Report Number: WUCS-89-35

1989-08-01

Host-Network Interface Architecture for Gigabit Communications

James P. G. Sterbenz

There are two complementary trends in the computer and communications fields. Increasing processor power and memory availability allow more demanding applications, such as scientific visualizations and imaging. Advances in network performance and functionality have the potential for supporting applications requiring high bandwidth communications. However, the bottleneck is increasingly in the host-network interface, and thus the ability to deliver high performance communications capability to applications has not kept up with the advance in computer and network speed. We have proposed a new architecture that meets these challenges, called Axon. The Axon thesis is that an essential requirement for the support... **Read complete abstract on page 2.**

Host-Network Interface Architecture for Gigabit Communications

James P. G. Sterbenz

Complete Abstract:

There are two complementary trends in the computer and communications fields. Increasing processor power and memory availability allow more demanding applications, such as scientific visualizations and imaging. Advances in network performance and functionality have the potential for supporting applications requiring high bandwidth communications. However, the bottleneck is increasingly in the host-network interface, and thus the ability to deliver high performance communications capability to applications has not kept up with the advance in computer and network speed. We have proposed a new architecture that meets these challenges, called Axon. The Axon thesis is that an essential requirement for the support of high performance distributed IPC is to provide a direct channel for object transfer between the communicating processes. Thus, this research centers on how to create an end-to-end pipeline to deliver this high bandwidth to applications. The goals are to develop a suitable architecture, determine the key issues and tradeoffs, and evaluate them as data rates scale beyond 1 Gbps. Novel aspects of this research include: an integrated design of hardware, operating systems, and communications protocols, stressing both performance and the required functionality for demanding applications; the proper division of hardware and software function; and reorganization of end-to-end protocols to take advantage of the increased functionality of the emerging high speed internetworks.

HOST-NETWORK INTERFACE ARCHITECTURE FOR GIGABIT COMMUNICATIONS
[dissertation proposal]

James P. G. Sterbenz

WUCS-89-35

August 1989

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

ABSTRACT

There are two complementary trends in the computer and communications fields. Increasing processor power and memory availability allow more demanding applications, such as scientific visualization and imaging. Advances in network performance and functionality have the potential for supporting applications requiring high bandwidth communications. However, the bottleneck is increasingly in the host-network interface, and thus the ability to deliver high performance communication capability to applications has not kept up with the advances in computer and network speed.

We have proposed a new architecture that meets these challenges, called Axon. The *Axon thesis* is that an essential requirement for the support of high performance distributed IPC is to provide a direct channel for object transfer between the communicating processes. Thus, this research centers on how to create an end-to-end pipeline to deliver this high bandwidth to applications. The goals are to develop a suitable architecture, determine the key issues and tradeoffs, and evaluate them as data rates scale beyond 1 Gbps.

Novel aspects of this research include: an integrated design of hardware, operating systems, and communications protocols, stressing both performance and the required functionality for demanding applications; the proper division of hardware and software function; and reorganization of end-to-end protocols to take advantage of the increased functionality of the emerging high speed internetworks.

References updated April 5, 1990.

James Sterbenz is on leave from IBM Corporation at Washington University in St. Louis.

HOST–NETWORK INTERFACE ARCHITECTURE FOR GIGABIT COMMUNICATIONS

[dissertation proposal]

James P. G. Sterbenz
+1 314 726 4203 *jps@wucs1.wustl.edu*

1. Introduction

The ongoing research in the computer communication and telecommunications fields suggests two emerging trends which are complementary to one another. First, as time goes on we will continue to witness communication networks which can support increasingly high data rates. For example, networks with data rates of a few hundred Mbps are being prototyped, and networks with data rates of a few Gbps are being planned. The future generation of internetwork, consisting of these high speed subnetworks, will be referred to as the *very high speed internetwork* (VHSI) [Pa89]. Second, a diverse application set having differing bandwidth, latency, and reliability requirements will have to be supported on the VHSI communication substrate. For example, video distribution, computer imaging, distributed scientific computation and visualisation, distributed file and procedure access, and multimedia conferencing are all target applications. These trends pose a number of new challenges and opportunities to the researchers in the field. One such challenge is how to support high performance interprocess communication (IPC) in this environment.

We argue that the existing approach of supporting IPC cannot deliver the underlying high bandwidth to newer and demanding applications because of a number of reasons: there is a lack of integration among host architecture, operating system, and communication protocols; there are numerous performance bottlenecks in the existing end-to-end protocols; and the shared memory paradigm is not well supported in a loosely coupled or network environment.

We propose a new communication architecture for distributed systems called **Axon**. The primary goal of the Axon architecture is to support a high performance data path delivering VHSI bandwidth directly to applications. The significant features of Axon are:

- An integrated design of host and network interface architecture, operating systems, and communication protocols. To serve the needs of high performance IPC, it is essential that the design and implementation of all of these areas are coordinated to minimise the overhead in their interaction.
- A network virtual storage facility which includes support for virtual shared memory on loosely coupled systems. This is accomplished by extending segmented, paged virtual storage management to allow segments to be addressed anywhere within the VHSI.

- A high performance, lightweight object transport facility which can be used by both message passing and shared memory mechanisms. Flow control is rate-based, and uses an underlying connection oriented internet substrate. Error control is decoupled from flow control, and to a great extent from end-to-end latencies, and specifically tailored to object transport.
- A pipelined network interface which can provide a path directly between the VHSI and host memory. This is accomplished by implementing critical protocol and operating system assist functions in hardware.

The *Axon thesis*[†] is that an essential requirement for the support of high performance distributed IPC is to provide a direct channel for object transfer between the communicating processes. Thus, a pipeline is set up that directly connects the memory addressed by the applications, without intermediate buffering, data copying, or transfer of control. The proposed research consists of of the design of the Axon *architecture* to satisfy the Axon *thesis*, and of exploring the issues that arise in maintaining an application-to-application pipeline through the VHSI as data rates scale upward.

1.1. Organisation of this proposal

This proposal indicates weaknesses of current communications models applied to the VHSI environment, and describes a more suitable model for high performance communication. The Axon architecture is described, and a number of research issues are identified along with methodology to address them.

This proposal is organised as follows: Section 2 gives motivation for the investigation of high performance IPC, and sumarises problems with current implementations. Section 3 provides background for further research by describing the Axon architecture at a high level. Section 4 discusses the issues that will be pursued in the proposed dissertation, narrowing the focus from the initial Axon research and design. The central theme is based on a high performance end-to-end IPC pipeline between applications. The current status of the research is indicated, along with a plan to complete the dissertation. Section 5 describes other related current and previous work, and Section 6 is the conclusion.

More detail on the Axon architecture can be found in the technical reports [StPa89a, StPa89b, StPa89c, St90]. Based on the information in the Axon overview section of this proposal (§3), details on NVS can be gotten from [StPa89b] §3–5, on ALTP-OT from [StPa89c] §3–4, and on the host and network interface architecture from [St90] §4–6. An example of an Axon segment transfer is presented in [StPa89a] §3.6.

An overview of the Axon architecture is presented in [StPa89a]; there is substantial overlap in the motivation and Axon overview sections with this proposal. If [StPa89a] is read first, §2.3 and §4 of this proposal may be read in isolation without significant loss of content.

2. Motivation

We are witnessing a number of revolutionary changes to the communication substrate. At least a few orders of magnitude increase in communication bandwidth is imminent, giving rates on the order of 1–10 Gbps. This is also accompanied with improved functionality and lower error rates of the communication substrate. For example, emerging networks are designed to carry multiple

[†] *thesis* is used in the sense of a hypothesis which will be assumed to be true, on which the proposed research is based

types of traffic: video, voice, and data, and designed to make performance guarantees based on their characteristics [PaTu89].

The scientific computation environment has also come a long way. There are a growing number of scientists in all disciplines solving bigger and more complex problems using computers, some of which are accessed remotely across a network (*e.g.* [Fe84, Pe84, KaSm87, Ni89]). Thus, the application set for the VHSI communication substrate includes application classes such as computer imaging, distributed scientific computation and visualisation, large distributed file and procedure access, and multimedia conferencing.

2.1. Target environment

Figure 1 depicts what we view as a typical future scientific and engineering computation environment [McDe87, Pa87, RaLa87, Kl88]. At the heart of this is a high bandwidth communication substrate (VHSI) which can support communication at rates of at least 1 Gbps, and provide performance guarantees in terms of throughput, delay, packet loss rate, and packet sequencing. The VHSI provides access to a number of large mass storage facilities. These facilities store data and images obtained from computation, such as simulations, finite element analysis, and molecular modeling, as well as from real-time data acquisition, such as from satellite telemetry and medical scanning. User applications can use and cause the generation of parts of this data during their execution.

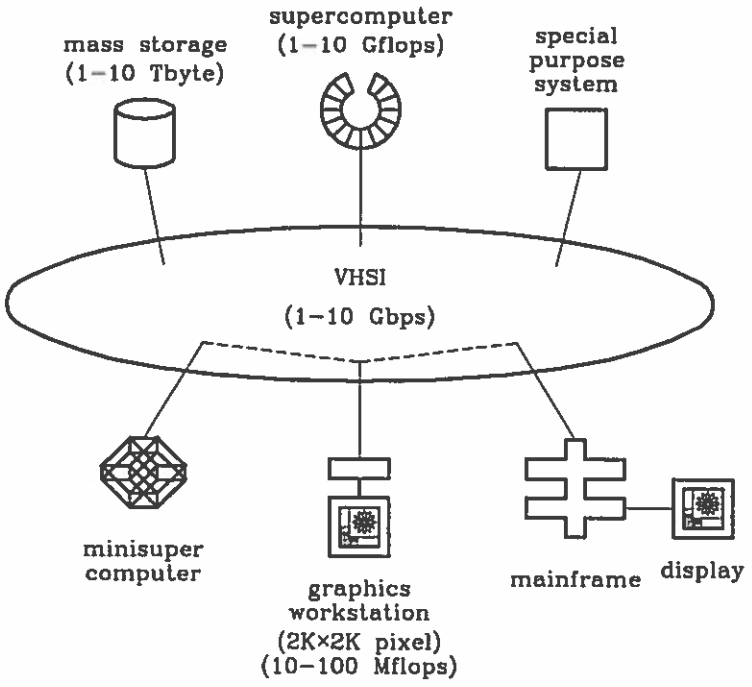

Figure 1: Target Environment

In this scenario, users have high performance graphics workstations with compute engines (such as the Ardent Titan [DiHa88] or Stellar GS1000/2000 [SpMo88, St88b]). These workstations include a bit mapped display (~2K×2K pixel), floating point processor(s) (10–100 Mflops), and a large fast memory (32-512 MB). In a local environment, the user will have access to modest parallel machines and mini-supercomputers (such as the Intel iPSC-2 [Ra85, In86a]) and to mainframe computers (such as a CDC Cyber 180/990 [CDC84a, CDC84b] and IBM 3090 [IBM86a, IBM88a]). The local

machines can be used to solve smaller problems or to perform trial runs of a bigger problem. Access to supercomputers (such as the Cray-2, ETA-10, or NEC SX [Hw87, Hw89]) and special purpose systems (such as simulation engines) will be provided across the VHSI.

An example computation might involve large modeling or simulation programs to be run on supercomputers and special purpose processors. Results may be accumulated on mass storage, or piped in a stream to the workstation as produced, allowing the progress to be viewed. The workstation takes raw output data, and performs visualisation computations to produce images. The local rendering of images allows for the user to examine the visualisation in real time, *e.g.* rotation of a 3-dimensional image, or variable speed playback of a time-varying animation. In another scenario, the user may want to view images generated directly by the supercomputer, or stored in an image database. Additionally, the user may wish to use the supercomputers in a time-shared mode, *e.g.* rerunning parts of simulations with differing parameters to render new images. This demands very low latency (ideally sub-second). The important points to note about this type of application environment are the following:

- An application is programmed as one distributed program which uses remote procedure calls, shared memory segments, streams of data, and/or message passing primitives for communication.
- Application communication needs are bursty, and the bursts require large amounts (megabytes) of data to be moved across the network.
- Applications require the interactive use of the resources across the network, to allow the control and modification of the computations, which requires the blocks of data to be delivered with low latency (sub-second). This coupled with the size of the data demands very high bandwidth ($\geq$1 Gbps).
- Both ends of communication may have to do processing of the data, allowing local processing of data and real-time image interaction, in addition to the remote data access and processing.
- Communication processing overhead must be minimised to provide low latencies at high bandwidth.
- As processor speed and network bandwidth increase, end-to-end latency is an increasingly dominant factor, affecting the performance of communicating processes and the policy and performance tradeoffs in system design.

2.2. Limitations of the existing model

These characteristics of communication pose a number of challenges to the designers of host interfaces. Over the past few years significant progress has been made in the fields of communications and computer architecture. Specifically, the communication networks are able to move increasingly large amounts data from one place to another at high bandwidths and with low latency. Similarly, computers can process data increasingly fast, and even support interactive visualisation. The major bottleneck, however, remains at the host–network interfaces. The ability to deliver high bandwidth communications to applications has not kept pace with other developments. For example, even on a high performance graphics workstations, current IPC implementations achieve throughputs of no more than a few Mbps.

