



High Performance Cluster Computing Architectures and Systems

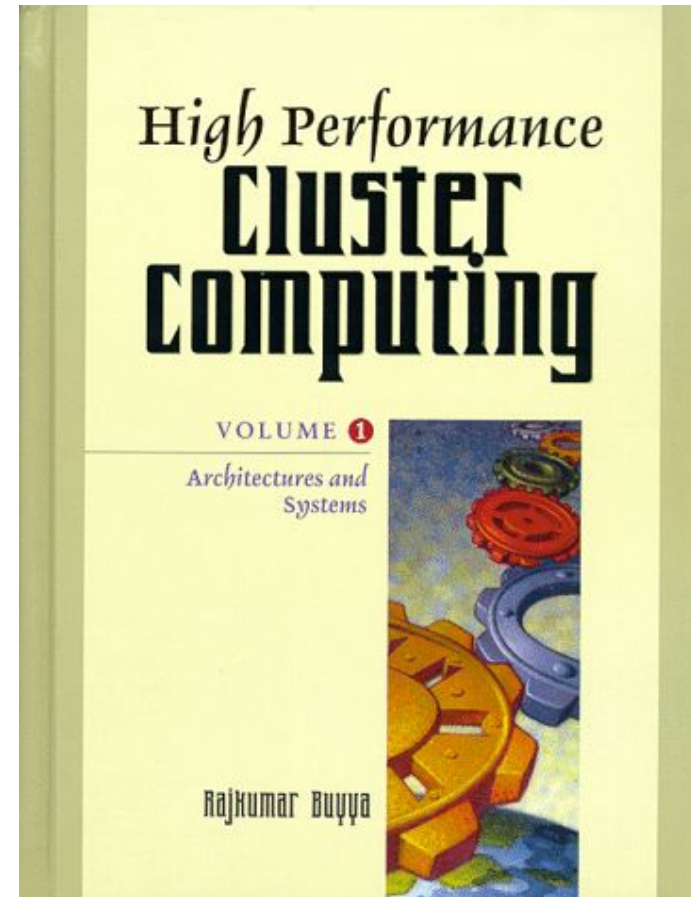
Hai Jin

Internet and Cluster Computing Center

Huazhong University of Science & Technology

Lightweight Messaging Systems

- Introduction
- Latency/Bandwidth Evaluation of Communication Performance
- Traditional Communication Mechanisms for Clusters
- Lightweight Communication Mechanisms
- Kernel-Level Lightweight Communications
- User-Level Lightweight Communications
- A Comparison Among Message Passing Systems



Introduction

- Communication mechanism is one of the most important part in cluster system
 - PCs and Workstations become more powerful and fast network hardware become more affordable
 - Existing communication software needs to be revisited in order not to be a severe bottleneck of cluster communications
- In this chapter
 - A picture of the state-of-the-art in the field of clusterwide communications
 - Classifying existing prototypes
- Message-passing communication
 - NOWs and clusters are distributed-memory architectures
 - These distributed-memory architectures are based on message-passing communication systems

Latency/Bandwidth Evaluation of Communication Performance

- Major performance measurements
 - Performance of communication systems are mostly measured by two parameters below
 - Latency, L
 - deals with the synchronization semantics of a message exchange
 - Asymptotic bandwidth, B
 - deals with the (large, intensive) data transfer semantics of a message exchange

Latency

■ Purpose

- Characterize the speed of underlying system to synchronize two cooperating processes by a message exchange

■ Definition

- Time needed to send a minimal-size message from a sender to a receiver
 - From the instant the sender starts a send operation
 - To the instant receiver is notified about the message arrival
- Sender and receiver are application level processes

■ Measure the latency, L

- Use a ping-pong microbenchmark
- L is computed as half the average round-trip time (RTT)
- Discard the first few data for excluding "warm-up" effect

End-to-end and One-sided Asymptotic Bandwidth (I)

■ Purpose

- Characterizes how fast a data transfer may occur from a sender to a receiver
- “Asymptotic”: the transfer speed is measured for a very large amount data
 - One, bulk, or stream

■ Definition of asymptotic bandwidth, B

- $B = S/D$
- D is the time needed to send S bytes of data from a sender to a receiver
- S must be very large in order to isolate the data transfer from any other **overhead** related to the synchronization semantics

End-to-end and One-sided Asymptotic Bandwidth (II)

- Measure the asymptotic bandwidth, B
 - End-to-end
 - Use a ping-pong microbenchmark to measure the average round-trip time
 - D is computed as half the average round-trip time
 - This measures the transfer rate of the whole end-to-end communication path
 - One-sided
 - Use a *ping* microbenchmark to measure the average send time
 - This measures the transfer rate as perceived by the sender side of the communication path, thus hiding the **overhead** at the receiver side
 - D is computed as the average data transfer time (not divided by 2)
 - The value of one-sided is greater than one of end-to-end

Throughput

- **Message Delay, $D(S)$**
 - $D(S) = L + (S - S_m)/B$
 - S_m is the minimal message size allowed by the system
 - half the round-trip time (ping-pong)
 - data transfer time (ping)
- **Definition of throughput, $T(S)$**
 - $T(S) = S/D(S)$
 - It is worth nothing that the asymptotic bandwidth is nothing but the throughput for a **very large message**
 - A partial view of the entire throughput curve
 - $T(S_h) = B / 2$
 - S_h : the message size

Traditional Communication Mechanisms for Clusters

- Interconnection of standard components
 - They focus on the standardization for interoperation and portability than efficient use of resources
- TCP/IP , UDP/IP and Sockets
- RPC
- MPI and PVM
- Active Message

TCP, UDP, IP, and Sockets

- The most standard communication protocols
- Internet Protocol (IP)
 - provides unreliable delivery of single packets to one-hop distant hosts
 - implements two basic kinds of QoS
 - connected, TCP/IP
 - datagram, UDP/IP
- Berkeley Sockets
 - Both TCP/IP and UDP/IP were made available to the **application level** through the API, namely Berkeley Sockets
 - Network is perceived as a **character device**, and sockets are **file descriptors** related to the device
 - Its level of abstraction is quite low

RPC

- Remote Procedure Call, RPC
 - Enhanced general purpose (specially distributed client-server applications) network abstraction atop socket
 - The de facto standard for distributed client-server applications
 - Its level of abstraction is high
- Familiarity and generality
 - sequential-like programming
 - Services are requested by calling procedures with suitable parameters. The called service may also return a result
 - hiding any format difference
 - It hides any format difference across different systems connected to the network in heterogeneous environment

MPI and PVM

- **General-purpose systems**
 - the general-purpose systems for message passing and parallel program management on **distributed platforms** at the application level, based on available IPC mechanisms
- **Parallel Virtual Machine (PVM)**
 - provides an easy-to-use programming interface for **process creation** and **IPC**, plus a **run-time system** for elementary application management
 - run-time programmable but inefficient
- **Message Passing Interface (MPI)**
 - offers a larger and more versatile set of routines than PVM, but does not offer run-time management systems
 - greater **efficient** compared to PVM

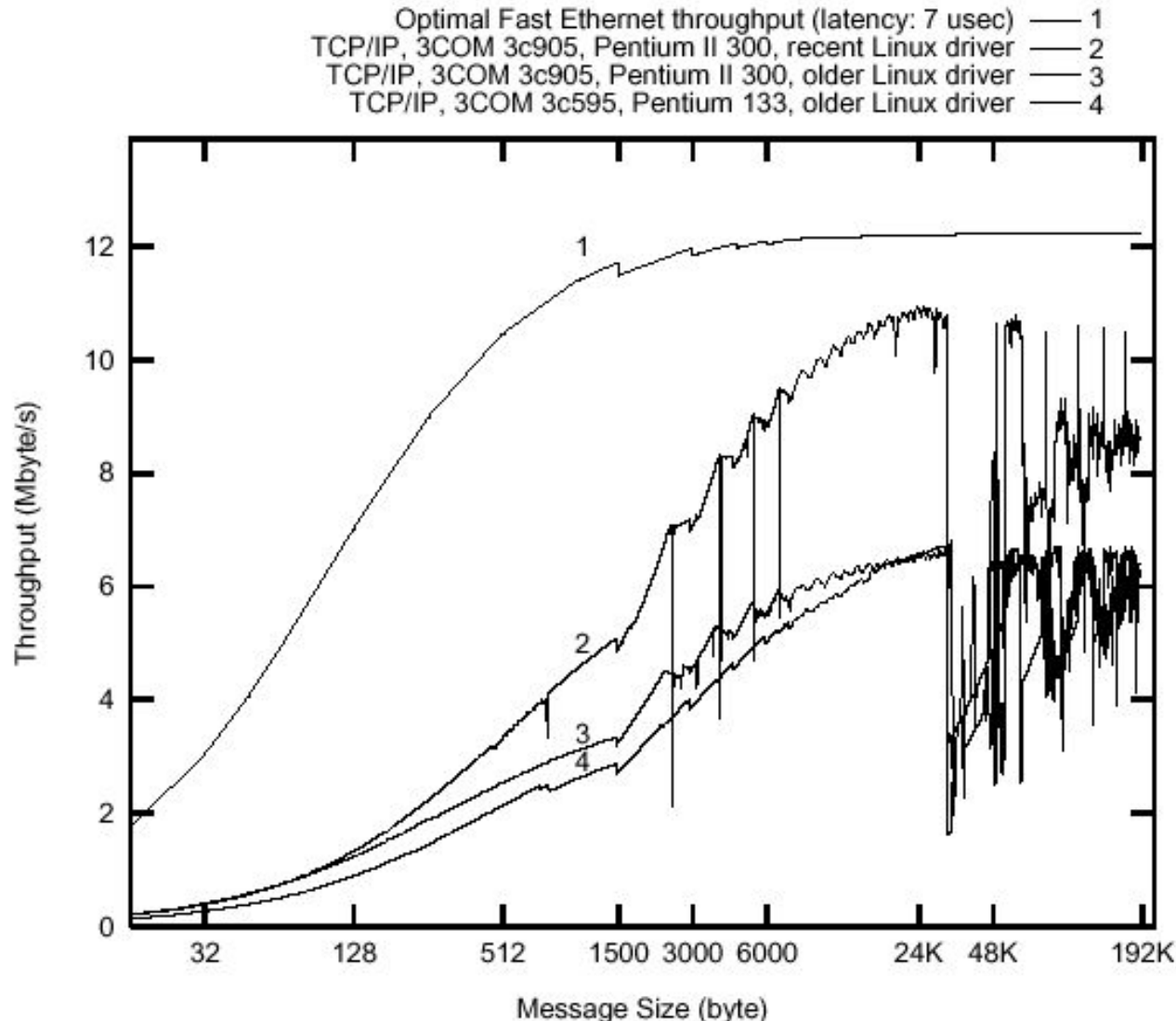
Active Message (I)

