End-to-end Communication in High Speed Networks

Advanced Networks Group
Computer and Communications Research Center
Washington University, St. Louis

1992

The Advanced Networks Group

The Advanced Networks Group (ANG) is concerned with new communication technologies that can support a wide range of different communication applications in the context of both public and private networks. Fast packet or ATM networks promise a far more flexible communications infrastructure than is currently available. ANG is particularly concerned with systems that are capable of supporting ubiquitous multicast communication, suitable for applications such as video distribution, voice/video teleconferencing and LAN interconnection. We are developing an experimental switching system supporting links operating at 100 Mb/s and have devised economical switch architectures that can support link speeds in excess of a gigabit per second and having total throughput exceeding a terabit per second.

Our work spans a variety of topics including switching system design and analysis, performance evaluation of switching systems and networks, multicast connection management, algorithms for multicast routing, buffer and bandwidth management in the presence of bursty traffic, internetworking of high speed networks, high speed transport protocol design, host-network interfacing, image and video compression, distributed visualization, and design of specialized computer-aided design tools. Our research program includes a strong experimental component, which currently centers on the development of a prototype fast packet switching system and implementation of protocols and applications. We have developed five circuits to be used in this prototype system and plan to assemble a network of four switches to demonstrate applications of fast packet switching. The experimental work is a crucial element of the overall research program, exposing detailed issues not apparent in higher level studies and providing a strong focus for the other activities.

Faculty

Jonathan Turner
Andreas Bovopoulos
Guru Parulkar

Doctoral Students

Akira Arutaki
Millind Buddhikot
Andy Fingerhut
Fengmin Gong
Saied Hosseini
Seyed Mahdavian
Nader Mirfakhraei
Gopal Raman
James Sterbenz
Einir Valdimarsson
Ellen Witte

Masters Students

Jim Anderson
Charles Cranor
Apostolos Dailianas
Zubin Dittia
Rex Hill
Sanjay Kapoor
Christos Papadopoulos

Related Activities

ANG is part of the Computer and Communications Research Center (CCRC), an interdepartmental research center that spans the departments of computer science and electrical engineering. In addition to the work in networking and communication work CCRC carries out research activities in parallel computer architecture, parallel algorithms, design automation and performance analysis and modeling. Recent activities have centered on the application of parallel computers to logic simulation and the associated problem of task allocation. This has included development of parallel simulated annealing algorithms and their application to task allocation for logic simulation.

The Center is headed by Dr. Mark Franklin and includes Dr. Roger Chamberlain, Dr. Ken Wong and several graduate students in addition to those in the Advanced Networks Group. The Center's facilities include three Sun file servers and over twenty workstations, a variety of software for VLSI circuit design, software development and performance modeling. In addition, a 64 processor NCUBE multiprocessor is available for work on parallel algorithms and a well-equipped electronics laboratory provides facilities for assembly and testing of systems. Other facilities available to the Center include a Pixar image computer and a video production facility.

The Computer Science Department's Applied Research Laboratory is a closely allied organization that carries out projects with the objective of transferring faculty research results into industrial practice. ARL currently has a full-time engineering staff of ten, including three long term visitors, and is engaged in a project whose objective is to demonstrate the potential of the broadcast packet switching technology, developed by the Advanced Networks Group. The project involves construction of a network of four switches at different locations in the St. Louis area, and demonstration of their use in support of video and medical imaging applications. The project also has a research component centering on algorithms and systems for video and image compression, and a second research component centering on image workstations for medical applications. Faculty and graduate students in Computer Science, Electrical Engineering and the Mallinkrodt Institute of Radiology are involved in various aspect of these activities. The project is supported by Southwestern Bell Corporation and NEC America.

ARL Staff

Jerome Cox, Jr., – Director

Hardware Design Group
- Pierre Costa – Group Leader
- Neil Barrett
- Tom Chaney
- Noritaka Matsuura (NEC)
- Randy Richards
- Kenichi Sato (NEC)

System Software Group
- Mike Gaddis – Group Leader
- Rick Bubenik
- John DeHart
- Ken Katsumata (NEC)

Companies currently supporting the work of the Advanced Networks Group include:

Bell Communications Research
Bell Northern Research
Digital Equipment Corporation
Italtel SIT
NEC America
Nippon Telegraph and Telephone
SynOptics Communications

Program Overview

The research program of the Advanced Networks Group includes a blend of experimental, analytical and theoretical work with a strong systems focus. We are committed to the proposition that successful engineering research requires an intimate knowledge of the practical issues involved in building complex systems, as well as strong analytical capabilities. We find that theoretical research, in the absence of a strong connection to practical concerns, too easily drifts into activities of interest only to an ingrown research community, and is ultimately sterile and unprofitable. On the other hand, experimental work that is uninformed by a deeper understanding of fundamental issues, can have only limited and short-term value. The program we have constructed, is based on this commitment to both engineering science and practice, and is unusual among university-based research programs in this regard.

Over the past several years, we have engaged in research activities focussing on packet switching systems operating at link speeds of about 100 Mb/s. This work has included the design and implementation of integrated circuit components and their use in experimental switching systems; performance evaluation of switching systems from a variety of different points of view; design of connection management protocols for multicast networks; design of multicast routing protocols; high speed internetworking and host-network interfaces; buffer and bandwidth management; distributed debugging systems; special purpose computer-aided design tools; and video coding algorithms. These activities are described briefly in the short articles that follow and more fully in the references. A prime distinguishing feature of our research activities has been our focus on the problem of multicast communication. The experimental switching system we are constructing will be one of the first high speed packet switching systems in the world that is capable of supporting large amounts of multicast communication and our research directed to understanding how to efficiently operate such systems is unique.

Our research agenda is now directed increasingly toward the design and analysis of switch architectures capable of supporting gigabit transmission rates. We are designing architectures capable of supporting such speeds using wide internal data paths and moderate speed circuit technology. We are also interested in switching systems suitable for high speed datagram switching, and in particular, the integration of datagram switching in a system supporting point-to-point and multicast connections. We also have a continuing interest in the network control and optimization problems associated with emerging high speed networks. In particular, we continue to work toward better methods for bandwidth allocation and management in fast packet networks, as well as more efficient distributed algorithms for multicast connection routing.

Another key element of our evolving research agenda is end-to-end communications in high speed networks which involves internetworking and the application of high speed networks to computer-based applications, particularly those involving high resolution images and multimedia workstations. Topics of research include the design and implementation of high speed internetworking, gateway architectures, application-oriented lightweight transport protocols, and host–network interfacing within the constraints of host operating system. We have recently started an effort to study a set of distributed visualization applications. Our aim is to understand the real communication requirements of these applications and appropriately tailor the protocols to provide efficient support.

Within the university we have recently initiated an effort called Project Zeus, which will apply the emerging ATM technology within a campus environment. This work will build on our earlier efforts, but will move toward the design and construction of operational networks, rather than research prototypes. The project will be carried out by the Applied Research Lab and the Office of the Network Coordinator (which operates the existing campus network) in cooperation with a number of industrial partners and collaborators throughout the university. We anticipate that Zeus will provide a rich source of research problems for ANG faculty and graduate students and a stimulating context within which to direct our

activities.

The program provides ample opportunities for discussion and collaboration with IPP members. In addition to providing members with timely access to technical reports and publications, our regular progress review meetings are designed to stimulate interaction between members and ANG's faculty and students. Through these discussions, members have the opportunity to point out new research directions and help guide the research activities in order to ensure its practical relevance. Interactions with students and staff can also be promoted through extended visits by ANG staff to a member's location or by extended visits by member personnel to Washington University. Such interactions have been a crucial part of our research program with several of our visitors being key collaborators who played a strong role in our program, while providing their companies with deeper insight and understanding of ANG's research activities.

High Speed Internetworking

High Speed Internetworking

Guru Parulkar, James Anderson, Milind Buddhikot, Chuck Cranor, Zubin Dittia, Sanjay Kapoor, Tony Mazraani

We have proposed a very high speed internet abstraction (called VHSI) which can efficiently support guaranteed levels of performance for a variety of applications, and can cope with the diversity of underlying networks [12, 15, 16]. Important components of this abstraction are shown in Figure 1. We continue to make good progress on the research and development of various components of the VHSI abstraction. Progress is summarized for each component of the VHSI abstraction in the following paragraphs:

Multipoint Congram-oriented High performance Internet Protocol (MCHIP). MCHIP is a novel multipoint congram-oriented high performance internet protocol, equal in status to IP in terms of the protocol hierarchy. The congram service primitive aims at combining the strengths of both classical connection and datagram approaches.

MCHIP includes two types of congrams: user congram (UCon) and persistent internet congram (PICon). UCons provide support for the connection-oriented applications, and PICons for datagram applications. Specification of PICon management was completed during this past year. PICon management involves specification of PICon setup, usage, and termination. We specify PICon management using specification of appropriate data and control packet formats, exchange of control messages, creation and management of various data structures to store state information, and interaction with resource managers to allocate resources for PICons. Details of PICon and UCon management can be found in [2, 3, 12, 13].

We have also made good progress with MCHIP implementation in the kernel of SunOS Unix 4.0 which is summarized in the next section.

Resource Server. The VHSI abstraction provides performance guarantees to applications by preallocating resources to congrams, based on the application needs. However, a number of networks do not do resource management on a per connection basis, and therefore the VHSI abstraction includes resource servers to provide this functionality.

We have completed a simulation study of a resource server for Ethernet with a variety of traffic sources, including Poisson and bursty [13]. This model uses a central resource manager in a directly connected gateway. The role of the resource manager is to keep track of all active congrams and their resource usage, and accept or block new congrams depending on resource availability. The resource manager is always consulted before a congram can be established.

The goal of the resource manager is to guarantee performance to established congrams, and to manage the network resources efficiently. Efficient resource management in this environment means maximum utilization of the network and minimum congram blocking, packet loss, and packet delay. Note, however, that minimizing packet delay may result in lower channel utilization. This study however shows that a simple resource management model can be devised to provide bounded packet loss and delay to various applications with reasonable channel utilization and blocking.

We have also made considerable progress on the simulation study of PICon multiplexing. Note that the PICons are provided to multiplex datagram traffic from a number of sources. A datagram source is considered to be a bursty source and its traffic output is characterized by three parameters: burst size, delay between bursts, and a time after which a burst is to be discarded. The simulation study is aimed at understanding how many and what kind of datagram sources can be successfully multiplexd on a given PICon. A number of scheduling schemes have been studied, and a deadline based scheduling algorithm is found to be provide the best results under the given conditions. For details of this study, refer to the MS thesis [2].

Gateway Architecture. The VHSI abstraction requires that the gateway architectures be able to

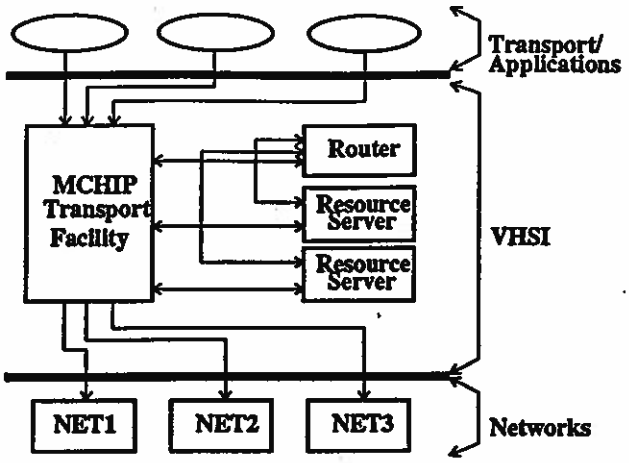


Figure 1: VHSI Abstraction

support data rates of at least a few hundred Mbps, to interface with diverse networks, and to implement MCHIP without becoming a performance bottleneck. An important part of the gateway design philosophy is to partition the functionality into critical and non-critical paths. The critical path consists of per packet processing, and should be implemented in hardware for speed and performance. The non-critical path consists of congram management and resource and route management and is best implemented in software due to complexity, need to do fine tuning with time, and flexibility. It is not an efficient approach to mix these paths as is generally done in current gateways.

To research high speed gateway architectures, a paper design and a simulation study of a two port ATM-FDDI gateway were undertaken [11, 10]. We believe that ATM and FDDI are excellent target networks to explore the VHSI gateway architecture because they pose the necessary challenges to the gateway designer due to their high data rates and significant diversity.

Design of the ATM-FDDI gateway requires appropriate separation of control and data paths, and design of protocol processors to implement the Segmentation and Reassembly (SAR) Protocol and MCHIP critical path in hardware. The SAR protocol processor implements the SAR protocol that is used in the gateway to reassemble incoming cells from the ATM network into FDDI frames, and also fragment FDDI frames into ATM cells. The MCHIP protocol processor implements some of the MCHIP protocol primitives. It contains channel translation tables and logic to append FDDI specific headers on reassembled frames, and ATM headers on incoming FDDI frames before they are forwarded for fragmentation. Protocol processor designs are sufficient in detail to allow a prototype implementation. A simulation study of the gateway has also been completed which has served two purposes: first to do functional verification of the design, and second, to characterize the performance of the gateway for different traffic patterns.

MCHIP Implementation

Charles D. Cranor, Guru Parulkar

Our MCHIP implementation essentially consists of two parts: COIP-K and MCHIP modules. COIP-K is the connection oriented internet protocol kernel which includes the common functionality necessary for most connection-oriented internet protocols. The MCHIP modules are protocol specific modules that use COIP-K as a toolkit to create an instantiation of MCHIP. We decided to partition the MCHIP implementation as described above because of the following reasoning.

A number of research groups, including ours, have proposed connection-oriented internet protocols (COIPs), which include MCHIP[12], ST[6, 23], and FLOW[24, 25]. The Internet Activities Board (IAB) and Internet Engineering Task Force (IETF) also recognized the need for exploring connection-oriented internetworking and initiated a COIP working group.