The problems with existing architectures can be explained as follows: At the user level, processes communicate either by shared variables or by passing messages. At the system level, the two corresponding paradigms are shared memory and message passing. The shared memory mechanism

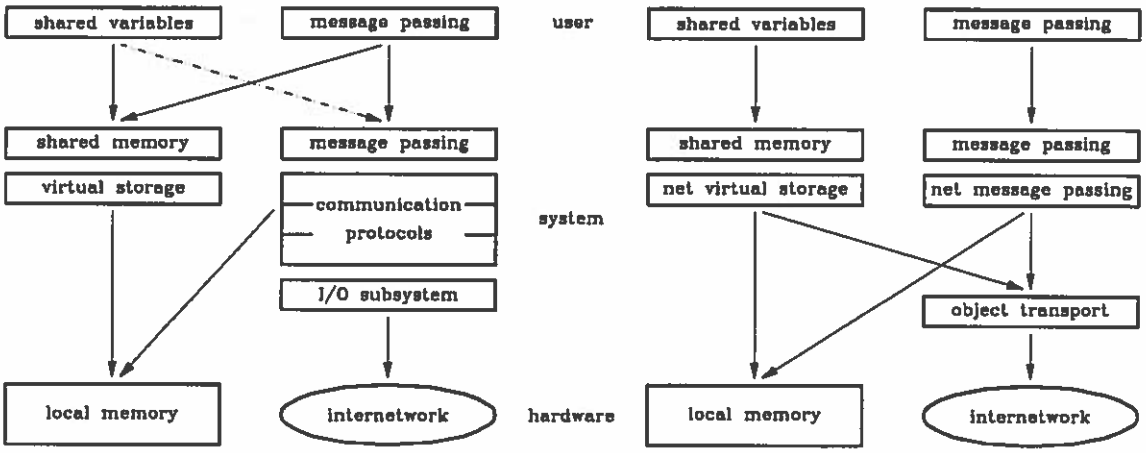

Figure 2: Interprocess Communication Implementation

is supported on tightly coupled systems with shared physical memory. Shared memory is not directly supported across a network, and must be mapped onto message passing at the system level. Message passing is supported on both tightly and loosely coupled systems. The support for message passing on tightly coupled systems is provided by reading and writing messages to shared physical memory, or by sending and receiving messages using queues or buffers. The support for message passing across a network typically consists of a stack of network protocols and mechanisms to treat the network as an I/O device, as shown in Figure 2a. The problems with this approach are the following:

Architectural: There is a lack of integration of hardware, operating system, and communication protocols. This results in considerable inefficiency and complexity for several reasons. The functionality of operating system and communications system components are not optimised for one another, thus the interaction and interfaces between them is inefficient and complex. There is a lack of correspondence between network and host data objects (*e.g.* packets and pages), and therefore inefficient synchronisation between components (*e.g. per* packet processing and page fault handling). This results in unnecessary control transfer, data buffering and reformatting.

The network interface is treated like an I/O device, and therefore, the *per* packet processing involves servicing interrupts, context switches, and data copying to protocol and I/O buffers. Furthermore, since I/O processors are designed to handle a wide diversity of I/O devices, ranging from slow character and unit record devices to high speed mass storage, I/O processors are not designed to perform optimally for VHSI type communications. In addition, the data stream is subject to the delays of both the I/O processors and network interface.

There is no way to directly use the shared variables paradigm for IPC across a wide area network, which leads to performance compromises for applications naturally suited for data sharing.

Transport protocols: Many existing and proposed transport protocols are general purpose, and are not designed to perform well for various classes of demanding applications.

General purpose error and flow control schemes are used which are complex to implement in hardware, and which do not exploit the improved functionality of the newer high speed networks.

End-to-end flow control schemes, such as the sliding window and variants, are unable to control the exact rate of transmission, due to interaction with error control.

Congestion control is prone to instability due to inaccurate estimates of end-to-end latency and lost packets, and other system interactions.

Flow and congestion control mechanisms are less able to respond to changing network conditions as data rates increase, *i.e.* by the time adjustments are made, the conditions that induced the adjustment may have drastically changed.

Network interface: Communication is handled through front end network interface or communication processors, which are stored-program processors that manipulate packets in a store-and-forward manner, resulting in latency due to their programmed operation and buffering of data. The network interface must also communicate with the host system using the standard I/O interface, which is not optimised for high speed communication (as mentioned above in the context of host architecture), resulting in an interface to the host not well suited for very high performance communication.

2.3. A model for high performance communication

A new host communications architecture is proposed, called Axon, to address the problems outlined above, and meet the requirements of high performance applications. The critical aspect of the Axon architecture involves providing an end-to-end data path between distributed processes with the characteristics of high bandwidth (sustained data rates of at least 1 Gbps), low latency, and the required functionality for communications across the VHSI. This requires that the data path be pipelined to a fine granularity (bit/byte level rather than packet level store-and-forward). This section will develop a model for high performance communication in an incremental manner, starting with a simple pipeline model, and indicating the difficulties in providing this end-to-end path. Then, a realistic model for pipelined communications will be developed that will be the basis for the Axon architecture presented in Section 3.

Simple pipeline model. An extremely primitive model is depicted in Figure 3. In this case, processes execute on CPUs with their code and data in a dual ported VRAM type memory, using the conventional random access ports. To communicate across the VHSI, data is shifted through the serial ports. As long as the serial ports are capable of shifting data at VHSI rates, the end-to-end pipeline has been established.

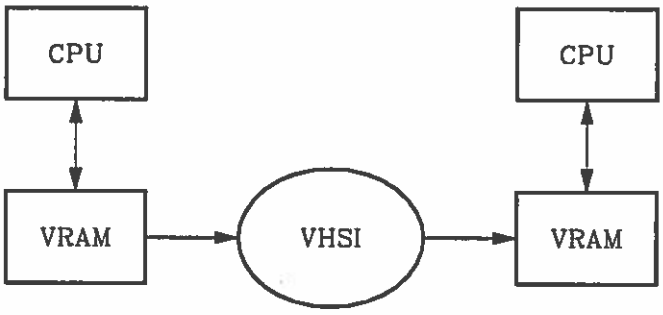

Figure 3: Simple Pipeline Model

Note that it is also important for the VHSI to provide the path internally to support the end-to-end pipeline, but for purposes of this research, the VHSI will be treated as a pipeline whose

characteristics can be described by a set of parameters to reflect its performance. This simple model suffers from a number of limitations when applied to a realistic setting:

- The path between VRAM modules through the VHSI has considerably higher latency than in the local environment, and with a non-uniform distribution. In a tightly coupled environment, the smaller uniform latency can be easily dealt with. Across an internetwork, the need for end-to-end flow control arises, resulting from internal network congestion and rate control, and the lack of clock synchronisation at the ends.

- Related to this is the presence of errors in data transmission, resulting from bit errors, and packet loss, duplication, and misordering. Some internetwork paths may have higher error probabilities, which the application may want to trade for performance and cost.

  These are the standard arguments for the need for transport level protocols, relating to conditions not present in a tightly coupled environment. Even though the underlying internetwork and protocols will provide a *quasi-reliable* connection [StPa89c], the requirements for flow and error control cannot simply be ignored.

- The objects to be communicated must be mapped into the application name (or address) space. In the case of the transmitting end, it may be reasonable to expect the application to be aware of the object binding to a range of real store locations to be transmitted serially out the VRAM. On the receiving end, however, the mapping is much more difficult. The receiving VRAM must accept all data presented to it at the prevailing data rate, and the application must decide where objects reside to perform the required binding. In addition to the performance implications of this, allowing processes to arbitrarily examine incoming data is not desirable from a security and privacy perspective. Since the end-to-end pipe must place data objects directly into the receiving process address space without the overhead of a store-and-forward operations, providing known buffer areas for incoming data is an unacceptable solution.

  This motivates the need for system level (host hardware and operating system) assistance in the symbol binding and address mapping. Since this requirement is similar to the motivation for providing virtual storage mechanisms, a good solution is to extend these mechanisms to include objects transported across the network. In addition to mapping remote objects for the application, this provides address space isolation between processes (unless sharing is explicitly desired). Note that some form of mapping is necessary regardless of the IPC paradigm in use. In the case of message passing IPC, the message buffers/queues are mapped into the process address space. In the case of shared variable IPC, objects (segments) are mapped into virtual address spaces as if a tightly coupled shared memory were available.

- The requirements for a transport protocol and mapping mechanism mentioned above require the appropriate system level support. Clearly, to support data rates above 1 Gbps, much of this support must reside in hardware. This includes the entire data path from the network interface to the host memory, as well as the routine control functions to support the pipelined data transport. This is referred to as the *critical path*. Thus, an interface is needed between the memory and the network. The network connection to host memory is part of the critical path, and requires network interface hardware to implement parts of the transport protocol and assist in the object mapping (as well as other desirable functions such as encryption).

Axon pipelined communications model. By building upon the simple pipelined model, and providing the required functionality motivated by its limitations, a realistic model for end-to-end data transport can be constructed. This is referred to as the *Axon pipelined communications model*, and is presented in Figure 4.

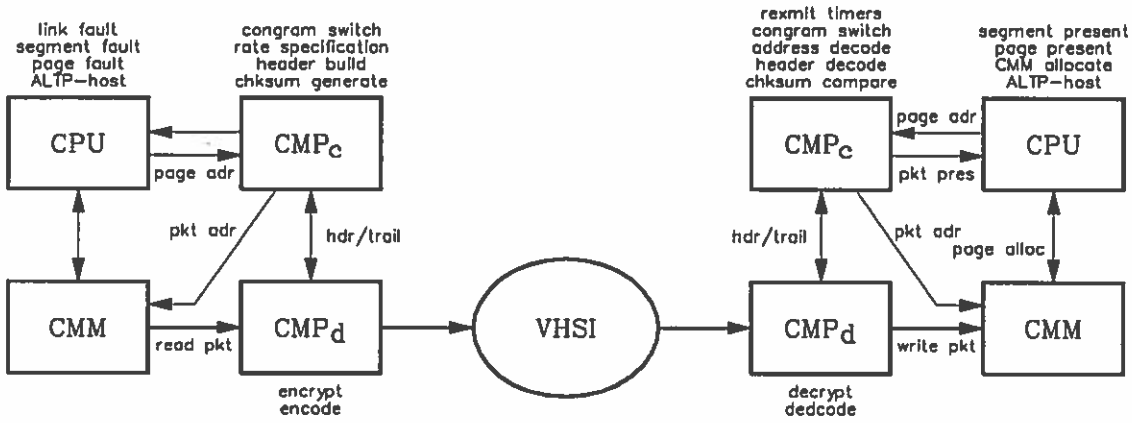

Figure 4: Axon Pipelined Communications Model

The CPUs access a CMM (communications memory module), which is similar in design to a VRAM. The symbolic namespace to virtual address to real address bindings are performed by the NVS (network virtual storage) mechanisms. The transport protocol is implemented as an ALTP (application-oriented lightweight transport protocol), which is specifically designed to have its critical path implemented in hardware, and is particularly oriented toward the end-to-end transfer of objects for IPC. The CMP (communications processor) is the network interface, implementing the required ALTP critical path and NVS assists. In the Axon pipeline model, the CMP consists of datapath (CMP_d) and control (CMP_c) functions. Thus the end-to-end data pipeline in the Axon architectural model consists of:

$$\text{CMM} \longleftrightarrow \text{CMP}_d \longleftrightarrow \text{VHSI} \longleftrightarrow \text{CMP}_d \longleftrightarrow \text{CMM}$$

It should be evident that the Axon pipelined communications *model* requires significant changes to existing operating systems, communications protocols, and host–network interfaces. The Axon *architecture* incorporates the necessary changes. The next section will present an overview of the Axon architecture as a whole. Then this proposal will focus on exploring the architectural requirements and performance tradeoffs in support of the Axon pipelined communications model.

3. Overview of the Axon Architecture

The Axon architecture provides the support for efficient, high performance (high bandwidth, low latency) IPC across an end-to-end pipeline through the VHSI. This is done by providing an object transport facility (ALTP-OT), which is used by the NVS (network virtual storage) subsystem and network message passing (NMP) interface. This is illustrated in Figure 2b.

This section provides an introduction to the Axon architecture, providing background on the research that has been done to date, and setting the stage for the central research issues to be discussed in Section 4. Axon is described in greater detail in technical reports [StPa89b, StPa89c, St90]. First, IPC primitives are discussed within the framework of the VHSI environment. Then, a brief description is presented of significant Axon architectural components. An example of the interaction of these components is given by describing the transport of a segment object across the VHSI in [StPa89a] §3.6.

3.1. IPC in the Axon architecture

IPC is supported as either a shared variable or message passing paradigm. Shared variable IPC is characterised by the use of *read/write* (r, w) primitives to shared data structures. Message passing IPC is characterised by the use of *send/receive* (s, r) primitives, and requires explicit message synchronisation by the programs communicating. A *generalised remote procedure call* (GRPC) is supported, in which the location of the procedure, data, and execution are all arbitrary. Additionally, *segment streaming* is supported, which transports multiple segments at high bandwidth with no *per* segment request overhead.

A *logical* view of the Axon protocol hierarchy is presented in Figure 5. It is important to note that this layered view is a logical view of functionality only, and does not imply that strict layering (in the ISO-OSI sense) is being adhered to.