- One-sided communication paradigm
 - Whenever the sender process transmits a message, the message exchange occurs regardless of the current activity of the receiver process
- Reducing overhead
 - The goal is to reduce the impact of communication overhead on application performance
- Active Message
 - Eliminates the need of many temporary storage for messages along the communication path
 - With proper hardware support, it is easy to overlap communication with computation
 - As soon as delivered, each message triggers a user-programmed function of the destination process, called receiver handler
 - The receiver handler act as a separate thread consuming the message, therefore decoupling message management from the current activity of the main thread of the destination process

Lightweight Communication Mechanisms

- Lightweight protocols
 - Cope with the lack of efficiency of standard communication protocols for cluster computing
- Linux TCP/IP is not good for cluster computing
- Performance test in Fast Ethernet
 - environments
 - Pentium II 300 MHz, Linux kernel 2.0.29
 - 2 PCs are connected by UTP ported 3Com 3c905 Fast Ethernet
 - results
 - latency = 77 μ s (socket) / 7 μ s (card)
 - bandwidth
 - large data stream: 86%
 - short message (<1500 bytes): less than 50%
- Drawbacks of layered protocols
 - memory-to-memory copy
 - poor code locality
 - heavy functional overhead

Linux 2.0.29 TCP/IP Sockets: Half-Duplex "Ping-Pong" Throughput with Various NICs and CPUs



What We Need for Efficient Cluster Computing (I)

- To implement an efficient messaging system
 - Choose an appropriate LAN hardware
 - ex) 3COM 3c905 NIC can be programmed in two way
 - In descriptor-based DMA (DBDMA), the NIC **itself** performs DMA transfers between host memory and the network by '**DMA descriptors**'
 - In CPU-driven DMA, this leads to a 'store-and-forward' behavior
 - Tailor the protocols to the underlying LAN hardware
 - ex) flow control of TCP
 - TCP avoids packet overflow at the **receiver side**, but cannot prevent overflow to occur in a **LAN switch**
 - In cluster computing, the overflow in a LAN switch is important

What We Need for Efficient Cluster Computing (II)

- To implement an efficient messaging system
 - Target the protocols to the user needs
 - Different users and different application domains may need different tradeoffs between **reliability** and **performance**
 - Optimize the protocol code and the NIC driver as much as possible
 - Minimize the use of memory-to-memory copy operation
 - ex) TCP/IP
 - TCP/IP is the layered structure needed memory-to-memory data movements

Typical Techniques to Optimize Communication (I)

- Using multiple networks in parallel
 - Increases the aggregate communication bandwidth
 - Can not reduce latency
- Simplifying LAN-wide host naming
 - Addressing conventions in a LAN might be simpler than in a WAN
- Simplifying communication protocol
 - Long protocol functions are time-consuming and have poor locality that generates a large number of cache misses
 - General-purpose networks have a high error rate, but LANs have a low error rate
 - Optimistic protocols assume no communication errors and no congestion

Typical Techniques to Optimize Communication (II)

- Avoiding temporary buffering of messages
 - Zero-copy protocols
 - remapping the kernel-level temporary buffers into **user memory space**
 - lock the user data structures into **physical RAM** and let the NIC access them directly upon communication via **DMA**
 - need gather/scatter facility
- Pipelined communication path
 - Some NICs may transmit data over the physical medium while the host-to-NIC DMA or programmed I/O transfer is still in progress
 - The performance improvement is obtained at both latency and throughput

Typical Techniques to Optimize Communication (III)

- **Avoid system calls for communication**
 - Invoking a system call is a time-consuming task
 - User-level communication architecture
 - implements the communication system entirely at the **user level**
 - all buffers and registers of the NIC are remapped from **kernel space** into **user memory space**
 - **protection** challenges in a multitasking environment
- **Lightweight system calls for communication**
 - Eliminate the need of system calls
 - save only a subset of CPU registers and do not invoke the scheduler upon return

Typical Techniques to Optimize Communication (IV)

- Fast interrupt path
 - In order to reduce **interrupt latency** in interrupt-driven receives, the **code path** to the interrupt handler of the network device driver is optimized
- Polling the network device
 - The usual method of notifying **message arrivals** by **interrupts** is time-consuming and sometimes unacceptable
 - Provides the ability of explicitly **inspecting** or polling the network devices for incoming messages, besides interrupt-based arrival notification
- Providing very low-level mechanisms
 - A kind of RISC approach
 - Provide only very low-level primitives that can be combined in various ways to form higher level communication semantics and APIs in an 'ad hoc' way

The Importance of Efficient Collective Communication

- To turn the potential benefits of clusters into widespread use
 - The development of parallel applications exhibiting high enough performance and efficiency with a reasonable programming effort
- Porting problem
 - An MPI code is easily ported from one hardware platform to another
 - But performance and efficiency of the code execution is not ported across platforms
- Collective communication
 - Collective routines often provide the most frequent and extreme instance of “lack of performance portability”
 - In most cases, collective communications are implemented in terms of point-to-point communications arranged into standard patterns
 - This implies very poor performance with clusters
 - As a result, parallel programs hardly ever rely on collective routines

A Classification of Lightweight Communication Systems (I)

- Classification of lightweight communication systems
 - kernel-level systems and user-level systems
- Kernel-level approach
 - The messaging system is supported by the OS kernel with a set of low-level communication mechanisms embedding a communication protocol
 - Such mechanisms are made available to the user level through a number of OS system calls
 - Fit into the architecture of modern OS providing protected access
 - A drawback is that traditional protection mechanisms may require quite a high software overhead for kernel-to-user data movement

A Classification of Lightweight Communication Systems (II)

■ User-level approach

- Improves performance by minimizing the OS involvement in the communication path
- Access to the communication buffers of the network interface is granted without invoking any system calls
- Any communication layer as well as API is implemented as a user-level programming library
- To allow protected access to the communication devices
 - single-user network access
 - unacceptable to modern processing environment
 - strict gang scheduling
 - inefficient, intervening OS scheduler
 - Leverage programmable communication devices
 - uncommon device
- Addition or modification of OS are needed

Kernel-level Lightweight Communications

- Industry-Standard API system
 - Beowulf
 - Fast Sockets
 - PARMA²
- Best-Performance system
 - Genoa Active Message Machine (GAMMA)
 - Net^{*}
 - Oxford BSP Clusters
 - U-Net on Fast Ethernet

Industry-Standard API Systems (I)

■ Portability and reuse

- The main goal besides efficiency is to comply an industry-standard for the low-level communication API
- Does not force any major modification to the existing OS, a new communication system is simply added as an extension of the OS itself

■ Drawback

- Some optimization in the underlying communication layer could be hampered by the choice of an industry standard

Industry-Standard API Systems (II)

■ Beowulf

- Linux-based cluster of PCs
- channel bonding
 - two or more LANs in parallel
- topology
 - two-dimensional mesh
 - two Ethernet cards on each node are connected to horizontal and vertical line
 - each node acts as a software router

Industry-Standard API Systems (III)

■ Fast Sockets

- implementation of TCP sockets atop an Active Message layer
- socket descriptors opened at fork time are shared with child processes
- poor performance: UltraSPARC 1 connected by Myrinet
 - 57.8 μ s latency due to Active Message
 - 32.9 MB/s asymptotic bandwidth due to SBus bottleneck

Industry-Standard API Systems (IV)

■ PARMA²

- To reduce communication overhead in a cluster of PCs running Linux connected by Fast Ethernet
 - eliminate flow control and packet acknowledge from TCP/IP
 - simplify host addressing
- Retain BSD socket interface
 - easy porting of applications (ex. MPICH)
 - preserving NIC driver
- Performance: Fast Ethernet and Pentium 133
 - 74 μ s latency, 6.6 MB/s (TCP/IP)
 - 256 μ s latency for MPI/PARMA (402 μ s for MPI)
 - 182 μ s latency for MPIPR

Best-Performance Systems (I)

- Simplified protocols designed according to a performance-oriented approach
- Genoa Active Message Machine (GAMMA)
 - Active Message-like communication abstraction called Active Ports allowed a zero-copy optimistic protocol
 - Provide lightweight system call, fast interrupt path, and pipelined communication path
 - Multiuser protected access to network
 - Unreliable: raise error condition without recovery
 - Efficient performance (100base-T)
 - 12.7 μ s latency, 12.2 MB/s asymptotic bandwidth

Best-Performance Systems (II)

■ Net*

- A communication system for Fast Ethernet based upon a reliable protocol implemented at kernel level
- Remap kernel-space buffers into user-space to allow direct access
- Only a **single** user process per node can be granted network access
- Drawbacks
 - no kernel-operated network multiplexing is performed
 - user processes have to explicitly fragment and reassemble messages longer than the Ethernet MTU
- Very good performance
 - 23.3 μ s latency and 12.2 MB/s asymptotic bandwidth

Best-Performance Systems (III)

■ Oxford BSP Clusters

- Place some structural restriction on communication traffic by allowing only some well known patterns to occur
 - good to optimizing error detection and recovery
- A parallel program running on a BSP cluster is assumed to comply with the **BSP computational model**
- Protocols of BSP clusters
 - destination scheduling is different from processor to processor
 - switched network
 - using exchanged packets as acknowledgement packets
- BSPlib-NIC
 - the most efficient version of the BSP cluster protocol has been implemented as a device driver called BSPlib-NIC
 - remapping the kernel-level FIFOs of the NIC into user memory space to allow user-level access to the FIFOs
 - no need to "start transmission" system calls along the whole end-to-end communication path
 - Performance (100base-T)
 - 29 μ s latency, 11.7 MB/s asymptotic bandwidth