The proposed COIPs have a number of similarities and differences. The members of the COIP working group decided that it is important to pursue these protocols and compare and contrast the alternate approaches of these protocols. However, implementation of these protocols completely independently was considered unwise for the following reasons. First, as the proposed COIP protocols have a number of common functions, independent implementations would lead to a lot of duplicate work. For example, all the protocols have a connection state machine and a resource allocation and enforcement function. Secondly, implementation of a protocol in the Unix kernel poses a number of challenges: coding or logic errors can result in system crashes, kernel debugging support is limited, kernel dynamic memory allocation mechanisms are complex, the protocol's code must co-exist with the rest of the kernel, and the existing kernel interface is not well documented.

In order to develop a more productive research environment, avoid duplication of work, and foster collaboration, we proposed the COIP-kernel (COIP-K) to the COIP working group. COIP-K forms the core of a COIP protocol and includes the minimum functionality necessary for a wide range of multicast connection-oriented protocols. It also includes appropriate provisions to interface other functional modules. COIP-K, when combined with a set of functional modules, will create an instance of a COIP such as MCHIP or ST. This process is shown in Figure 2.

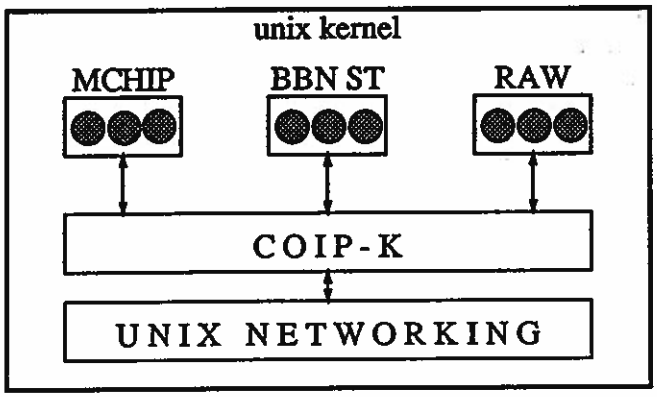


Figure 2: COIP-K structure

This approach to protocol development yields important benefits to the research effort. Many of the functions that a COIP must provide can be supported by a number of alternative mechanisms. These mechanisms can be implemented in different experimental modules and integrated with COIP-K to produce different instantiations of COIPs. These instantiations represent different mechanisms which

```
s = socket(PF_COIP, SOCK_RAW, 0)

err = setsockopt(s, level, CIN_SETPREQ, &preq,
sizeof(preq))


err = connect(s, addr, addrlen)


err = read(s, buf, buflen)
err = write(s, buf, buflen)


close(s)
```

Figure 3: COIP-K client

can be compared under controlled experimental conditions. As a result, it will be possible to describe under what conditions each of the alternate mechanisms behave well or poorly, and thus define a COIP that is optimal for a given target environment.

COIP-K includes routing and resource functions to set up a connection, functions to forward data packets based on connection identifiers, and functions to terminate the connection. Thus, COIP-K can run the basic state machine necessary for a connection-oriented protocol, and its implementation in the Unix kernel can provide the standard interface to its higher-level protocols. However, it is important to note that COIP-K leaves a number of options open and delegates important decision making to other functional modules. For example, COIP-K talks to the resource manager for resource availability and allocation, but the actual resource allocation algorithm is part of the resource allocation module.

A MS thesis [4] has explored the challenge of realizing the COIP-K vision and demonstrated its feasibility and viability. This section attempts to summarize this effort.

The BSD Unix networking model has three layers: the socket layer, the protocol layer, and the network interface layer. The protocol layer consists of a few communication domains with a number of protocols under each domain. For example, INET is a domain for the Internet suite of protocols, and protocols such as TCP, UDP, IP are part of this domain. For the purpose of connection-oriented protocols, we have created a new domain called COIP, and COIP-K is a protocol (actually a subset of a protocol) under this domain.

Application Interface. The socket layer is the layer the applications programmers use to interface to various protocol suites. Since COIP-K is built into the normal BSD UNIX networking software system, the application programmer interface is very similar to any other protocol that runs under BSD Unix (e.g. TCP/IP). A typical COIP client and a typical COIP server are shown in figures 3 and 4 respectively. Note that the socket interface has not changed except that the `setsockopt` is used to specify the performance requirements of the connection and `sockaddr_cin` structure has been modified to allow multiple addresses for a multipoint connection.

COIP-K In Protocol Layer. COIP-K has been designed to work well with the BSD Unix networking model. COIP-K lives in the protocol layer of the SUNOS/BSD kernel and has its own communications domain. In the protocol layer, COIP-K was designed to support multiple COIP protocols concurrently, to efficiently handle per-packet processing, and to support multipoint connections.

```
s = socket(family, type, protocol)

err = bind(s, addr, addrlen)

err = listen(s, 5)

s_new = accept(s, addr, addrlen)


err = read(s, buf, buflen)
err = write(s, buf, buflen)

close(s)
```

Figure 4: COIP-K server

The COIP-K system can be divided into two main parts: the first part is the core COIP-K code which is common to all COIP-K protocols; the second part is the group of protocol-specific modules which are plugged in on top of the core code to form an implementation of a COIP protocol. The main assumption that the COIP-K code makes is that IP addresses will be used (this allows COIP-K to ignore issues such as address resolution (ARP) by allowing the normal IP code to handle it).

COIP-K maintains a per-socket protocol control block (PCB). The COIP-K PCB contains state information such as connection IDs (CIDs), addresses, port numbers, routing information, timer values, logical channel numbers (LCN), the state of the connection, and a pointer to a per-protocol PCB which allows protocols built with COIP-K to have their own protocol specific control blocks.

COIP-K also maintains the interface between itself and the socket and network interface layers. The interface with the socket layer is maintained using a COIP-K user request function which acts as a dispatcher for all calls received from the socket layer. COIP-K calls the standard socket funnctions to pass data and control information to the socket layer in the reverse direction. For the interface with the network layer, COIP-K provides an interrupt function which the network layer calls for received packets. COIP-K uses an output function to send and route packets to an appropriate network interface. There are also other functions for PCB manipulations and running the connection state machine and forwarding data in the case of a gateway. On the other hand, the protocol specific modules are expected to support protocol specific functions such as packet creation, data extraction from a packet, control packet processing, and PCB lookup and setup functions.

COIP-K Feasibility and Viability. COIP-K has been successfully implemented in the Unix kernel within the framework of the Unix networking model. A subset of MCHIP implementation has also been realized using COIP-K. This exercise has served two purposes. First it has helped demonstrate COIP-K's usefulness in creating implementations of a COIP protocol. Second, it has helped us thoroughly debug and test COIP-K.

Easy and efficient module interchange is critical in realizing COIPs using COIP-K and in comparing alternate solutions of COIP modules. We show by an example that the COIP-K organization does indeed make the module interchange easy. We implemented two multipoint behaviors: treating a multipoint connection as a broadcast channel among endpoints (allowing many-to-many communication) and treating multipoints connection as one-way broadcast to allow one-to-many communication. A user can select one of these bahaviors by specifying an appropriate flag at the time of connection set up, and COIP-K

uses a corresponding subset of modules to provide the right functionality.

A number of performance measurement experiments have been carried out to characterize the per-packet processing effectiveness of COIP-K and to also quantify the cost of the COIP-K concept. The cost of COIP-K is defined as the additional overhead of packet processing using COIP-K as compared to direct protocol-implantation. This is quantified in terms of the number of additional function calls and protocol independent tasks needed to do per-packet processing. The COIP-K code has efficient per-packet processing. We found that COIP-K's throughput and delay performance are comparable to UDP, and are much better than those of TCP. Also, we found that COIP-K could easily overwhelm the ethernet hardware interface on Sun Sparc stations. Based on the processing latency of COIP-K on a Sparc1, the theoretical maximum data rate of COIP-K can be as high as 86.2 Mbps.

A number of demonstration applications have also been created on COIP-K. These demonstrations serve three objectives. First, they test and verify several capabilities (point-to-point, multipoint, gatewaying, etc.) of COIP-K. Second, they show that applications using the standard socket interface can be ported to work on COIP-K with minimal effort. Finally, these applications show that COIP-K can be used to create useful applications. Example applications include a file transfer protocol, BSD Unix telnet, multipoint chat program, multipoint network monitoring, etc.

Design of an ATM-FDDI Gateway

Sanjay Kapoor, Milind Buddhikot, Guru Parulkar

In this section we summarize the design of an ATM-FDDI gateway that implements the VHSI functionality and is able to sustain high throughput. We believe that ATM and FDDI are excellent target networks to explore the VHSI gateway architecture for two reasons: these networks represent two important component networks of the next generation internet, and thus the ATM-FDDI gateway has a lot of practical value; also these networks are sufficiently different to pose the necessary challenges to the gateway designer. For example, an ATM network is essentially connection-oriented using small fixed size cells with explicit resource management. An FDDI network on the other hand is a datagram network using relatively large variable size frames with no explicit resource management.

An important part of the gateway design philosophy is to partition the functionality into critical and non-critical paths. The critical path consists of per packet processing, and should be implemented in hardware for speed and performance. The non-critical path consists of connection management, resource and route management, and is best implemented in software due to complexity, need to do fine tuning, and flexibility. It is not an efficient approach to mix these paths as is generally done in present day gateways.

Figure 5 shows a high level schematic of a two port gateway. The gateway architecture consists of the following main components: ATM Interface Chip (AIC), SAR Protocol Processor (SPP), MCHIP Protocol Processor (MPP), Node Processing Element (NPE), reassembly buffer, FDDI interface chipset (SUPERNET[1]). We now explain each of these building blocks within the gateway.

ATM Interface Chip (AIC). This chip interfaces with the fiber optic link from the ATM network/switch. It synchronizes the incoming bit stream from the fiber optic link to an internal clock. It also performs an error check on the 5 byte ATM header. Similarly, it generates a CRC check for the ATM headers of outbound cells. Any cells with an error in the header are simply discarded by the AIC.

SAR Protocol Processor (SPP). This chip interfaces to the AIC and MPP. The SPP receives ATM cells from the AIC, and reassembles them into frames, using the SAR protocol headers. Similarly, it fragments FDDI frames received from the MPP into ATM cells and appends the necessary SAR headers to them. Some of the SPP's functionalities include: reassembly buffer management, reassembly timer management, writing cells into the reassembly buffer and reading frames out of the reassembly buffer concurrently for up to 16 active congrams.

MCHIP Protocol Processor (MPP). This chip interfaces with the SPP, NPE, and transmit buffer memory. The SPP passes the reassembled frames to the MPP, which then decodes the frame type and routes them appropriately. Control frames are routed to the NPE without any table lookup. There are two types of control frames: network level control frames (that is, frames for ATM signaling), and internet or (MCHIP) control frames. For data frames, MPP does a table lookup to determine the outgoing destination address on FDDI. Similarly it reads an FDDI frame from the receive buffer memory, and appends the ATM header to it. It then loads the frame into the SPP FIFO. The MPP can receive control frames and initialization data from the NPE, meant for both the SPP and the MPP. The MPP forwards SPP initialization and control frames to the SPP without modification.

SUPERNET. After loading frames in the transmit buffer, the MPP sends a transmission request to the NPE. This request is forwarded to SUPERNET by the NPE. Once the SUPERNET captures a valid

[1]SUPERNET is a registered trademark of Advanced Micro Devices Inc.

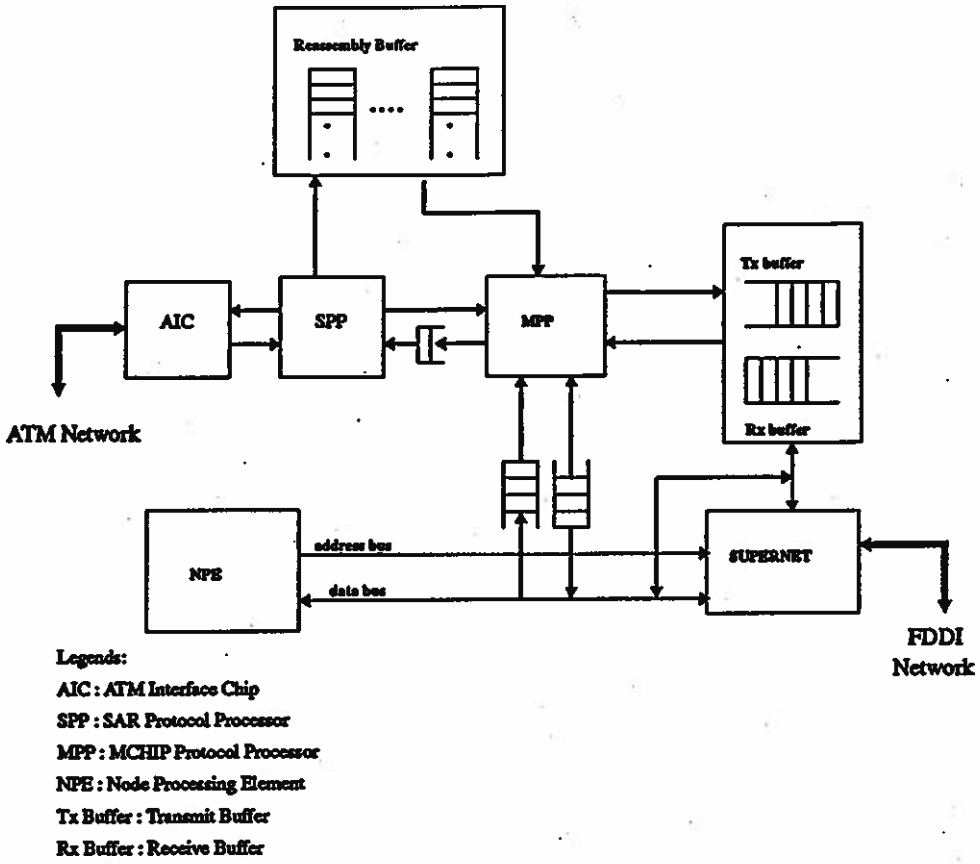


Figure 5: ATM-FDDI Gateway Architecture

token, it transmits the frames on the ring. When frames are received by the gateway, the NPE informs the MPP that there is a data frame in the receive buffer. The SUPERNET chip set implements the PHY layer and MAC layer protocols in VLSI. Station management and connection management are not part of the SUPERNET implementation, though it provides some registers to keep track of ring statistics, that can be used by the node processor to do station management and monitoring.