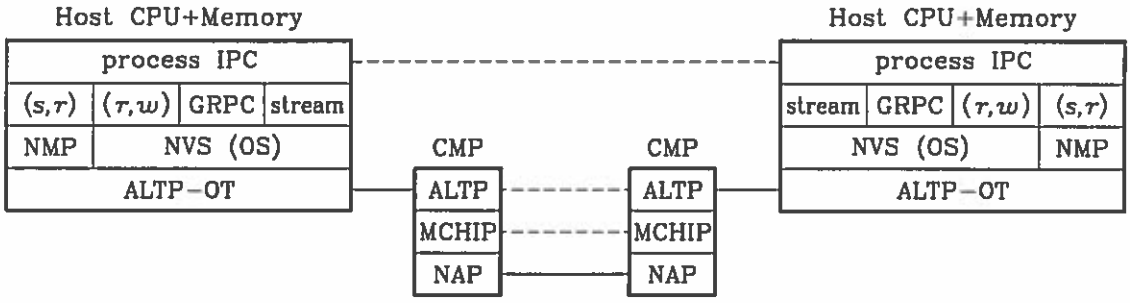

Figure 5: Axon Logical Protocol Hierarchy

The shared memory mechanism for IPC across the VHSI is implemented by NVS (network virtual storage). This can be utilised by an application either by referencing segments that are non-local, through the facilities provided by GRPC (generalised remote procedure call), or by the use of segment streaming. GRPC and segment streaming are described in [StPa89b] §2.1.

Support for message passing IPC is provided by a network message passing interface (NMP), which provides the support for invocation of the appropriate message based transport protocol calls. The transport mechanism is supplied by an *application-oriented lightweight transport protocol* (ALTP) tailored for the class of applications using IPC *object transfer*; this type is ALTP-OT. ALTP-OT resides as a set of software modules in the host system, and as hardware in the CMP (communications processor). The underlying internet/network layer of function is provided by a *multipoint congram-oriented high-performance internet protocol* (MCHIP) [Pa89, MaPa89], and *network access protocols* (NAP), which will be assumed to be present for the purpose of this proposal.

To provide support for IPC three major areas of the Axon architecture are involved: system level support for IPC (NVS and NMP), transport protocol support for the movement of objects involved in IPC (ALTP-OT), and the host hardware architecture and communications processor (CMP). Each of these will be sumarised briefly to provide an introduction to the Axon architecture, and framework for remainder of this proposal.

3.2. System level IPC support and NVS

The system level support for the various application level IPC paradigms is provided by two components: NVS and NMP. NVS is the system level shared memory interface for shared variables, GRPC, and segment streaming. NMP is the system level message passing interface. NMP performs

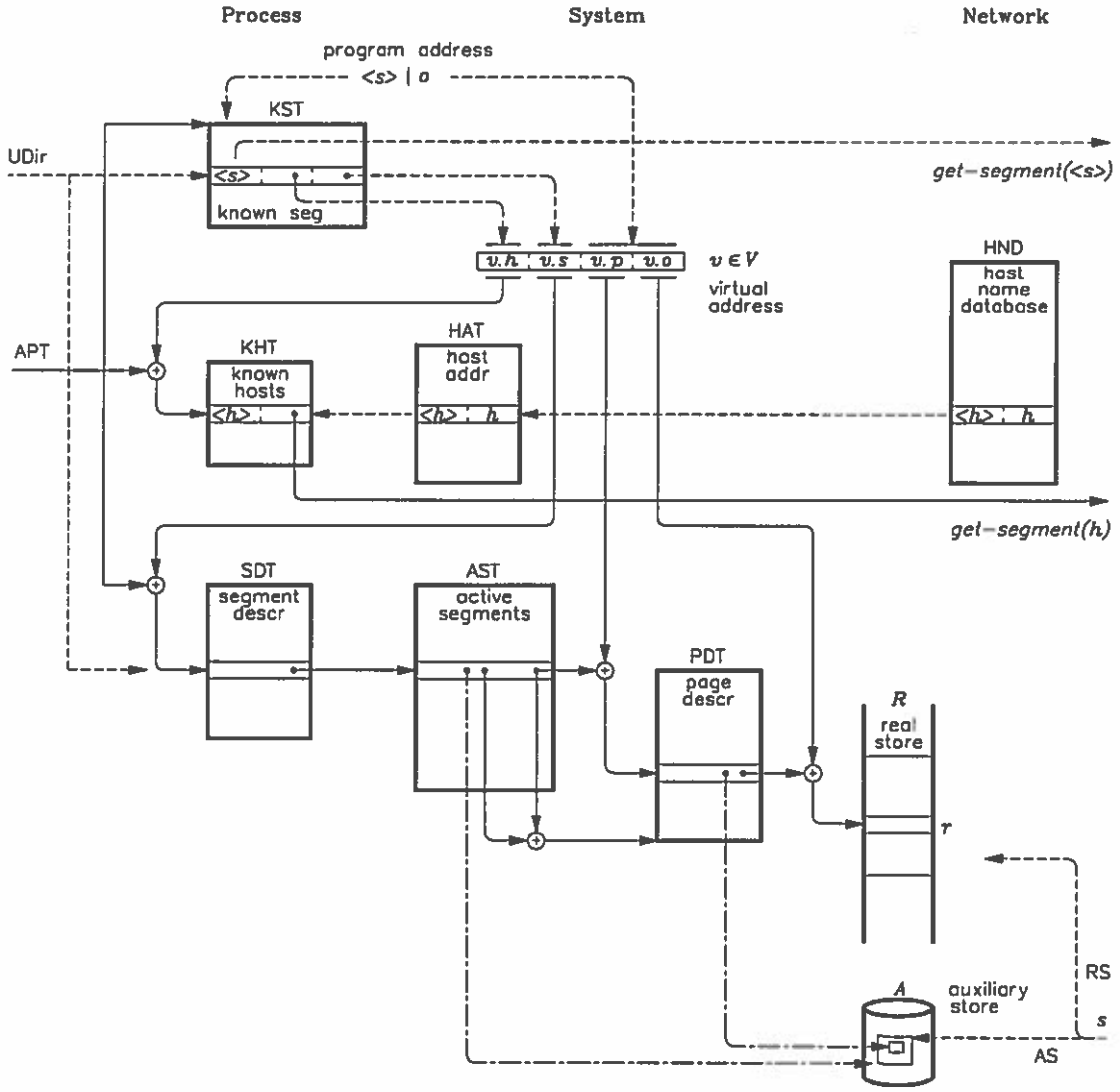

Figure 6: Network Virtual Storage Address Translation

a relatively straight-forward transformation of program (*send, receive*) primitives to corresponding transport protocol message-object transfer calls. Thus, the emphasis in this section will be on describing NVS.

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, in a manner based on the Multics operating system [Be72, Or72, MaDo74]. Details on segmented paged virtual storage are provided in [StPa89b] §3; this section only briefly discusses the NVS extensions.

Nvs extensions allow the segments to be addressed when resident on a non-local host. This is accomplished by either including a host id. field in the virtual address (*network virtual address*), or the host id. in the *segment descriptor table* (SDT) entry (*local virtual address*). When a segment fault occurs for a nonlocal segment (indicated in the segment descriptor), the dynamic address translation facility invokes ALTP-OT to get a copy of the segment from the appropriate system.

When 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 6. Address pointers are represented by arrows on solid lines, the copying of data 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 to assist in n-way IPC using multipoint connections (not shown in Fig. 6). Depending on the method used for network-to-host object mapping, a packet presence bit vector may be in *page descriptor table* (PDT) entries.

NVS in Axon also involves extensions and additions to storage management policies. The replacement policy is affected as a result of pages from remote segments in the locality set, which requires redefinition of the working set to account for non-local segments. 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 (RS), auxiliary store (AS), a combination (RAS), or frame buffer (FB) placement, with a number of sub-policy options (swappable, nailed, *etc*). The NVS storage management policies are described in detail in [StPa89b] §5.

3.3. 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 rate, once a transport operation has been initiated. By optimising 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. ALTPs can be optimised to provide the kind of performance guarantees and functionality the specific applications need.

The ALTP type that will be investigated is designed to support IPC by the transfer of objects, with primary consideration in supporting NVS segments. This will be referred to as ALTP-OT. ALTP-OT uses rate based flow control, where the rate specification consists only of parameters important to IPC, and efficient error control streamlined to include only what is necessary for object transfer. The various error conditions are handled by ALTP-OT as follows: duplicate packets are discarded; corrupted packets are discarded with retransmission request; missing packets are detected by the expiration of a timer with retransmission request; packet sequence is ignored since the packets are placed directly in the proper location of the target store. 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 (avoiding sequence buffers) than would be the case for a general purpose transport protocol.

Information is transferred throughout the internetwork in packets. A structured group of packets corresponding to a single ALTP-OT semantic action is a *super-packet*, consisting of an initial control packet (which may also contain data), and optionally followed by data packets. Bits in the packet

header indicate whether the packet is control (MCHIP or ALTP) or data. ALTP-OT control packets require processing by the ALTP-OT logic in the CMP (communications processor), as well as by the host system hardware and OS. Data packets require considerably less processing, all of which can be done in real time by the CMP hardware. The significant point is that most of the usual *per packet* control processing is only performed *per super-packet* in Axon. In addition, since ALTP-OT is an integrated system program, it has direct access to the appropriate operating system facilities (*via* lightweight system calls) and data structures. The format of a data packet is presented in Figure 7.

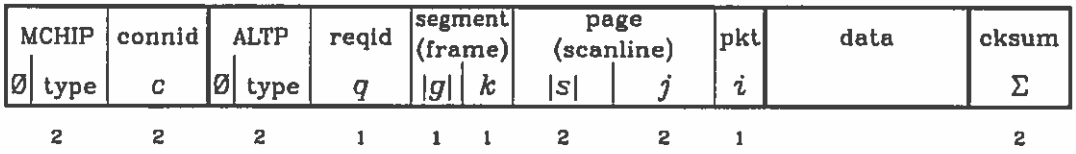

Figure 7: ALTP-OT Data Packet Format

Thus a structuring of the data that is recognised by ALTP-OT allows the *per* packet processing to be simplified to the extent that VLSI implementation is reasonable and efficient.

The ALTP-OT requests and operations include:

- Connection management: *join-ipc, respecify-rate, leave-ipc, invalidate-segment*
- Object receive: *get-segment, acquire-segment, get-page, get-copy, get-stream, receive-message, retransmit-packets*
- Object transmit: *release-segment, release-page, remote-execute, send-copy, send-stream, send-message*

ALTP-OT is described in detail in technical report [StPa89c], including arguments in favor of the choices made for error and rate control.

3.4. Host and network interface architecture

High performance computer systems typically consist of one or more central processors, which communicate with memory banks and I/O processors through an interconnection network. In addition, various caches may be present to utilize fine-grained locality, and perform speed-matching of data rates. Additionally, the memory system may consist of a multi-level hierarchy including extended memory, such as for virtual backing store. Two high-level host configurations are part of the Axon architecture, which give the CMP (communications processor) direct access to memory.

The first gives the CMP a relationship to the system similar to that of I/O processors, thus interfacing the CMP directly into the processor–memory interconnection network. This is referred to as *interconnect interface architecture* (IIA). The second, interfaces the CMPs to the back end of a special dual-ported CMM (communications memory module), which is referred to as *memory interface architecture* (MIA). In this case, 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 design of the CMM is similar in concept to VRAM design. If all real storage is not CMM, the physical address space of the system must be partitioned between conventional and communications memory.

The Axon architecture interfaces the CMP directly to the processor or memory. On the network interface side, the CMP must be capable of receiving and transmitting packets at the full network data rate. On the host side, the CMP must either interface to the processor–memory interconnect,

or the CMM, depending on the host architecture (IIA or MIA, respectively). More details on Axon host architecture configurations are presented in [St90] §4.

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 function in VLSI. This may be realised by organising the CMP as a pipeline, dynamically reconfigurable based on the ALTP type and options for a particular connection. The pipeline organisation allows packets to be processed while moving at the VHSI interface data rates.

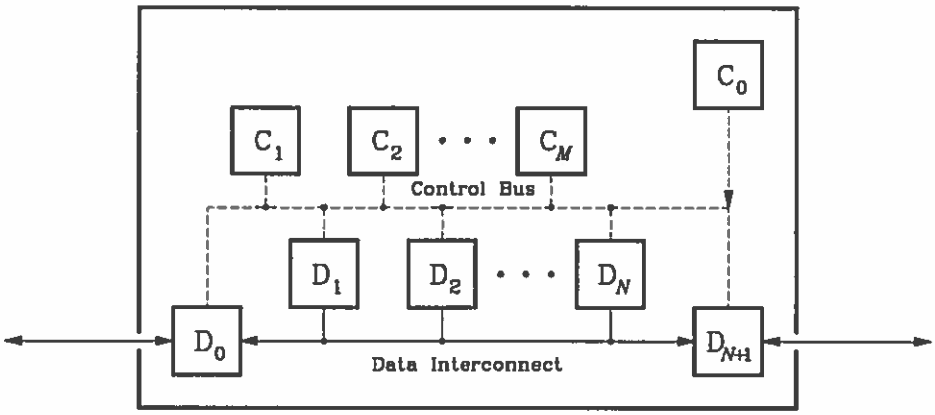

Figure 8: Communications Processor Implementation Model

The CMP implementation model (Figure 8) consists of a set of *datapath modules* (DMs) $D = \{D_0, D_1, \ldots D_{N+1}\}$, and control modules (CMs) $C = \{C_1, C_2, \ldots C_M\}$. The DMs perform data manipulation

and transformation on packets as they pass through the CMP. In general, each DM should be designed to perform its function without buffering a packet, except for the pipeline delay as the packet passes through. One of the CMs (C_0) is responsible for pipeline configuration and control.

