. This separates the request/response activity

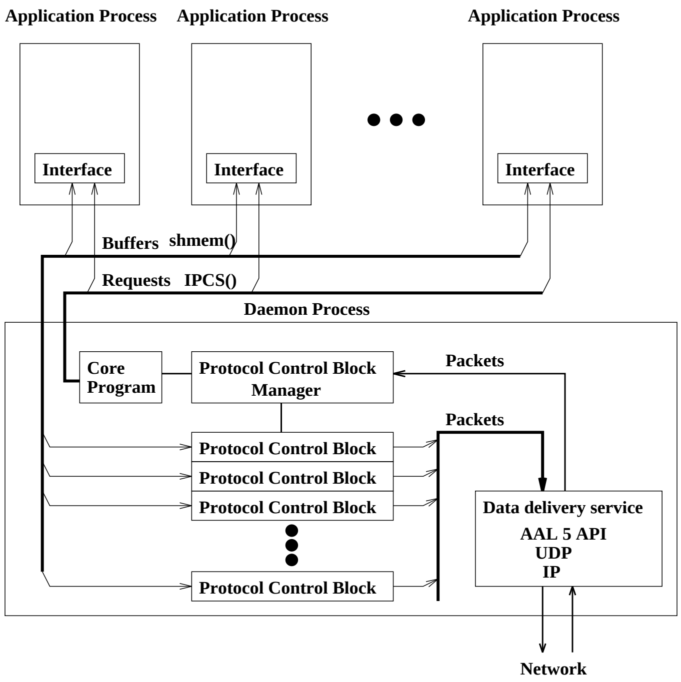


Figure 4.6: Implementation of a user-level transport protocol

from the maintenance of data buffers. The core program in the daemon is a loop that accepts user requests and invokes the appropriate entry point into the actual protocol processing. These entry points are procedures (functions or methods) in the protocol control block manager. The state structure for control information is called the protocol control block or context. The context manager owns all of the contexts in the daemon and steers incoming packets to the proper context. The contexts implement the protocol-specific procedures, some of which generate packets. The daemon also owns a data delivery service objects that manage the use of the network.

Figure 4.7 shows how the buffer managers, the shared memory segments, and the client and daemon processes are related. A client process has two data buffers: one for sending and the other for receiving. Each of the client's buffers

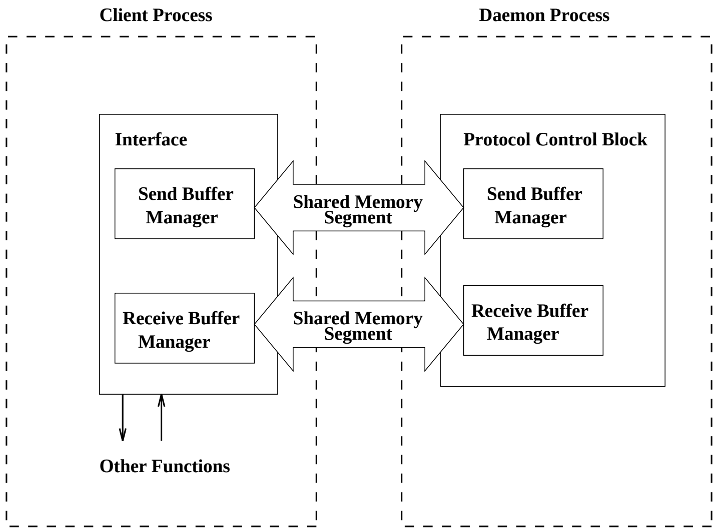


Figure 4.7: Buffer Management between an application process and a daemon process

is controlled by a buffer manager. When the client registers these buffers with the daemon, the context (or protocol control block) assigned to the client process attaches its two buffer managers to the client's buffers. In this way, each buffer is controlled by two buffer managers: one at the user application side and the other in the protocol control block associated with this client process. In general, one of the managers writes to the buffer and the other manager reads from it, so there must be some way of coordinating the head and tail markers for the buffer. This is done via user request commands: When a send request is issued, the interface object places the new marker values into the request so that the context's buffer manager can acknowledge which data to send. A similar exchange occurs for receiving data through the receiver buffer.

The buffers are implemented as segments of shared memory. The user application, through the user interface library, creates two shared memory segments. The identifiers for these segments are relayed to the daemon, where the newly initialized context for this user attaches the daemon process to the shared memory segment.

Error Handling The host-network interface on the receiver checks CRC bits for header and payload, length indicator of data, and sequence number of the cell. The receiver host checks the multiplexing identifier (MID). The checked error information is sent to the transmitter, then the transmitter corrects the error based on the forward error correction and automatic repeat request.

There are two CRC checking hardware units: CRC-8 for header error check and CRC-32 for the payload error of the cell. When a cell arrives, the CRC-8 hardware unit checks header error before any other functions serve. If there is no error, the destination address is checked to protect the mis-routed cell. The partial checksum with CRC-32 is processed by the host. The CRC-32 checksum is processed only at the source-end and destination-end systems, not in the any ATM switch system. Error correction schemes are implemented by FEC and ARQ as in the case of the simple interface.

Flow Control The receiver maintains the flow of incoming cells through counting the arrival rate of packets and reports it to the transmitter by the packet arrival rates. Then, the transmitter controls the rate of transmitting cells.

Specially, the flow control via the medium access protocol is a backward explicit congestion notification mechanism. When a buffer in a switch exceeds a threshold, the control logic of the switch sends congestion notification cells back to the sources of the virtual channels currently submitting traffic to this switch. On receiving a congestion notification cell on a particular virtual channel, called signaling channel, a source must reduce its transmission rate for the indicated virtual channel. If there is no such a cell on the intended channel for a certain period of time, a source may gradually restore its transmission rate on that channel.

The SONET interface on the source-end counts the number of outgoing cells for a particular channel for a given time and computes the transmitting cell rate. The SONET interface on the destination-end also counts the incoming cells for the channel for a given time. If there is a need to control the cell rate, the destination host puts the information for flow control on the signaling channel for the intended channel. Thus, the source-end host can adjust the optimized cell rate between the source and the destination-ends.

4.3 Summary

In this chapter, two host-network-interface designs have been discussed with the new taxonomy: simple and intelligent interfaces.

Major philosophies in designing simple interface are flexibility of protocol implementation, minimal hardware components, and data transfer between main memory and FIFOs by direct memory access. By the descriptive feature of the taxonomy, the simple host-network interface is designed and explained that cell-header CRC is checked by a hardware, the host implements the protocol processing, and the DMA transfers data. However, we can predict from the predictive feature of the taxonomy that the host is likely to be overloaded by the protocol processing since simple interface does not have any on-board processor.

In designing the intelligent interface, the major objectives are to support very high-speed and low-latency ATM networks. With the descriptive feature of the taxonomy, we explained each components. In the architectural context, an on-board protocol processor, CRC-8, CRC-32, and cache-based network buffer memory are explained. In the protocol implementation context, data copying as cache memory, error handling, cell flow rate counting, flow reporting by the protocol processor are described. In detail, the intelligent interface supports no data copying in main memory. Data is moved from a user space of main memory to the transmitting FIFO through dual-ported cache memory which is working as a network buffer memory. Since dual-ported, pipelined, and burst mode SRAM is used for cache and network buffer memory, we can predict that it can support faster data transfer and it results in supporting real-time applications very well.

Chapter 5

Performance Analysis

In this chapter, we analyze the latency of data transfers between a host and host-network interfaces designed by our taxonomy and show their effective transfer rates. We also analyze performances of a simple interface and an intelligent interface, then compare them with performance of a conventional host-network interface. In designing host-network interfaces, we have focused on reducing overheads of unnecessary protocol processing and parallelizing procedures as possible. Thus, the analysis is focused on the data transfer between the host and a host interface.

5.1 Introduction

The custom-designed host-network interface is the key system component which connects each computer to a port of an ATM switch. Depending on the purposes of applications or the required loads of the given network, the host-network interface should be selected as an optimized component.

In analyzing performances of our designed host-network interfaces, we made an assumption that the host processor is the Alpha 21164 processor from Degital Equipment Corp. The AlphaPC consists of Alpha 21164 microprocessor (500 MHz clock speed) with a second-level external cache via 128-bit bidirectional data bus, a 36-bit bidirectional address bus, and several control signals [32]. The 21164 external interface is flexible such that read and write speeds (33 or 66 MHz) of the external cache array can be programmed by means of register bits. Read and write speeds are independent of each other and the system interface clock frequency.

The simple interface is designed for low-data transfer rates, for requirements of small board size of interface, and thus for low cost. With the simple interface, the host is connected to the interface via main memory through the system bus, PCI bus.

On the other hand, the intelligent host-network interface is connected to an external cache slot which uses the 128-bit data bus of the Alpha 21164 processor. The external cache slots connect the host system and the network adaptor board.