Node Processing Element (NPE). The main functions of the node processing element are: running the ATM signaling, MCHIP connection management, and FDDI station management protocols and maintaining the resource server and the routing tables. Besides this, it also performs the housekeeping and initialization functions for the various chips of the gateway.

Buffer Memories. It should be noted that there are three distinct buffer memories in the gateway, and they are: the reassembly buffer, transmit buffer, and receive buffer memory. We decided on separate reassembly and transmit buffer memories in order to keep the SUPERNET interface clean and simple. The exact size of the buffers is a matter of further study, and can only be determined after a thorough simulation of the expected traffic intensity, and the type of traffic possible.

There are 3 FIFOs used in the gateway, labeled A, B, and C in Figure 5. FIFO-A buffers control frames

received from the ATM network to be processed by the NPE. FIFO-B buffers ATM control frames and the initialization data for the MPP and the SPP originating from the NPE. The MPP decodes the type of packet and then routes SPP control and initialization frames to the SPP. The FIFO-C provides buffering for the FDDI fame being fragmented by the SPP and being sent to the ATM network.

Detailed designs of the AIC, SPP, AND MPP have been completed and can be found in [11]. The designs are in sufficient detail to allow a prototype implementation. A detailed simulation study of the design has been also completed which includes performance analysis and evaluation of various tradeoffs associated with the design.

SPP and MPP Complexity. The implementation complexity of the SPP and MPP consists of two components: fixed cost and cost proportional to the number of congrams to be supported. The `Reassembly Logic` and `Fragmentation Logic` represent cost proportional to the number of congrams. It is important to note that the significant fraction of the reassembly and fragmentation cost is contributed by look up tables and reassembly timers that store state information for active congrams. Similarly, in the case of the MPP, most of its hardware cost is contributed by two look up tables which store state information for active congrams.

An approximate but overly generous transistor count analysis of the MPP and SPP suggests that these protocol processors can easily support 64 congrams with the transistor count in the range of $60K$. Thus, each processor can be packaged into a single custom PGA (pin grid array) VLSI chip. It is important to note that the current design has scope for further optimizations which will lead to even lower transistor count. However, the current design does not include functionality to allow diagnosis and/or testing of the chip which is typically part of any production chip. We believe that these two factors cancel out, and the transistor count of $60K$ is a reasonably accurate estimate of the complexity.

We want to emphasize that providing support for 64 active congrams in the SPP and MPP is sufficient. If there are more than 64 congrams active at the same time, it would mean that the data rate of most congrams is relatively small, because all congrams share the 100Mbps of the FDDI network. Congrams requiring low data rates can be served by the NPE with minimal hardware support. In other words, the SPP and MPP would support 64 congrams which would consume most of the 100Mbps of the FDDI network, and other congrams (over 64) would be supported by the NPE with software processing.

SUPERNET and Gateway Performance. The SUPERNET architecture imposes two constraints that have implications on the performance of the gateway as explained in the following paragraphs.

- The SUPERNET protocol requires that the NPE be involved in transmission and receiving of every frame. For example, for a frame to be transmitted on the FDDI, the NPE requests SUPERNET to capture a token and initiate transmission. Similarly, when a frame is received, SUPERNET notifies to the NPE, which decodes the type of the frame. For data frames, the NPE requests the MPP to forward them on the ATM network. The NPE involvement is contrary to our principle of keeping the critical path as a simple hardware pipeline, and it also adds considerable delay in the critical path, especially compared to the delay contributed by the SPP and MPP

  The NPE delay becomes a bottleneck and is the primary reason for cell/frame loss for short frames in the gateway as explained in the following subsection.

- The SUPERNET chip set allows the maximum transmit buffer size of $256K$ which is not sufficient to avoid frame loss for very bursty (burst lengths of 1200 cell and more) ATM sources. A simulation experiment shows that for an ATM link rate of 155Mbps and the aggregate average data rate of all ATM connections higher than 70Mbps, the transmit buffer of $256K$ overflows and leads to significant frame loss [10].

Reassembly Buffer Organization. It may seem attractive to make both the reassembly buffer and the transmit buffer one and the same. However, we chose to keep them separate for the following reasons.

- The design uses a commercially available SUPERNET chip set as the FDDI interface, and its architecture makes it difficult to use the transmit buffer space as the reassembly as well as transmit buffer. Also the maximum possible transmit buffer of $256K$ severely limits the maximum number of active congrams.
- The RBC (RAM Buffer Controller) supports three DMA channels to the transmit and receive buffer which are used by the NPE, MPP, and SUPERNET. If the reassembly logic were to use the same buffer, it will require an additional DMA channel which would, of course, lead to the increased design complexity and possible performance deterioration.
- A separate reassembly buffer statically segmented on the basis of VCIs makes realization of the reassembly process in hardware much simpler.

Another important aspect of the current design is that it allows two reassembly buffers per connection. The simulation results demonstrate that two buffers per connection is sufficient if the size of reassembled frames is larger than a threshold or the date rate of each connection is smaller than a threshold [10]. For example, if the frame size is at least 8 cells, a congram with data rate of 100Mbps may not lose any cells/frames due to an overflow in the reassembly buffer. Otherwise, if the frame size is as small as 2 cells, then the congram should be limited to only 30Mbps. Both these thresholds essentially ensure that while the next frame is being reassembled, there is enough time to allow the SPP and MPP to forward the assembled frame to the transmit buffer, in spite of the NPE delay. It is, however, important to note that the current design does static partitioning of the reassembly buffer memory and gives the same number of reassembly buffers to each connection. This is not particularly attractive if different connections have different characteristics in terms of the data rate and frame size. A dynamic scheme of memory usage needs to be investigated for the next version of this design.

Resource Enforcement. It is important to note that connection oriented internetworking with performance guarantees requires resource allocation, resource enforcement, and scheduling policies. Clearly, the resource enforcement and scheduling are part of the critical path and should be realized in hardware. However, a number of research groups have been investigating alternate resource management schemes, and the *right* solution has not yet matured enough to consider a hardware implementation. As the continued research leads to a better understanding of these issues, future generation designs will be appropriately modified to incorporate these functionalities.

Performance Analysis of the Ethernet under Bursty Traffic Conditions

Tony Mazraani, Guru Parulkar

In this section we summarize a study on performance analysis of Ethernet under conditions of bursty traffic. This study is motivated by the following two step reasoning.

First, we believe that at least in the first phase of deployment of broadband networks, they will be used as backbone networks at the campus, regional, and national level, and the access networks will continue to be local area networks (LANs) such as Ethernet, token ring, and FDDI.

Second, broadband network deployment will also bring with it a whole set of new applications, such as scientific imaging and visualization, multi-media conferencing, video distribution, and others. We claim that most of the broadband applications need to be modeled as bursty sources, and therefore, previous analyses of Ethernet with Poisson sources cannot be used to accurately characterize the performance obtained by applications running on the Ethernet in a broadband communication environment. In this section we report results of our simulation study aimed at characterizing Ethernet performance for broadband applications.

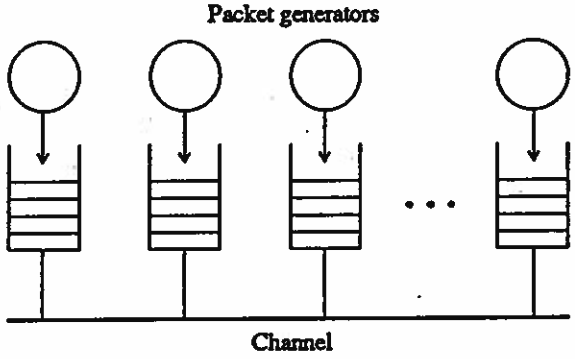


Figure 6: Network configuration

The simulation model simulates the operation of the standard Ethernet at the level of the data link layer. Some features of the physical layer, such as cable propagation and repeater delay are also included. The network consists of a single channel and a set of stations. Each station has an independent packet generator and a buffer of infinite capacity as shown in Figure 6. Hence, packet loss in this model occurs only when the Ethernet protocol discards a packets after 16 unsuccessful retransmission attempts[2]. In each station, the first packet in the queue is completely transmitted before a second packet, if it exists, is dequeued. A *balanced star* topology is assumed, *i.e.* the propagation delay between any two stations on the channel is constant. The simulator uses discrete-event simulation techniques, and is written in C and runs under the Unix operating system.

In this study, a bursty source is modeled as a *Markov chain* consisting of two states, *active* and *idle*, as shown in Figure 7(a). When the source is in the active state, it generates packets at a rate λ_p. In the idle state the source is shut off. The holding times in both the active and the idle states are exponentially distributed with means $1/\beta$ and $1/\alpha$, respectively. These holding times are computed using the peak bandwidth, average bandwidth, and burst factor.

[2]However, we do monitor the packet queue of each station, and have found that the average and maximum queue lengths remain reasonable.

Note that for Poisson traffic, the mean holding times in the active and idle states of the packet generator are not specified. The packet generator is configured to remain in the active state for the duration of the experiment (*i.e.* $1/\alpha = 0$ and $1/\beta = \infty$). The only remaining packet generation parameter is λ_p, the mean packet arrival rate per station.

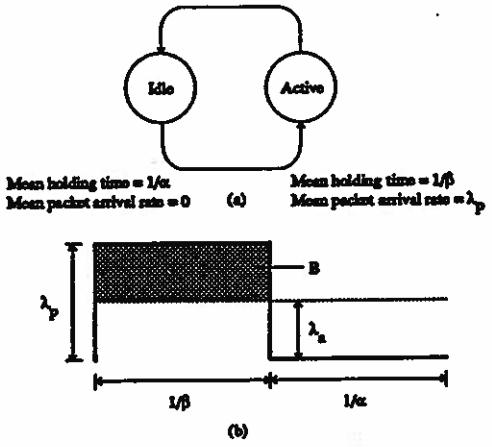


Figure 7: Model of a bursty source

Figures 8 and 9 present mean packet delay and packet loss rate vs. throughput for different values of burst factor. The results clearly show that Ethernet performance under conditions of bursty traffic is different from the performance obtained when using traditional analyses with Poisson traffic. The simulation results show that the performance of Ethernet deteriorates as the burst factor increases. For instance, assuming a peak-to-average ratio of 4 and 1500 byte packets, Figure 8 shows that in order to guarantee less than 10 msec mean packet delay, the Ethernet utilization has to be much lower than the channel capacity, which is roughly 90%. Under conditions of Poisson traffic, the Ethernet utilization should be lower than 72%. However, under conditions of bursty traffic, this value deteriorates quickly to 57% and 43% for burst factors of 10 and 100, respectively. Bursty traffic has a more dramatic effect on packet loss. Figure 9 shows that in order to guarantee less than 10^{-4} packet loss under conditions of Poisson traffic, the Ethernet utilization should always be less than 62%. Under conditions of bursty traffic, this value decreases quickly to 39% and 27% for burst factors of 10 and 100, respectively.

The study also includes results that show how Ethernet performance changes with peak-to-average ratio and source cycle time as parameters of bursty sources [13].

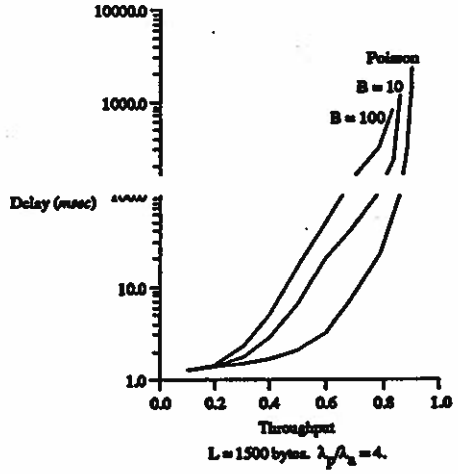


Figure 8: Mean packet delay vs. throughput and burst factor

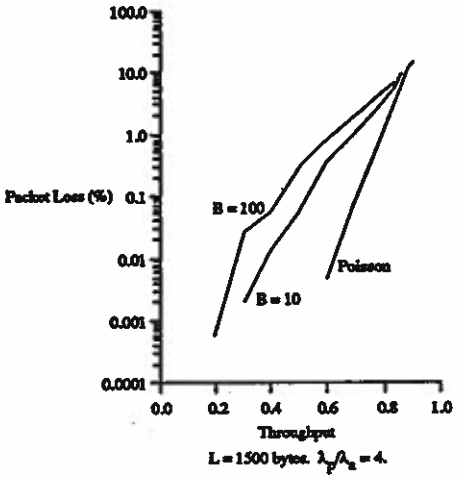


Figure 9: Packet loss vs. throughput and burst factor

Resource Management in Local Area Networks

Tony Mazraani, Guru Parulkar

The internet layer must support emerging broadband applications which include video distribution, distributed scientific computation and visualization, computer imaging, and multimedia conferencing. Supporting such applications means that the internet layer has to provide variable grade service with performance guarantees. If the underlying network is capable of managing its own resources and guaranteeing performance to applications, the internet layer can simply take advantage of this network facility. However, in the case of networks with no resource management capabilities, it is the responsibility of the internet layer to do resource management on behalf of the network. In order to provide performance guarantees at the internet level, the VHSI uses *resource managers* which manage the resources of those networks with no resource management capabilities.