A particular configuration of the pipe C_i, consists of a permuted sequence of data modules $d = \langle d_0, d_1, \ldots d_n \rangle \subseteq D$, forming a logical pipeline, along with a set of control modules controlling them $c = \{c_0, c_1, \ldots c_m\} \subseteq C$. The DMs are connected by a high speed interconnection network, which is capable of transferring data between the modules subject to the configuration C_i. Each type of ALTP requires a particular configuration of the pipeline, *e.g.* ALTP-OT induces a configuration C_{OT} of the pipeline. Note that while this model allows for reordering of the DMs, a fixed sequence pipeline may be sufficient in most implementations.

Examples of data modules include the network receive and transmit interfaces, CMM interface, parallel/serial conversion, data format conversion (including generic network standards), encryption/decryption, and video widow coordinate translation. Control modules include rate control, checksum generate/compare, CMM address generate, header generate/decode, and packet presence timers and retransmission logic. The CMP implementation is discussed in more detail in [St90] §6, particularly with respect to the supporting ALTP-OT.

4. Focus of Proposed Research

Previous sections have motivated the need for delivering high bandwidth communications directly to applications that require the transfer of large objects with low latency. The Axon pipelined

communications *model* was introduced (§2.3) as an end-to-end data path allowing applications to utilise VHSI bandwidth. The Axon host communications *architecture* was proposed (§3) to meet the requirements of this model, and to provide the framework for high performance IPC using various paradigms.

This section provides the focus for continued research, by returning to the end-to-end pipeline, and identifying the research issues of concern, centered around data transport at gigabit data rates, with scope appropriate for the proposed dissertation. The thrust of this research will be to explore the architectural requirements and performance tradeoffs in support of this end-to-end pipeline, starting at 1 Gbps, and scaling toward 1 Tbps. First, the main problems to be pursued will be stated. Then, a more formal statement will be made, indicating parameters of interest and tradeoffs to be investigated in their resolution. Finally, a plan of action for the continuation of this research will be indicated, describing the design, specification, simulation, and evaluation activities, along with the type of results expected.

4.1. Research questions and issues

The research questions to be pursued within the scope of this dissertation will be presented, with a high level introduction to each one. Note that there are a large number of other interesting problems that arise from the proposed Axon model and architecture (such as in the area of object coherency and storage management), but these are left outside the scope of this thesis. At this level each question will be posed, with a brief indication of why it is important to solve. The first three of these will receive the greatest emphasis in the proposed research.

Critical path: It is clear that to sustain data rates above 1 Gbps, the critical path function will have to be implemented in hardware. This includes the data path and routine control functions necessary to keep the data path moving at the required rates. An important question is then: What are the critical and non-critical path functions to allow end-to-end pipelined communications at 1 Gbps, and how do these scale with higher data rates? Note that some function, such as the data path from network to memory is clearly part of the critical path, as are some control functions that govern routine packet processing, such as the rate control logic. Other control functions may or may not need to be part of the critical path, depending on the data rate, and time–space complexity tradeoffs (discussed later). Examples of functions that may or may not be completely in the critical path are packet retransmission timers and host–network object mapping. Parallelism in the data path can be used to provide a speed advantage, allowing higher data rates for a given CMP processing rate. In addition, pipeline delay can be used to allow control functions the necessary time to operate at high data rates. A time–space complexity analysis will indicate which functions should reside in the critical path.

There are two possibilities for the implementation of non-critical path functions. The first is implementation in host software. In some cases, however, it may make sense to have a microprocessor assist to the CMP, which will allow a performance level greater than that of host software, and without impacting the normal program flow on the host. This CMP *assist processor* (CAP) may be useful for functions involving control between the CMP and host CPU.

Host–network interface control: The main path between the CMP and host system is *via* the serial data port of the CMM, which is primarily used for data transfer. It is necessary, however to pass control information as well, *e.g.* for host–network object mapping (discussed below), ALTP-OT operation initiation, real storage allocation for incoming objects, *etc.* The question is: What is the best way to handle the transfer of control and synchronisation between the CMP and host CPU (and operating system)? Additionally, if a CAP (CMP assist processor) is

used, it will be involved in, and may actually be primarily responsible for the CMP's control communication with the host.

Object mapping: Network data objects are packets and super-packets; host data objects are words, swords (interleaved memory data path width), cache lines, pages, segments, and segment groups. The resulting question is: How is the mapping between network and host data objects handled? Since page-level objects are used for storage management, it is the packet-to-page mapping that is of greatest concern (as well as the super-packet-to-segment mapping). In the ideal case, the packet size equals the page size. This allows the placement of packets directly into the target page location, and a packet arrival corresponds to a page fault recovery. It is likely, however, based on current experience with host page sizes and network packet sizes, that packets will be smaller than pages. In considering heterogeneous systems that may have differing page sizes, communicating across a VHSI constructed of subnetworks with differing packet sizes, it becomes essential to consider how to manage the arrival, presence, and movement of smaller network data objects that are mapped within larger host objects. This mapping must, in part, be within the critical path, since it is part of routine packet handling.

Latency: As network bandwidth and processor performance increase, the speed-of-light latency becomes an increasingly serious problem. The total end-to-end delay of an object transmission consists of several components: the speed-of-light latency, the queuing delay due to VHSI subnetwork switching, the pipeline delay at the transmitting and receiving host/CMP, and finally the delay involved in receiving the entire object at the prevailing data rate. All but the first of these latency components will be reduced by the Axon and VHSI architectures. Unfortunately, there is no reasonable expectation to scale the speed-of-light downward, and it will thus increasingly dominate as other delay components scale. One solution is to prefetch objects before their reference. This is not accomplished without difficulty and cost. The question is then: How to best deal with the increasing dominance of the speed-of-light latency, and to either prefetch earlier or with larger granularity, based on application behavior and locality sets?

Note that by using the segment as the granularity of object transfer, rather than pages, a certain amount of prefetching at the page level is automatically done, which helps reduce the impact of this problem.

Error control: A number of error control strategies are possible, in particular governing the manner in which retransmission requests are made. The Axon architecture allows application-oriented selective retransmission of error or lost packets, which gives considerable flexibility in retransmission strategy. There are, however, a number of alternatives that can be considered to optimise the retransmission policy:

location of retransmission request timers (Axon uses the receiving end since it is best able to estimate when packets should arrive [Cl87a])

granularity of retransmission and timer values: selective packet requests can be accumulated to packet (PKT), page/scanline (PGE), segment/frame (SEG), or segment-group/image (GRP) quantities, allowing multiple packets to be retransmitted in a single operation

fetch policy determines whether packets are always requested for retransmission (anticipatory – AR), or only if a page is referenced that contains them (demand – DR)

preemption of the primary data stream by the packet retransmissions (PE), or non-preemptive (NP), since error control is in band

An important issue is to determine the appropriate selection of retransmission policies from this space of sub-options, as various application and network performance parameters vary,

and based on other Axon architectural decisions. Related to this is how to determine the timer values and mechanisms to trigger retransmission. Note that this policy selection interacts with the issues previously described. The retransmission policies are described in [StPa89c] §4.

4.2. Investigation of problems

This section looks at each of the research questions described in the previous section in a somewhat more formal manner, indicating some of the tradeoffs and parameters of interest that will be investigated as part of this research. These formal statements will be developed and incorporated in the simulation models (§4.3) and analysis, to evaluate associated tradeoffs and find optimal operating points.

Critical path: The determination of what control function should be implemented as part of the critical path involves a time-space complexity tradeoff.

Time complexity: The impact on time complexity of a control function is the time taken for a hardware implementation (in clock cycles), *vs.* the time taken for a software implementation (in host/CAP instruction cycles).

For the host, the time that control function C_i takes to complete t_i is based on the instruction cycle t and the number of instructions to execute m_i, as well on any extra overhead m_i' in terms of context switches, system calls, *etc.*, thus: $t_i = (m_i + m_i')t$. For simplicity, CAP implementation is not considered in this discussion.

For the CMP, define the *minor cycle* τ to be the inverse of the serial data rate on VHSI communications links, (*e.g.* for 1 Gbps, τ=1ns). Define the CMP *major cycle* τ_w as the clock cycle internal to the CMP within the parallel data path w bits wide, (*e.g.* for 1 Gbps and octet-wide data paths, τ_8=8ns). By allowing n_i stages of pipeline delay for a particular control function C_i to take place, the time it takes to complete is then $\tau_i = n_i\tau_w$. Thus, define a metric for *critical path time savings*: $\mathsf{CP}_i = t_i - \tau_i$.

Note that by implementing a control function in the critical path, there may be an associated cost, in increasing the latency through the CMP by increasing the number of pipeline stages by Δn_i, *i.e.* the object transmission latency for *all* objects would increase by $\tau_w \Delta n_i$.

Space complexity For the host, space complexity consists of memory used for software implementation of the function. This will be ignored for this discussion, since a given operation is a sufficiently small fraction of total host or CAP memory.

For the CMP, space complexity has two measures: chip area and off-chip interconnect lines. Define a_i as the area [m^2] required to implement C_i. Control functions can be classified by their complexity sensitivity to other parameters: [1] *Fixed-cost control functions* utilise a fixed area, relatively independent of other parameters (an example of this is the rate control logic). [2] *Datapath-width sensitive control functions* have an area that is a function of the data path width of the CMP: $a_i = f(w)$, and thus constrain the possible speedup of the CMP by greater parallelism, in the same manner that total available chip area limits the possible datapath width (an example of this is the checksum logic). Note that a_i need not be linear in w. [3] *State-sensitive control functions* are those that utilise on chip memory in maintaining state. (An example is the packet presence function, if implemented using a memory array of presence bits, which can be estimated by $a_i \doteq f(|\hat{c}||\hat{q}||\overline{\sigma}||\overline{s}||p|/|\pi|)$, where $|\hat{c}|$ is the expected number of concurrent connections, $|\hat{q}|$ is the expected number of outstanding requests, $|\overline{\sigma}|$ is the average number of segments being transported *per* request, $|\overline{s}|$ is the average segment size in pages, $|p|$ is the page size in bits, and $|\pi|$ is the packet size in bits.) The constraint to be met is that the sum of all the control and datapath functions implemented on the CMP must not

exceed the available area for a given process technology: $\sum_i a_i \leq a_{\text{CMP}}$. It will be assumed that a complementary logic family will be used so that power dissipation is not the dominant constraint.

The other space complexity measure is the number of off-chip interconnect lines (pinout using conventional packaging techniques). Define l_i as the number of off-chip interconnect lines needed to implement control function C_i on the CMP. Thus, l_i is constrained by the threshold of available interconnect on a chip using a given packaging technology: $\sum_i l_i \leq l_{\text{CMP}}$.

Thus, in determining if a function should be in the critical path, the tradeoff is in time saving CP_τ, *vs.* acceptable chip complexity (a_i, l_i). Particularly as data rates scale upward, the decision can be made determining which functions should be included into the critical path.

Object mapping: This mapping must be part of the critical path, since it is part of routine packet handling, either explicitly or implicitly. The CMP will place packets directly into the proper locations of the target page. In the explicit case, the CMP critical path records packet arrivals, and determines page presence when all of the page's packets have arrived. One option is for the CMP to have a packet presence bit array μ, with a bit for each packet of each page of each segment of each group of each request of each active connection. Thus the size of the array required is $|\mu| \doteq |\hat{c}||\hat{q}||\overline{\sigma}||\overline{s}||p|/|\pi|$, as indicated above. This scheme is simple, particularly in the random addressability of packets, but requires significant memory. Another possibility is to structure queues that are tagged by connection and request id (c, q), and contain the packet number $\pi_i p_j s_k$ only for missing packets of pages that are not yet fully present. When the entire page has arrived (initially or after retransmission), it is marked present in the PDT, and the CMP memory is freed. Note that both explicit mappings allow the CMP to request retransmission of missing or corrupted packets based on the best retransmission policy, *e.g.* whenever the timers fire for each packet.

In the implicit case, when packets arrive, packet presence bits are set in the corresponding host PDT (page descriptor table) entry. When the host page faults, the packet presence vector is examined to determine if the whole page is present. Note that while relieving the CMP of a certain amount of complexity (especially memory for the packet presence bit vectors), this restricts the ability to request missing or corrupted packets to page fault events, rather than based on CMP timers. An alternative implicit scheme uses additional structure in the CMM to tag packet chunks of memory with presence bits, which are used by the host (or CAP) to set PDT presence bits.