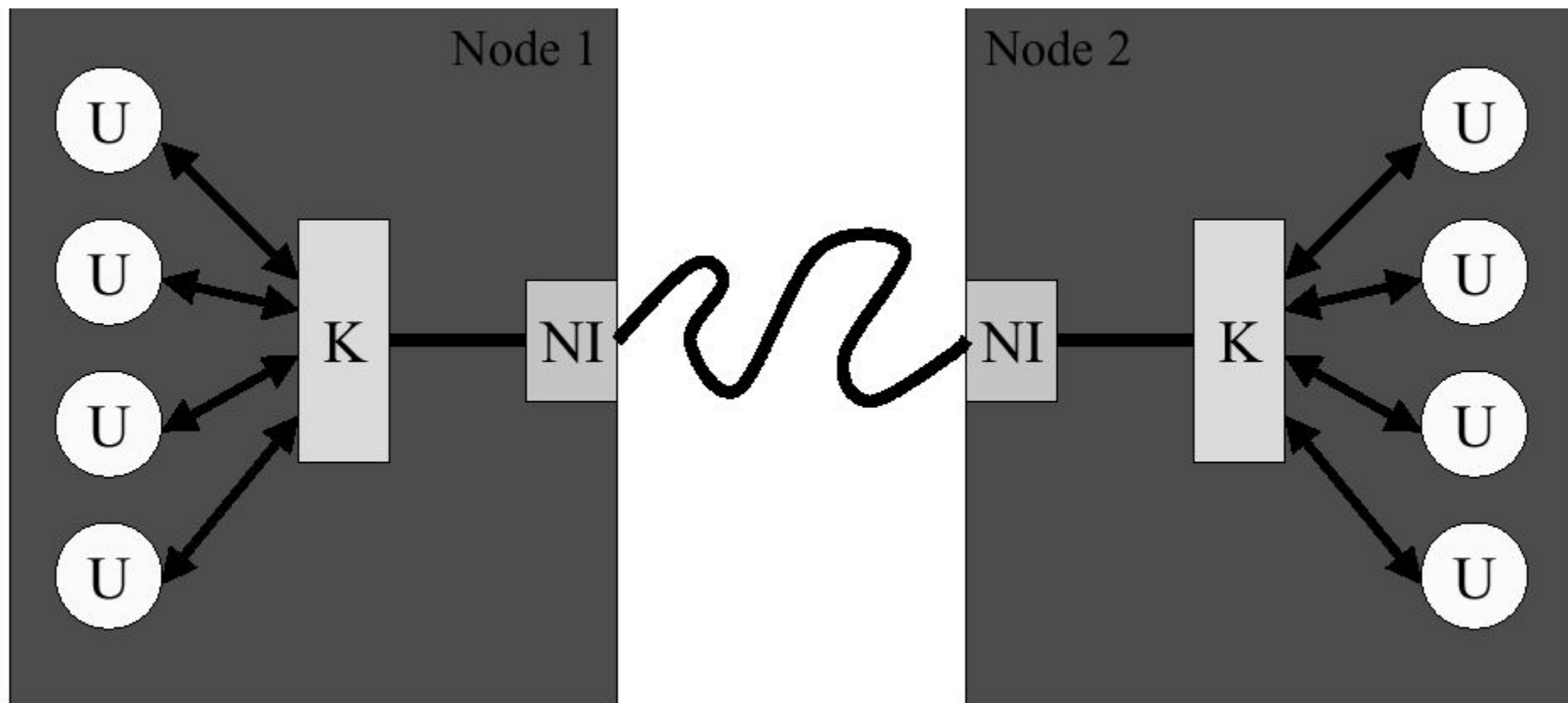
Best-Performance Systems (IV)

- U-Net on Fast Ethernet
 - Require a NIC's programmable onboard processor
 - The drawback is the very raw programming interface
 - Performance (100base-T)
 - 30 μ s one-way latency, 12.1 MB/s asymptotic bandwidth

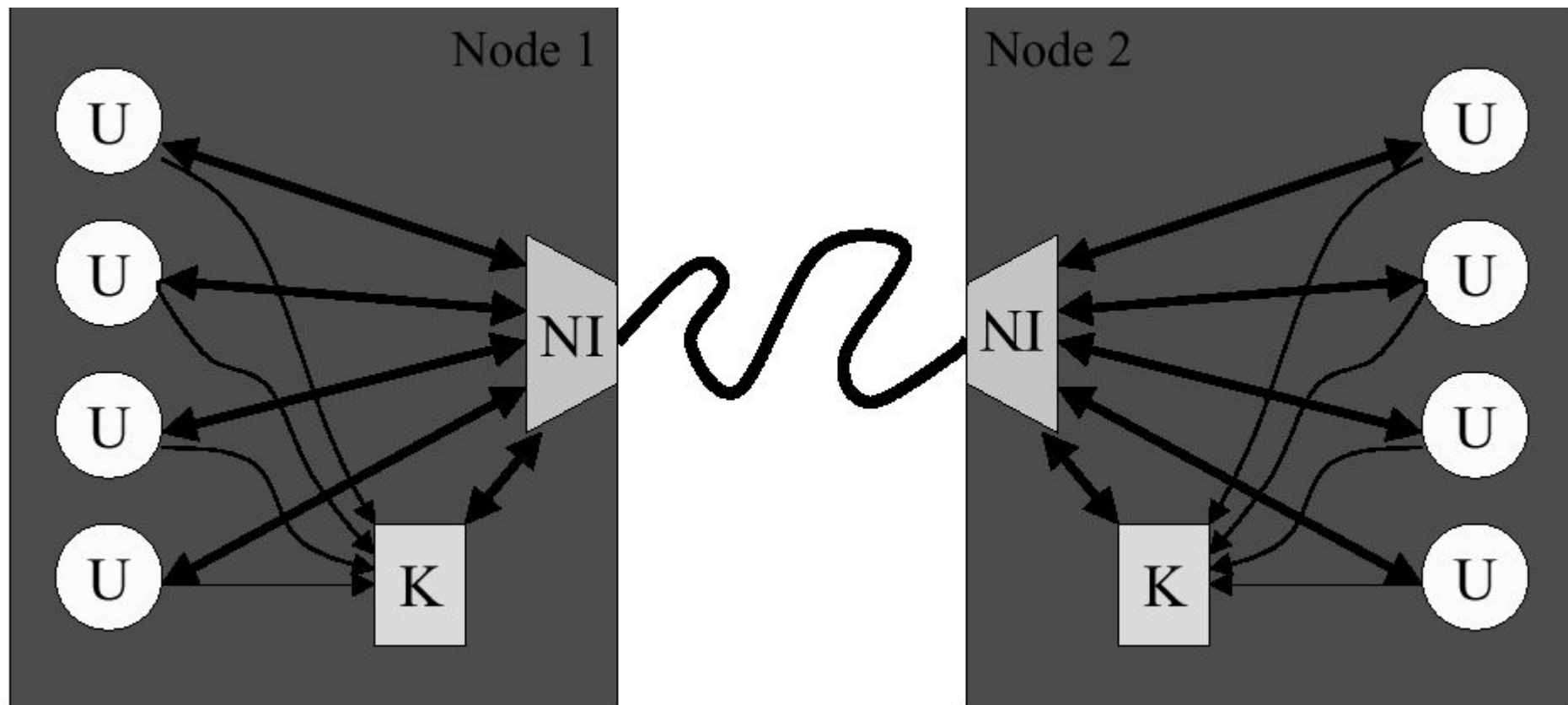
U-Net

- User-level network interface for parallel and distributed computing
- Design goal
 - Low latency, high bandwidth with small messages
 - Emphasis protocol design and integration flexibility
 - Portable to off-the-shelf communication hardware
- Role of U-Net
 - Multiplexing
 - Protection
 - Virtualization of NI

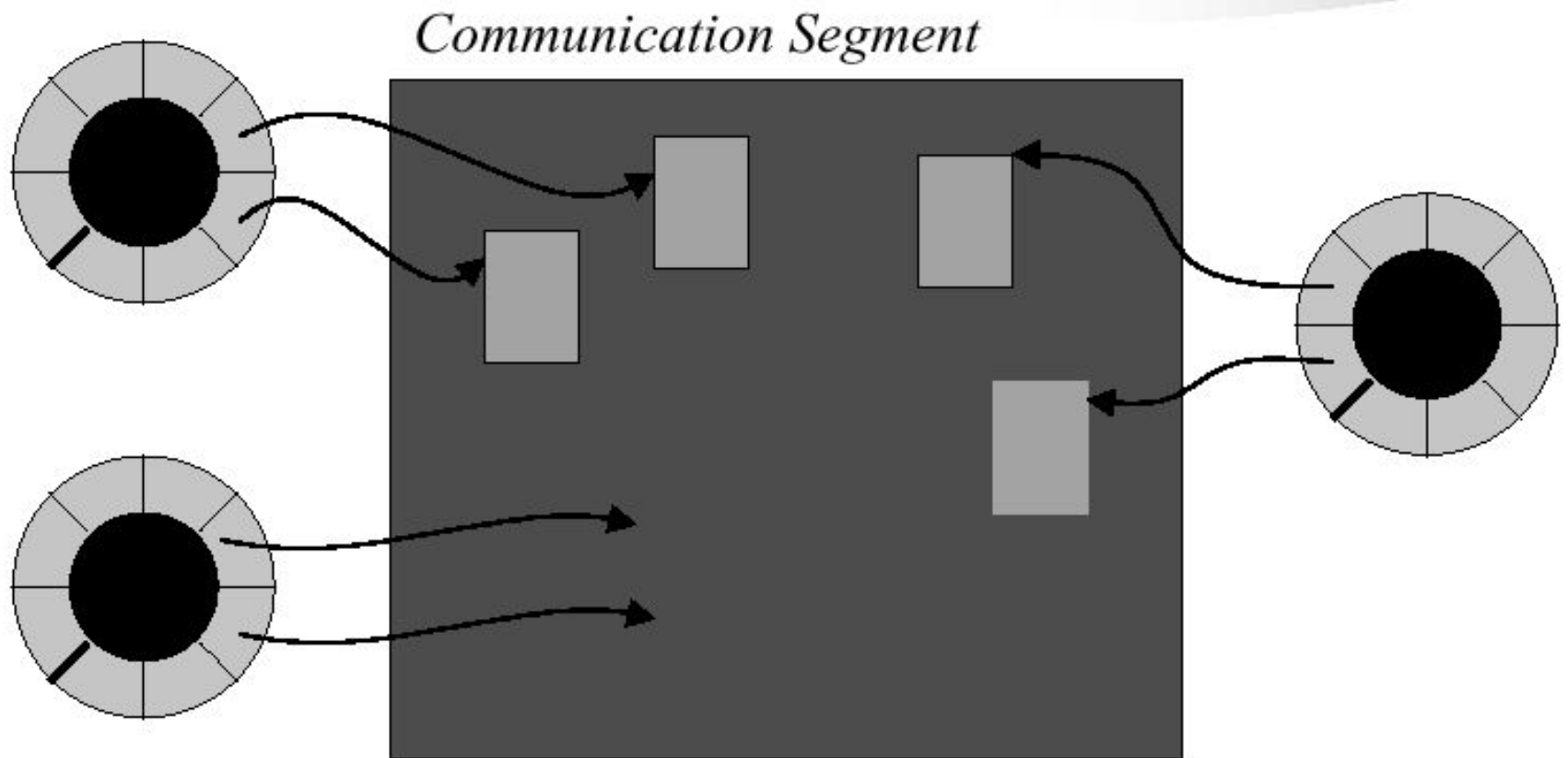
Traditional Communication Architecture



U-Net Architecture



Building Blocks of U-Net



U-Net Endpoint

User-Level Lightweight Communications (I)