In this section, we summarize a resource management model for the Ethernet in a connection-oriented communication environment, under conditions of bursty traffic. This model uses a central resource manager in a directly connected gateway. The role of the resource manager is to keep track of all active connections and their resource usage, and accept or block new connections depending on resource availability. The resource manager is always consulted before a connection can be established.

The goal of the resource manager is to guarantee performance to established connections, and to manage the network resources efficiently. Efficient resource management in this environment means maximum utilization of the network and minimum connection blocking, packet loss, and packet delay. Note, however, that minimizing packet delay may result in lower channel utilization. This study however shows that a simple resource management model can be devised to provide bounded packet loss and delay to various applications with reasonable channel utilization and blocking.

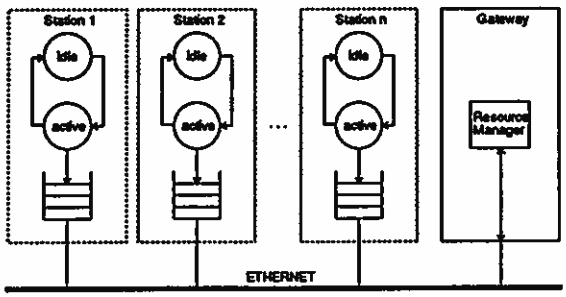


Figure 10: Network Configuration

The network for this study is shown in Figure 10. It consists of a set of stations and one gateway, all connected via an Ethernet. Communication in this network is connection-oriented, and connection management uses the new internet protocol [12]. This protocol requires that a station ready to start a connection send a connection request to the resource manager which resides in the directly connected gateway[3]. The resource manager gets the connection attributes from the request, and then accepts or blocks the connection depending on resource availability on the network. When the connection is terminated, the resource manager deallocates reserved resources for the connection. Traffic from each station is assumed to be bursty. Stations use five attributes to characterize their resource requirements; peak bandwidth, average bandwidth, burst factor, delay, and packet loss. Stations abide by their bandwidth and burstiness attributes, and use the standard Ethernet protocol to access the channel.

[3]Note that the resource manager does not have to be in the gateway. Since Ethernet is a broadcast network, any station on the channel may host the resource manager.

The resource manager accepts a new connection as long as its attributes (as well as those of the the most constraining active connection) are satisfied. It makes its decision based on a resource database which contains simulation data obtained from the performance analysis of the Ethernet under conditions of bursty traffic. The data is arranged in a matrix format with six columns representing the following six parameters: peak-to-average ratio, packet length, burst factor, average bandwidth, delay, and packet loss. Since the resource database consists of a set of discrete points, the resource manager uses linear interpolation to estimate those points that are not covered by the database. The details of the resource database and decision making algorithm of the resource manager can be found in [13].

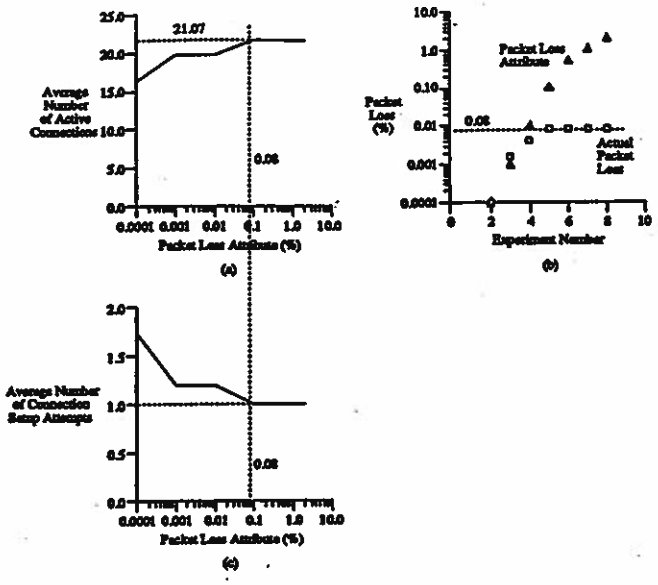
We decided to use a database of simulation data because there are no accurate analytical models that describe Ethernet performance in terms of the three parameters we use to characterize bursty sources (peak bandwidth, average bandwidth, and burst factor). The database approach has the advantage of being simple to implement and can also be easily expanded to incorporate new data about the performance of the network. This expansion is achieved by simply adding the new data to the resource database of the resource manager. The disadvantage of the database approach is its dependency on six variable parameters, especially packet length. This dependency means that a large amount of data is required to predict the performance of the network under all possible real-time conditions.

Figures 11 and 12 show simulation results obtained using connections with a peak-to-average ratio of 2 and average bandwidth of 0.2 Mbps. The average number of active connections as a function of the packet loss attribute is shown in Figure 11(a). The solid lines correspond to use of the resource manager, and the dotted lines correspond to results with no resource manager. These results show that when the resource manager is off, the average number of active connections on the channel is 21 and the mean packet loss rate is 0.08%. However, when the resource manager is on, packet loss rate less than 0.08% can be achieved at the expense of decreasing the average number of active connections. Note that as the packet loss attribute becomes larger than 0.08%, the average number of active connections starts to converge to 21. At this point, all connections are established on the first attempt. The actual packet delay on the channel ranges from 1.97 msec for experiment 1 to 3.17 for experiment 8.

Figure 11(b) shows the packet loss attribute requested by connections and the corresponding actual packet loss on the channel for each experiment (simulation run). These results show that by using the resource manager, the actual packet loss is always less than the requested loss attribute. When the packet loss attribute increases beyond the worst-case packet loss of 0.08%, all connections requests can be satisfied, and thus the actual packet loss converges to 0.08%.

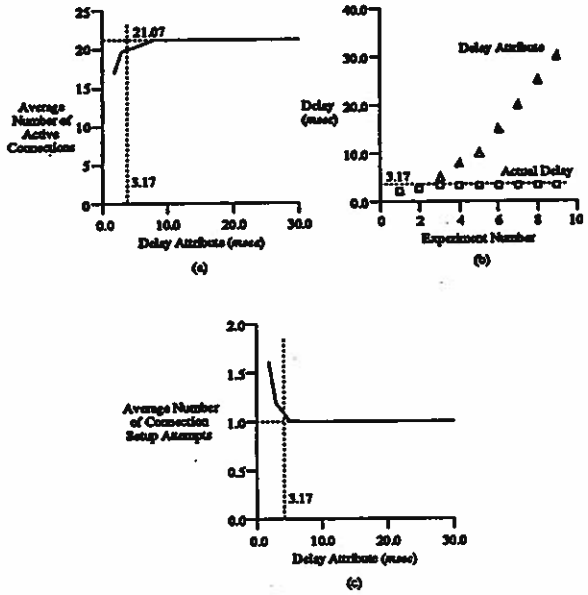
The average number of attempts made to set up a connection, which can also be considered as a measure of blocking, is shown in Figure 12(c) as a function of the packet loss attribute. When connections request very low loss rate, the resource manager blocks connections more often to guarantee the required performance. As a result, the average number of attempts to set up a connection increases, as shown in Figure 12(c) for loss attributes less than 0.08%. However, when the delay attribute becomes higher than the worst-case packet loss on the channel, the average number of connection setup attempts converges to 1, which means that connections are established on the first attempt.

More details of our performance study can be found in [13].



L = 1500 bytes. B = 10 packets.
Peak-to-average Ratio = 2. Average Connection Bandwidth = 0.2 Mbps.

Figure 11: Resource manager performance – similar connection attributes, variable packet loss attribute



L = 1500 bytes. B = 10 packets.
Peak-to-average Ratio = 2. Average Connection Bandwidth = 0.2 Mbps.

Figure 12: Resource manager performance – similar connection attributes, variable delay attribute

Axon: Host Communication Architecture

The Axon Host Communication Architecture for Gigabit Applications

James Sterbenz, Guru Parulkar

The research and development of high speed switching systems and networks will result in the ability to construct networks and internetworks that support data rates up to a few Gbps. We refer to this as the VHSI (very high speed internetwork). Similarly, processor and workstation power and functionality is rapidly increasing which make network applications, such as distributed scientific computation and visualization, video distribution on demand, multimedia conferencing, and remote imaging possible. For such applications to fully utilize VHSI performance and functionality, however, the host-network interface architecture must be capable of delivering the high bandwidth to the applications with minimum latency.

The Axon architecture satisfies this need by providing (1) an integrated design of host and network interface architecture, operating systems, and communication protocols stressing both performance and the required functionality for demanding applications such as visualization and imaging, (2) a proper division of functionality in hardware and software for optimal performance, (3) reorganization of end-to-end protocols to take advantage of the increased functionality of emerging high speed networks. The overall Axon architecture is described in [19]. Significant features of the Axon architecture are summarized below, and presented in Figure 13.

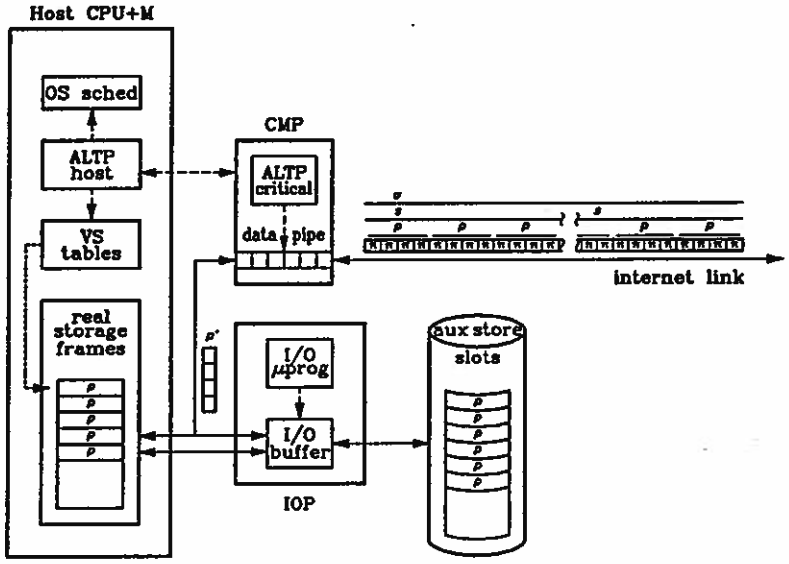


Figure 13: Axon Architecture

System level IPC support and NVS. The system level support for the various application level interprocess communication paradigms is provided by two components: NVS and NMP. NVS (network virtual storage) is the system level shared memory interface for shared variables, GRPC (generalized remote procedure call), and segment streaming. NMP (network message passing) is the system level message passing interface. NMP performs a relatively straightforward transformation of program (*send, receive*) primitives to corresponding transport protocol message-object transfer calls. NVS extends the typical virtual storage mechanisms to include systems throughout the VHSI. A segmented programming model is used, with underlying paging to facilitate storage management, as in the Multics operating

system.

Transport protocol. At the transport level, applications using the VHSI are best supported by a set of simple ALTPs (application-oriented lightweight transport protocols) for various classes of applications. Key issues in the design of an ALTP are the implementation of critical functions in hardware, rate based flow control, application-oriented error control, and structured collections of packets.

ALTPs have their critical path functions implemented in VLSI hardware. The critical path consists of the data path, and routine control functions allowing data to flow at peak network rates, once a transport operation has been initiated. By optimizing the critical path functions, and by processing multiple packets in a single transport level operation, the per packet processing can be performed in real time at the full sustained data rate. For the protocol to be efficiently implemented in hardware, the protocol, hardware design, and host operating system should be well integrated.

Host and network interface architecture. The Axon architecture interfaces the CMP (communications processor) to the back end of a special dual-ported CMM (communications memory module). The CMM has a conventional random access port which appears like any other memory bank to the processor-memory interconnect. The second port is a high speed serial access interface to the CMP.

The goals for the design of the CMP include the ability to perform critical path functions in real time, with no packet buffering, and the ability to incorporate the necessary functions in VLSI. The CMP implementation consists of a pipelined set of datapath modules and control modules.

Work is in progress on analytical and simulation models to evaluate the associated tradeoffs more rigorously, and on a detailed design of the communication processor. Successful completion of this effort will result in the following:

- Demonstration of viable Axon design (functional, performance) from the design specifications, implementations, and overall simulations. This provides the proof of concept.
- Determination of function that is part of the critical path, as data rates scale above 1 Gbps, based on the time–space complexity analysis.
- Specific solutions for the network-host object mapping, determination of components to be included in the critical path, OS interaction, and determination of whether implicit or explicit mapping is better.
- Understanding of relationship between latency and memory requirements in terms of locality and working sets, based on latency, processor performance, and network bandwidth, with respect to the incorporation of function in the critical path.

Axon: Network Virtual Storage Design

James Sterbenz, Guru Parulkar

This note presents an overview of the design of *network virtual storage* (NVS) in the Axon host communications architecture for distributed applications. NVS extends segmented paged virtual storage management and address translation mechanisms to include segments located across an internetwork. This provides the ability to efficiently use the shared memory paradigm in non-local environments, as well as the support for a very high speed end-to-end data path between demanding applications such as scientific visualization and imaging. Additionally, segments that are transported across the VHSI are mapped into the address spaces of processes by NVS. This eliminates the need to copy segments from intermediate system buffers into the process address space, resulting in lower latency and system overhead.