5.1.1 Formal Performance Metrics

Eric Cooper *et al.* formalized a performance analysis parameters in [19]. Performance parameters are simplified but translate each characteristics very well. Since it presents the comparison of bottlenecks related to the data transfers, this model is formalized as the performance metrics of host-network interfaces in this chapter.

This model is parameterized using the following set of performance characteristics for loads and stores of 32-bit words.

1. $T_{\mu p}$: The clock cycle time of CPU data bus in nanoseconds. For Alpha 21164, it is $33MHz$ or $30.3nsec$.

2. $T_{\mu c}$: Clock cycle time in data bus for adaptor control processor. MC88110 has

25 nsec data bus clock cycle time for 64 bits.

3. $L^f_{\mu p}$: The number of clock cycles per load from cache to host CPU, or from network buffer to host CPU.

4. $L^f_{\mu c}$: The number of clock cycles per load from network buffer to network-adaptor control processor. It takes 3-1-1-1 data transfer clock cycles.

5. $L^s_{\mu p}$: The number of clock cycles per load from main memory to host CPU. For typical EDO (Expanded Data Output) dynamic RAMs, 7-2-2-2 data transfer clock cycles in a pipelined-burst mode.

6. $L^s_{\mu c}$: The number of clock cycles per load through bus. It is supposed to have the same value with $L^s_{\mu p}$.

7. $S^f_{\mu p}$: The number of clock cycles per store to the network buffer, or to the cache from the host CPU.

8. $S^f_{\mu c}$: The number of clock cycles per store to one cell FIFO in network adaptor.

9. $S^s_{\mu c}$: The number of clock cycles per store to main memory from host CPU or to cell FIFO from main memory in conventional architecture.

Figure 5.1 shows different connections between a host and a host-network interface with describing whether a processor can access a component fast or not.

This modeling simplifies all arithmetics and control instructions to take one clock cycle. Pollings or message waiting durations are not considered. Only time delays during transferring data are assumed.

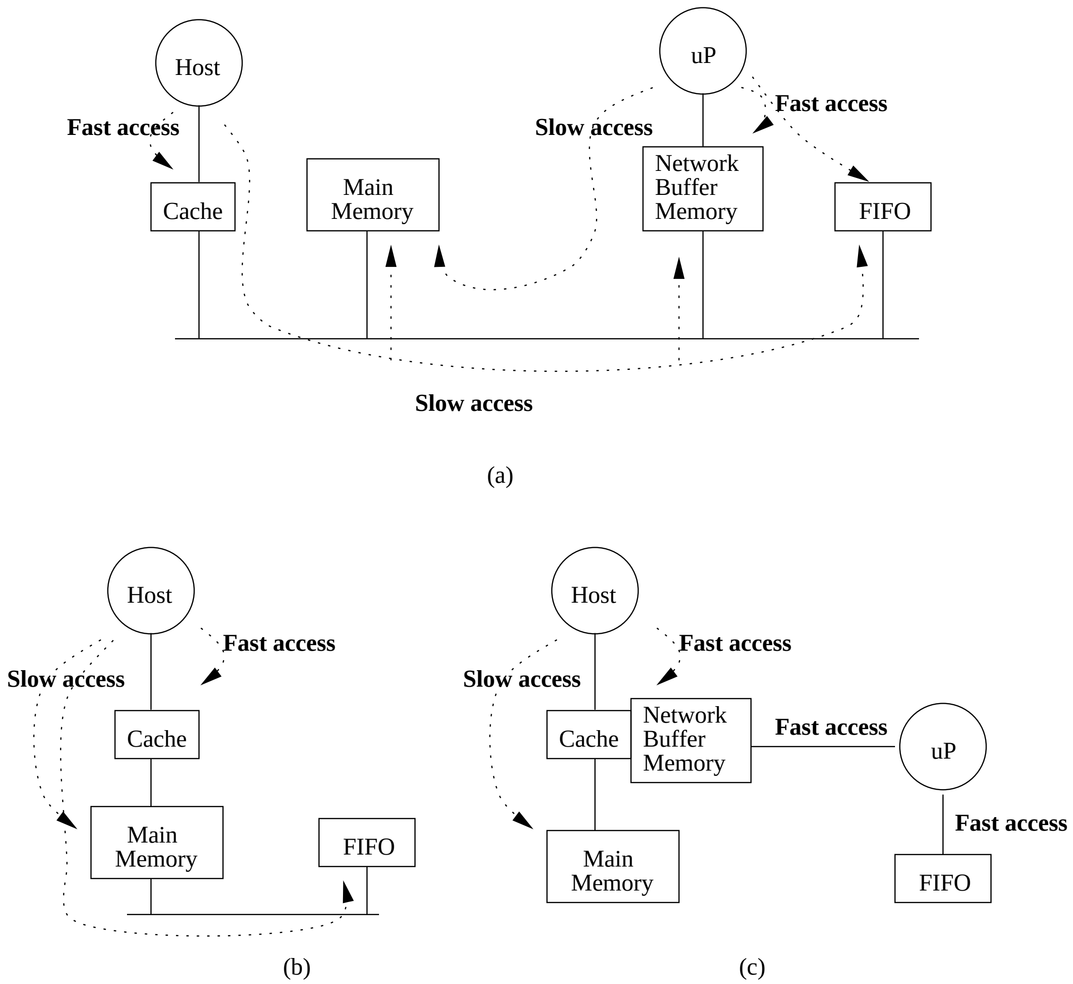


Figure 5.1: The connections between host and network adaptor

5.2 Performance Analysis of Simple Interface

The simple host-network interface is designed with the concept of minimal hardware support. The host processor, therefore, implements protocol processing, instead of protocol processor on the interface board. It may cause the system bus, PCI, to be overloaded or to be the bottleneck of data transfers. It can be formalized in terms of performances on transmitting and receiving sides whether the system bus cause data transfer to be slow down. The following is the performance analysis of simple interface.

5.2.1 Transmitting Performance

Since the host performs the segmentation processing, the host informs the start address and the offsets of intended payload-sized data to the DMA controller on the interface board, while the host informs the start address and length of message-sized data to the on-board protocol processor in intelligent interfaces such as the OSIRIS or ORBIT of AURORA project. Thus, the segmented data, payload, goes simply over the system bus, PCI, to the host interface. However, it may cause the data load by DMA on the interface to be slow.

Here, we should discuss some assumptions to make our analysis simple. After the host has a store instruction, the data on the cache is assumed to be written back into the memory before the DMA controller tries to move data. It results in ignoring the cache coherence problem. Also, the host does not have the context switching during sending data to the host interface and transferring data on the middle of message. The PCI system bus is assumed to send data in 64 bit wide in a cycle between main memory and the host interface. Figure 5.1 (b) shows how fast the host processor can access cache, main memory, network buffer (FIFOs). Additionally, in ATM network the ATM adaptation layer 5 seems to be optimized for real time data service since there is only one CRC field on the trailer of the last cell. Therefore, AAL-5 is assumed for this analysis.

Since the DMA controller has the information of message segmentation from the host, the loads go over system bus to the interface. The stores of segmented data into the network FIFO also are slow since the data must go over the system bus. In this case, the throughput for a payload is given by the following equation:

$$\begin{aligned} Throughput &= \frac{48\ byte/cell \times 8\ bit/byte}{(6L^s_{\mu p} + 7S^s_{\mu p} + 6computation)T_{\mu p}} \\ &= \frac{384}{(22 + 22 + 6)(30.3 \times 10^{-9})} \\ &= 248.50Mbps \end{aligned}$$

The number of instructions is based on the reference [19]. For example, to send 1 payload data (384 bits), 6 slow loads and 6 slow stores are needed through the 64-bit PCI system bus. For the header information, 1 store is approximated since there is less than 1 fast load for the header information and CRC field. The header information is simply loaded and stored without more clock cycles to look up because the information is managed on a special place such as cache.

As a result, Figure 5.2 shows the throughputs related to the packet sizes of messages. Since the bus is fully utilized, there is no high slope on the graph. The above analysis shows that the burst mode (e.g. DMA) supports data transfer of several words in significantly fewer cycles than data transfer by the individual loads or stores such as programmed I/O. With a small amount of pipelining, it is also possible to overlap the transfer of the payload with the processing of the header.