- User-level approach
 - Derived from the assumption that OS communications are inefficient by definition
 - The OS involvement in the communication path is minimized
- Three solutions to guarantee protection
 - Leverage programmable NICs
 - support for device multiplexing
 - Granting network access to one single trusted user
 - not always acceptable
 - Network gang scheduling
 - exclusive access to the network interface

User-Level Lightweight Communications (II)

- Basic Interface for Parallelism (BIP)
 - Implemented atop a Myrinet network of Pentium PCs running Linux
 - Provide both blocking and unblocking communication primitives
 - Send-receive paradigm implemented according to rendezvous communication mode
 - Policies for performance
 - a simple detection feature without recovery
 - fragment any send message for pipelining
 - get rid of protected multiplexing of the NIC
 - the register of the Myrinet adapter and the memory regions are fully exposed to user-level access
 - Performance
 - 4.3 μ s latency, 126MB/s bandwidth
 - TCP/IP over BIP: 70 μ s latency, 35MB/s bandwidth
 - MPI over BIP: 12 μ s latency, 113.7MB/s bandwidth

User-Level Lightweight Communications (III)

■ Fast Messages

- Active Message-like system running on Myrinet-connected clusters
- Reliable in-order delivery with flow control and retransmission
- Works only in single-user mode
- Enhancement in FM 2.x
 - the programming interface for MPI
 - gather/scatter features
 - MPICH over FM: 6 μ s
- Performance
 - 12 μ s latency, 77 MB/s bandwidth (FM 1.x, Sun SPARC)
 - 11 μ s latency, 77 MB/s bandwidth (FM 2.x, Pentium)

User-Level Lightweight Communications (IV)

- Hewlett-Packard Active Messages (HPAM)
 - an implementation of Active Messages on a FDDI-connected network of HP 9000/735 workstations
 - provides
 - protected, direct access to the network by a single process in mutual exclusion
 - reliable delivery with flow control and retransmission.
 - Performance (FDDI)
 - 14.5 μ s latency, 12 MB/s asymptotic bandwidth

User-Level Lightweight Communications (V)

■ U-Net for ATM

- User processes are given direct protected access to the network device with no virtualization
- The programming interface of U-Net is very similar to the one of the NIC itself
- Endpoints
 - The interconnect is virtualized as a set of 'endpoints'
 - Endpoint buffers are used as portions of the NIC's send/receive FIFO queues
 - Endpoint remapping is to grant direct, memory-mapped, protected access
- Performance (155 Mbps ATM)
 - 44.5 μ s latency, 15 MB/s asymptotic bandwidth

User-Level Lightweight Communications (VI)

- Virtual Interface Architecture (VIA)
 - first attempt to standardize user-level communication architectures by Compaq, Intel, and Microsoft
 - specifies a communication architecture extending the basic U-Net interface with remote DMA (RDMA) services
 - characteristics
 - SAN with high bandwidth, low latency, low error rate, scalable, and highly available interconnects
 - error detection in communication layer
 - protected multiplexing in NIC among user processes
 - reliability is not mandatory
 - M-VIA
 - the first version of VIA implementation on Fast/Gigabit Ethernet
 - kernel-emulated VIA for Linux
 - Performance (100base-T)
 - 23 μ s latency, 11.9 MB/s asymptotic bandwidth

A Comparison Among Message Passing Systems

- Clusters vs. MPPs
- Standard Interface approach vs. other approach
- User level vs. kernel-level

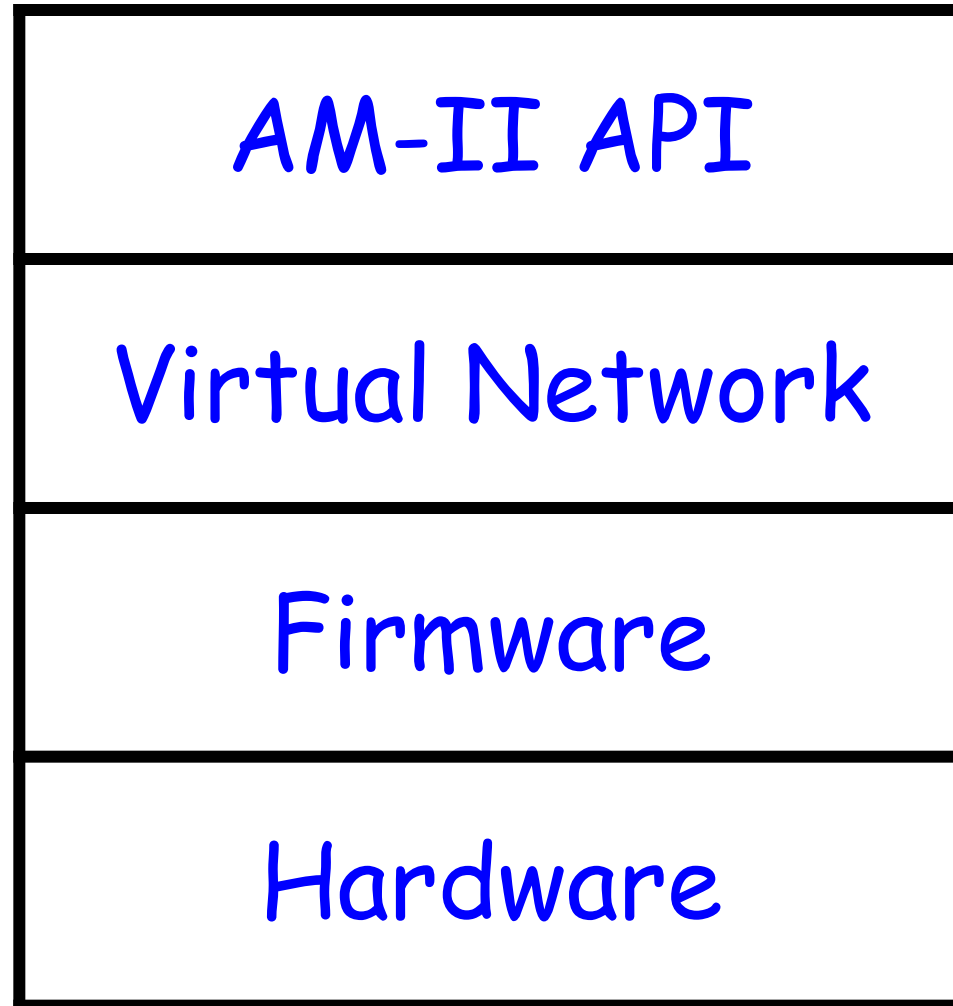
"Ping-Pong" Comparison of Message Passing Systems

Platform	Latency (μ s)	Max. throughput (MByte/s)
Linux 2.0.29 TCP/IP sockets, half-duplex (3COM 3c595 100base-T, Pentium 133 MHz CPU-driven DMA)	113.8	6.6
(3COM 3c905 100base-T, Pentium II 300 MHz DBDMA)	77.4	10.8
Fast Sockets (Myrinet,UltraSPARC)	57.8	32.9
PARMA ² sockets (3COM 3c595 100base-T, Pentium 133 MHz)	74.0	6.6
GAMMA [4] (3COM 3c905 100base-T, Pentium 133 MHz)	12.7	12.2
Net* (raw) (100base-T) [15]	23.3	12.2
Oxford BSP Clusters (100base-T) [5]	29.0	11.7
U-Net/FE (DEC DC21140 100base-T) [20]	30.0	12.1
M-VIA (DEC DC21140 100base-T) [1]	33.0	11.9
BIP (Myrinet,PPro) [13]	4.3	126.0
HPAM (FDDI) [12]	14.5	12.0
FM2.x (Myrinet,PPro) [10]	11.0	77.0
U-Net/ATM (FORE PCA-200 ATM) [20]	44.5	15.0
CM-5 CMAML ports	10 - 15	8.5
SP2 MPL [11, Table 2]	44.8	34.9
T3D PVMFAST [11, Table2]	30.0	25.1