Data structures. Addressing of a segment resident on a non-local host is accomplished by including a *host id* field in either the virtual address, or in the segment descriptor table (SDT) entry. When a segment fault occurs for a nonlocal segment (indicated in the segment descriptor), the dynamic address translation facility invokes the transport protocol (ALTP-OT) to get a copy of the segment from the appropriate system. As the segment is returned, the appropriate page and segment descriptor presence bits are set, so that program execution can resume with the normal fault recovery mechanisms. The address translation data structures are presented in Figure 14. Address pointers are represented by arrows on solid lines, the movement of data/requests is represented by arrows with dashed lines.

The local storage management data structures are extended to allow the addressing of segments on other hosts. This is accomplished by adding a *host id* field to the *known segment table* (KST), which holds the symbolic segment bindings. This is an index into the per process *known host table* (KHT), which holds the symbolic host name to address/path bindings. This binding is resolved by searching the *host address table* (HAT) for each host, which gets its binding by invoking an internet name server, using the *host name database* (HND). There are also tables (not shown in the figure) to assist in n-way IPC using multipoint connections.

Segment types. Axon segments are of two types: *memory* and *video*. Memory segments are either *code* or *data* subtype. Memory segments are divided into pages, and may be organized into segment groups, for performance reasons. Video segments are either *text* or *graphics* subtype. Graphics segments are bit-mapped video image frames; text segments correspond to a text window on a workstation. Video graphics segments are divided into scanlines, and may be organized into multi-frame images (eg. a color image of R,G,B frames).

Segments have attributes of *read, write, execute*, indicating the type of access allowed. Code segments are assumed to be pure (refreshable), and therefore always have access attributes of execute-only. Data segments may be readable and/or writable.

Storage management policies. NVS in Axon involves extensions and additions to storage management policies. The fetch policy is not affected by NVS, except that demand-segment implies a degree of anticipatory-page movement across the network and is, in fact, desired to counter latency effects. The (real) placement policy is not affected by NVS at all, since placement is trivial for paged storage management, and unaffected by NVS.

An entirely new policy, the *remote placement policy*, is used to determine where remote segments are placed while being used by the local system. These include real store, auxiliary store, a combination, or frame buffer placement, with a number of sub-policy options, such as swappable and nailed. Due to the presence of segments from remote hosts, the conventional replacement policy is affected. In particular, placement of entire remote segments in real store will result in presence of some of the pages that are

not really in the process locality set in the real store. This indicates that the estimation of working sets must consider local and remote segments differently.

Nvs and its storage management policies are described in more detail in [18, 20].

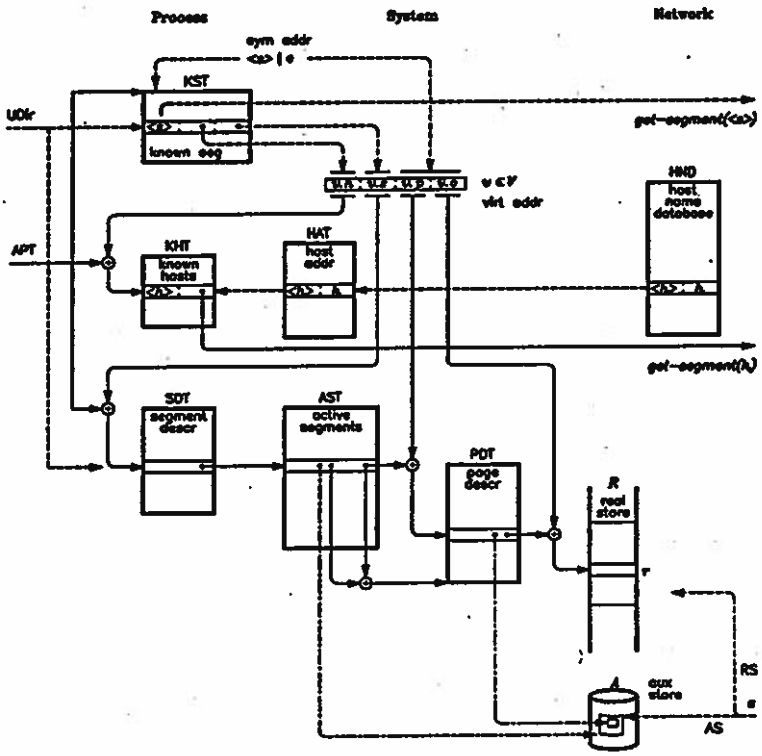


Figure 14: Network Virtual Storage Address Translation

Application-oriented Lightweight Transport Protocol

James Sterbenz, Guru Parulkar

Most current transport protocols (including TCP, X.25, and SNA) are *general purpose*, providing complete flow and error control to all applications. This results in complexity of implementation and operational overhead that is not necessary for particular applications. It may be possible to *functionally partition* a transport protocol, to provide only the functionality needed for various classes of applications, while still allowing the use of a single protocol by adjusting the appropriate parameters. A similar strategy that does not require that a single transport protocol serve all applications, is to have a small set of *application-oriented* transport protocols. A possible set might consist of application oriented transport protocols for object transfer, for voice and for video distribution.

Somewhat orthogonal to the *scope* of the transport protocol is the simplicity of design and efficiency of operation. A protocol that is simple, streamlined, and efficient is referred to as a *lightweight* protocol. Note that while it is possible to design even a general purpose protocol to be lightweight to some degree, it is much easier to do so with an application oriented protocol that can efficiently serve the corresponding application class with the appropriate (and simplified) error and flow control mechanisms. This is the approach taken in Axon with ALTPs (application-oriented lightweight transport protocols).

In the Axon transport level, IPC across the VHSI is supported by ALTP-OT (ALTP for object transfer), which has its critical path function implemented in VLSI hardware, is optimized to provide the kind of performance guarantees and functionality required for object transfer. The ALTP-OT requests include connection establishment/termination, segment/page/message transfer, and packet retransmission. ALTP-OT is described in detail in [21]. ALTP-OT design is summarized in the following paragraphs.

Flow control. When ALTP-OT opens a connection, it specifies attributes of the connection in terms of parameters such as average and peak bandwidth, and a factor reflecting the burstiness of the transmission. These parameters can be translated into buffer requirements, based on a rate between the average and peak specifications. Initial exploration of this allocation has been researched in [1]. Since the connection set up is end-to-end, all the intermediate systems, including various packet switches and gateways, as well as the endpoint hosts that this connection goes through, can make appropriate buffer and resource reservations. The rate specification will have to be negotiated between ALTP-OT and the internetwork/network layers, to ensure that the requested rate does not exceed the capacity of internal network nodes and gateways. As a result, as long as both ends transmit subject to the rate specification, the probability of packet loss due to buffer overruns is very low.

It is assumed that the internet level below has the functionality to support connections with specified bandwidth requirements, and furthermore, that the probability of packet loss, errors, and resequencing is low enough to design the critical path with the assumption that handling such problems is the exceptional case.

The only explicit flow control exercised by ALTP-OT is the control of the CMP (communications processor) data transmission rate to correspond to the rate specification. The benefit of the ALTP approach is manifest in that only attributes significant for object transfer need to be considered and that the rate control functionality necessary on a per-packet basis can be implemented in hardware.

Error control. In the VHSI environment error control is performed, as much as possible, on an end-to-end basis, and is decoupled from flow (rate) control, as described above. The ALTP error control is as simple as possible, based on the target application characteristics. For ALTP-OT, the packet handling is as follows:

- duplicate packets are discarded
- corrupted packets are discarded, and retransmission requested based on application need
- missing packets are detected by the expiration of a timer, and retransmission is requested
- packet sequence is irrelevant because each packet is directly placed into the proper location of application memory space.

The Axon architecture allows application specific selective retransmission of corrupted or missing packets, which gives considerable flexibility in retransmission strategy. Note that due to the orientation of ALTP-OT to this application, the handling of duplicate and out-of-sequence packets is considerably simpler and more efficient than would be the case for a general purpose transport protocol. Since data packets have sufficient header information to indicate the connection and request, and are placed directly into the proper location of target store, the overhead of sequence buffering is not necessary. The simplified error control of ALTP-OT can be efficiently implemented in VLSI hardware.

The Axon Host-Network Interface Architecture

James Sterbenz, Guru Parulkar

The Axon architecture interfaces the CMP (communications processor) to the back end of a special dual-ported CMM (communications memory module). The CMM has a conventional random access port which appears like any other memory bank to the processor–memory interconnect. The second port is a high speed serial access interface to the CMP. The goals for the design of the CMP include the ability to perform critical path functions in real time, with no packet buffering, and the ability to incorporate the necessary functions in VLSI.

The ALTP-OT critical path, consisting of the data path and per-packet processing, is implemented in the CMP. The CMP consists of datapath (CMP_d) and control (CMP_c) portions. The CMP datapath interfaces to the VHSI optical links and the sequential ports of the CMM, and performs such functions as encryption/decryption and format conversion (encode/decode). The CMP control functions are those directly related to the datapath such as header build/decode, checksum generate/compare, CMM address generation, rate specification timing, as well as the per-packet congram multiplexing and control.

The Axon interface in fact also includes a high performance microprocessor, the *CMP assist processor* (CAP), which performs functions that are not part of the critical path, but require higher performance than could provided by the host CPU. The CAP is responsible for building control packets and passing them to the CMP for transmission and checksumming. Similarly, control packets received by the CMP are passed to the CAP for full decoding and subsequent action, which may involve interaction with the host CPU. The CAP is involved in the timer management for error packet retransmission, and in the packet arrival to page presence mapping ($\pi \rightarrow p$), in particular setting page and segment presence indicators and subsequent host notification for fault recovery.

The host CPU is responsible for link/segment/page fault handling (NVS), and per-congram functions (ALTP-host) directly corresponding to application requests.

In the following paragraphs we summarize the design of the CMP (Figure 15).

Congram control

The multiplexing of congrams (which may be considered to be soft connections) is handled by the *congram control logic*:

MPX – MULTIPLEXING control logic performs the hardware context switching of the CMP in response to transmission requests and received congram ids.

CSR – CONGRAM STATE REGISTERS hold all of the state information for each active congram, to allow rapid control and pipeline configuration changes for multiplexed congrams using the CMP. There is a set of CSRs for both the transmitting and receiving side of the CMP.

Packet data paths

The *transmit data pipe* and *receive data pipe* are the main data paths of the CMP. The transmit data pipe modules are:

ECD – ENCODE performs any transformations required by internetwork data format standards and accommodates heterogeneous hosts (such as byte ordering).

ECR – ENCRYPT performs data encryption of packet data and internal control fields.

P2S – PARALLEL-TO-SERIAL datapath conversion.

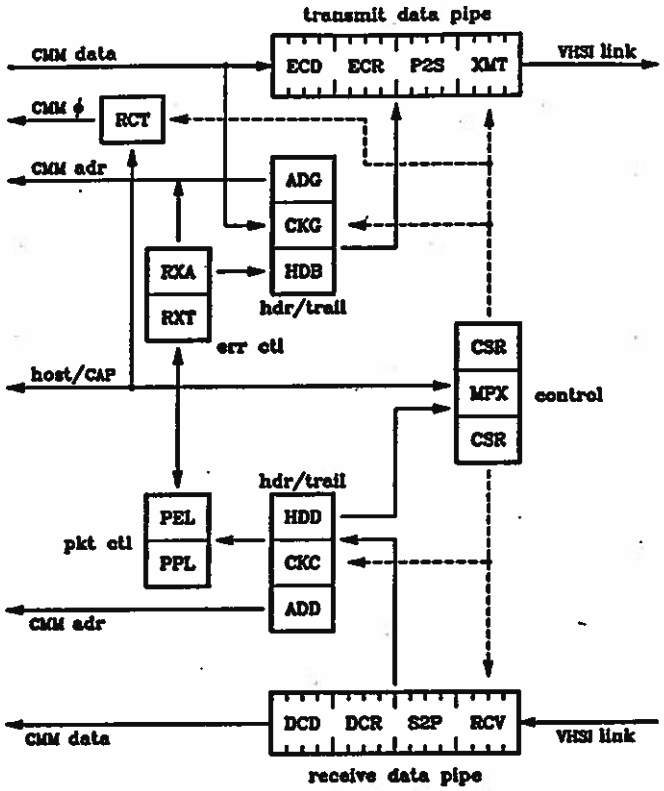


Figure 15: Communications Processor Block Diagram

XMT – TRANSMIT performs line coding and transmission functions.

The receive data pipe modules are:

RCV – RECEIVE takes the bit stream derived from the optical receiver, handles line coding and clock recovery. for the receive data pipeline.

S2P – SERIAL-TO-PARALLEL datapath conversion.

DCR – DECRYPT performs data decryption of the packet data and internal control fields.

DCD – DECODE performs any required internetwork-to-host data format transformations.

Per packet control

Associated with the transmit data pipe are control modules:

RCT – RATE CONTROL uses the rate specification r_i for each congram i to determine the timing of data reads from the CMM, and thus the rate at which packets are clocked out of the transmit pipeline for each congram. The r_i values for each congram are obtained from the corresponding CSR.

ADG – ADDRESS GENERATE uses the initial CMM address of each page to form the addresses for each packet read from the CMM.

CKG – CHECKSUM GENERATE sums the packet data fields as they pass through the data pipeline. The computed checksum is then inserted in the packet trailer.

HDB – HEADER BUILD uses the congram id and control information from the corresponding CSR to build the header for the packet. All protocol levels of encapsulation (ALTP, MCHIP, and network) are done at one time. The congram and request ids (c, q) and packet id (segment id k, page number j, packet number i) are inserted into a header template.

Associated with the receive data pipe are control modules:

HDD – HEADER DECODE determines the congram id for CMP configuration, and determines whether the packet type is control or data. Control packets are passed to the CAP. For data packets, HDD determines the packet address in the CMM from the packet index (ijk) and the base address of the page from the corresponding CSR. The congram and request ids (cq) are used by MPX to select the appropriate CSR.