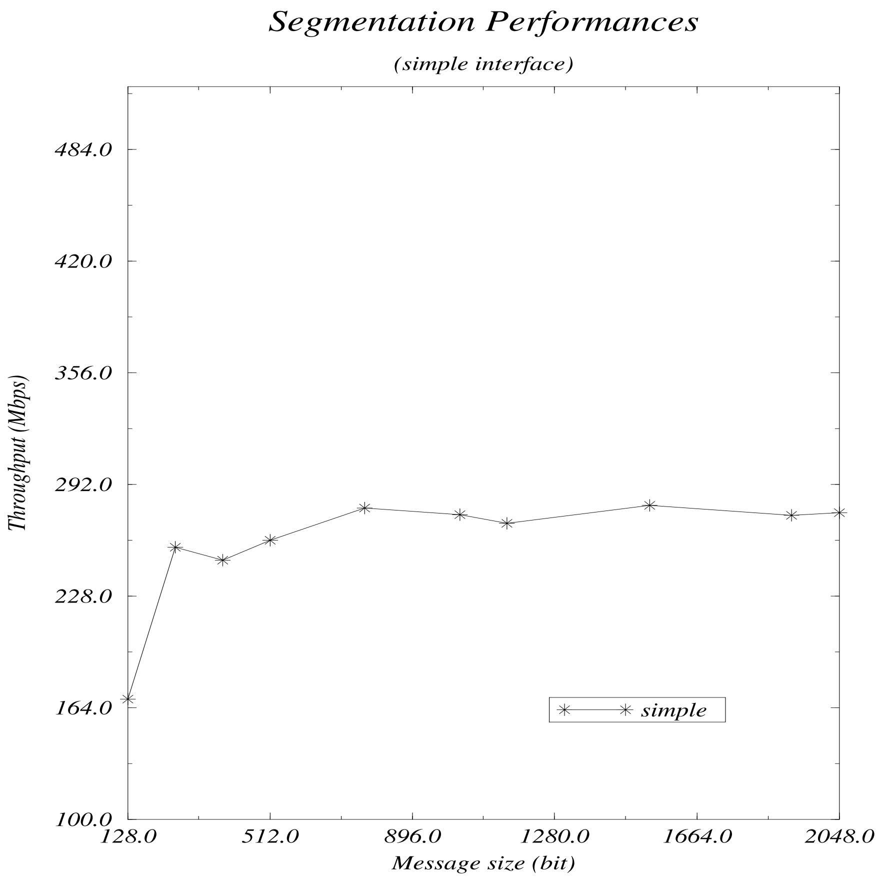


Figure 5.2: Throughputs of transmitting procedure on simple interface with variable packet sizes

5.2.2 Receiving Performance

In this simple interface model, the network interface receives ATM cells from ATM switch, then splits header and payload parts. As a first protocol processing, CRC-8 hardware component checks any errors on the header fields. If no error exists on the header part, SONET Linker checks the addresses of the destination and the source. With correct addresses on the header, payload can be moved to main memory by the DMA controller. Since the interface board does not have network buffer memory, the host performs the reassembly processing with payloads saved on the main memory.

Before any further steps, preassumptions should be considered. The protocol processing is a loop processing using the reassembly state information. Thus, it is the best when the host processor keeps the protocol state information of the most recent reassembly in registers. This technique is called hints [19]. An expected reassembly header for the next cell can be computed, which is used as a hint. If it matches, then there is no need to check any additional fields of cell headers. Thus, the cached reassembly state information is assumed to be always valid. It is possible when an ATM cell arrives at a time.

Since the host performs the reassembly processing and DMA transfers data from network FIFO to the main memory, the reassembly processing must require more clock cycles of the host on data transfer than the reassembly processing by an on-board processor. After DMA transfers data into the main memory, the host reassembls the received payloads into a message. Thus, there are slow loads from network FIFO, then slow loads into the host (refer Figure (b) of 5.1). On the other hand, if the host is on idle, then payloads can be reassembed directly by the host without storing to the memory, which can save clock cycles of processor. In this case, there are slow loads from network FIFO and slow stores to main memory after reassembly processing by the host. Let us evaluate both cases for performance of the simple interface.

The throughput of reassembly by a lightly loaded host for a ATM cell with simple interface is as follows: The DMA transfers data with 6 loads (7-2-2-2-7-2) for a ATM cell since the DMA channel has 64 bit bandwidth. The host processor requires 3 (7-2-2) stores for reassembled data since the host has 128 bit bandwidth to main memory.

$$
\begin{aligned}
Throughput &= \frac{48Bytes}{(6L^{s}_{\mu p} + 3S^{s}_{\mu p} + 12\ computation)T_{up}} \\
&= \frac{384}{(22+11+12)30.3 \times 10^{-9}} \\
&= 281.63\ Mbps
\end{aligned}
$$

However, the throughput of the reassembly processing by a heavy loaded host for a ATM cell with simple interface is as follows:

$$
\begin{aligned}
Throughput &= \frac{48Bytes}{(6L^{s}_{\mu p} + 3L^{s}_{\mu p} + 3S^{s}_{\mu p} + 12\ computation)T_{up}} \\
&= \frac{384}{(22+11+11+12)30.3 \times 10^{-9}} \\
&= 226.31\ Mbps
\end{aligned}
$$

Hence, Figure 5.3 shows the throughputs given by the packet sizes. From this figure, the real throughputs can be expected to be better than the throughputs with heavy loaded host and less than the throughputs with lightly loaded host.

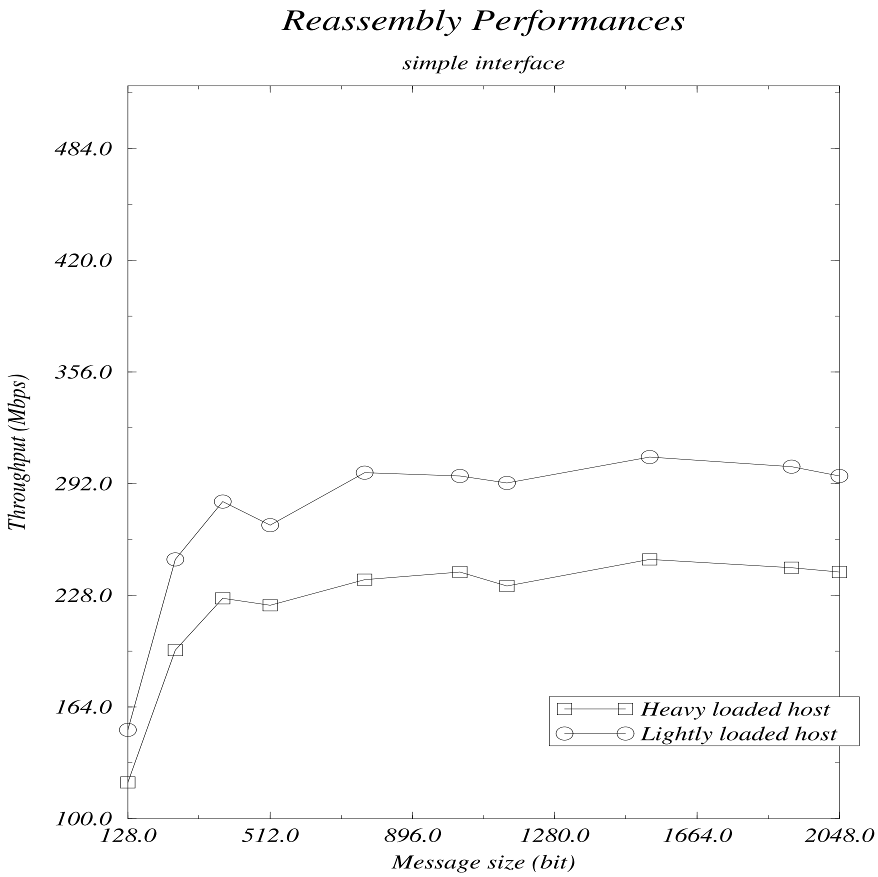


Figure 5.3: The throughputs of receiving procedures with variable packet sizes

5.3 Performance Analysis of Intelligent Interface

A processor on the host-network interface can perform the SAR processing more efficiently than the host processor since it is dedicated only to this task and has low-latency, high-bandwidth access to the network interface. Especially, the rate of data transfers between the host and network buffer memory is expected to be very high since the network buffer is a kind of cache in the host system. This can be proved by comparing throughputs as in the following.

5.3.1 Event Analysis

This section unveils the full design of the intelligent interface and the performance of its components for the Alpha 21164 microprocessor [32] from Digital Equipment Cooperation, with IDT's synchronous pipelined burst dual-port SRAM IDT709279 as a network buffer. The detail event description of the host processor can be from [79] which has the same operations for the processing of the host processor.

The 21164 partitions physical address, PA[39:5], into an index field and a tag field. The 21164 presents PA[25:4] and PA[38:20] as an index address and a tag field address, respectively to the external L2 cache interface. Each cache block would contain status bits for the valid, shared, and dirty status bits along with a part or all of PA[30:20].