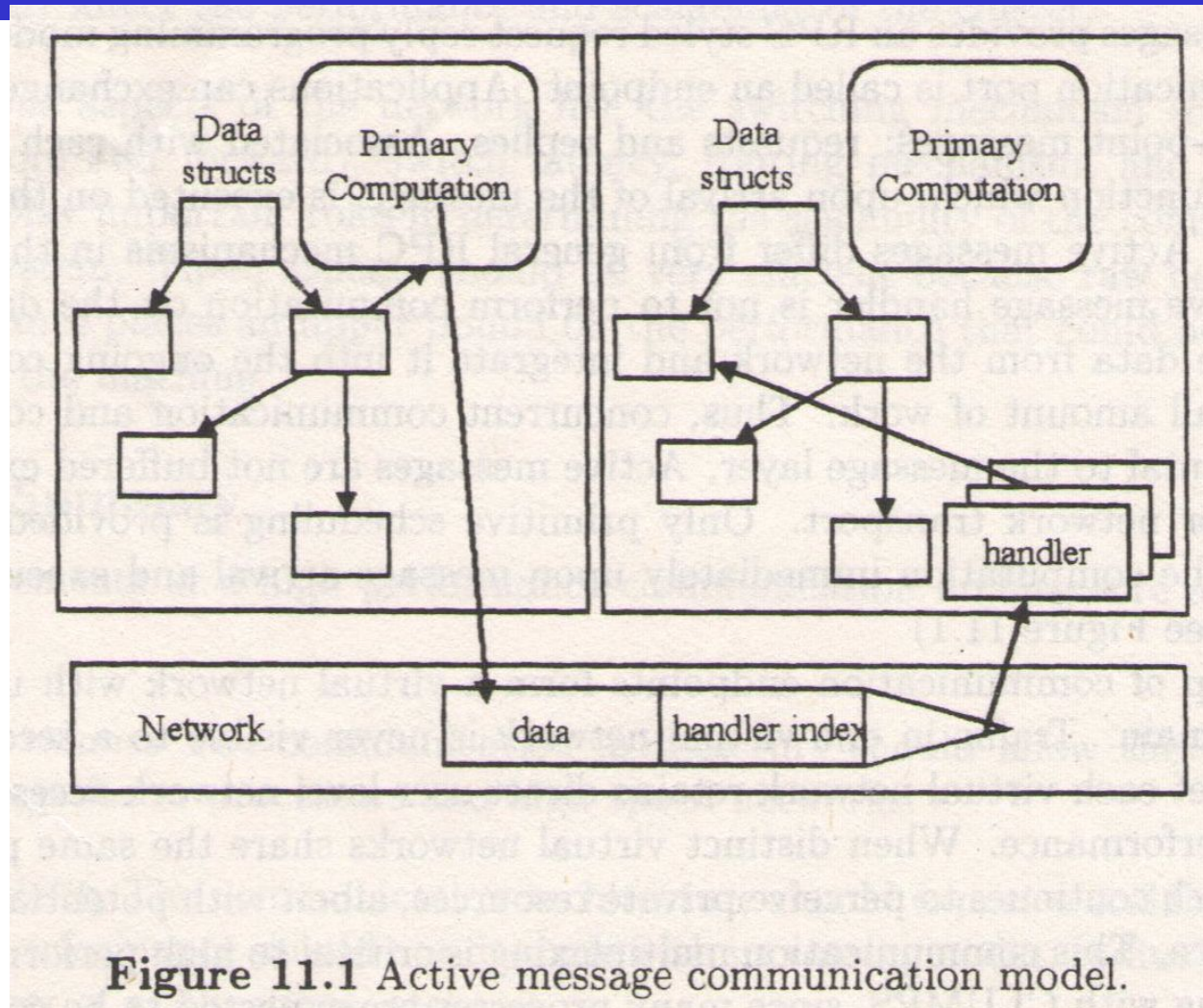
Chapter 11

ACTIVE MESSAGES

Active Message Architecture



Active Message Communication Model



Anatomy of endpoint

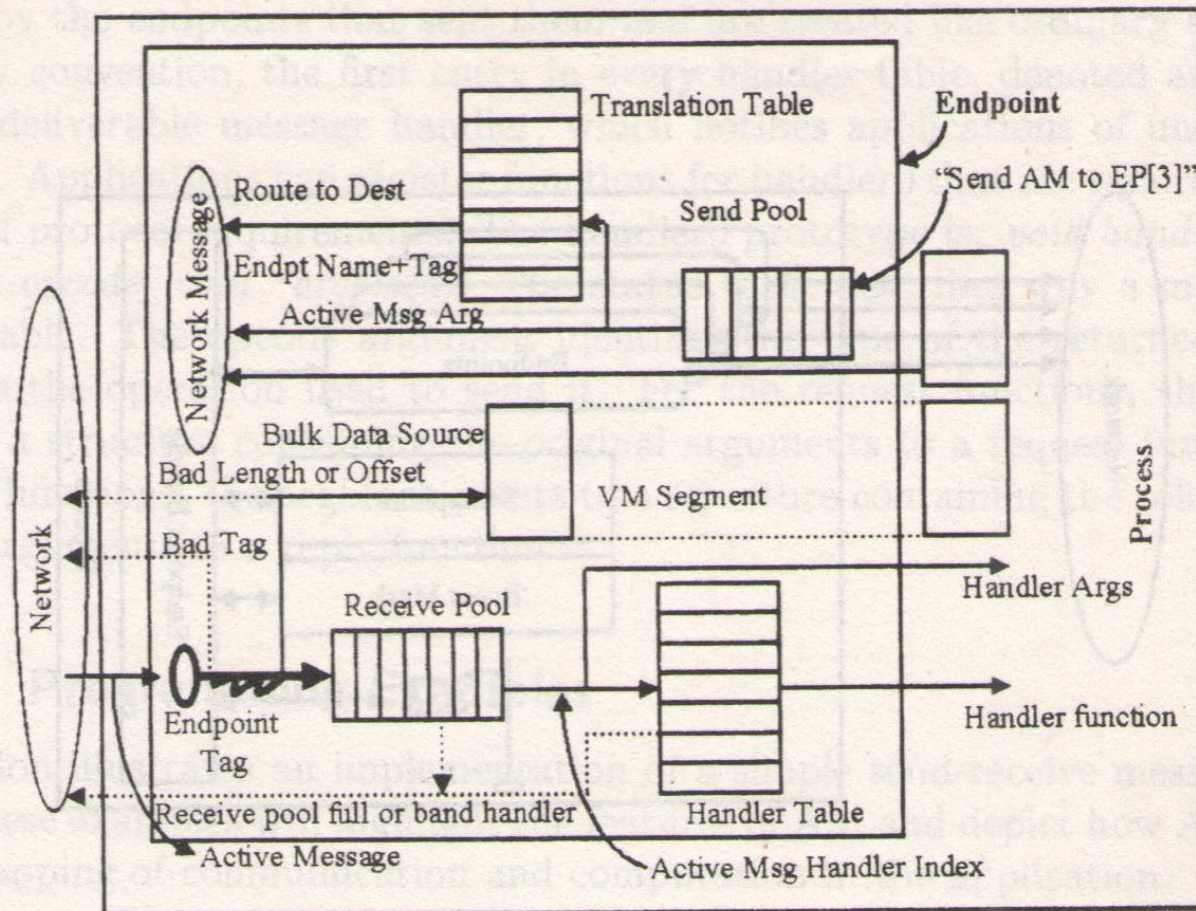


Figure 11.2 Anatomy of an endpoint. Reproduced from [8] with permission.

Anatomy of endpoint bundle

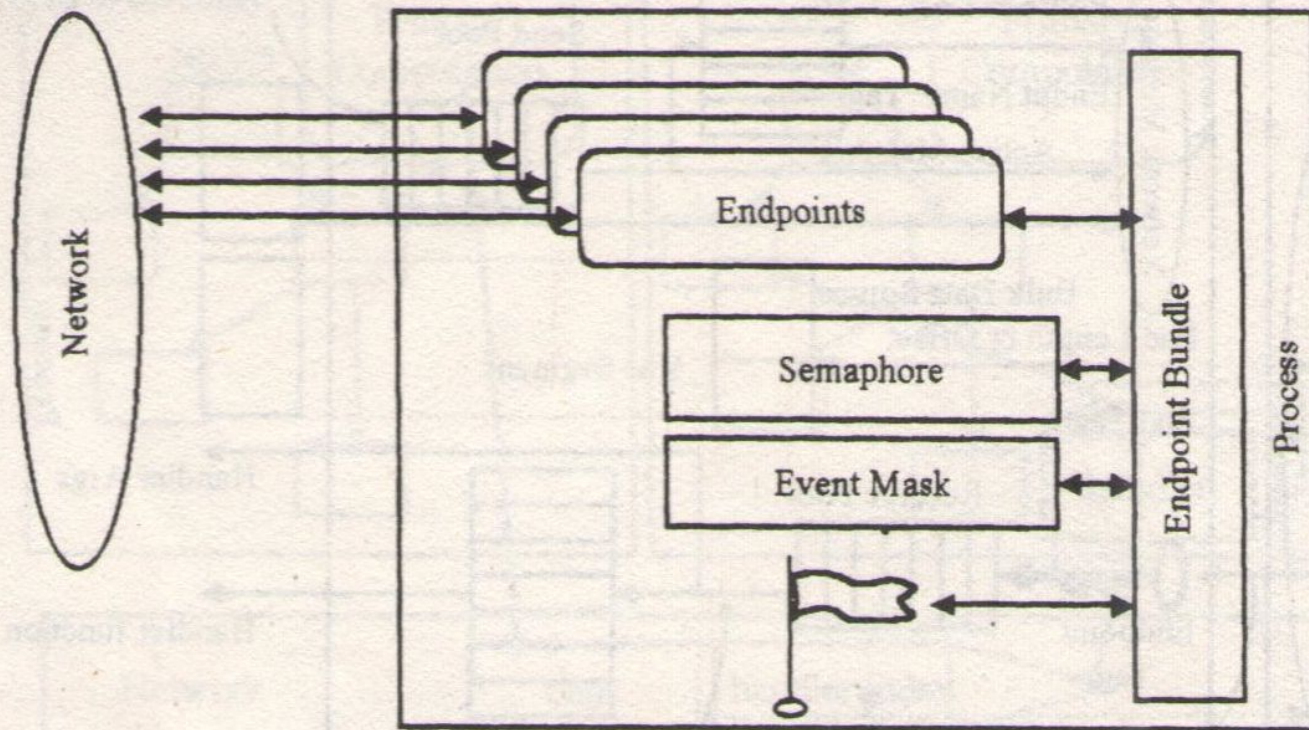


Figure 11.3 Anatomy of an endpoint bundle. *Reproduced from [8] with permission.*

Programming example : send-receive for small message

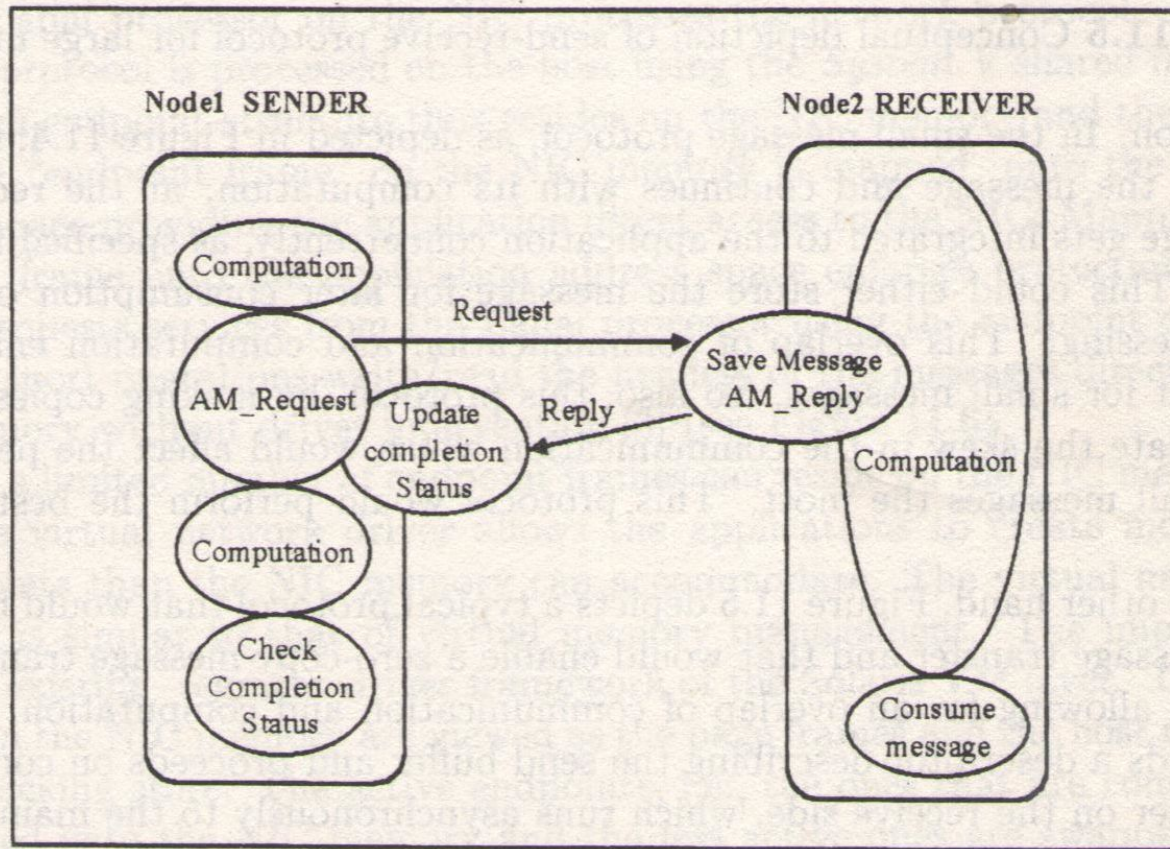


Figure 11.4 Conceptual depiction of send-receive protocol for small messages.