CKC – CHECKSUM COMPARE sums the packet data fields as they pass through the data pipeline. The computed checksum is then compared with the actual checksum in the packet trailer. If a mismatch is found, the PPL and PEL (packet presence and error logic – see below) are notified to indicate that the packet has been discarded after initial receipt.

ADD – ADDRESS DECODE uses the initial CMM address of each page (from the CSR), and the packet index (ijk) to form the CMM address for the writing of each packet into the CMM.

The *packet control* logic is responsible for recording packet arrivals and missing or corrupted packets.

PPL – PACKET PRESENCE LOGIC keeps track of packet arrival to allow the CAP and host to determine the presence of complete pages and segments, so that the appropriate page descriptor table and segment descriptor table presence bits can be set, and host CPU application resumed.

PEL – PACKET ERROR LOGIC keeps track of corrupted (from CKC) and missing (from RXT) packets so that retransmission requests can be made, and also invalidates corrupted packets to the PPL.

The *error control* logic is responsible for generating the appropriate selective retransmission requests and packet addresses at the receiving end, and retransmitting packets on the sending end.

RXT – RETRANSMIT TIMERS determine (in conjunction with the CAP) when packet retransmission requests should be made, based on the retransmission policy. Timer values are accumulated to the proper granularity (packet, page, segment, or group). The fetch and preemption options are then used to determine when the retransmit-packets control packet should be sent, and the PEL is used to construct the retransmit packet bit map.

RXA – RETRANSMIT ADDRESS generates the (local) CMM addresses for packets to be retransmitted, using the base address of the corresponding page and the retransmit packet bit map in the retransmit-packets control packet received.

Simulation of the Axon Host-Network Interface Architecture

James Sterbenz, Milind Buddhikot, Guru Parulkar

This section describes the Axon architecture in the context of its simulation which was done using the BONeS™ simulation package [5]. The simulation of the Axon architecture serves two purposes. First, it is a verification of design and the ability to implement key mechanisms in hardware. In particular, the simulation model of the CMP (communications processor) uses modules that correspond to functions available in a VLSI cell library. Secondly, the simulation provides a platform to evaluate design options and tradeoffs before a prototype system is built.

The Axon system level simulation consists of two hosts connected by the VHSI, as shown in Figure 16.

Figure 16: Axon System Level Simulation

The entire system model consists of 7 levels of hierarchy, which when fully flattened contains 1018 blocks. The upper levels of the Axon model will now be described.

VHSI. The VHSI is modeled at a high level of abstraction for the purpose of simulating the Axon host–network interface. The model includes end-to-end latency (and its variance), packet loss (including burst errors), missequencing, duplication, and bit errors. These are used as parameters into the behavioral simulation of paths connecting hosts through the VHSI.

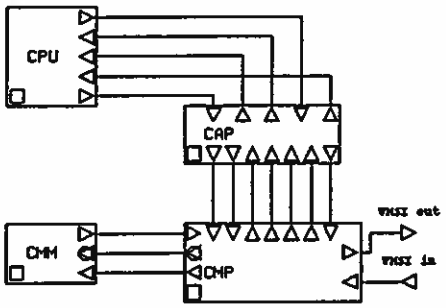


Figure 17: Axon Host

Hosts. In a real Axon implementation, there would be a number of symmetric hosts scattered throughout the internet. The simulation model consists of two hosts, which will be referred to as the local and remote hosts depending on where a request has been initiated. The characteristics of load are important rather than the routing of traffic between hosts. Thus it is necessary to ensure that local host making requests has load induced by another host, and similarly that a remote host satisfying requests is also loaded by request issuance.

The Axon host model is presented in Figure 17. Each host contains a CPU, CAP (CMP assist processor),

[3] ™BONeS (Block Oriented Network Simulator) is a trademark of Comdisco Systems, Inc.

CMP (communications processor – the network interface chip), and CMM (communications memory module). The local host is responsible for generating requests based on the address reference trace model. The remote host receives requests and returns segments based on the request.

This organisation allows a direct connection through the CMP between the CMM and VHSI. Thus, packets can be transmitted and received without any host interaction, with the CMP performing all critical *per* packet processing. The CAP serves to perform functions that do not need to be implemented in CMP hardware, but involve protocol processing that may be offloaded from the CPU.

CPU. The CPU simulation model generates requests from a process model consisting of a dispatching queue served by the CPU instruction processor. Processes receive service for a specified burst length (in number of instructions), and are returned to the dispatching queue. The program execution model assumes that process execution can be divided into phases during which a set of segments is in the locality set. This phase behavior is a common way of modeling program locality. In the Axon environment, segments may be located on other hosts across the VHSI. Thus, with a given probability, a remote segment fault will take place. This results in transfer of control to ALTP-OT which issues a get-segment request, which is then passed to the CAP.

When pages within the segment return, the CPU is notified of their arrival by the CAP. The CPU can then mark them present in the corresponding page table and recover from a page fault if necessary.

CAP. A high performance microprocessor, the *CMP assist processor* (CAP), performs functions that are not part of the critical path, but require high performance that would be inadequately provided by the host CPU and would adversely impact the performance of other host processes. The CAP is responsible for building control packets and passing them to the CMP for transmission and checksumming. Similarly, control packets received by the CMP are passed to the CAP for full decoding and subsequent action, which may involve interaction with the host CPU. The CAP is involved in the timer management for request retransmission, and notifies the host when pages have been completely received.

Since the CAP is a microprocessor running software processes to perform its function, operations are modeled as processing delays with parameters indicating the number of instruction cycles. The simulator insures the serialisation of processing delays. Additionally, there is some overhead modeled in task switching between operations.

The CPU sends requests to the CAP, such as for a remote segment fault. The CAP then builds the appropriate control packet and passes this to the CMP. The CAP also decodes incoming control packets from the CMP and takes the appropriate action. This may involve interrupting the CPU, for example when a link fault is required in response to a get-segment control packet received.

CMM. The method for providing direct access between host memory and the network interface without any store-and-forward hops is through the use of a special multi-ported *communications memory module* (CMM), similar in concept to VRAM (video-RAM) design. The CMM has a conventional random access port which appears like any other memory bank to the processor–memory interconnect, out of which the CPU may execute code and access data. The other ports are high speed sequential access interfaces to the CMP (transmit and receive), and must operate at a rate of the VHSI optical links scaled by the CMP datapath width.

A delay models the access time to start up the read from the sequential output port. The data packet length field is inserted in the packet for use by simulator probes in computing throughput. Individual read operations take place for a sequence of packets in a given page, given the page base address. A page transmission request from the CMP `Rate Control` results in a page burst of packets at full VHSI link rate.

Incoming data packets are sinked, since it is the processing of the CMP, and not the actual destination

of the data that is of concern for this paper.

CMP. The goals for the design of the CMP include the ability to perform critical path functions in real time with no packet buffering and to incorporate the necessary function in VLSI. This may be realised by organising the CMP as a dynamically reconfigurable pipeline, based on the ALTP type and options for a particular connection. The pipeline organisation allows packets to be processed at the VHSI data rate.

It is important to note that the simulation model is constructed to reflect the actual hardware design of the Axon CMP. Since a major thrust of this research is to investigate the implementation of critical path function in hardware, simulation blocks are chosen carefully to represent function easily implementable in VLSI. Thus, high levels of simulation abstraction are not used within the CMP model. For the sake of brevity, details of the CMP simulation are not presented here but can be found in [17].

Simulation Results

Axon simulation results are divided into four groups: overall system behavior, rate control, error control, and functional partitioning. Two example results are presented in this section from rate and error control. A complete set of simulation results can be found in [17]. A metric of primary concern is the time that a process is blocked waiting for the return of a data segment. The time interval between the process referencing the segment and its complete return to the local host is defined as T_S. More important is the time a process is *blocked*. This is measured by the interval between reference and the return of the *first* page in the segment, since execution can begin when this page is marked present; this is defined as T_s.

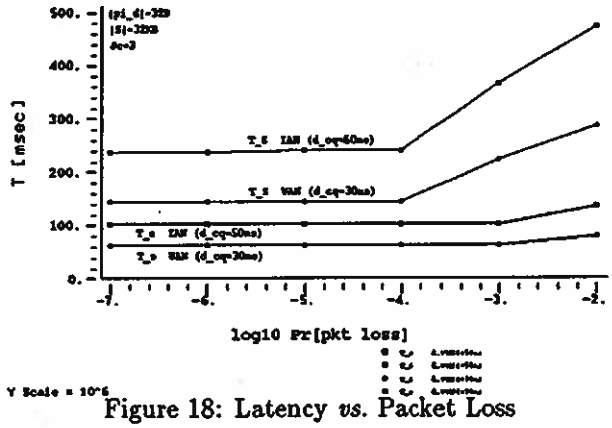


Figure 18: Latency *vs.* Packet Loss

A number of simulation experiments have been run to indicate the performance of the Axon system with respect to errors. An example is shown in Figure 18. In this case 3 connections on each host share 90% of a 1Gbps link. The packet size is 32 bytes of data with 16 bytes of header. The page size is 1 Kbyte, with an average segment size of 8 pages. Results are shown for both a WAN latency of 30ms and an internetwork (IAN) latency of 50ms.

Packets are lost with a probability varying from 10^{-8} to 10^{-2}. Note that artificially high error rates with very small packet sizes are simulated to push the error control mechanisms to extreme limits. The VHSI environment is expected to be much more reliable than this.

The latency performance is quite flat, even up to extremely high error rates, which can be attributed to the use of selective retransmission. The T_S (upper) curves indicate the blocking that would occur

Table 1: Network Throughput by Connection [Mbps]

$\|s\|$	1		32		1K
$\|\pi\|$	1040B		144B		48B
$\#c$	10	100	10	100	10
min	3.92	3.93	61.5	9.99	94.8
max	3.93	3.93	73.0	10.38	99.5
mean	3.93	3.93	70.5	10.23	96.8
ideal	3.93	3.93	70.8	9.96	96.8
total	39.30	392.00	708.0	996.00	968.0

if a process would have to wait for an entire segment to arrive (as is the case with common general purpose protocols). The T_s (lower) curves indicate the advantage of knowledge by ALTP-OT of the page structure of segments, allowing processes to resume execution more quickly (the height of T_s related to the page size). Note that while the segments simulated are rather small, T_s is independent of segment size. This is the case since T_s is only dependent on the correct arrival of the first page in the segment, and retransmitted pages preempt the primary segment transmission in progress. As segment sizes increase, so will the difference between T_s and T_S, and thus the performance benefits of ALTP-OT.

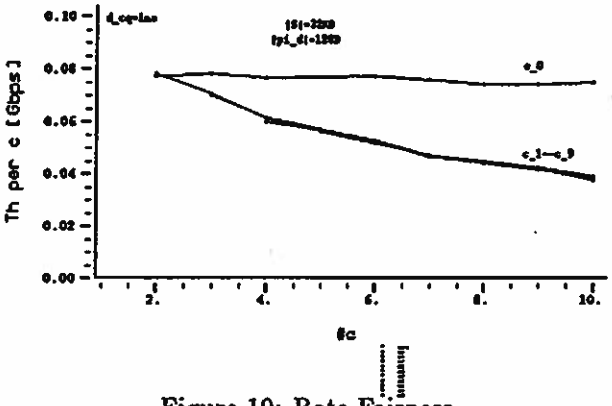


Figure 19: Rate Fairness

For a rate control scheme to be fair, all processes should receive the specified share of bandwidth, and in the case where rate specifications are identical across processes none should receive significantly different service. Figure 19 compares the throughput across connections as they are added. A single connection c_0 was given a peak bandwidth of $\lambda_p = 0.5$Gbps with additional connections evenly sharing an additional 0.5Gbps as they were added. The curve for c_0 is flat as desired, and the curves for the other connections decrease and track one another closely.

Table 1 indicates the network throughput *per* connection for various combinations of packet size $|\pi|$ and segment size $|s|$ (with a 1KB page size), for 10 and 100 active connections *per* processor, indicating close correspondence across connections. Note that since the application is subjected to the rate control mechanism and has no direct access to flow control once the requested rate has been granted, a firewall is established preventing applications from harming one another in terms of bandwidth utilised. Simulations were also run verifying the insensitivity to individual connections as the error rates on other connections were driven extremely high.

Remote Visualization

VISUALIZATION OF CELLS IN BIOLOGICAL ORGANISMS

Christos Papadopoulos, Guru Parulkar

In collaboration with scientists in the department of biology, we are developing a distributed visualization scheme for the display of cells of biological organisms in 3D, in particular, the cellular slime mold which is an amoeba like organism. This project has been motivated by the following two observations:

- Very little is understood about the real communication requirements for distributed computations on wide area networks, despite the fact that such understanding is necessary for design of efficient protocols to support these computations.
- Although pipelining is a simple special case of general distributed computation, it is readily applicable to many televisualization applications; and televisualization is a very important and challenging class of network applications because it is intensive both in computation and communication.

  Thus, the goal of this project is to better understand the remote visualization applications and to characterize the performance bottlenecks of the protocols, if any, for this class of applications.

The amoebae of the organism being studied undergo a very interesting development as they starve: first, they aggregate in a hemispherical mount of about 10^5 cells; the mount is soon transformed into an upright finger called a slug; then, the slug topples over and moves as a unit before eventually standing upright again to differentiate into two parts, the stalk and the spores. This last process of differentiation is the focal point of the biological research.

The significance of this study lies in the promise that insight can be gained on how global shape changes are produced by coordinated movements of cells in a developing organism. An important question is how cells in different parts of the mount are determined to become a part of the stalk or the spores. Answers to this question can lead to better understanding of the development of any embryo. One of the visions of this project is that understanding of embryo development may lead to the ability to control regeneration of damaged organs in the body, by simulating the conditions of growth. For such biological applications, we have found volume visualization techniques to be very useful. In our current volume visualization scheme there are four major steps: data collection, thresholding, rendering and display. They are described in the following paragraphs.