Latency: For the speed-of-light latency problem, define the total end-to-end transmission delay of an object o by $D_o = d_c + d_q + d_x + d_r + d_r + d_o$, with components: the speed-of-light latency $d_c = g/c_f$ (g = distance, c_f = velocity of light in fiber), the queuing delay due to VHSI subnetwork switching d_q, the pipeline delay at the transmitting and receiving CMM/CMP $d_x + d_r = (n_x + n_r)\tau_w$, and finally the delay involved in receiving the entire object at the prevailing data rate $d_o = |o|\tau$. The goal is to minimise D_o. As indicated in the previous section, it is the responsibility of the VHSI to minimise d_q, and the Axon architecture to minimise $d_x + d_r$. As these scale downward, (with t and τ), d_c dominates d_x, d_r, d_q. Also note that if $|o|$ scales up with application demands and host CPU power and memory size, d_o may remain a significant part of the latency.

The time a program must wait to use an object for the first time is dependent on the total delay D_o, and how far in advance prefetching occurs t_δ, *i.e.* $\min(D_o, D_o - t_\delta)$. Ideally t_δ can be increased, so that the latency D_o is not significant, but this of course has a number of costs: Segment and segment group transfers utilise program structure and data structure locality, but may prefetch data not in the programs current locality set, *e.g.* if an entire segment is placed in real store, the induced working set of pages P may be much greater than the actual locality set P_ℓ, resulting in extremely inefficient use of real store ($P - P_\ell$ wasted). If

an object must be exclusively used, and therefore be locked, processes waiting for the lock remain blocked for longer periods as the prefetching time and granularity increase. Finally, while some applications may be able to predict address reference traces ρ, it is unreasonable to expect this in the general case. This means that "fine-tuning" t_δ to optimally match D_o seems to be a difficult goal.

Note that the range of latencies can be partitioned into three regions, based on the geographic diameter g [km] of the network. This results in differing magnitudes of d_c (and also to some extent d_q). Assuming fiber optic transmission technology, there is $d_{\text{LAN}} \leq 10\mu\text{s}$ for local area networks ($g \leq 2$), $10\mu\text{s} \leq d_{\text{MAN}} \leq 500\mu\text{s}$ for metropolitan area networks ($2 < g \leq 100$), and $500\mu\text{s} \leq d_{\text{WAN}} \leq 100\text{ms}$ for wide area networks and internetworks ($100 < g \leq 20\times10^3$). Clearly, the latency problem is worst for long haul networks, but it will be useful to evaluate the latency tradeoffs in the context of the three classes of networks. For example, it may make sense to *page* across a LAN, but not across the VHSI, where *segments* are a better granularity of object transfer.

Error control: The optimal choice of retransmission strategy is sensitive to a number of parameters, with each of the three policy sub-options (granularity, preemption, fetch policy), having a somewhat differing dependency. The parameters of interest are defined as:

- ρ the program address reference trace $\rho = \langle \rho_0, \rho_1, \ldots \rho_v \rangle$, $\rho_i \in V$
- x_π the event of a lost packet as random variable
- D_π the end-to-end delay waiting for a retransmitted packet
- $|o|$ the object size in transit (if NP – non-preemptive)
- n_o the number of retransmission requests (o granularity)

The policy that minimises D_π is PKT-ARPE (packet granularity with anticipatory preemptive retransmission). The policy that minimises overhead (in terms of n_o and CMP processing) is GRP-NP (segment group granularity with non-preemptive retransmission; the relative efficiency of demand *vs.* anticipatory retransmission is dependent on ρ). Thus, there is a tradeoff between delay and efficiency. Other factors affect the policy choice. The object latency component to receive a packet $d_\pi = |\pi|\tau$ will be much smaller than $d_c + d_q$ for WANs. Since the page is the unit of reference at the host, and $d_\pi \approx d_p \ll d_c$, it may make sense to use at least PGE granularity. In addition, if implicit host PDT mapping is being used, it is at page reference time that missing packets will be discovered, and the minimum useful granularity is PGE.

Identify a particular packet as $\pi_i p_j s_k$. The timer for this packet expires after [StPa89c]:

$$\delta_{\pi_i p_j s_k} = \pi_r + 2(d_x + d_c + d_q + d_r) + \tau|\pi| \left[i + (j-1)\frac{|p|}{|\pi|} + \sum_{\kappa=1}^{k-1} \kappa \frac{|p||s_\kappa|}{|\pi|} \right]$$

where π_r is the time to fetch a packet at the remote end, the delay components d are as used above, and the expression in brackets corresponds to the packet number. Note that the purpose of the timers depends on the fetch policy in use. For AR the timer value indicates *when* a packet retransmission request should be made, and for DR the timer indicates *how long to wait* before a referenced packet is assumed to be missing, and thus retransmitted.

Some combination schemes will be worth investigating, *e.g.* PGE-DRPE/SEG-ARNP. This policy uses a page granularity, requesting preemptive retransmission of any page referenced (*i.e.* page fault). Otherwise, the primary data stream is allowed to complete before all other error packets are retransmitted. This would provide a compromise between the desire to maximise efficiency (by accumulating requests for the entire segment), *vs.* minimising the time for a page to obtain all of its packets on reference. This policy may be superior to either a pure NP or PE scheme.

4.3. Plan of action and current status:

To deal with the limitations of the current communications model, and to satisfy the requirement of the Axon pipelined communications *model*, the components of the Axon *architecture* have been identified, along with the research questions to be pursued within the scope of this dissertation. The purpose of this section is to state the scope of the dissertation, report on the current status, and suggest a plan of action for continued research.

The scope of this research involves a number of activities, which can roughly be categorised as design and specification, or as simulation and evaluation of research issues. The main emphasis so far has been on the design and specification activities, which have been necessary to determine how the simulation and evaluation activities should proceed, and to narrow the scope from an extremely large range of potential issues.

The design and specification activities are:

- Host architecture: Suitable configurations of host architecture are identified, to provide direct VHSI access to host memory. The IIA (interface interconnect architecture) and MIA (memory interface architecture) configurations are defined. This work is described in [St90]. This dissertation will use MIA for the exploration of research issues.
- NVS design: A segmented paged virtual storage system is extended to include segments located throughout the VHSI. Data structures are added for symbolic host name binding, and extended to allow non-local segments to be handled, as well as to allow distributed n-way IPC. Link and segment fault resolution procedures are extended, especially for remote segment faults. Additionally, the impact on storage management policies and working/locality set considerations is defined. The replacement policy is reconsidered, in the light of locality sets with remote pages, and working sets with remote segments. A remote placement policy and space of sub-options is defined. This work is described in [StPa89b].
- ALTP-OT design and specification: The requirements for a transport protocol to operate in the VHSI environment, in response to the deficiencies of current and proposed protocols, are identified (as first outlined in [BhSt88]). The rate and error control strategies appropriate for object transfer are specified, along with detailed strategies for packet retransmission and timers. The ALTP-OT operations are specified, in terms of interface to higer levels, and operational description. Key operations are specified in detail, in the form of state machines and procedural description. This work is described in [StPa89c]; and the detailed specification of many of the ALTP-OT operations remains to be done.
- CMP/CMM design: A generic, high level reconfigurable pipeline model for the CMP is described. Appropriate DMs (datapath modules) and CMs (control modules) to implement ALTP-OT that are clearly part of the critical path are identified. The first estimation of other candidate critical path CMs are also identified. Potential bottlenecks for 1 Gbps operation are identified (such as for packet presence and retransmission timers) that will be explored in particular. CAP implementation of some of this function will be considered. This work is currently in progress, and described in sumarised in [St90]. A CMP and CMM based on these blocks is being designed.

A high level simulation model is constructed to evaluate the research issues and confirm the validity of the Axon design decisions, and is presented in Figure 9. This model consists of five major sub-models, whose relationships are indicated in the figure by directed arcs between the sub-models. External parameters and policy choices are indicated by arcs into the sub-model blocks. The five sub-models are briefly sumarised as follows:

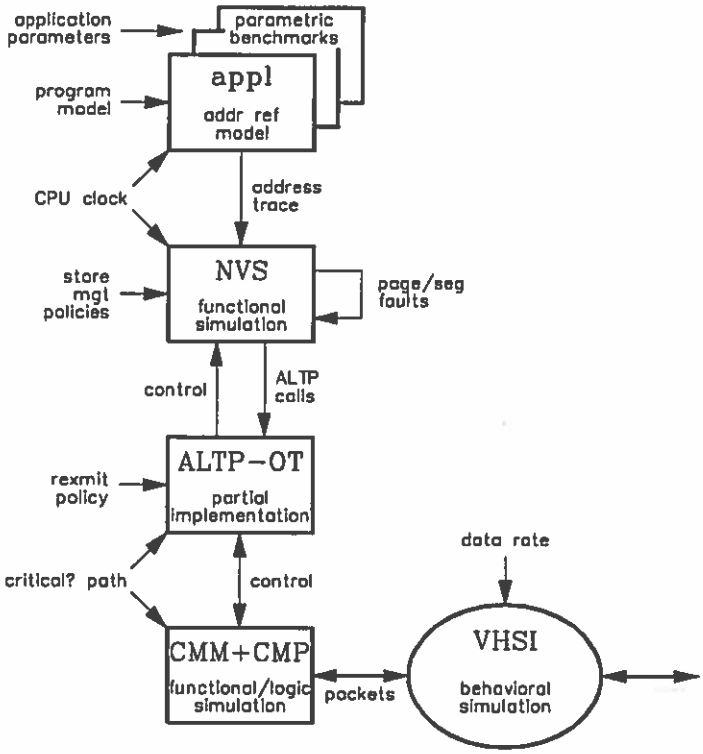

Figure 9: Axon Simulation Model

- Application behavior: Application behavior may be modeled in two ways: A generic program behavior can be used and extended to model applications using NVS. For example, using the hierarchical BLI (bounded locality interval) model [Ma76, Ma82], non-local segments can be included. A model for determining the distribution of non-local segments can be constructed. It is likely that some form of *topological locality* will be a useful concept, and this may correspond to the top level BLI in some manner. Alternatively, specific models of program behavior corresponding to one of the target VHSI applications (such as scientific visualisation and remote image access) can be constructed, (referred to as *parametric benchmarks* in Fig. 9). The simulation will use one or both of these methods.

- NVS: The simulation will incorporate the functional behavior of NVS, in particular link, segment, and page fault resolution, as well as the use of a TLB for address mapping. An extended queuing model has been constructed to model NVS behavior [St88d].

- ALTP-OT: A partial implementation of ALTP-OT will be made for demonstration purposes, and for use in the simulation of research issues, tradeoffs, and policy decisions. This will specifically include the object transport operations for representative calls, such as for *get-segment* and *get-stream*, as well as the appropriate error control strategies. An extended queueing model has also been constructed to model ALTP-OT and the VHSI [St88d].

- CMP/CMM: A high level functional simulation of the CMP and CMM will be constructed to model their behavior. By altering the CM content of the CMP, along with their chip complexity analysis, the appropriate critical/non-critical path choices can be made, as the data rate scales upward. If a CAP is determined to be desirable, it will be included in the simulation.

- VHSI/MCHIP: A behavioral model of the VHSI will be constructed, consisting primarily of the latency d_q and packet loss x_π random variables. The behavior of MCHIP will be specified for connection/congram management and rate control.

These simulations, implementation, and models will be used to explore the research questions described in the last section (§4.2). The questions will be further formulated, in terms of detailed optimisation problems and tradeoff evaluations. The simulations will be used to evaluate the proper parameter operational ranges and policy choices. When possible this will consist of optimal choices. When tradeoffs are involved, reasonable operating ranges of parameters and policies can be determined from the appropriate plots. Additionally, it is expected that further important issues will arise as this research proceeds.

4.4. Expected contribution of dissertation

The goals of this research are to design an architecture capable of supporting an end-to-end pipeline between applications, at data rates above 1 Gbps, and to understand what the constraints and bottlenecks are as the data rate scales upward. The types of solutions expected from this research are:

- Demonstration of viable Axon design (functional, performance) from the design specifications, implementations, and overall simulations. This provides for 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. Justification for VLSI implementation, in terms of overall chip complexity, particularly memory space requirements and off-chip interconnect.
- 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 (especially LAN/MAN/WAN), processor performance, and network bandwidth, with respect to the incorporation of function in the critical path.

5. Related Work

This section describes background and ralated projects and research efforts. Motivation for the proposed research has been given in Section 2, and is also indicated, for example, in [RaLa87, Kl88, Le88].

5.1. Network virtual storage

A segmented, paged virtual store was first implemented by Multics [Be72, Or72, MaDo74] on a GE-645 and the IBM 360/67 [IBM72] running TSS/360 [Co65, Le65, IBM78a]. The Multics line continued on the HIS 645, 6180, DPS-60/68, DPS-8/M, but has now been terminated. Modern systems that owe significant heritage to Multics include the CDC Cyber 800 NOS/VE [CDC84a, CDC84b, CDC84c] and Prime 50 Primos [AuLa83a, AuLa83b]. Segmented virtual store was not used by other operating systems in the IBM System/360 and 370 family, until the addition of features provided by ESA/370 [Pl89, ScGa89, IBM88a, IBM88b] under MVS/ESA.

Additionally, systems that provide a segmented paged virtual store include the IBM AS/400 [IBM88c] and System/38 CPF [IBM86b, IBM85a, IBM78b], AT&T 3B series [HeKu83, ATT86a, ATT86b], Intel iAPX432 [In81, In83a, In83b, Or83] 80386 [In86b, In87] i486 [In89] and 80960 [In88], and Motorola 68030 [Mo87].