In this memory system hierarchy, external L2 cache consists of 1 MBytes and a network buffer of 1 MBytes. The block size must be the same for both internal and external. Thus,

$$cache\ size = 1\ MBytes = 2^{24}$$

$$cache\ block\ size = 32\ Byte\ block = 2^{8}$$

$$the\ number\ of\ blocks = \frac{2^{24}}{2^{8}} = 2^{16}$$

Each of the L2 cache and network buffer has 2^{16} 32-byte blocks. Therefore, the 40-bit physical block address is divided into an 18-bit tag and a 16-bit index:

$$40 - 16 - 5\ for\ block\ offset - 1\ for\ interleaving = 18\ bits.$$

The most significant bit is used to distinguish data for network buffer from data for external L2 cache. For example, the capacity of physical address is roughly 33MB ($32MB\ Main\ Mem{+}1MB\ for\ MSG\ BUF$) or more, even though the size of physical main memory is 32 MB.

The L2 cache reads the tag from that index and if it matches, the cache returns the critical 32 bytes in the first 2 clock cycles (internal L2) or 3 clock cycles (external L2 or network buffer) and the next block in 1 clock cycle after the first 3 clock cycles since the cache uses pipelined burst mode. At the same time, a request is made for the next sequential 32-byte block, which is loaded into the instruction prefetch buffer in the next 1 clock cycle with pipelined burst mode.

Specific instructions from the network are found in the network buffer completely. If not, processing will be stalled until the instruction comes from the network. However, if the instruction except network specific instructions is not found in the L2 cache or the network buffer, the translated physical address is sent to main memory. When the Alpha 21164 reads miss command and then enters the main memory logic, the memory controller generates the appropriate address signals for the memory array. The memory controller then waits for the memory access delay before instructing the bus interface to accept the memory data.

Now that the processor needs to process data to send into network, it adds some overhead bits in Convergence-Sublayer of AAL. Thus, we suppose that the instruction is a load data from cache or even from main memory. The instruction will send the page frame of its data address to the data TLB (DTLB) at the same time as the index from the page offset is sent to the data L1 cache. The 64-entry, fully associative, dual-ported data translation buffer (DTLB) stores recently used data stream page table entries. In the worst case, the page is not in main memory, and the operating system gets the page from disk storage. Since millions of instructions could execute during a page fault, the operating system will swap in another process if there is something waiting to run.

Assuming that the valid page table entry is in the data TLB, the cache tag and the physical page frame can be compared with a match, which is sending the desired 16 Bytes from the 32-Byte block to the CPU. If the data is not found in the data cache, Dcache, (a miss), then the address, target register number, and formatting information are entered in the miss address file. The miss address file performs a load-merging function. When a load miss occurs, each entry of the miss address file is checked to see whether it contains a load miss that addresses the same data cache block. If it does, and certain merging rules are satisfied, then the new load miss is merged with an existing entry of the miss address file. If it does not, the miss goes to the L2 cache, which proceeds exactly like an instruction miss.

It is time to store the data to the proper space in the network buffer. That instruction is a store. The page frame portion of the data address is again sent to the data TLB and the data cache, which checks protection violations as well as translates the address. The physical address is then sent to the data cache. Since the data cache uses write through, the store data are simultaneously sent to the write buffer and the data cache. The data address of this store is checked for a match, and at the same time the data from the previous write hit are written to the cache because the processor pipelines write hit steps. If the address check was a hit, then the data from this store are placed in the write pipeline buffer. On a miss, the data are just sent to the write buffer since the data cache does not allocate on a write miss. While data for an L2 cache block or a main memory block may have write-hit, data for a block in the network buffer will not have write-hit since the valid bit is masked after receiving acknowledgment from the receiver.

The write buffer takes over now. It has six 32-Byte entries, each of which holds the data from one or more store instructions that access the same 32-Byte block in memory or in network buffer until the data is written into the internal L2 cache. The write buffer provides a finite, high-bandwidth resource on receiving a store data to minimize the number of CPU stall cycles. If the buffer is full, then the CPU must stall until a block is written to the network buffer. If the write buffer is not full, the CPU continues and the address of the word is presented to the write buffer.

All writes are eventually passed on to the network buffer. Even though the network buffer uses write back, it can pipeline writes because writing data are sequential: a first full 32-Byte block write takes 3 clock cycle to check the address and 4 clock cycles to write the data (2 times writing of 16 Bytes). However, the next data will take 1 clock cycle to check the address and 2 clock cycles to write the data. Since CS-PDU (Convergence Sublayer of AAL Protocol Data Unit) is much bigger than a block size and the latter overrules the former, storing the block to the network buffer is assumed to have latter clock cycles, that is, to take 3 clock cycles. In this case, the network buffer marks the block as dirty in status and control flags.

On the other hand, the 21164 processor has instructions to poll a control message area of the network buffer to receive data from the network. If there are some data to be received in the network buffer, the 21164 processor gives a load instruction to the network buffer, extracting the header and trailer parts from CS-PDU's, then it gives a store instruction to store data into main memory.

Almost all events have the same clock cycles as in the case of transmitting data into the network except writing data to internal L2 cache and to main memory. A full 32-Byte block write to internal L2 cache takes 1 clock cycle to check the address and 2 clock cycles to write the data.

If the access to the L2 cache is a miss, the victim block is checked to see whether it is dirty; if so, the victim block is placed in the victim buffer to get out of the way of new data as before in instruction. If the new data is a full block, then the data is simply written and marked dirty. If the new data is a partial, a partial block write results in an access to main memory since the L2 cache policy is to allocate on a write miss.

5.3.2 Network Buffer Efficiency

The event analysis for the cell processing is discussed in the above section. The model includes an Alpha 21164 micro processor, the second level cache, and the network buffer. An ATM-network-adaptor control processor can read and write data in the network buffer as a local memory. Thus, the network buffer is used to transfer data for control commands of SAR (Segmentation and Reassembly) processing as well as data for cell payloads.

Transmitting Performance

The mapping from a variable-size message, CS-PDU, to a sequence of fixed-length ATM cells is called segmentation. CS-PDU (Protocol Data Unit for Convergence Sublayer of ATM Adaptation Layer) is usually much bigger than an ATM cell. As mentioned in the previous chapter, the AAL-5 protocol will be used for this performance evaluation. To evaluate the performance of a cell in transmitting side, only 48 bytes are assumed to be stored to the network buffer from the host even though the transmitting host sends CS-PDU to the network buffer in one or a few times. In segmentation, slightly more complicated operations are needed at the beginning of message and at the end of message because there are some initialization commands. To make simple situation, we assumed only the continuation of message, thus each cell is fully loaded with data.

The protocol processing time is very dependent on the application program since every application program has different memory accessing method. In the worst case, the host will load all data from main memory. Since the memory bus transfers data of 16 Bytes per load, it will take 3 slow loads for 48 Bytes of data. It is assumed to have 7-2-2 cycles in processor cycle unit. To store 48 bytes into the network buffer, the host needs 3 stores since data bus passes data in 16 bytes per store. It will take 5 clock cycles (1 address and 3 data stores such as 3-1-1) by the pipelined data storing of 48 bytes into the network buffer.

As a future work, the cache memory controller can be modified. In this analysis, the data kept in main memory are supposed to be transferred into the network buffer memory by the programmed I/O in which the host controls each transfer. However, it is possible for a host processor to give an instruction to the cache controller. The new extended instruction has the start address and the length of intended data. Thus, with only one instruction from the host processor, the intended data can be transferred to the network buffer without further host's works. It seems to be a direct memory access scheme but adopted to the cache memory, not to the main memory. It might be called a direct cache access (DCA) scheme.

In the network adaptor, the control processor, MC88110, has a 64-bit pipelined burst mode (3-1-1-1-1-1-1-1) data bus with back-to-back cycles (eight 64-bit access). For example, it requires 6 clock cycles (1 for address translation and 5 for data, 3-1-1-1) to load a 32-byte data block. Thus, the control processor takes 8 (3-1-1-1-1-1) clock cycles for the data load of 48 bytes in the pipelined burst mode. The network-adaptor control processor adds SAR-header bits, SAR-trailer bits, and ATM header bits. The control processor takes 8 fast loads, 9 (8 for payloads and 1 for header) stores, 5 arithmetic instructions and 1 branch instruction ([19]) for SAR-PDU processing (computations of start address and offsets from information about the message given by host) and CRC bits. The 8 stores are fast stores in this design since cells are stored into the local memory and even in pipelined from the network buffer into SONET interface.

This breakdown is used to predict the throughput for two configurations: one with pipelined burst dual-port network buffer, and the other in conventional system bus to transfer 48 byte data.