Data is collected using a computer-controlled microscope used for time lapse 3D imaging. 3D images are collected in the form of 2D slices obtained by rapidly stepping through focal planes spanning the specimen. The whole data collection process takes about 2 hours, collecting a 3D image every 2 minutes, for a total of 60 images. Typical image sizes are $256 \times 256 \times 20$ at 2 bytes per voxel, giving a total data size of about 157 Mbytes. Each 2D slice collected by this method is blurred by out of focus light from the surrounding slices. Removal of the blurring is accomplished by thresholding the data at an appropriate value obtained from the data histogram. Rendering of the cleaned up data is accomplished using a Simulated Fluorescence Process (SFP) algorithm. This algorithm closely resembles a fluorescence process in that it simulates the flow of excitation light through the 3D data and then creates a projection by gathering the fluorescence from each individual voxel. Finally, the projected images are displayed using the X-window system. Each image is saved at the display stage, the researcher can choose to view them individually or as an animation sequence.

The four steps described above are naturally implemented as a pipeline of separate modules. While the raw data is collected and stored on a lab computer, the data viewing needs to be on the biologist's workstation remote from the lab. Instead of either doing all the computation on the lab computer and

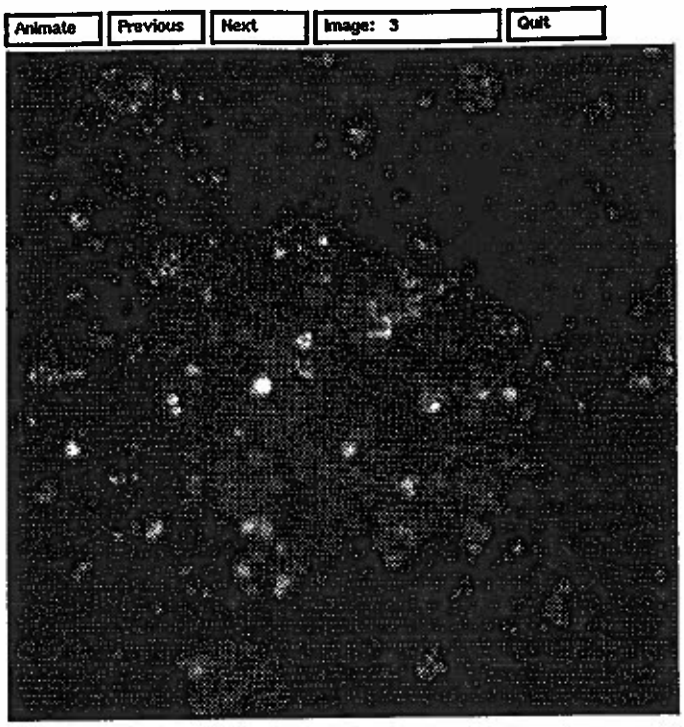
shipping images to the workstation for display, or shipping the raw data to the workstation first and doing all computation there, we assign each stage to a different machine so that pipelined parallelism can be achieved. Each of the modules is designed such that it can run on different computers to better utilize the computing resources. Communication between modules is through Unix IPC socket interface on top of TCP/IP over our campus network (10 Mbps Ethernet). Each module consists of three processes: reading, computing and writing. These processes communicate locally using shared memory. This allows communication to proceed independent of computation, better utilizing the local computing resources.

Figure 20 shows the 3D organism as seen by a user from the microscope. Clearly, it is very difficult to trace various cells and their trajectories using such images. Figure 21 shows the processed image which is quite good for identifying cells, and an animation of a sequence of such images is also found useful for studying the cell trajectories.

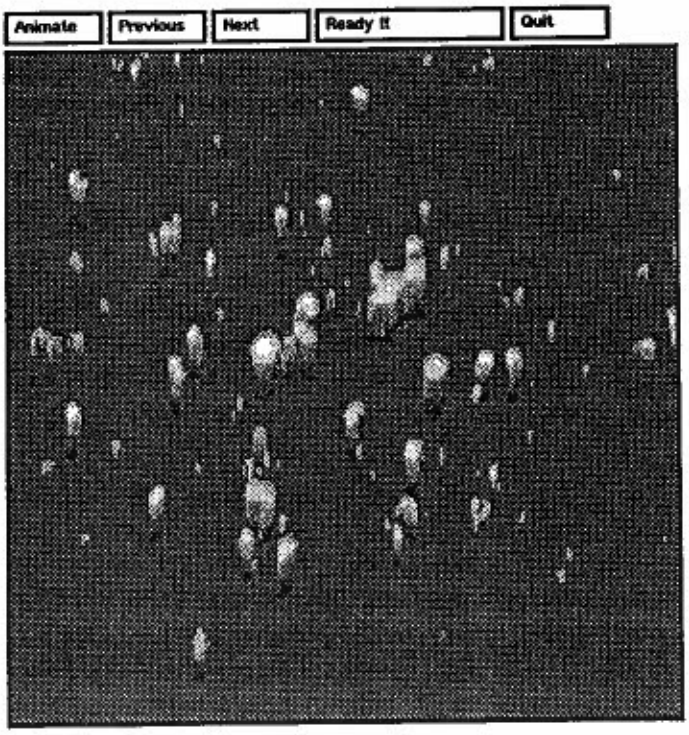
Initial experience with the implementation does show that there is speedup when using the pipeline. The speedup can be attributed to two factors: parallelism resulting from pipeline operation and the reduction in disk swapping due to availability of increased memory at different stages and reduced memory requirement at any given stage of the pipeline.

If the work load is carefully partitioned, communication becomes the bottleneck and it will get worse as the data resolution increases. The communication overhead includes moving the data between user and system spaces, protocol processing, and data transmission (data transmitting time + propagation delay). The transmitting time is expected to decrease as faster networks become available, but the propagation delay will remain the same and become more significant overhead. The protocol processing delay will also increase since transmitted data size is also expected to increase. Locating bottlenecks in the current IPC paradigm, (*i.e.*, the UNIX socket interface on top of TCP/IP) may provide insight into what problems need to be addressed in the design of future protocols to support pipelining. To identify those problem areas, we plan to run the example application for different data sizes and pipeline configurations with a minimal number of probes strategically placed in the networking code. This will help us acquire a better understanding of the different processing delays in the individual protocol layers and characterize them as either protocol or operating system dependent. Currently such experimentation is in progress.

Figure 20: An unprocessed image



Raw Image



Processed Image

Interprocess communication for Remote Visualization

Larry Gong, Guru Parulkar

During the past year, we have initiated a research project to develop an efficient interprocess communication (IPC) facility at the transport level that is targeted for distributed pipelines. A distributed pipeline is a pipeline whose stage processors are physically distributed across (inter)networks. We believe that such a pipeline can be effective in supporting a class of remote visualization applications. Therefore we use pipelined televisualization (PTV) as the main application model in our IPC research. This recognition is also apparent in the design of apE and AVS systems[4]. A typical configuration of a distributed pipeline for televisualization is shown in Figure 22. The first stage of the pipeline is usually a computer accessing a mass storage device or a supercomputer generating a large amount of data from simulation and/or computation. There are intermediate stages that perform various data transformations. The last stage performs graphics rendering and display, and accepts user input to steer the simulation.

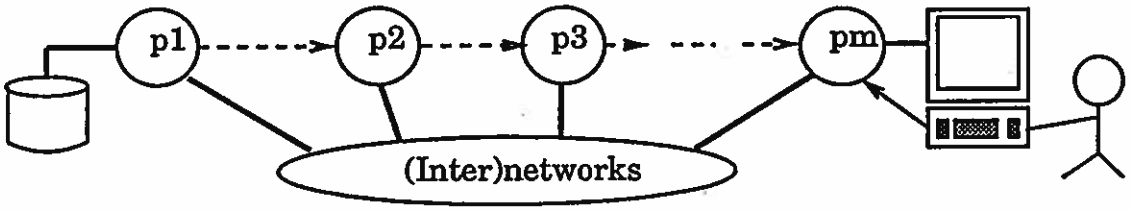


Figure 22: A Typical Televisualization Pipeline Configuration

In a distributed pipeline such as the one described above, interstage communication incurs significant delay due to the physical distribution of pipeline stages. The slowest stage becomes the bottleneck and limits pipeline speed. In order to achieve efficient pipelining, the interstage communication has to satisfy a number of conditions:

- A visualization application typically involves a large number of data segments of considerable size. These segments have to be streamed through the pipeline with minimum delay to allow overlapped processing.
- Computation and communication speed should be well balanced for optimal utilization of both resources.
- An application should be able to specify its error tolerance requirements to the IPC mechanism and the IPC mechanism should enforce it only to the degree that is necessary to satisfy the requirements, thus avoiding any unnecessary error control overhead.
- Buffer overflow in the pipeline should be avoided because loss of partially processed data due to overflow may require restarting the pipeline from an earlier stage, thus wasting computing cycles and introducing unnecessary (and unacceptable) delay.

Existing IPC mechanisms and underlying communication protocols do not have the functionality to meet these demands. Significant progress is being made in the development of high-speed packet switching

[4]apE is a visualization software from the Ohio Supercomputer Center. AVS is a visualization software from the Stardent company. Both systems allow a user to create a visualization application in the form of a pipeline using a graphical editor.

networks, internetworks, and host-network interfaces. This development sets the stage for the development of IPC facilities that are necessary for converting the high bandwidth of networks into high performance for distributed applications.

To provide efficient interstage communication for a distributed pipeline, we have proposed an IPC solution which consists of three important parts: a segment streaming IPC paradigm, a two-level flow control method, and an application-oriented error control method. We summarize each of the components in the following paragraphs. A more detailed description is available in [9].

Segment Streaming. The new IPC paradigm, called *segment streaming*, is specifically proposed for supporting exchange of a stream of segments among processes with high bandwidth and low latency. Segment streaming is provided through two transport level operations: **send-stream** and **get-stream**. They are invoked by applications by making the corresponding system calls. For a distributed pipeline, the set of data items to be processed is defined as a segment group with each data item being a segment. A network connection is established for each segment stream upon the single request (get-stream or send-stream) by the application process. Each segment will be transmitted when ready across the connection without the latency of request or setup. Therefore, it supports segment prefetching and allows overlapping between the visualization computation and the communication processing.

Two Stege Flow Control. With our distributed asynchronous pipeline, flow control becomes an important issue. Since each pipeline stage is usually a multiprogrammed system, change of multiprogrammed tasks will cause a corresponding change in the effective speed of the pipeline process. Moreover, underlying network connections can only provide statistical guarantees for bandwidths but not absolute rates. Without an effective flow control, buffer overflow may occur at bottleneck stages. Any overflow can in turn cause significant performance degradation due to the linear data dependency among the pipeline stages. Our objective for flow control is to maintain the flow rate of the pipeline at that of the bottleneck stage and avoid any overflow. To achieve this objective, we use a two-level flow control method. The two-level method consists of a simple window flow control on top of the rate control mechanism. The simple window control only serves the interstage (end-to-end) flow control purpose and it is not overloaded with error control and congestion control as in TCP and other transport protocols. The rate control mechanism enforces data transmission and reception rates which are derived from the flow requirement of the simple window control and agreed upon by the underlying network at the time of connection setup.

Application-Oriented Error Control. Error control has two aspects: detection of packet corruption, loss, and duplication; and compensation for these error conditions. Traditionally, error control either corrects all error conditions to achieve 100% reliability as in TCP, or does nothing as in UDP (User Datagram Protocol). We argue that the goal of the error control should be to satisfy the error tolerance of applications with minimum error control overhead. There are several observations that support this argument.

- With the combination of large bandwidth-delay product of very high speed internetworks and demanding PTV applications, the delay for rigorous error control becomes a significant overhead and may not be acceptable for many applications.
- Visualization applications have very diverse reliability requirements from the underlying transport protocol. A visualization application may deal with different types of data (e.g. video frames, sensory data, text, and model data) at different stages of processing. The error tolerances for these data types spread over the whole spectrum from those of model data transfer to video distribution. A typical data transfer cannot tolerate any errors and thus requires recovery from every packet loss or corruption. A typical video distribution, on the other hand, requires no error recovery (as long

as the error rate is reasonably low) because previous and subsequent frames provide a context that compensates for the loss in the current frame. Also, transmission of real time data will have more tolerance for corruption than for time delay. Therefore, it is desirable for the transport protocol to provide variable grades of error recovery to avoid unnecessary control overhead.

We address this problem by using an application-oriented error control strategy which has several important aspects to it:

Detection at receiver. The loss detection is located at the receiving end of a connection. This is consistent with our objective to allow receivers to make application-dependent retransmission decisions.

Selective retransmission. In the case of errors, a selective retransmission scheme is used. A receiver saves correctly received data packets and requests retransmissions for only the missing or corrupted packets. Therefore, application processes can get access to partially received data without much delay, and extraneous retransmissions (as in a cumulative retransmission scheme) are avoided. While the unit of retransmission is a packet, we perform loss detection and send acknowledgment on a per segment basis, thus reducing the control overhead. Retransmissions preempt the primary data stream, which is consistent with the assumption that the application expects segments in the sequential order.

Application error tolerance. Since the main objective of the application-oriented error control is to satisfy an application's requirement using minimum retransmission, an accurate characterization of application error tolerance is necessary to determine the optimal control strategy. Application error tolerance has three key components: packet loss rate, burstiness of the loss, and end-to-end delay. They are specified as the length of a "window" W over which error requirement is enforced, E the maximum number of packet losses tolerable over W, and B the maximum length of error burst tolerable. We denote the tolerance as a triplet (B,E,W) $(0 \leq B \leq E \leq W)$. By satisfying the application tolerance with minimum retransmission overhead, we automatically achieve the minimum end-to-end-delay possible for the given connection. An application is expected to specify its tolerance, and the error control mechanism uses this tolerance to do application-oriented selective retransmission.