For quite some time, there has been an ongoing discussion in the research community about the relative advantages of the shared memory and I/O models of IPC. Though this discussion has not been conclusive in favor of either approach, it is clear that the application program semantics should use the communications model most appropriate for its function [AnSc88, Re82], and for a large class of problems, the shared memory model provides a natural paradigm [Fl87]. These include certain IPC and RPC (remote procedure call) problems, as well as process, procedure, and file servers, and some types of distributed database access.

Advantages typically associated with the shared memory model are the close mapping of the primitives to computer hardware, and the idea that the reading and writing of shared variables is *fundamentally* primitive [FiFr84, LyFi81], to be used as a base for other programming paradigms. Additionally the shared memory paradigm provides the best abstraction for handling shared state in distributed systems [Ch86a], allowing distributed systems to be more easily constructed, understood, and proved correct [AlGo89].

It is interesting to note that arguments for the *duality* of the two models have been made [LaNe78], which indicate that a relatively simple transformation can be made between the program primitives of the two models. But more recently, it has been shown that the duality arguments can only be applied to indicate that an underlying structure can support either model [Re82]. Problems which are naturally suited to the shared memory paradigm can be somewhat difficult to map onto an I/O based physical implementation. On the other hand, problems that are naturally formulated as message passing can easily use shared memory, by reading and writing to *mailboxes* [AlGo89]. Additionally, in the cases where *read-only* or *execute-only* access is required, explicit program synchronisation is unnecessary using the shared memory paradigm, but would be required under the I/O paradigm (due to the necessity to deal with message transmission).

Early work in the research community on IPC and the design of distributed systems was done in the context of tightly coupled multiprocessor systems, as opposed to loosely coupled systems situated across local and wide area networks. There were only a few exceptions to this trend, including DCS [Fa88, Fa75, Fa74, FaFe73, FaHe70], which is one of the first distributed systems to have implemented IPC for processes communicating across a local area network. Other early work included an implementation on a Newhall ring [MaPe75], which provided for a shared memory transfer of segments around the ring, and considered the value of a CMP implementing critical path protocol function. Note that both of these designs involved local area networks.

Berkeley UNIX successfully demonstrated that the IPC mechanisms can be interfaced to communication protocols, with the result that IPC can be done across any network. Berkeley UNIX IPC [Se86, LeFa86] is strictly based on the I/O paradigm, and does not significantly address performance issues. However, this effort has spurred a lot of interest among the research community to improve IPC over the communication networks in a number of ways. Some of these activities have resulted in systems such as Amoeba, Accent, and V. Most of these efforts are derived from UNIX in some sense, and use its I/O paradigm for IPC. Also, the I/O paradigm interfaces well with the existing protocols, and requires little change to host architecture and the underlying mechanisms of an existing operating system. Some of these improvements can summarised as follows:

Amoeba [ReSt88, ReTa87, TaMu86, MuTa85] is based on client workstations and server nodes, with a primary design goal of high performance, achieved in part with simple communications protocols and contiguous disk file allocation. It is interesting to note that the high performance in this context has meant ability to cope with some fraction of the rather modest maximum bandwidth of existing local area networks.

Accent [FiRa86, Ra86, RaRo81] uses *copy-on-write* semantics to avoid the copying of large data objects. This is more reasonable in a local environment than in a high-bandwidth internet, since a process must block to receive a copy of a small data object that has been modified. Its successor, Mach [RaTe88, YoTe87, Ra86] is designed to be more UNIX compatible. It is designed to be portable, at the expense of performance, and the virtual storage management is highly decoupled from the hardware architecture.

The extension of IPC across wide area networks is manifest in the evolution of proprietary mainframe network architectures, such as SNA and BNA. In these systems, the I/O mechanisms are also used. In BNA [Bu81, Un87b], programs communicate by standard file I/O to special *port* files. In SNA [GrHa83, IBM85c, IBM81, IBM83], programs communicate with one another as *logical units* (LU type 6), but a message based paradigm is used. In both cases, the underlying communication passes through I/O subsystem software, I/O processors (or channels), and communications front end processors.

There has been some research on exploring the shared memory paradigm for IPC over the network, exemplified by Memnet and LOCUS. LOCUS [PoWa85, WaPo83, PoWa81] is a UNIX variant, based on a distributed file system. Implementation problems were incurred in the original version of LOCUS, due to the use of a message based model, rather than shared memory [PoWa85]. To improve LOCUS IPC, UNIX System V communication primitives [Ba86] have been added, specifically, *messages*, *semaphores*, and *shared-memory* [Fl86]. Due to the difficulty of distributing the System V shared memory mechanisms (which are not reliant on hardware segmentation), this function was restricted to use on a single CPU. Current work [Fl87] is extending this support to provide a distributed shared memory to LOCUS.

In the case of Memnet [De88, DeSe88, DeFa85], processes communicate across a ring local network by reading and writing into shared memory. Memnet's emphasis has been on studying caching algorithms and their hardware implementations, to reduce the network traffic, and to avoid network latency for remote memory accesses. However, Memnet assumes a perfect communication medium with no errors, and does not allow virtual storage. Thus, the Memnet effort suggests viability of the basic idea, that is, the shared memory model can be used for IPC across the network. However, there are a number of research questions about the suitability of this model that need to be addressed. The purpose of this dissertation is to take this model a few steps further by including support for virtual storage, and providing mechanisms so that the shared memory model can work in a real wide-area internetwork with errors, congestion, and longer latencies. This involves evaluation of a number of tradeoffs in the areas of virtual storage management, and protocol design and implementation. The CapNet project [TaFa89] is also extending the Memnet work in similar directions, but with somewhat different emphasis. The Apollo DOMAIN [LeLe83, Ap85, Ap87] system also provides a shared memory interface on a LAN ring.

There are also other research groups that are starting to explore use of shared memory for IPC across network, including current work on Ivy, Mermaid, Shiva, Ra, and the Tapestry project [CaRe88, CaRu88]. The Ivy [Li86], Mermaid [LiSt88], and Shiva [LiSc89] research explores a shared virtual memory, with particular emphasis on providing page level coherency, and accommodating heterogeneous systems. Unlike Axon, the granularity of object transfer is the page, rather than the segment. The Ra [AuHu87] kernel project for the Clouds distributed system includes an investigation of distributed shared memory (DSM). This consists of exploring alternative address translation schemes and memory management hardware [RaKh88b], with particular emphasis on the object orientation of the system [RaKh88a].

5.2. Transport protocols

Currently, most transport protocols in wide use [MePe82] (such as TCP [Co88, St88a, RFC793], and the transport levels of X.25 [St87], SNA [Cy78, MaCh87, IBM85b, IBM85c], BNA [Un87a, Bu81], DCA [Un88, Sp81], *etc.*) are designed to be general purpose, and are therefore not optimised to different classes of applications requiring high performance. These protocols are sufficiently complex that their implementation must reside largely in the software of the host system and front end communications processor. They are also not specified to allow the separation of critical functions to be optimally implemented in hardware. The classical layering mechanisms (exemplified in the extreme by the ISO-OSI model), result in significant penalties if layer boundaries are strictly adhered to. The layered model also frequently results in a number of functions (such as error and flow control) replicated at multiple layers, when much functionality could be moved to the ends of the connection [LeLe83, SaRe84, PoWa81]

TCP has been the most popular and successful transport protocol in the current DOD Internet, and there have been constant efforts to make it better [Ja88, KaPa87, Zh86]. Although these efforts have significantly improved TCP performance, the fundamental changes dictated by the VHSI environment indicate that assumptions about the communications substrate made by TCP are becoming increasingly outdated. This justifies the consideration of completely new transport protocols.

Several other protocols have been proposed for use in higher performance versions of the DOD Internet. These include VMTP [Ch86b, Ch88a] and NETBLT [Cl87a, Cl87b]. VMTP is designed as a general purpose transport protocol, with emphasis on RPC (remote procedure call) and page level file access. Significant aspects of VMTP design applicable to this research include the packet grouping and selective retransmission based on bit vectors. NETBLT is a protocol designed for transport of large blocks of data with high throughput. The most significant aspect of NETBLT design applicable to this work is the decoupling of error and flow control. In addition it is based on a simple rate-based flow control mechanism, with selective retransmission determined by timers at the receiving end of a transmission. Both of these protocols group packets to increase efficiency of transport.

Another approach to the performance problem is to implement existing transport protocol mechanisms in hardware. This is manifest in the work on the express transport protocol (XTP) and the protocol engine (PE) [Ch86c, ChEi88, Ch88c]. While the goals for XTP are similar to those for ALTPs in VHSI, there are also some significant differences. The XTP approach is to take the existing protocols mechanisms, streamline the packet format for pipeline processing, and implement each step in the pipeline using a customised VLSI processor.

In summary, NETBLT, VMTP, and XTP have contributed a number of interesting ideas to the design of transport protocols, and they do improve upon TCP within the current DOD internet for the applications they were originally designed for. However, we believe that these protocols are not appropriate solutions for the VHSI model, because the underlying assumptions and trade-offs that these protocols are based on are very different in the VHSI model. Specifically, these assumptions include the quasi-reliability provided by an underlying connection-oriented internet protocol (MCHIP) [Pa89, MaPa89], and data rates that are several orders of magnitude greater than these proposed protocols assume. We argue that the transport protocols in the VHSI model should avoid end-to-end flow control as much as possible, and make the end-to-end error control application specific and independent of the end-to-end latency. In general, the transport protocols should be simpler, designed to be mostly implemented in VLSI, well integrated with the host architecture and operating system, and targeted for a specific class of applications. More detail concerning the incompatibility of these protocols extended to the VHSI environment, and the justification of ALTPs has been discussed in [BhSt88].

Acknowledgments

It may be somewhat premature to include an acknowledgement section in a dissertation *proposal*, but I feel the need emphasise the particular gratitude I feel toward several people.

I owe much to Gurudatta Parulkar, who is everything that one could possibly want in an advisor. He is an exceptionally supportive and motivating influence as this research progresses. I am indebted to to Jerome Cox, whose insight and guidance in the formulation of the focus of this research has been invaluable, as has been his general support. I would also like to thank the other members of the dissertation committee for their interest in this research: Gary Delp (IBM Yorktown), Martin Dubetz, David Farber (U. Penn.), and Jonathan Turner. Finally, I am indebted to my wife Kris, for the continuing understanding and the sacrifice of living with a graduate student.

References

[AlGo89] Almasi, George S. and Allan Gottleib, *Highly Parallel Computing*, Benjamin/Cummings, Redwood City, Calif., 1989.

[AnSc88] Andrews, Gregory R. and Fred B. Schneider, "Concepts and Notations for Concurrent Programming", in *Concurrent Programming*, Narian Gehani and Andrew D. McGettrick, eds., Addison-Wesley, Wokingham, England, Mass, 1988, pp. 3-69.

[Ap85] *Programming with System Calls for Interprocess Communication*, Apollo Computer, Inc., Chelmsford, Mass., 005696, rev 00, 1985.

[Ap87] *Network Computing System Reference*, Apollo Computer, Inc., Chelmsford, Mass., 010200, rev 00, 1987.

[ATT86a] *WE 32100 Microprocessor Information Manual*, AT&T Technologies, Allentown, Penn., 307-730 issue 2, 1986.

[ATT86b] *WE 32101 Memory Management Unit Information Manual*, AT&T Technologies, Allentown, Penn., 307-731, 1986.

[AuHu87] Aubán, José M. Bernabéu, Phillip W. Hutto, and M. Yousef Amin Khalidi, *The Architecture of the Ra Kernel*, Georgia Institute of Technology, School of Information and Computer Sciences, technical report GIT-ICS-87/35, Atlanta, 1987.

[AuLa83a] August, Martha and Sarah Lamb, *PRIME 50 Series Technical Summary*, Prime Corporation, Framingham, Mass., rev 19.1, DOC6904-191, 1st ed., 1983.

[AuLa83b] August, Martha, Alice Landy, and Marilyn Hammond, *PRIME System Architecture Reference Guide*, Prime Corporation, Framingham, Mass., rev 19.2, DOC3060-192P, 3rd ed., 1983.

[Ba86] Bach, Maurice J., *The Design of the UNIX Operating System*, Prentice-Hall, Engelwood Cliffs, N.J., 1986.

[Be72] Bensoussan, A., C.T. Clingen, and R.C. Daley, "The Multics Virtual Memory: Concepts and Design", *Communications of the ACM*, Vol.15 #5, ACM, New York, May 1972, pp. 308–318.

[BhSt88] Bhatia, Anil, James Sterbenz, and Guru Parulkar, *Comments on Proposed Transport Protocols*, Washington University Computer Science Department, technical report WUCS-88-30, St. Louis, Oct. 1988.

[Bu81] *Burroughs Network Architecture (BNA) Architectural Description, Operating and Programming Reference Manual*, Burroughs Corporation (Unisys), Blue Bell, Penn., 1132172, 1981.

[CaRe88] Campbell, Roy H. and Daniel A. Reed, *Unifying Shared and Distributed Memory Parallel Systems*, University of Illinois Department of Computer Science Department, UIUCDCS-R-88-1449, Urbana, Illinois, Aug. 1988.