In case of the cache-based host-network interface:

$$\begin{aligned}
Throughput &= \frac{48\ byte/cell \times 8\ bit/byte}{(3L^{s}_{\mu p} + 3S^{f}_{\mu P})T_{\mu P} + (6L^{f}_{\mu C} + 7S^{f}_{\mu C} + 6\ computation)T_{\mu C}} \\
&= \frac{384}{(11+5)(30.3\times 10^{-9}) + (8+9+6)(25\times 10^{-9})} \\
&= 362.33 Mbps
\end{aligned}$$

In contrast to this cache-based network adaptor, the conventional network adaptor loads control information first, then loads and stores directly cell payload into cell FIFO, [19]. There are 1 load for control information, 6 loads and 7 stores for data transfer. These 6 slow loads take 7-2-2-2 and 7-2 clock cycles for data transfer which is a 64-bit wide. There are 7 stores that are 6 for data and 1 for header. The conventional network adaptor has the following throughput equation:

$$\begin{aligned}
Throughput &= \frac{48\ byte/cell \times 8\ bit/byte}{(L^s_{\mu p} + S^s_{\mu p})T_{\mu p} + (6L^s_{\mu c} + 7S^f_{\mu C} + 6computation)T_{\mu C}} \\
&= \frac{384}{(7+7)(30.3 \times 10^{-9}) + (22+9+6)(25 \times 10^{-9})} \\
&= 284.61 Mbps
\end{aligned}$$

The throughputs will be compared in terms of variable-sized CS-PDUs for transmitting procedure.

Figure 5.4 shows the comparisons for segmentations of the cache-based and the conventional.

Receiving Performance

When the network adaptor receives any ATM cell from the ATM network switch, it splits header bytes from the ATM cell, checking CRC, and if no error is detected then the adaptor-control processor stores the payload to the network buffer. The SAR layer performs the reassembly of ATM cells into the original large data (CS-PDUs) of the higher layers at the receiving unit.

The information payload for each ATM cell contains the AAL-specific information such as the information field length and the sequence number that must be passed between peer AAL entities. This information is stored in the AAL layer protocol processing, not in the header of each cell.

The protocol processing can be improved by using the technique which keeps the protocol state information of the most recent reassembly in registers. It is called hints [19]. After processing each cell, an expected SAR header for the next cell can be computed, which is used as a hint. If it matches, then there is no need to check

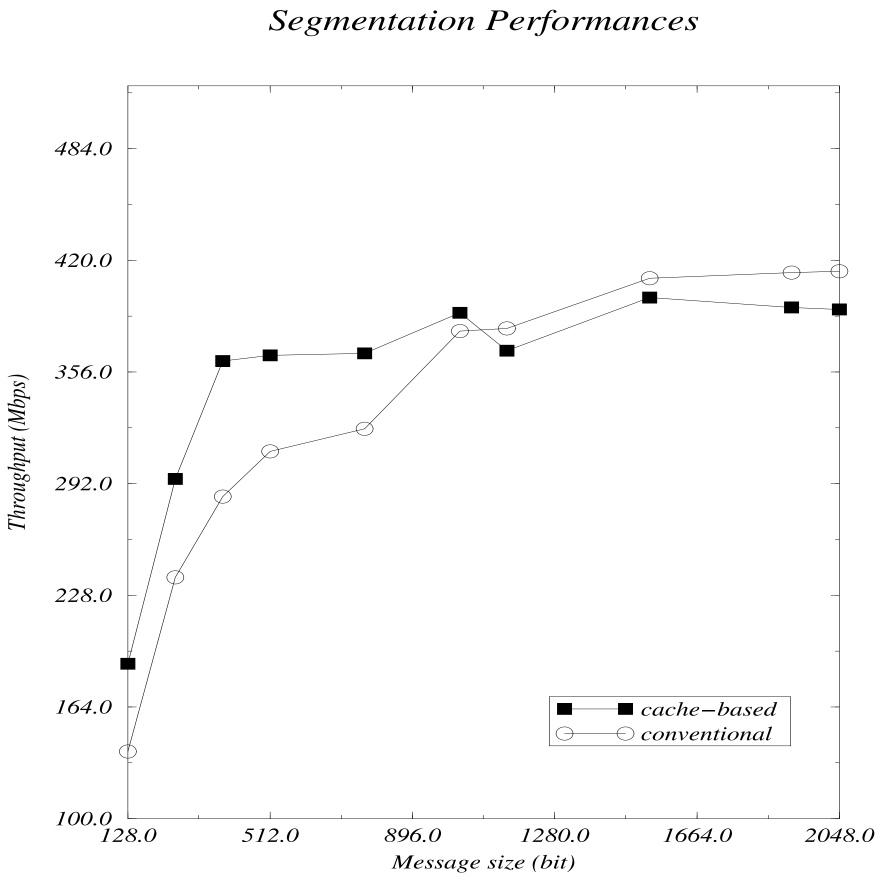


Figure 5.4: Throughput of transmitting procedure with variable CS-PDU sizes

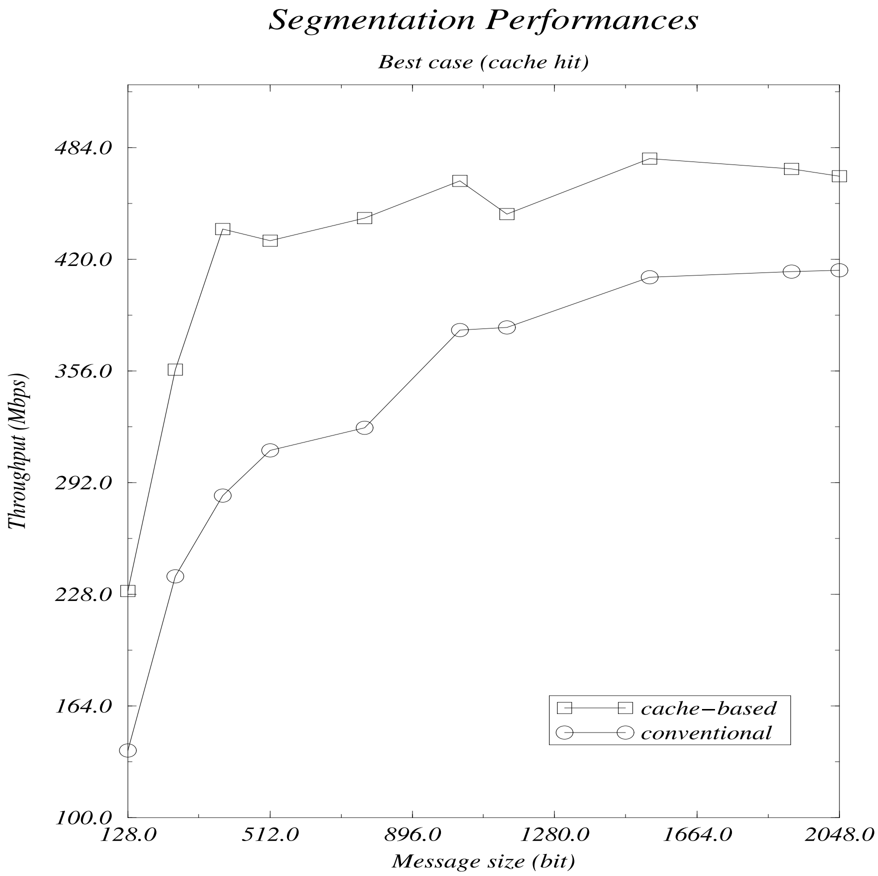


Figure 5.5: Throughputs of transmitting procedure with variable CS-PDU sizes on cache

any additional fields.

With this technique, the best case occurs when the cached reassembly state information is always valid where only single ATM cell arrives at a time. In this case, reassembly processing requires the following: Alternatively, the reassembly processing of the intelligent interface does not need the bandwidth of the system bus for the slow loads in which the conventional architecture needs because the host processor can access directly to the reassembly buffer (dual-port SRAM) as cache access (fast loads). By the back-to-back cycling of data transfer, 6 fast loads (3-1-1-1-1-1) and 6 stores (3-1-1-1-1-1) are required for 48 Bytes of a ATM cell payload. Therefore, the throughput for a ATM cell is as follows:

$$\begin{aligned} Throughput &= \frac{48Bytes}{(6L^f_{\mu c} + 6S^f_{\mu c} + 12\ computation)T_{uC}} \\ &= \frac{384}{(8+8+12)25 \times 10^{-9}} \\ &= 548.57\ Mbps \end{aligned}$$

On the other hand, the reassembly on the conventional architecture requires the data transfer from the network buffer to the main memory. Moreover, the best case is with data transfers of fully pipelined, that is, 3 loads and 3 stores consecutively (for detail description, see previous section, transmitting performance).

$$\begin{aligned} Throughput &= \frac{48Bytes}{(3L^s_{\mu p} + 3S^f_{\mu p})T_{\mu p} + (6L^f_{\mu c} + 6S^f_{\mu c} + 12\ comp)T_{uC}} \\ &= \frac{384}{(11+5)30.3 \times 10^{-9} + (8+8+12)25 \times 10^{-9}} \\ &= 324.11\ Mbps \end{aligned}$$

The performance comparisons are in Figure 5.6.