Programming example : send-receive for large message

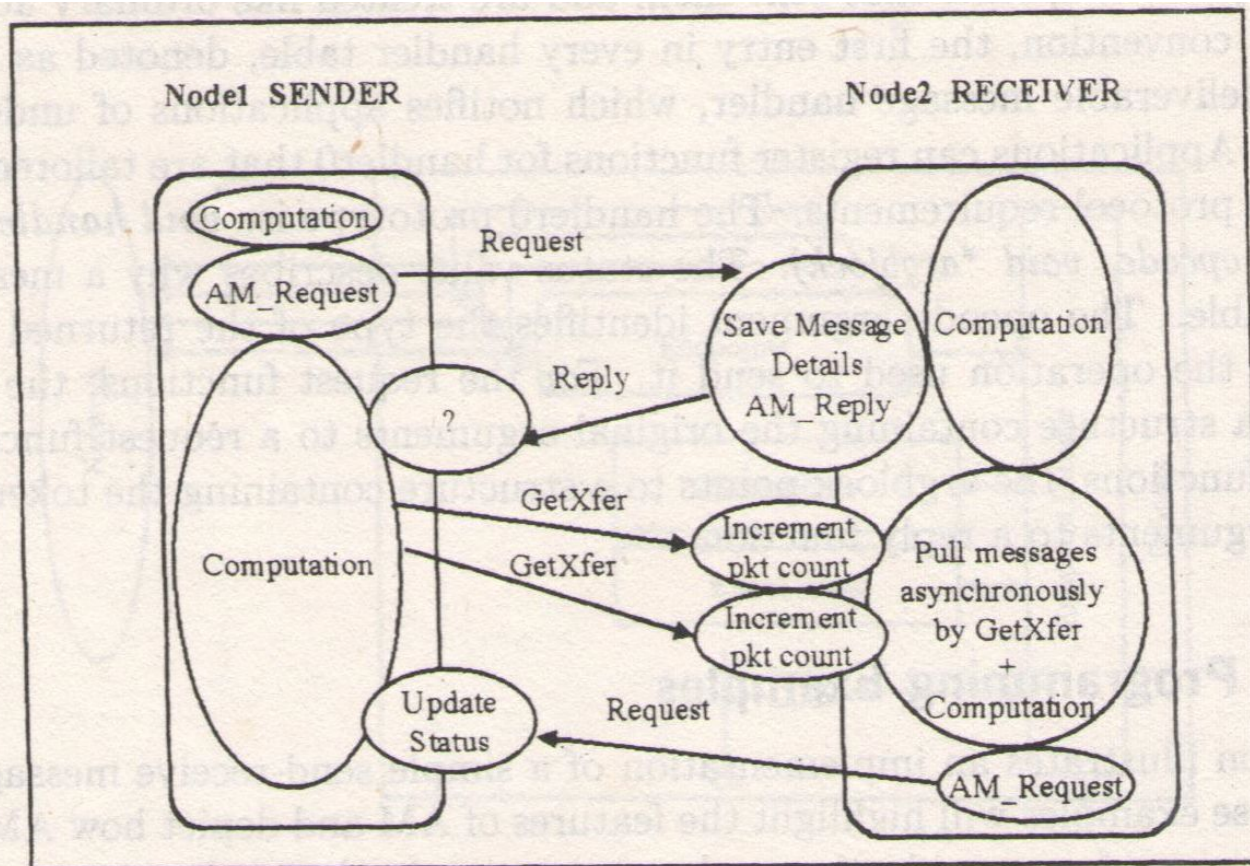


Figure 11.5 Conceptual depiction of send-receive protocol for large messages.

AM Implementation - NIC interaction

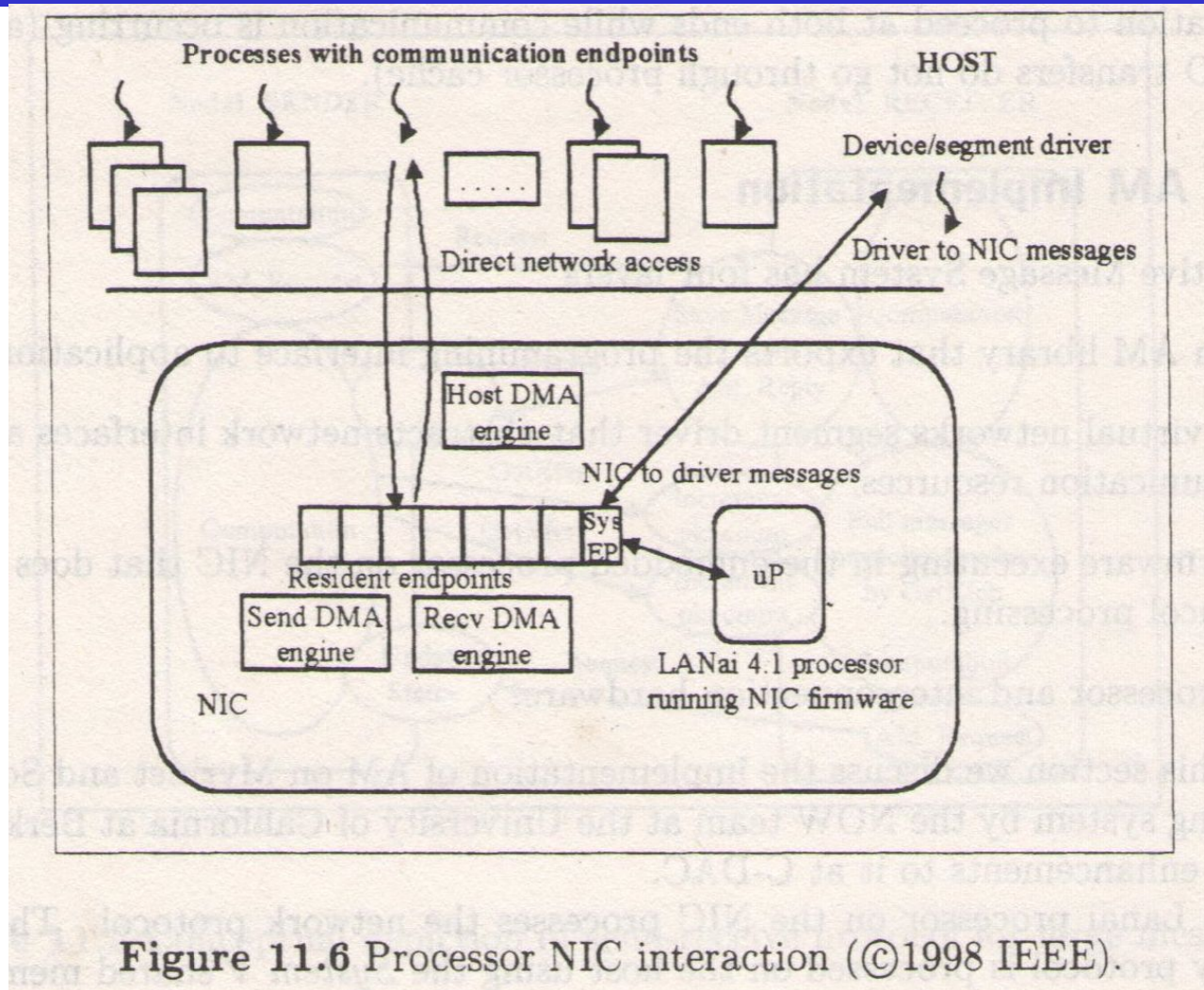


Figure 11.6 Processor NIC interaction (©1998 IEEE).