[CaRu88] Campbell, Roy, Vincent Russo, and Gary Johnston, *A Class Hierarchical, Object-Oriented Approach to Virtual Memory Management in Multiprocessor Operating Systems*, University of Illinois Department of Computer Science Department, UIUCDCS-R-88-1459, Urbana, Illinois, Sept. 1988.

[CDC84a] *System Architecture: Cyber 180 Systems*, Control Data Corporation, Minneapolis, 204 137, 1984.

[CDC84b] *CDC Cyber 170 and Cyber 180 Volume I: Virtual State System Description, Functional Descriptions*, Control Data Corporation, Minneapolis, 60462090, 1984.

[CDC84c] *CDC Cyber 170 and Cyber 180 Volume II: Instruction Descriptions, Programming Information*, Control Data Corporation, Minneapolis, 60458890, 1984.

[Ch85] Cheriton, David, "Preliminary Thoughts on Problem-Oriented Shared Memory: A Decentralized Approach to Distributed System Design", *Operating Systems Review*, Vol.11 #4, ACM SIGOPS, New York, Oct. 1985, pp. 26-33.

[Ch86a] Cheriton, David, "Problem-Oriented Shared Memory: A Decentralized Approach to Distributed System Design", *Sixth International Conference on Distributed Computer Systems*, IEEE, 1986, pp. 190-197.

[Ch86b] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.16 #3, ACM SIGCOMM, New York, 1986, pp. 406-415.

[Ch86c] Chesson, Greg, "Protocol Engine Design", *Proceeding of the Usenix Conference*, 1986.

[Ch88a] Cheriton, David, "VMTP: Versatile Message Transaction Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-1045, Arlington Va., Feb. 1988

[Ch88c] Chesson, Greg, "XTP/PE Overview", Protocol Engines, Inc., PEI 88-63, Santa Barbara, Calif., 1988.

[ChEi88] Chesson, Greg, Brendan Eich, Vernon Schryver, Andrew Cherenson, and Al Whaley, "XTP Protocol Definition", Revision 3.1, Protocol Engines, Inc., PEI 88-13, Santa Barbara, Calif., 1988.

[ChGr88] Chesson, Greg, and Larry Green, "XTP - Protocol Engine VLSI for Real-Time LANs", *EFOC/88 Amsterdam*, Protocol Engines, Inc., PEI 88-53, Santa Barbara, Calif., 1988.

[ChGu88] Cheriton, David, Anoop Gupta, Patrick D. Boyle, and Hedrick A. Goosen, "The VMP Multiprocessor: Initial Experience, Refinements and Performance Evaluation", *IEEE*, 1988, pp. 410-421

[ChZw83] Cheriton, David, R. and Willy Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations", *Ninth ACM Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.17 #5, ACM SIGOPS, New York, 1983, pp. 129-140.

[ChZw85] Cheriton, David, R. and Willy Zwaenepoel, "The Distributed Process Groups in the V Kernel", *ACM Transactions on Computer Systems*, Vol.3 #2, ACM, New York, 1983, pp. 77-107.

[Cl82] Clark, David D., "Modularity and Efficiency in Protocol Implementation", *DARPA Internet Program*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-817, Arlington Va., July 1982.

[Cl85] Clark, David D., "The Structuring of Systems Using Upcalls", *Tenth ACM Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.19 #5, ACM SIGOPS, New York, 1985, pp. 171-180.

[Cl87a] Clark, David D., Mark L. Lambert, and LiXia Zhang, "NETBLT: A High Throughput Transport Protocol", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol.17 # 5, ACM, New York, 1987, pp. 353-359.

[Cl87b] Clark, David D., Mark L. Lambert, and Lixia Zhang, "NETBLT: A Bulk Data Transfer Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-998, Arlington Va., Feb. 1988.

[Co65] Comfort, Webb T., "A Computing System Design for User Service", *Proceedings of the Fall Joint Computer Conference*, Vol.27 Part I, AFIPS, Spartan Books, Washington D.C., 1965, pp. 619–626.

[Co88] Comer, Douglas E., *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice-Hall, Engelwood Cliffs, N.J., 1988.

[Cy78] Cypser, R.J., *Communication Architecture for Distributed Systems*, Addison-Wesley, Reading, Mass, 1978.

[De65] Dennis, J.B., "Segmentation and the Design of Multiprogrammed Systems", *Journal of the ACM*, Vol.12 #4, ACM, New York, Oct. 1965, pp. 589–602.

[De68] Denning, P.J., "The Working Set Model of for Program Behavior", *Communications of the ACM*, Vol.11 #5, ACM, New York, May 1968, pp. 323–333.

[De70] Denning, P.J., "Virtual Memory", *ACM Computing Surveys*, Vol.2 #3, ACM, New York, Sept. 1970, pp. 153–189.

[De88] Delp, Gary S., *The Architecture and Implementation of Memnet: A High-Speed Shared-Memory Computer Communication Network*, University of Delaware Department of Electrical Engineering, technical report #88-05-1, Newark, Del., May 1988.

[DeFa85] Delp, Gary S. and David J. Farber, *Memnet: An Experiment in High Speed Memory Mapped Local Network Interfaces*, University of Delaware Department of Electrical Engineering, technical report #85-11-1, Newark, Del., Nov. 1985.

[DeSe88] Delp, Gary S., Adarshpal S. Sethi, and David J. Farber, "An Analysis of Memnet: An Experiment in High-Speed Shared-Memory Local Networking", *SIGCOMM '88 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.18 #4, ACM SIGCOMM, New York, 1988, pp. 165–174.

[DiHa88] Diede, Tom, Carl F. Hagenmaier, Glen S. Miranker, Jonathan J. Rubenstein, and William S. Worley, Jr., "The Titan Graphics Supercomputer Architecture", *Computer*, Vol.21 #9, IEEE Computer Society, Los Alamitos, Calif., Sept. 1988, pp. 13–30.

[Fa74] Farber, David J., "Software Considerations in Distributed Architectures", *Computer*, Vol.7 #3, IEEE Computer Society, March 1974, pp. 31–35.

[Fa75] Farber, David J., "A Ring Network", *Datamation*, Feb. 1975, pp. 44–46.

[Fa88] Farber, David J., "Some Thoughts on the Impact of Ultra-High-Speed Networking on Processor Interfaces", University of Pennsylvania Distributed Systems Laboratory unpublished note, April, 1988.

[FaFe73] Farber, David J., Julian Feldman, Frank R. Heinrich, Marsha D. Hopwood, Keneth C. Larson, Donald C. Loomis, and Lawrence A. Rowe. "The Distributed Computing System", *COMPCON 73 Digest of Papers: Computing Networks from Minis to Maxis - Are They for Real?*, IEEE Computer Society, New York, 1973, pp. 31–34.

[FaHe70] Farber, David J. and Frank R. Heinrich, "The Structure of a Distributed Computer System – The Distributed File System", *First International Conference on Computer Communication*, Washington, D.C., 1970, pp. 364–370.

[Fa88] Farber, David J., "Some Thoughts on the Impact of Ultra-High-Speed Networking on Processor Interfaces", University of Pennsylvania Distributed Systems Laboratory unpublished note, April, 1988.

[FaPu88] Farber, David J. and Guru Parulkar, "Some Thoughts on the Impact of Ultra-High-Speed Networking on Processor Interfaces", Washington University Computer and Communications Research Center internal memo, 1988.

[Fe84] Fernbach, Sidney, "Applications of Supercomputers in the U.S. – Today and Tomorrow", in *Supercomputers: Design and Applications*, Kai Hwang (ed.), IEEE Computer Society Press, Silver Spring, Md., 1984, pp. 421–428.

[FiFr84] Filman, Robert E. and Daniel P. Friedman, *Coordinated Computing: Tools and Techniques for Distributed Software*, McGraw-Hill, New York, 1984.

[FiRa86] Fitzgerald, R. and R.F. Rashid, "The Integration of Virtual Memory Management and Interprocess Communication in Accent", *ACM Transactions on Computer Systems*, Vol.4 #19, ACM, New York, May 1986, pp. 147–177.

[Fl86] Fleisch, Brett D., "Distributed System V IPC in LOCUS: A Design and Implementation Retrospective", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.16 #3, ACM SIGCOMM, New York, 1986, pp. 386–396.

[Fl87] Fleisch, Brett D., "Distributed Shared Memory in a Loosely Coupled Distributed System", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol.17 #5, ACM SIGCOMM, New York, 1987, pp. 317–327.

[GrHa83] Gifford, J.P., P.J. Hansen, P. Homan, M.A. Lerner, and M. Pozefsky, "Advanced Program-to-Program Communication in SNA", *IBM Systems Journal*, Vol.22 #4, IBM Corporation, 1983.

[HeKu83] Hetherington, I.K. and P. Kusulas, "3B20D Processor Memory Systems", *Bell System Technical Journal*, Vol.62 #1 Part 2, AT&T Co., New York, 1983, pp. 207–220.

[Hw87] Hwang, Kai, "Advanced Parallel Processing with Supercomputer Architectures", *Proceedings of the IEEE*, Vol.75 #10, IEEE, New York, Oct. 1987, pp. 1348–1379.

[Hw89] Hwang, Kai, "Exploiting Parallelism in Multiprocessors and Multicomputers", in *Parallel Programming for Supercomputers and Artificial Intelligence*, Kai Hwang and Douglas DeGroot (ed.), McGraw-Hill, New York, 1989.

[IBM81] *Systems Network Architecture – Sessions Between Logical Units*, IBM Corporation, GC20-1868-2, 1981.

[IBM72] *IBM System/360 Model 67 Functional Characteristics*, IBM Corporation, Poughkeepsie, New York, GA27-2719-2, 1972.

[IBM78a] *IBM Time Sharing System Concepts and Facilities*, IBM Corporation, Poughkeepsie, New York, GC28-2003-6, 1978.

[IBM78b] *IBM System/38 Technical Developments*, IBM Corporation, Rochester, Minn., G580-0237, 1978.

[IBM81] *Systems Network Architecture – Sessions Between Logical Units*, IBM Corporation, GC20-1868-2, 1981.

[IBM83] *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2*, IBM Corporation, GC30-3084-1, 1983.

[IBM85a] *IBM System/38 Functional Reference Manual, Volume 1*, IBM Corporation, Rochester, Minn., GA21-3331-6, 1985.

[IBM85b] *Systems Network Architecture: Concepts and Products*, IBM Corporation, GC30-3072-2, 1985.

[IBM85c] *Systems Network Architecture: Technical Overview*, IBM Corporation, GC30-3073-1, 1985.

[IBM86a] *IBM 3090 Processor Complex: Functional Characteristics*, IBM Corporation, Poughkeepsie, New York, SA22-7121-3, 1986.

[IBM86b] *IBM System/38 Functional Concepts Manual*, IBM Corporation, Rochester, Minn., GA21-9330-6, 1986.

[IBM88a] *IBM System/370 Enterprise System Architecture Principles of Operation*, IBM Corporation, Poughkeepsie, New York, SA22-7022-0, 1988.

[IBM88b] *MVS/Enterprise System Architecture System Programming Library: Application Development – Extended Addressability*, IBM Corporation, Poughkeepsie, New York, GC28-1854-0, 1988.

[IBM88c] *IBM Application System/400 Technology*, IBM Corporation, Rochester, Minn., SA21-9540-0, 1988.

[In81] *Introduction to the iAPX 432 Architecture*, Intel Corporation, Santa Clara, Calif., 171821-001, 1981, reprinted in: *Tutorial on Advanced Microprocessors and High-Level Language Computer Architecture*, Veljko Milutinović (ed.), IEEE Computer Society Press, Washington, D.C., 1986, pp. 358–421.

[In83a] *iAPX 432 General Data Processor Architecture Reference Manual*, Intel Corporation, Santa Clara, Calif., 171860-004, 1983.

[In83b] *iMAX 432 Reference Manual*, Intel Corporation, Santa Clara, Calif., 172103-003, 1983.

[In86a] *iPSC System Overview Manual*, Intel Scientific Computers, Beaverton, Ore., 310610-001, 1986.

[In86b] *80386 Programmer's Reference Manual*, Intel Corporation, Santa Clara, Calif., 230985-001, 1986.

[In87] *80386 System Software Writer's Guide*, Intel Corporation, Santa Clara, Calif., 231499-001, 1987.

[In88] *80960MC Programmer's Reference Manual*, Intel Corporation, Santa Clara, Calif., 271081-001, 1988.

[In89] *i486 Microprocessor*, Intel Corporation, Santa Clara, Calif., 240440-001, 1989.

[Ja88] Jacobsen, Van, "Congestion Avoidance and Control", *SIGCOMM '88 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.18 #4, ACM SIGCOMM, New York, 1988, pp. 314-329.

[KaPa87] Karn, Phil and Craig Partridge, "Improving Round-Trip Estimates in Reliable Transport Protocols", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol.17 #5, ACM SIGCOMM, New York, 1987, pp. 2-7.

[KaSm87] Karin, Sidney and Norris Parker Smith, *The Supercomputer Era*, Harcourt Brace Jovanovich, Boston, 1987.

[Kl88] Kleinrock, Leonard, *et al*, "Toward a National Research Network", National Research Network Review Committee; Computer Science and Technology Board; Commission on Physical Sciences, Mathematics, and Resources; National Research Council; National Academy Press, Washington, D.C., 1988.