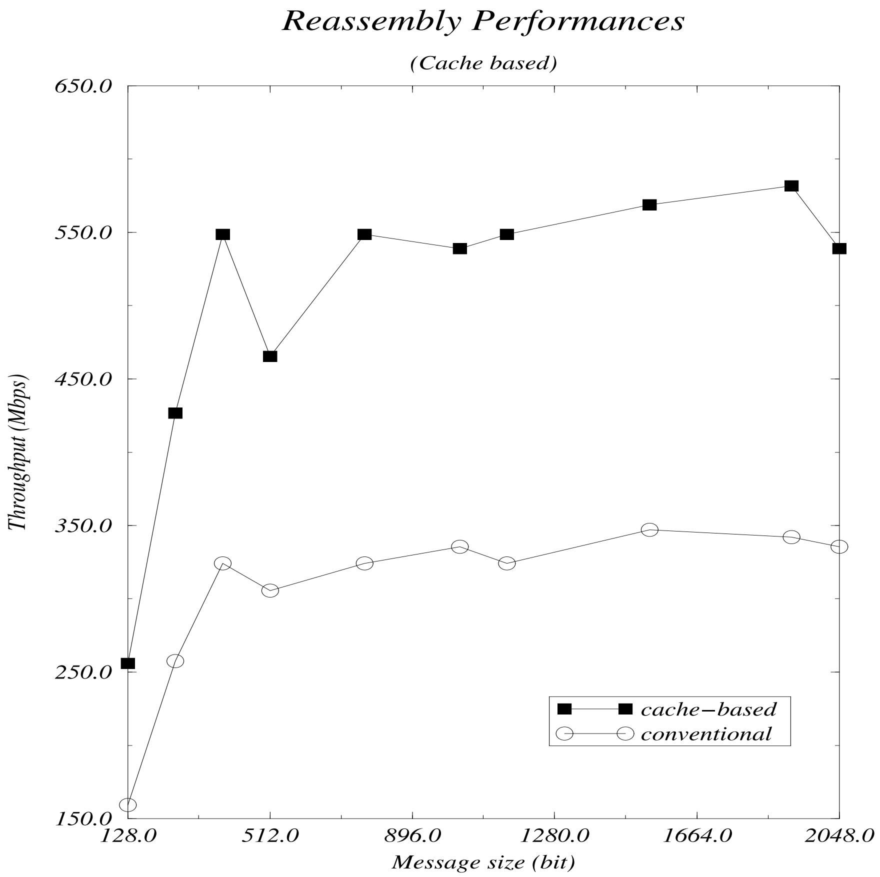


Figure 5.6: Throughputs of receiving procedures with variable CS-PDU sizes

5.4 Summary

In this chapter, we have analyzed latencies of data transfers between a host and host-network interfaces designed by our taxonomy and shown their effective transfer rates under different host-network interfaces. We have analyzed performances of a simple interface and an intelligent interface, and compared those with performance of a conventional host-network interface.

In the simple interface, the segmentation processing has been performed by the host processor. It caused the system bus to be the bottleneck of data transfer. In segmentation, the host must have been interrupted much less because DMA transferred data directly into network FIFO with burst mode. However, in reassembly, the DMA transferred data to the memory through the system bus without transport layer CRC checking. If there are errors which can not be cured, the data might cause the system component to be wasted.

In the intelligent interface, the network buffer seems to be a cache. It saves time to transfer data from the host because the host can access cache in fast access mode. The segmented data, therefore, can be sent to the network buffer memory without precomputation of start addresses and offsets which are usually used on conventional host interfaces. In reassembly processing, cache-based interface does not use the system bus and data moves in fast access mode to the host. The performance has supported the functionality of cache-based interface for real-time application support, low latency, multiprotocol support, and so on.

In designing host-network interfaces, we have focused on reducing overheads of unnecessary protocol processing and parallelizing procedures as possible. As supposed, using taxonomy on design of host-network interface saves time to develop and deploy for the special purpose.

Chapter 6

Conclusion

6.1 Summary and Conclusion

The recent advances in high-speed network technologies have made parallel and distributed computing over a Network of Workstations (NOW) an attractive and cost-effective computing environment.

However, the performance of parallel and distributed applications running on NOW resources suffers from high latency and low throughput even when the network speed is in Megabit or Gegabit per second range.

The main reasons for the low performance are contributed to the design of the host-network interface and the software mechanisms which are used to transmit data from the host to the network and vice versa.

In this dissertation, we have reviewed host-network interface designs and propose a hierarchical taxonomy to characterize and described the architectures of host network interface. The taxonomy consists of two levels of characterization: Architectural level and protocol level.

In the architectural level, we have described the main components used to implement any host-network interface and how they interact with each other to perform the host-network interface functions. The host CPU unit is to execute instructions of the user application and to transform data. Main memory unit is an intelligent storage device that passes data to and from the host CPU. Cache unit consists of a small fast memory that acts as a buffer between the main memory and the host CPU. Network buffer unit is a staging and speed-matching area for data in transmit between the host and the network. Protocol processor unit manages packet processing and various bookkeeping functions associated the protocol. And the switch unit provides connectivity between other functional units in one way of programmed I/O, DMA, burst transfer, or register accessing.

In the protocol level, we have discussed how the software functions (data copying, packetization, error handling, flow control, routing) are mapped into the main host-network-interface components identified at the architecture level.

We have also provided several examples that show how our taxonomy can accurately describe the architectures of host-network interface and characterize the performance limitations of any host-network interface design as well.

Simple host-network interface design is flexible for protocol implementation, reduced implementation complexity, minimal hardware components, and data transfer between main memory and FIFOs by direct memory access. However, the host is likely to be overloaded by the protocol processing since the simple interface does not have on-board processor.

The intelligent interface supports very high-speed and low-latency ATM networks. To make a good cost-performance efficiency, components are already developed or used in common. Above all, the intelligent interface supports no copying of data in main memory. Data moves from user space of main memory to the transmitting FIFO through dual-ported cache memory working as a network buffer memory. Since dual-ported pipelined and burst mode SRAM is used for cache and network buffer memory, it can support faster data transfer than data transfer using DMA. It results in supporting real-time applications better.

We have analyzed the performance of the proposed intelligent host-network interface. On data transmitting, the message buffer has different performance throughput whether the data is in cache or not. When the data is not in cache, the interface has improved performance throughput only with small-size messages. For large-size message, cache adds more overhead on writing back least recently used data into the main memory than for small-size message. It reduces the performance throughput. When data is in cache, the protocol processor can access directly without data copying. It causes a leap of performance improvement. On receiving data from the network, after the protocol processor executes the intended protocol processing, the host can access directly data through the network buffer. It saves the overhead given on data copying by the I/O bus. Thus, the user applications which require low level of communication latency can get a real advantage of high-speed network.

6.2 Future Work

Under the proposed taxonomic system, the organization of an efficient host-network interface heavily depends on the external constraints placed on the host-network interface such as the nature of the buses and memory system on the host, the application programming interface used for communication, and the placement of the checksum in the packet. That is, for the same host-network interface in the architectural level, the minimum number of data transfers across the bus can differ depending on the protocol-processing constraints. This relationship also suggests a number of unexplored possible architectures that might be fruitful places to look for new and innovative designs. We envision some key directions for future extension of the research presented in this thesis.

For the taxonomic system, the protocol implementation level can be extended to include application programming interface (API). The overhead of operating system affects API implementations and causes the whole communication network environment to slow down, even though the network interface has low latency and the host processes applications with high computing power. Thus, the API should be treated as one important functional unit in protocol implementation level of the taxonomic system to analyze more accurately host-network interfaces. One example is Jaguar which is a Java API to improve performance of communication and I/O methods in Java programming.

The fundamental issues in designing balanced machine are to provide the ability to overlap communication and computation and to reduce communication overhead. Thus, there can be some extension for the design of host-network interface.

The transport protocol can be extended to the TCP/IP implementation. Since the TCP/IP suite was developed long time ago, it has been advanced to support high-speed communication applications with multithreads, shared memory user space processing, and so on.

Therefore, our taxonomy has great potential to be applicable to development of optimal network environment by estimating the exact system's requirement and parallelizing desings of hardware and software simultaneously, which comes from descriptive and hierarchical features of our taxonomy.

Bibliography

[1] Virtual Interface Architecture Specification. Draft revision 1.0, Compaq, Intel, and Microsoft Corporations, http://www.viarch.org, December 1997.

[2] Selim G. Akl. *"The Design and Analysis of Parallel algorithms"*. Prentice-Hall, 1989.

[3] Jaime Jungok Bae and Tatsuya Suda. Survey of Traffic Control Schemes and Protocols in ATM Networks. *IEEE proceedings of*, 79(2):170–189, February 1991.

[4] David Banks and Michael Prudence. "A High-Performance Network Architecture for a PA-RISC Workstation". *IEEE Journal on Selected Areas in Communications*, 11(2):191–202, February 1993.