AM Implementation - endpoint and bundles

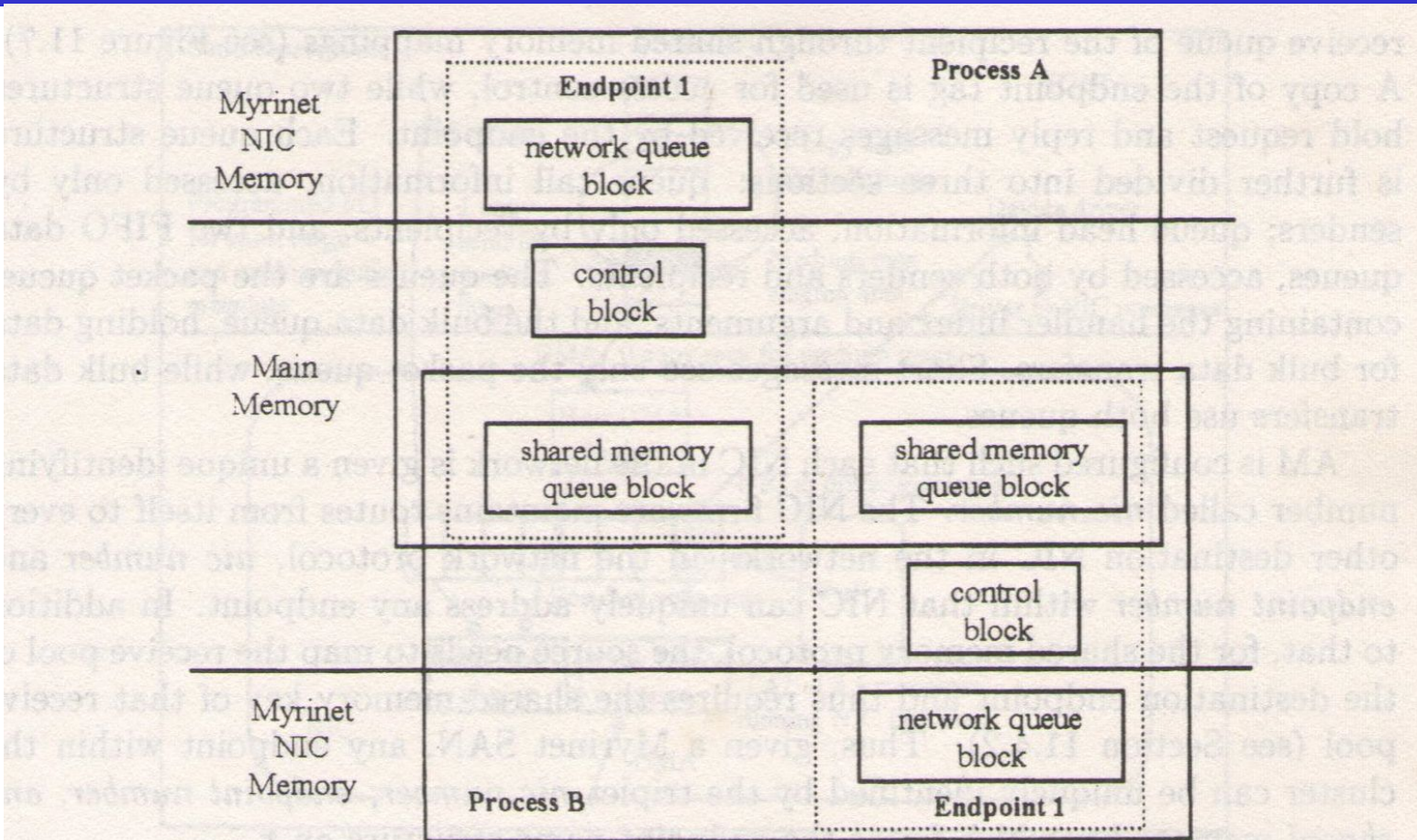
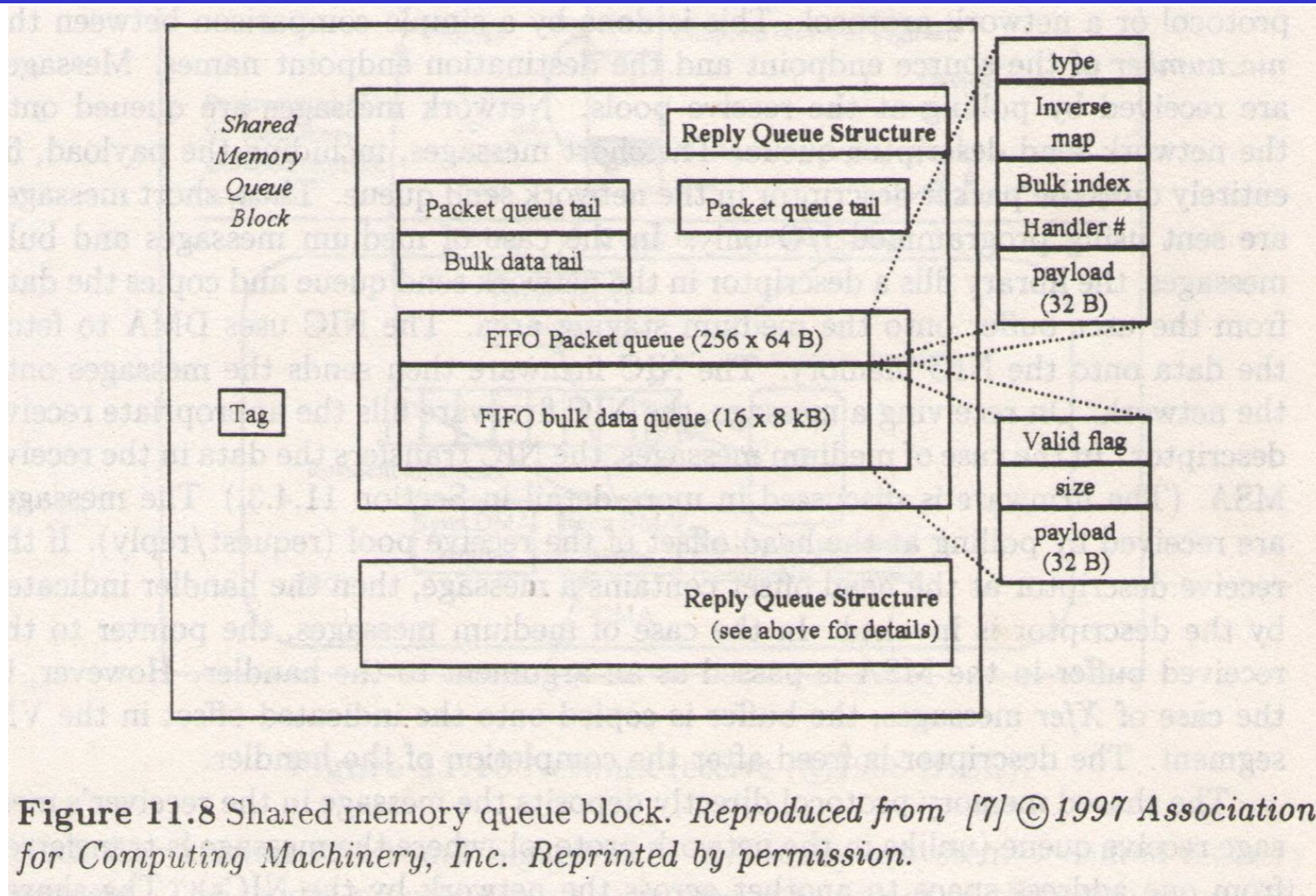
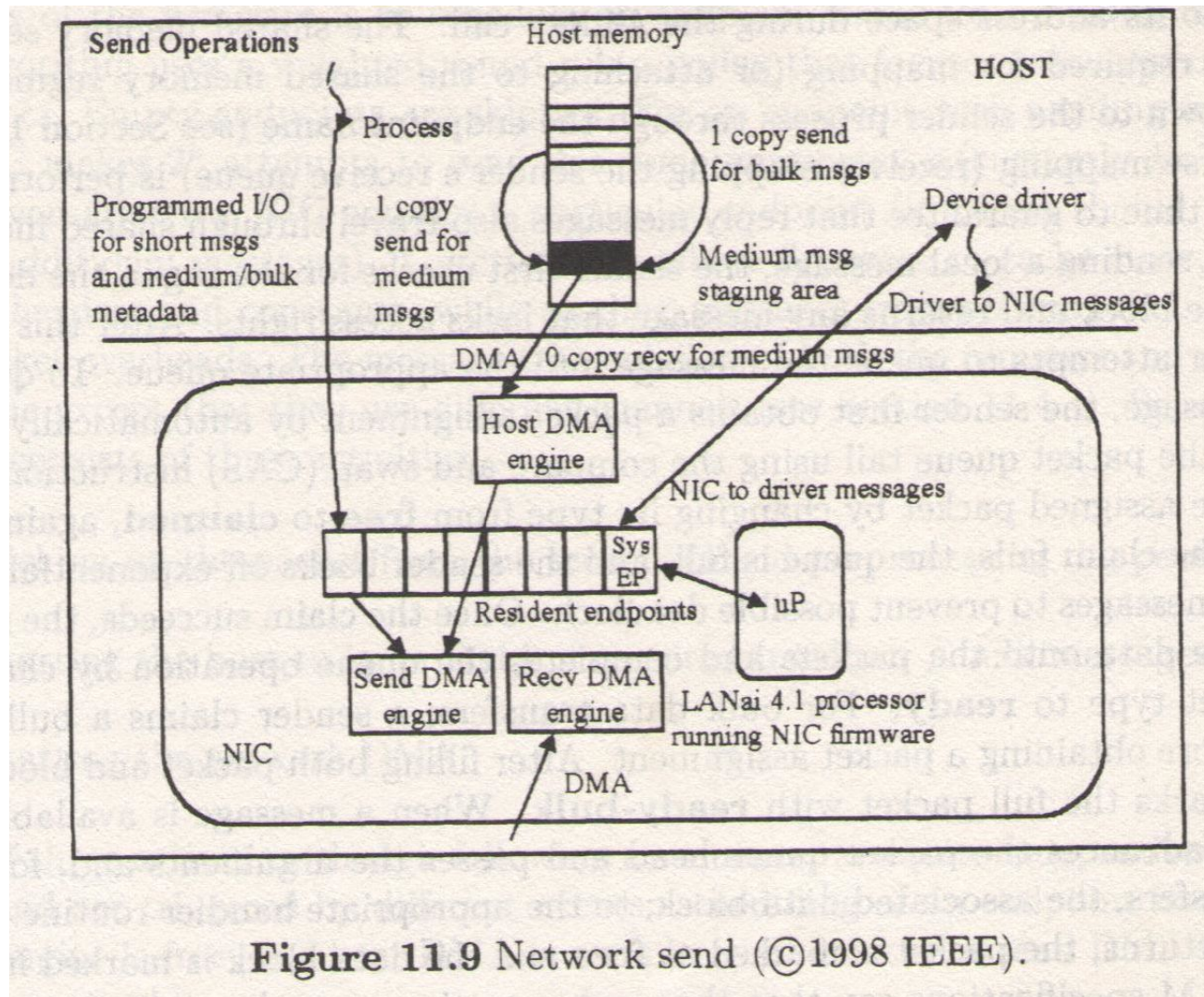


Figure 11.7 Endpoint data layout. Reproduced from [7] ©1997 Association for Computing Machinery, Inc. Reprinted by permission.