When a televisualization pipeline is set up for an application, the specific requirement for error control is registered by the transport service. During the pipeline operation, error control mechanisms (at both the sending and receiving ends of a connection) will respond dynamically to various error conditions and correct them just to the degree required by the application. With this scheme, applications will have the flexibility to choose appropriate error control strategy to suit their purpose and to endure transport overhead for error control only if necessary.

We plan to create an experimental implementation of the IPC facility and implementations of typical visualization applications using the PTV model. See [9, 7] for more details.

A Recurrence Model for Asynchronous Pipeline Analysis

Fengmin Gong, Zubin Dittia, Guru Parulkar

Motivation. We have proposed pipelining across high-speed networks as an efficient televisualization model, called pipelined televisualization (PTV) [9]. This has led us to develop tools for the analysis of asynchronous pipelines. In general, a pipeline can either operate in lock-step fashion by a global clock or operate by having each stage processor synchronize only with its neighbors using a handshaking protocol. In the first case, the pipeline is called a *synchronous* pipeline; the second case corresponds to an *asynchronous* pipeline. Issues surrounding the synchronous pipeline design are well understood. Synchronous pipelines are generally simpler to design, and they can be very efficient if the computation task can be partitioned into stages of nearly equal processing time. However, asynchronous pipelines are absolutely necessary when pipeline stages are physically distributed or have varying processing delays, for example, in the case of a pipelined televisualization.

There are still two main obstacles to the wide use of asynchronous pipeline. The complexity of asynchronous circuitry design is still high; and characterization of the asynchronous behavior has been a difficult task leading to limited understanding of asynchronous pipelines. We have developed a simple yet accurate method for analysis of asynchronous pipelines using recurrence relations. This development was done in two steps: (1) derivation and experimentation for pipelines of linear topology, and (2) derivation and verification for pipelines with general DAG (directed acyclic graph) topology. This note will present the recurrence relations for only the DAG topology and example verification results. For details refer to [8].

Notation. We view the set of dependencies among processors of the DAG as a set of dependency chains. We use a two dimensional indexing scheme for the identification of processors. The processors are partitioned into stages as follows. Let m be the total number of stages, which is defined as the number of processors on the longest dependency chain. Each source processor is labeled as a stage 1 processor and a sink processor as a stage m processor. For any internal processor (i.e., not a source or sink processor), if i is the maximum distance from the processor to any sink processor, this processor is labeled as a stage $(m-i)$ processor. If w_i is the number of processors at stage i, we label these processors by tuples $(i,1),(i,2),\cdots,(i,w_i)$. Also, we use $pred(i,j)$ to denote the set of predecessors for processor (i,j) and $succ(i,j)$ to denote the set of successors to processor (i,j).

Now we introduce recurrence variables. Let s_{ijk} be the time at which processing of data item k begins at processor (i,j). Let d_{ijk} denote the processing delay for data item k at processor (i,j), and let f_{ijk} denote the finish time of data item k at processor (i,j). Finally, l_{ijk} represents the time at which data item k leaves processor (i,j) and arrives at all processors in $succ(i,j)$. We assume that a processor begins execution when all input data items have arrived and can forward its output data item to its successors only when they are ready to receive the data item. A set of recurrence relations in these variables can be derived and organized as initial conditions, boundary conditions and recurrence body.

Initial Conditions. Since the pipeline is empty initially, the first data item ($k=1$) can be processed at a given processor as soon as all the corresponding input data items are received from predecessors; and the resulting output data item can move to successors immediately after its processing is finished. The first data item is available to the source processors at time 0. Therefore, the following initial conditions hold for the pipeline:

$$s_{1j1} = 0$$
$$f_{1j1} = d_{1j1}$$
$$l_{1j1} = f_{1j1}$$

$$s_{ij1} = \max_{(i',j') \in pred(i,j)} (l_{i'j'1})$$

$$f_{ij1} = s_{ij1} + d_{ij1}$$

$$l_{ij1} = f_{ij1}$$

where:

$$2 \leq i \leq m$$

$$1 \leq j \leq w_i$$

Boundary Conditions. Operation of source and sink processors is also special. First, source processors never have to wait for input data items because they are assumed to have all data items to begin with. Sink processors never have to wait to forward an output data item because they consume their own output. These correspond to the boundary conditions of the pipeline:

$$s_{1jk} = l_{1jk-1}$$

$$f_{1jk} = s_{1jk} + d_{1jk}$$

$$l_{1jk} = \max_{(i',j') \in succ(1,j)} (l_{i'j'k-1})$$

$$s_{mjk} = \max_{(i',j') \in pred(m,j)} (l_{i'j'k})$$

$$f_{mjk} = s_{mjk} + d_{mjk}$$

$$l_{mjk} = f_{mjk}$$

where:

$$1 \leq j \leq w_i$$

$$2 \leq k \leq n$$

Recurrence Body. Processors of internal stages ($2 \leq i \leq m-1$) have to wait for input data items from all predecessors to arrive before the processing can start; once the processing is complete, the output data item has to be held until all successors are ready to receive it. This logic is formulated into the recurrence body:

$$s_{ijk} = \max_{(i',j') \in pred(i,j)} (l_{i'j'k})$$

$$f_{ijk} = s_{ijk} + d_{ijk}$$

$$l_{ijk} = \max[\max_{(i',j') \in succ(i,j)} (l_{i'j'k-1}), f_{ijk}]$$

where:

$$2 \leq i \leq m-1$$

$$1 \leq j \leq w_i$$

$$2 \leq k \leq n$$

Given a real-life asynchronous pipeline, we first apply two simple transformations (see [8] for details) that models the effect of interstage communication overhead and data buffers. We then apply the recurrence relations to the resulting pipeline. The evaluation of the recurrence relations provides a complete trace of the start time, finish time, and departure time for every data item at each processor.

Results. The recurrence model has been implemented as a single C program. Results of two experiments have been reported that help verify the correctness of recurrence relations. These experiments

involved a 3-stage linear pipeline and a 3-stage pipeline with two processors at the second stage, referred to as a parallel pipeline. These pipelines are evaluated using the recurrence model as well as a discrete event simulation using BONeS, a network simulator from COMDISCO Systems Inc.. In this study, all stages (computation/communication) were assumed to have a normally distributed processing latency with mean equal to 50. The standard deviation was varied from 0 to 50 with step size 10 and four buffer sizes (0,1,10,20) were examined. Buffer sizes were always the same across stages. In all cases, both simulation and the recurrence model were run with more than 9000 data items to ensure statistical significance for the results. Figures 23 and 24 presents respectively the results for the linear and the parallel pipelines from the recurrence model. The same set of results were also obtained with the BONeS simulation model, and they match almost exactly with results of the recurrence model. Hence, the simulation results are omitted from the plot. It should be noted that the metric used in these plots is the average cycle time $AC^+ = \frac{t_{mn} - t_{m1}}{n-1}$ $(n > 1)$.

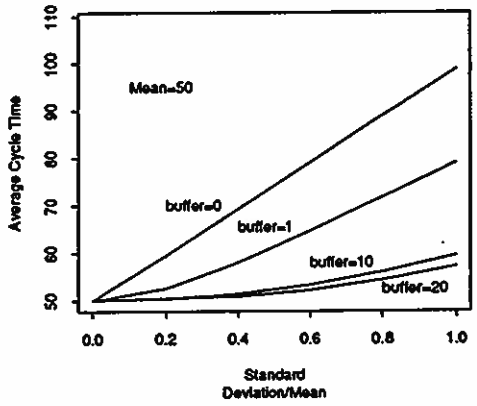


Figure 23: Linear Pipeline

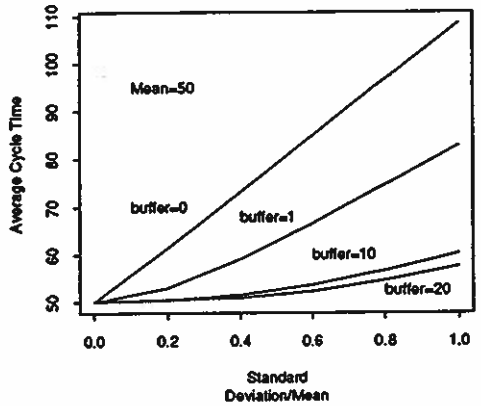


Figure 24: Parallel Pipeline

First we examine the linear pipeline results shown in Figure 23. When processing delay for all stations

is constant at 50 (i.e., standard deviation = 0), the average cycle time is 50 regardless of buffer size. As the standard deviation is increased, and there is no buffering, the average cycle time shows a linear increase; with buffer size 20, the increase in average cycle time is very little. Thus, the family of curves (for buffer sizes 0,1,10,20) shows that the increase of average cycle time with the increase in standard deviation is slower with larger buffers. This result fully conforms to our expectation of the effects of buffering.

In the case of a parallel pipeline, Figure 24 shows a similar trend, that is, the average cycle time increases linearly with increase in standard deviation when buffer size is 0. As larger buffers are used, average cycle increases slowly with increasing standard deviation. Comparison of Figures 23 and 24 show that for the same processing delay and buffer size, a parallel pipeline has a larger average cycle time than that of a linear pipeline except when standard deviation is zero. This is correct, because in the case of parallel pipelines, the waiting resulting from splitting and merging of stages contributes to a longer average cycle time. This kind of waiting is due to synchronization among parallel data streams, and it cannot always be eliminated by using extra buffers.

References

[1] Akhtar, S., "Congestion Control in a Fast Packet Switching Network," *MS Thesis, Department of Electrical Engineering, Washington University in St. Louis,* December 1987.

[2] Anderson, James A., "Persistent Connections in High Speed Internets," MS Thesis, Department of Computer Science, Washington University. 1991.

[3] Anderson, Jim, Zubin Dittia and Guru Parulkar. "Persistent Connections in High Speed Internets," Proceedings of the IEEE Globecomm'91, December 1991. Also, Technical Report WUCS-91-13, Department of Computer Science, Washington University.

[4] Cranor, Charles, D., "An Implementation Model for Connection-oriented Internet Protocols," MS Thesis, Department of Computer Science, Washington University. 1992.

[5] *Block Oriented Network Simulator™ (BONeS™) User's Guide*, version 1.5, Comdisco Systems, Inc., Nov. 1990.

[6] Forgie, J., "ST - A Proposed Internet Stream Protocol," IEN 119, MIT Lincoln Laboratory, 7 September 1979.

[7] Gong, F. and G. Parulkar, "A Novel Interprocess Communication Paradigm for Pipelined Televisualization," submitted for publication.

[8] Gong, Fengmin, Zubin Dittia and Guru Parulkar, " A Recurrence Model for Asynchronous Pipeline Performance Analysis", Technical Report WUCS-91-14, Department of Computer Science, Washington University (also submitted for publication).

[9] Gong, Fengmin "Segment Streaming for Efficient Pipelined Televisualization," Technical Report WUCS-90-39, Department of Computer Science, Washington University.

[10] Kapoor, S., "Design and Simulation of an ATM-FDDI Gateway," MS Thesis, Department of Electrical Engineering, Washington University.

[11] Kapoor, Sanjay and Guru Parulkar. "Design of an ATM-FDDI Gateway," Proceedings of the ACM SIGCOMM'91. Also, Technical Report WUCS-91-11, Department of Computer Science, Washington University.

[12] Mazraani, Tony and Guru Parulkar. "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol," *Proceedings of IEEE INFOCOM 90*, 6/90.

[13] Mazraani, Tony. "High Speed Internet Protocols and Resource Management in the Ethernet," Washington University Computer Science Department, MS thesis, 8/90.

[14] Parulkar, G.M. (editor), "The Connection-oriented Internetworking: Collaboration Plan," Technical Note, Department of Computer Science, Washington University in St. Louis, 1989.

[15] Parulkar, Guru and Jonathan Turner. "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment," *IEEE Network* 3/90.

[16] Parulkar, Guru. "The Next Generation of Internetworking," ACM SIGCOMM, *Computer Communication Review*, January 90.

[17] Sterbenz, James P.G., "Axon: A Host-Network Interface Architecture for Gigabit Communications," D.Sc. Thesis, Department of Computer Science, Washington University in St. Louis. December 1991.

[18] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon: Network Virtual Storage Design", ACM SIGCOMM, *Computer Communication Review*, Vol.20 #2, ACM, New York, April 1990.

[19] Sterbenz, James and Guru Parulkar. "AXON: a High Speed Communication Architecture for Distributed Applications," *Proceedings of IEEE INFOCOM 90*, 6/90.

[20] Sterbenz, James P.G. and G.M. Parulkar, "Axon Network Virtual Storage for High Performance Distributed Applications", *Proceedings of ICDCS*, 6/90.

[21] Sterbenz, James P.G. and G.M. Parulkar, "Axon: Application-Oriented Lightweight Transport Protocol Design", *Proceedings of ICCC*, 11/90.

[22] Sterbenz, James P.G. and G.M. Parulkar, "Axon: Host-Network Interface Architecture for Gbps Communication," in *Protocols for High Speed Networks, II*, Marjory J. Johnson ed., Elsevier Science Publishers.

[23] Topolcic, C., "Experimental Internet Stream Protocol: Version 2 (ST-II)," RFC-1190, October 1990.

[24] Zhang, Lixia, "A New Architecture for Packet Switching Network Protocols," Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, July 1989.

[25] Zhang, Lixia, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," ACM Transactions on Computer Systems, vol 9, no 2, pp. 101-124, May 1991.