[5] Edoardo Biagioni, Eric Cooper, and Robert Sansom. "Designing a Practical ATM LAN". *IEEE Network*, page 32, March 1993.

[6] Matthias A. Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki, and Edward W. Felten. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153, April 1994.

[7] C. Brendan, S. Traw, and Jonathan M. Smith. "A High-Performance Host Interface for ATM Networks". *Computer Engineering Design (ACM)*, pages 317–325, 1991.

[8] C. Brendan, S. Traw, and Jonathan M. Smith. Hardware/Software Organization of a High-Performance ATM Host Interface. *IEEE Journal on selected areas in communications*, 11(2):240–253, February 1993.

[9] Geoffrey M. Brown, Mohamed G. Gouda, and Raymond E. Miller. Block Acknowledgement: Redesigning the Window Protocol. *Proc. ACM SIGCOMM Symp. on Commun., Architectures, and Protocols*, pages 128–135, 1989. Austin, TX.

[10] P. Buonadonna, A. Geweke, and D. Culler. "An Implementation and Analysis of the Virtual Interface Architecture". *In Proceedings of SC'98*, November 1998.

[11] Yau chau Ching and H. Sabit Say. "SONET Implementation". *IEEE Comm. Magazine*, page 34, September 1993.

[12] Chien Chen, Yizhi Lu, and Anthony Wong. Microarchitecture of HaL's Cache Subsystem. Technical report, HaL Computer Systems, 1996. HaL Computer Systems, Inc. 1315 Dell Ave.

[13] David R. Cheriton. VMTP: A Transport Protocol for the Next Generation of Communication Systems. *Proc. ACM SIGCOMM Symp.: Commun., Architectures, and Protocols*, pages 353–415, 1986.

[14] G. Chesson. The Protocol Engine Design. *Proceedings of the Summer 1987 USENIX Conference*, pages 209–215, November 1987.

[15] Nim K. Cheung. "The Infrastructure for Gigabit Computer Networks". *IEEE Comm. Magazine*, page 60, April 1992.

[16] D. D. Clark and D. L. Tannenhouse. Architectural Considerations for a New Generation of Protocols. *Proceedings of the SIGCOMM Symposium on Comm. Arch. and Protocols*, pages 200–208, September 1990.

[17] David D. Clark, Van Jacobson, John Romkey, and Howard Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications Magazine*, pages 23–29, June 1989.

[18] David D. Clark, Mark L. Lambert, and Lixia Zhang. NETBLT: A High Throughput Transport Protocol. *ACM SIGCOMM Workshop: Frontiers in Computer Net.*, pages 353–359, 1987.

[19] Eric Cooper, Onat Menzilcioglu, Robert Sansom, and Francois Bitz. "Host Interface Design for ATM LANs". *IEEE Computer Society*, pages 247–258, October 1991.

[20] Eric C. Cooper, Peter A. Steenkiste, Robert D. Sansom, and Brian D. Zill. Protocol Implementation on the Nectar Communication Processor. *ACM*, pages 135–144, 1990.

[21] Digital Equipment Corporation. *VT-220 Owner's Manual*, 1984.

[22] Intel Corporation. *LAN Components User's Manual*, 1984. order number 230814-001.

[23] Vitesse Semiconductor Corporation. *Preliminary VSP947/VSP948 Data Sheet.* Vitesse Semiconductor Corporation, August 1994.

[24] Roger Cummings, Carl Zeitler, and Don Tolmie. High-Performance Parallel Interface - Mapping to Asynchronous Transfer Mode. working draft proposed american national standard for information system, X3T11, E-mail: Roger_Cummings@Stortek.com, June 1994.

[25] Chris Dalton, Greg Watson, David Banks, Costas Calamvokis, Aled Edwarrds, and John Lumley. Afterburner. *IEEE Network*, pages 36–43, July 1993.

[26] Subrata Dasgupta. A Hierarchical Taxonomic System for Computer Architectures. *IEEE Computer*, pages 64–74, March 1990.

[27] Bruce S. Davie. "A Host-Network Interface Architecture for ATM". *ACM*, pages 307–315, 1991.

[28] Bruce S. Davie. The Architecture and Implementatin of a High-Speed Interface. *IEEE Journal on selected areas in communications*, 11(2):228 – 239, February 1993.

[29] Keith Diefendorff and Michael Allen. "Organization of the Motorola 88110 Superscalar RISC Microprocessor". *IEEE Micro*, pages 40–63, April 1992.

[30] Willibald A. Doeringer, Doug Dykeman, Matthias Kaiserswerth, Bernd Werner Meister, and Harry Rudin. A Survey of Light-Weight Transport Protocols for High-Speed Networks. *IEEE Transactions on Communications*, 38(11):2025–2039, November 1990.

[31] Peter Druschel, Larry L. Peterson, and Bruce S. Davie. Experiences with a High-Speed Network Adaptor: A Software Perspective. *ACM SIGCOMM*, August 1994.

[32] Order Number: EC-QP99A-TE. *Digital Semiconductor 21164 Alpha Microprocessor: Hardware Reference Manual.* Digital Equipment Corporation, March 1996.

[33] A. E. Eckberg. "B-ISDN/ATM Traffic and Congestion Control". *IEEE Network*, page 28, September 1992.

[34] David C. Feldmeier. Multiplexing Issues in Communication System Design. *In Proc. ACM SIGCOMM*, pages 209–219, 1990.

[35] M. J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, September 1972.

[36] Reinhard Franz, Klaus D. Gradischnig, Manfred N. Huber, and Rolf Stiefel. ATM-Based Signaling Network Topics on Reliability and Performance. *IEEE Journal on Selected Areas in Communications*, 12(3):517–524, April 1994.

[37] A. G. Fraser and William T. Marshall. Data Transport in a Byte Stream Network. *IEEE Journal on Selected Area of Communications*, 7(7):1020–1033, September 1989.

[38] Jin fu Chang and Rong feng Chang. The Behavior of a Finite Queue with Batch Poisson Inputs Resulting from Message Packetization and a Synchronous Server. *IEEE Trans. on Comms*, COM-32(12):1277–1285, December 1984.

[39] George W. Gorsline. *Computer Organization: hardware,software.* Prentice-Hall, second edition, 1986.

[40] Kunio Goto, Yutaka Takahashi, and Toshiharu Hasegawa. An Approximate Analysis on Controlled Tandem Queues. *TK5105 .5 M63 1984*, 1984.

[41] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir. The NYU Ultracomputer–Designing an MIMD Shared Memory Parallel Computer. *IEEE Computer*, c-32(2):175–189, February 1983.

[42] Graham, Knuth, and Patashnik. *Concrete mathematics*. Addison-Wesley, 1990.

[43] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley, second edition, 1985.

[44] Zygmunt Haas. A Protocol Structure for High-Speed Communication over Broadband ISDN. *IEEE Network Magazine*, pages 64–70, January 1991.

[45] Salim Hariri. *High Performance Distributed Systems: Network, Architecture and Programming*. Artech House, 1998 expected.

[46] Peter G. Harrison and Naresh M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1992.

[47] John Heinlein, Kourosh Gharachorloo, Scott Dresser, and Anoop Gupta. Integration of Message Passing and Shared Memory in the Stanford FLASH Multiprocessor. *Proceedings of the sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

[48] John L. Hennessy and David A. Patterson. *Computer Architecture a Quantitative Approach*. Morgan Kaufmann, 1996.

[49] Dana S. Henry and Christopher F. Joerg. The Network Interface Chip. Technical report, Massachusetts, June 1991.

[50] Dana S. Henry and Christopher F. Joerg. "A Tightly-Coupled Processor-Network Interface". *ACM*, pages 111–123, 1992.

[51] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet Desk Area Network: Architecture, Implementation, and Experience. *IEEE Journal of Selected Areas in Communications*, pages 710 – 721, May 1995.

[52] Kai Hwang. *Advanced Computer Architecture: parallelism, scalability and progrmmability*. QA 76.9.A73H87. McGrall-Hill, 1993.

[53] IDT. High-Speed 32K x16 Synchronous Pipelined Dual-Port Static RAM. Preliminary IDT709279, Integrated Device Tech., September 1996.

[54] intel semiconductor. *82430FX PCIset Cache/Memory Subsystem: 82437FX System Controller; 82438FX Data Path*, June 1995.

[55] Niraj Jain, Mischa Schwartz, and Theodore R. Bashkow. Transport protocol processing at gbps rates. *Symp. SIGCOMM'90-Commun. Architecture and Protocols, Philadelphia, PA*, pages 188–199, October 1990.

[56] Raj Jain. A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks. *IEEE Journal on Selected Areas in Communications*, SAC-4(7):1162–1167, October 1986.

[57] Hemant Kanakia and David R. Cheriton. "The VMP Network Adapter Board (NAB)". *ACM*, pages 175–187, August 1988.