AM Implementation - Shared memory



Transport Operations - network send



Transport operations - network receive

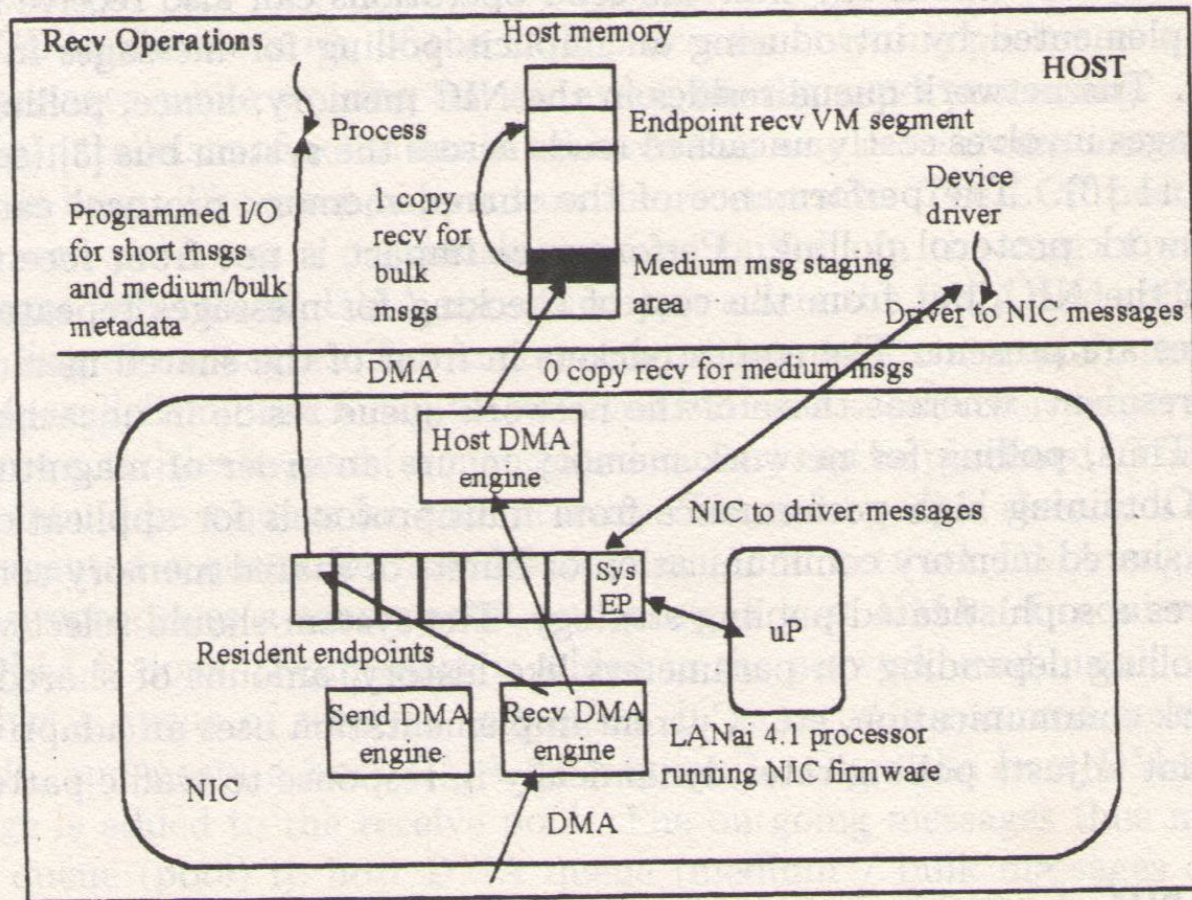
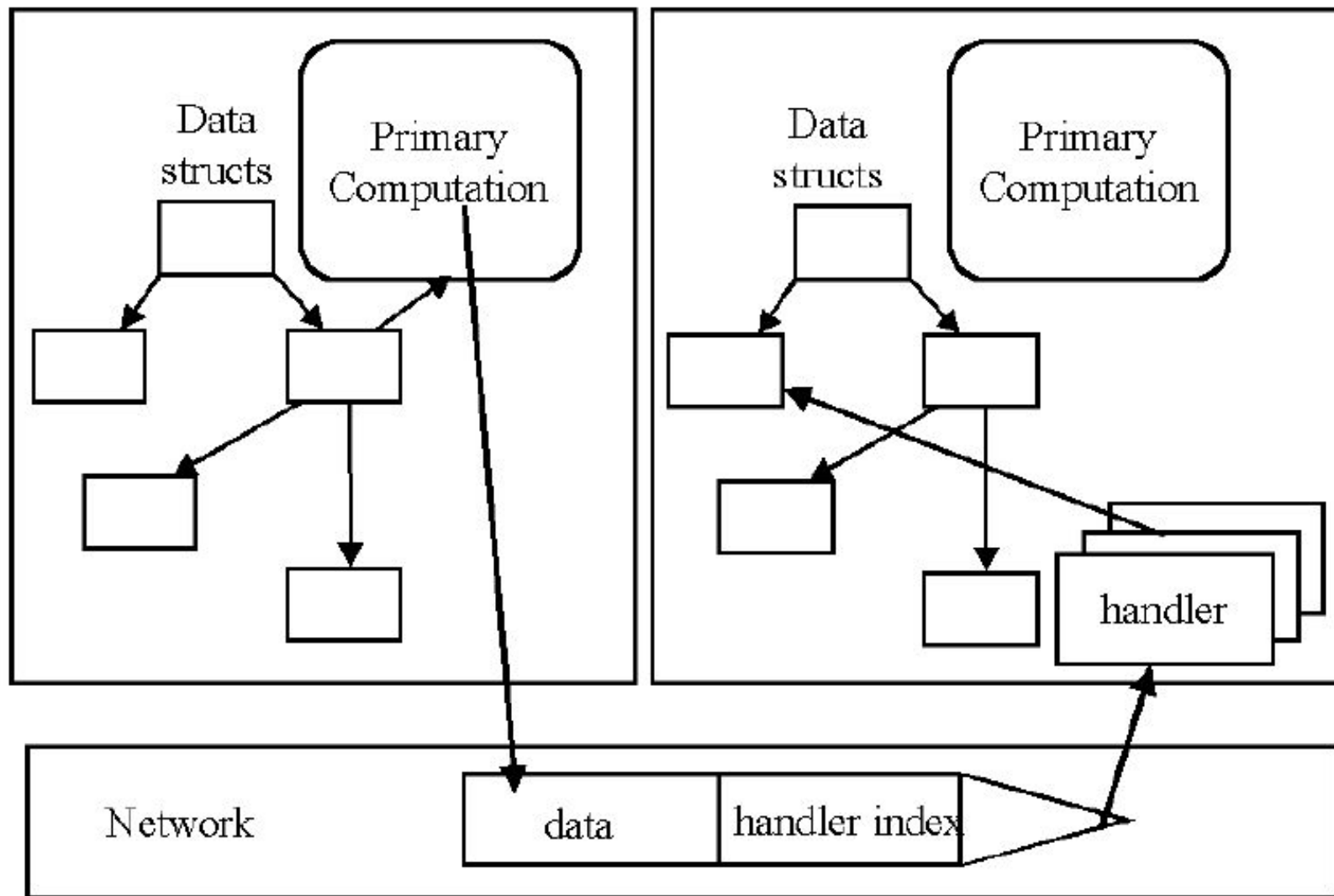
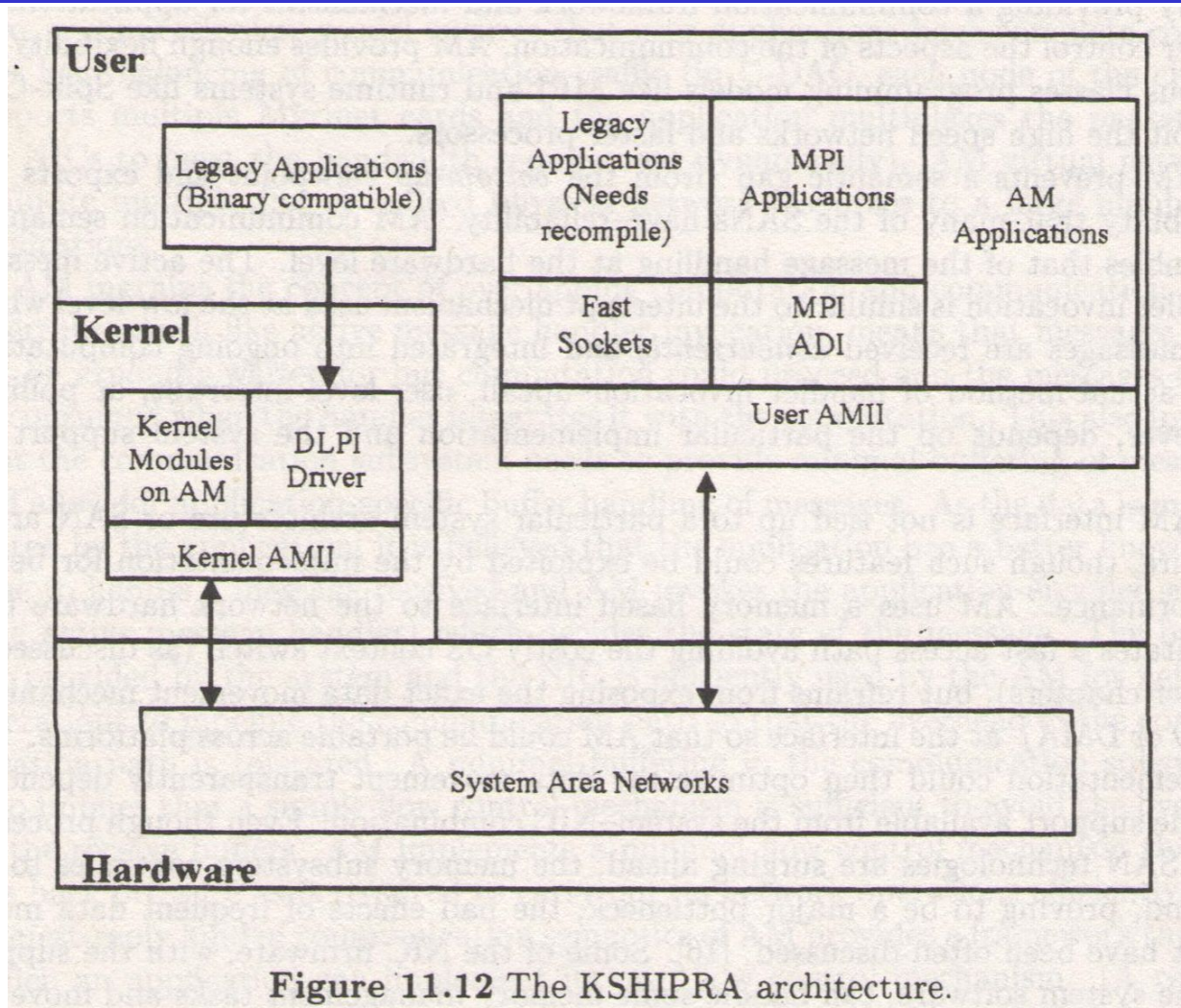


Figure 11.10 Network receive (©1998 IEEE).

Active Message Communication Model



KSHIPRA Architecture



Active Message (II)

■ AM-II API

- Support three types of message
 - short, medium, bulk
- Return-to-sender error model
 - all the undeliverable messages are returned to their sender
 - applications can register per-endpoint error handlers

Active Message (III)

■ Virtual networks

- An abstract view of collections of endpoints as virtualized interconnects
- Direct network access
 - an endpoint is mapped into a process's address space
- Protection
 - use standard virtual memory mechanism
- On-demand binding of endpoint to physical resources
 - NI memory: endpoint cache – active endpoint
 - host memory: less active endpoint
 - endpoint fault handler

Active Message (IV)

■ Firmware

- Endpoint scheduling
 - weighted round-robin policy
 - skip empty queues
 - 2^k attempts to send, where $k=8$
- Flow control
 - fill the communication pipe between sender and receiver
 - prevent receiver buffer overrun
 - three level flow control
 - user-level credit base flow control
 - NIC level stop-and-wait flow control
 - link-level back pressure
 - channel management tables
 - channel-based flow control
 - timer management
 - timeout and retransmission
 - error handling
 - detect duplicated or dropped messages
 - sequence number & timestamp
 - detect unreachable endpoints
 - timeout & retransmission
 - detect & correction other errors
 - user-level error handler

Active Message (V)

■ Performance

- 100 Ultra SPARC stations & 40 Myrinet switches
- 42 μ s round trip time
- 31 MB/s bandwidth