[58] Leonard Kleinrock. *Queueing Systems*, volume 1. John Wiley & Sons, 1975.

[59] Leonard Kleinrock and Richard Gail. *Solutions Manual for Queueing Systems volume I: Theory*. Technology Transfer Institute, Santa Monica, CA 90402, 1982.

[60] John Kubiatowicz and Anant Agarwal. Anatomy of a Message in the Alewife Multiprocessor. *ACM International Conference on Supercomputing*, July 1993.

[61] Paul J. Kuehn. Approximate Analysis of General Queuing Networks by Decomposition. *IEEE Trans. on Comms*, COM-27(1):113–126, January 1979.

[62] Paul J. Kuhn, Charles D. Pack, and Ronald A. Skoog. Common Channel Signaling Networks: Past, Present, Future. *IEEE Journal on Selected Areas in Communications*, 12(3):383–393, April 1994.

[63] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. The Benjamin/Cummings, 1994.

[64] H. T. Kung, Trevor Blackwell, and Alan Chapman. Credit-based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statiscal Multiplexing. *ACM SIGCOMM '94 Symposium on Communications Architectures, Protocols and Applications*, 1994.

[65] J. Bryan Lyles and Daniel C. Swinehart. "The Emerging Gigabit Environment and the Role of Local ATM". *IEEE Comm. Magazine*, page 52, April 1992.

[66] Alan M. Mainwaring and David E. Culler. Design Challenges of Virtual Networks: Fast, General-Purpose Communication. *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP). Atlanta, Georgia*, May 1999.

[67] David R. Manfield and P. Tran-Gia. Analysis of a Finite Storage System with Batch Input Arising out of Message Packetization. *IEEE Trans. on comm.*, COM-30(3):456–463, March 1982.

[68] David E. McDysan and Darren L. Spohn. *ATM Theory and Application*. McGraw-Hill, 1995.

[69] Onat Menzilcioglu and Steven Schlick. "Nectar CAB : A High-Speed Network Processor". *IEEE*, pages 508–515, 1991.

[70] Ron Minnich, Dan Burns, and Frank Hady. The Memory-Integrated Network Interface. *IEEE Micro*, pages 11–20, February 1995.

[71] Ateven E. Minzer. "Broadband ISDN and Asynchronous Transfer Mode(ATM)". *IEEE Comm. Magazine*, page 17, September 1989.

[72] Shubhendu S. Mukherjee, Babak Falsafi, Mark D. Hill, and David A. Wood. Coherent Network Interfaces for Fine-Grain Communication. *Computer Architecture News*, 24(2):247–258, May 1996.

[73] Sape Mullender. *Distributed Systems*. Addison-Wesley, 2nd edition edition, 19993.

[74] A. Netravali. "Design and Implementation of a High-Speed Transport Protocol". *IEEE Transactions on communications*, 38(11):2010–2024, November 1990.

[75] Gerald W. Neufeld, Mabo Robert Ito, Murray Goldberg, Mark J. McCutcheon, and Stuart Ritchie. "Parallel Host Interface for an ATM Network". *IEEE Network*, page 24, July 1993.

[76] Peter Newman. Backward Explicit Congestion Notification for ATM Local Area Networks. *IEEE Globecom*, 1993.

[77] Christos Papadopoulos and Gurudatta M. Parulkar. Experimental Evaluation of SUNOS IPC and TCP/IP Protocol Implementation. *IEEE Transactions on Networking*, 1(2):199–216, April 1993.

[78] Achille Pattavina. Nonblocking Architectures for ATM Switching. *IEEE Communications Magazine*, pages 38–48, February 1993.

[79] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware and Software Interface*. Morgan Kaufmann, 1995.

[80] Thomas F. La Porta and Mischa Schwartz. Architectures, Features, and Implementation of High-Speed Transport Protocols. *IEEE Network Magazine*, pages 14–22, May 1991.

[81] Franklin P. Prosser and David E. Winkel. *The Art of Desigital Design*. Prentice-Hall, 2 edition, 1987.

[82] Martin De Prycker. *"ASYNCHRONOUS TRANSFER MODE"*. Ellis Horwood, second edition, 1993.

[83] C Raghavendra. *Interprocess Communication Networks for parallel system*. M, 1994.

[84] K. K. Ramakrishnan. Performance Considerations in Designing Network Interfaces. *IEEE Journal on selected areas in Communications*, 11(2):203–219, February 1993.

[85] Steven K. Reinhardt, Robert W. Pfile, and David A. Wood. Decoupled Hardware Support for Distributed Shared Memory. *The Proceedings of the 23rd International Symposium on Computer Architecture (ISCA)*, May 1996.

[86] Leszek Reiss. *Introduction to Local Area Networks with Microcomputer Experiments.* Prentic-Hall, 1987.

[87] Marcel-Catalin Rosu, Karsten Schwan, and Richard Fujimoto. Supporting Parallel Applications on Clusters of Workstations: the Intelligent Network Interface Approach. *IEEE proceedings on 6th HPDC conference*, pages 159–168, August 1997.

[88] Yoshito Sakurai, Nobuhiko Ido, Shinobu Gohara, and Noboru Endo. Large-Scale ATM Multistage Switching Network with Shared Buffer Memory Switches. *IEEE Communications Magazine*, pages 90–96, January 1991.

[89] Robert M. Sanders and Alfred C. Weaver. *The Xpress Transfer Protocol(XTP)–A Tutorial.* ACM Computer Communication Review, rms4t@virginia.edu, weaver@virginia.edu, October 1990.

[90] Mischa Schwartz. *Telecommunication Networks; Protocols, Modeling and Analysis.* Addison-Wesley, 1987.

[91] Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *Computer Structures: Principles and Examples.* McGraw-Hill, 1982.

[92] David B. Skillicorn. A Taxonomy for Computer Architectures. *IEEE Computer*, pages 46–57, November 1988.

[93] Donald E. Smith and H. Jonathan Chao. Sizing a Packet Reassembly Buffer at a Host Computer in an ATM Network. *IEEE/ACM Trans. on Networking*, 3(6):798–808, December 1995.

[94] Peter A. Steenkiste. A Systematic Approach to Host Interface Design for High-Speed Networks. *IEEE Computer*, pages 47–57, March 1994.

[95] Peter A. Steenkiste, Brian D. Zill, H. T. Kung, Steven J. Schlick, Jim Hughes, Bob Kowalski, and John Mullaney. *"A Host Interface Architecture for High-Speed Networks"*, pages 31–46. High Performance Networking IV. Elsevier Science, 1993.

[96] James P. G. Sterbenz and Gurudatta M. Parulkar. Design of a Gigabit Host-Network Interface. Technical Report jhsn93.ps, Washington University at St. Louis, MO, 1993.

[97] James P.G. Sterbenz and Gurudatta M. Parulkar. Axon: Host-Network Interface Architecture for Gigabit Communications. *Protocols for High Speed Networks, II*, pages 211–236, 1991.

[98] W. Richard Stevens. *"UNIX Network Programming"*. Prentice-Hall, 1991.

[99] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall, 3rd edition, 1996.

[100] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active Messages: a Mechanism for Integrated Communicatin and Computation. *Proceedings of the 19th International Symposium on Computer Architecture, ACM*, May 1992.

[101] Frederick A. Ware. *Base RDRAM Design Guide.* Number 3.0. Rambus Inc., 1996.

[102] Matt Welsh and David Culler. Jaguar: Enabling Efficient Communication and I/O from Java. *Concurrency: Practice and Experience, Special Issue on Java for High-Performance Applications*, page http://www.cs.berkeley.edu/ mdw/proj/jaguar, December 1999.

[103] Marek R. Wernik and Ernst A. Munter. Broadband Public Network and Switch Architecture. *IEEE Communications Magazine*, pages 83–89, January 1991.

[104] J. H. Wilkinson. *"The Algebraic Eignevalue Problem"*. Clarendon Oxford, 1965.

[105] Ellen E. Witte. "A Quantitative Comparison of Architectures for ATM Switching Systems". Technical report, Washington Univ., Dept. of CS Campus Box 1045 Washington Univ. ST. Louis, MO 63130-4899, October 1991. mosaic http://wuarchive.wustl.edu/techreport/CS/.

[106] Lixia Zhang. Why TCP Timers Don't Work Well. *Proc. ACM SIGCOMM Symp. on Commun., Architectures and Protocols*, pages 397–405, 1986.

[107] M. Zitterbart. Parallelism in Communication Subsystems. *IBM Research Report RC 18327*, September 1992.

[108] Brian Zucker. Advances in PC Memory Subsystems: Pipelined-Burst Cache and EDO DRAM. http://www.us.dell.com/prodinfo/R&D_Briefs/vectors/1/vect_edo.html, Dell Dimension Product Line, 1996.