[Ko81] Kogge, Peter M., *The Architecture of Pipelined Computers*, McGraw-Hill, New York, 1981.

[LaNe78] Lauer, H.C. and R.M. Needham, "On the Duality of Operating Systems Structures", *Proceedings Second International Symposium on Operating Systems*, IRIA, 1978, pp. 17-43.

[Le65] Lett, Alexander S. and William L. Konigsford, "TSS/360: A Time-Shared Operating System", *Proceedings of the Fall Joint Computer Conference*, Vol.30, AFIPS, Thompson Book Co., Washington D.C., 1968, pp. 15–28.

[Le88] Leiner, Barry, ed., "Critical issues in High Bandwidth Networking", Defense Advanced Research Projects Agency – Gigabit Working Group, Network Working Group, RFC-1077, Arlington Va., Nov. 1988

[LeFa86] Leffler, Samuel J., Robert S. Fabry, William N. Joy, Phil Lapsley, Steve Miller, and Chris Torek, "An Advanced 4.3BSD Interprocess Communication Tutorial", *UNIX Programmer's Supplementary Documents, Vol. 1 (PS1), Virtual VAX-11 Version*, Computer Systems Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Calif., printed by USENIX, 1986, pp. 8-1–8-40.

[LeLe83] Leach, Paul J., Paul H. Levine, Bryan P. Douros, James A. Hamilton, David L. Nelson, and Bernard L. Stumpf, "The Architecture of an Integrated Local Network", *IEEE Journal on Selected Areas in Communication*, Vol. SAC-1 #5, IEEE, New York, Nov. 1983, pp. 842–856.

[LeMc89] Leffler, Samuel J., Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, Reading, Mass., 1989.

[Li86] Li, Kai, *Shared Virtual Memory on Loosely Coupled Multiprocessors*, Deptartment of Computer Science, Yale University, YALEU/DCS/RR-492, New Haven, Conn., Sep. 1986.

[LiSc89] Li, Kai and Richard Schaefer, *An Operating System Transforming a Hypercube into a Shared-Memory Machine*, Department of Computer Science, Princeton University, CS-TR-217-89, Princeton, N.J., April. 1989.

[LiSt88] Li, Kai, Michael Stumm, David Wortman, and SongNian Zhou, *Shared Virtual Memory Accommodating Heterogeneity*, Computer Systems Research Institute, University of Toronto, technical report CSRI-220, Toronto, Dec. 1988.

[LyFi81] Lynch, N.A. and M.J. Fischer, "On Describing the Behavior and Implementation of Distributed Systems", *Theoretical Computer Science*, Vol.13 #1, 1981, pp. 17-43.

[Ma76] Madison, A.W. and A.P. Batson, "Characteristics of Program Localities", *Communications of the ACM*, Vol.19 #5, ACM, New York, May 1976, pp. 285–294.

[Ma77] Matick, Richard E., *Computer Storage Systems and Technology*, Wiley-Interscience, New York, 1977.

[Ma82] Madison, Alan Wayne, *Characteristics of Program Localities*, UMI Research Press, Ann Arbor, Mich., 1982.

[MaCh87] Martin, James and Kathleen Kavanagh Chapman, *SNA: IBM's Networking Solution*, Prentice-Hall, Engelwood Cliffs, N.J., 1987.

[MaDo74] Madnick, Stuart E. and John J. Donovan, *Operating Systems*, McGraw-Hill, New York, 1974.

[MaPe75] Manning, Eric and R.W. Peebles, "Segment Transfer Protocols for a Homogeneous Computer Network", *Proceedings ACM SIGCOMM/SIGOPS Interprocess Communication Workshop (Operating Systems Review)*, Vol.9 #3, ACM SIGOPS, New York, 1975, pp. 65–73.

[MaPa89] Mazraani, Tony Y. and Gurudatta M. Parulkar, *Specification of a Multipoint Congram-Oriented High Performance Internet Protocol*, Washington University Computer Science Department, technical report WUCS-89-20, St. Louis, Aug. 1989.

[McDe87] McCormick, Bruce H., Thomas A. DeFanti, and Maxine D, Brown, "Visualization in Scientific Computing", *Computer Graphics Newsletter)*, Vol.21 #6, ACM SIGGRAPH, Baltimore, Md., Oct. 1987.

[MePe82] Meijer, Anton and Paul Peeters, *Computer Network Architectures*, Computer Science Press, Rockville, Md., and Pitman, London, 1982.

[Mo87] *MC68030 Enhanced 32-Bit Microprocessor User's Manual*, Motorola, Inc., Phoenix, MC68030UM/AD, 1987.

[MuTa85] Mullender, S.J. and Andrew S. Tannenbaum, "A Distributed File Service Based on Optimistic Concurrency Control", *Tenth ACM Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.19 #5, ACM SIGOPS, New York, 1985, pp. 51–62.

[Ni89] Nielson, Gregory M., ed., "Visualization in Scientific Computing", *IEEE Computer*, special issue Vol.22 #8, IEEE Computer Society, Los Alamitos, Claif, Aug. 1989.

[Or72] Organick, Elliot I., *The Multics System: An Examination of Its Structure*, MIT Press, Cambridge, Mass., 1972.

[Or83] Organick, Elliot I., *A Programmer's View of the Intel 432 System*, McGraw-Hill, New York, 1983.

[Pa87] Padua, David A., Vincent A. Guarna Jr., and Duncan H. Lawrie, *Supercomputer Programming Environments*, Center for Supercomputing Research and Development, University of Illinois, CSRD-673, Urbana, Illinois, June 1987.

[Pa89] Parulkar, Gurudatta M., *The Next Generation of Internetworking*, Washington University Computer Science Department, technical report WUCS-89-19, St. Louis, May 1989.

[PaTu89] Parulkar, Gurudatta M. and Jonathan S. Turner, "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment", *Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom '89)*, IEEE Computer Society, Washington, D.C., Vol.II, pp. 655–667, also, Washington University Computer Science Department, technical report WUCS-88-7, St. Louis, 1988.

[Pe84] Peterson, Victor, L., "Impact of Computers on Aerodynamics Research and Development", *Proceedings of the IEEE*, Vol.72 #1, IEEE, New York, Jan. 1984, pp. 68-79, reprinted in: *Supercomputers: Design and Applications*, Kai Hwang (ed.), IEEE Computer Society Press, Silver Spring, Md., 1984, pp. 462–473.

[Pl89] Plambeck, K.E., "Concepts of Enterprise Systems Architecture/370", *IBM Systems Journal*, Vol.28 #1, IBM Corporation, Armonk, New York, 1989, pp. 39–61.

[PoWa81] Popek, G., B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudidin, and G. Thiel, "LOCUS: A Network Transparent, High Reliability Distributed System", *Eighth ACM Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.15 #5, ACM SIGOPS, New York, 1981, pp. 169–178.

[PoWa85] Popek, Gerald J. and Bruce J. Walker, *The LOCUS: Distributed System Architecture*, MIT Press, Cambridge, Mass., 1985.

[Ra85] Rattner, Justin, "Concurrent Processing: A New Direction in Scientific Computing", *Proceedings of the National Computer Conference*, Vol.54, AFIPS Press, Reston, Va., 1985, pp. 157-166, reprinted as, Intel Technical Paper TP-9, Intel Corporation, Santa Clara, Calif., 280265-002.

[Ra86] Rashid, Richard F., "From RIG to Accent to Mach: The Evolution of a Network Operating System", *Proceedings 1986 Fall Joint Computer Conference*, IEEE Computer Society Press, Washington, D.C., 1986, pp. 1128–1137.

[Ra77] Ramamoorthy, C.V. and H.F. Li, "Pipeline Architecture", *Computing Surveys*, (special issue: "Parallel Processors and Processing"), Tse-Yun Feng, ed., Vol.9 #1, ACM, New York, Mar 1977, pp. 61–102.

[RaKh88a] Ramachandran, Umakishore and M. Yousef Amin Khalidi, *An Implementation of Distributed Shared Memory*, Georgia Institute of Technology, School of Information and Computer Sciences, technical report GIT-ICS-88/50, Atlanta, Dec. 1988.

[RaKh88b] Ramachandran, Umakishore and M. Yousef Amin Khalidi, *An Evaluation of Memory Management Structures for Object-based Systems*, Georgia Institute of Technology, School of Information and Computer Sciences, technical report GIT-ICS-88/53, Atlanta, Dec. 1988.

[RaLa87] Raveché, Harold J., Duncan H. Lawrie, and Alvin M. Despain, ed., "A National Computing Initiative: The Agenda for Leadership", *Report of the Panel on Research Issues in Large-Scale Computational Science and Engineering (SIAM workshop)*, SIAM, Philadelphia, 1987.

[RaRo81] Rashid, Richard and George G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel", *Proceedings of the Eighth Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.15, #5, ACM SIGOPS, New York, Dec. 1981, pp. 64–75.

[RaTe88] Rashid, Richard, Avadis Tevanian Jr., Michael Young, David Golub, Robert Baron, David Black, William J. Bolosky, and Jonathan Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures", *IEEE Transactions on Computers*, Vol.37, #8, IEEE Aug. 1988, pp. 896–908.

[Re82] Reid, Loretta Guarino, *Control and Communication in Programs*, UMI Research Press, Ann Arbor, Mich, 1982.

[ReSt88] Renesse, Robert van, Hans van Staveren, and Andrew S. Tannenbaum, "Performance of the World's Fastest Distributed Operating System", *Operating Systems Review*, Vol.22, #4, ACM SIGOPS, New York, Oct. 1988, pp. 25–34.

[ReTa87] Renesse, Robert van, Andrew S. Tannenbaum, Hans van Staveren, and J. Hall, "Connecting RPC-Based Distributed Systems using Wide Area Networks", *Seventh International Conference on Distributed Computer Systems*, IEEE, Washington, D.C. 1987, pp. 28–34.

[RFC791] "Internet Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-791, Arlington Va., Sep. 1981

[RFC793] "Transmission Control Protocol", *DARPA Internet Program Protocol Specification*, Defense Advanced Research Projects Agency – Information Processing Techniques Office, RFC-793, Arlington Va., Sep. 1981

[SaRe84] Saltzer, Jerome H., David P. Reed, and David D. Clark, "End-to-End Arguments in System Design", *ACM Transactions on Computer Systems*, ACM, New York, Vol.2, #4, Nov. 1984, pp. 277-288.

[ScGa89] Scalzi, C.A., A.G. Ganek, and R.J. Schmalz, "Enterprise Systems Architecture/370: An Architecture for Multiple Virtual Address Space Access and Authorization", *IBM Systems Journal*, Vol.28 #1, IBM Corporation, Armonk, New York, 1989, pp. 15–38.

[Se86] Sechrest, Stuart, "An Introductory 4.3BSD Interprocess Communication Tutorial", *UNIX Programmer's Supplementary Documents, Vol. 1 (PS1), Virtual VAX-11 Version*, Computer Systems Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Calif., printed by USENIX, 1986, pp. 7-1–7-25.

[SlKr87] Sloman, Morris and Jeff Kramer, *Distributed Systems and Computer Networks*, Prentice-Hall (UK) International, London, 1987.

[Sp81] *Sperry Univac Telecon System Description*, Sperry Univac (Unisys), UP-8455, 1981 Rev. 2.

[SpMo88] Sporer, Michael, Franklin H. Moss, and Craig J. Mathias, "An Introduction to the Architecture of the Stellar Graphics Supercomputer", *33rd IEEE Computer Society International Conference*, IEEE Computer Society, Los Alamitos, Calif., 1988, pp. 464–467.

[SpOr75] Spier, Michael J. and Elliot Organick, "The Multics Interprocessor Communications Facility", in *Software Systems Principles: A Survey*, Peter Freeman (ed.), Science Research Associates, Chicago, 1975, pp. 133–167.

[St87] Stallings, William, *Handbook of Computer Communication Standards, Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards,*, McMillan, New York, 1987.

[St88a] Stallings, William, Paul Mockapetris, Sue McLeod, and Tony Michel, *Handbook of Computer Communication Standards, Volume 3: Department of Defense (DOD) Protocol Standards,*, McMillan, New York, 1988.

[St88b] *Stellar Graphics Supercomputer Model GS1000 System Overview*, Stellar Computer, Inc., Newton Mass., MD-0001, 1988.

[St88d] Sterbenz, James P.G., *High Performance Host and Network Interface Architecture (request for comments)*, Washington University Computer Science Department, research note JPS-88-6, St. Louis, October 1988

[StPa89a] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: A High Speed Communication Architecture for Distributed Applications*, Washington University Computer Science Department, technical report WUCS-89-36, St. Louis, Sept. 1989, presented at the Fourth IEEE Communications Society Workshop on Computer Communications, Dana Point, California, Oct–Nov 1989.

[StPa89b] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: Network Virtual Storage Design*, Washington University Computer Science Department, technical report WUCS-89-13, St. Louis, May 1989.

[StPa89c] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: Application-Oriented Lightweight Transport Protocol Design*, Washington University Computer Science Department, technical report WUCS-89-14, St. Louis, Oct. 1989.

[St90] Sterbenz, James P.G., *Axon: Host–Network Interface Design*, Washington University Computer Science Department, technical report WUCS-90-7, St. Louis, March 1990.

[Su86a] "Network Services Guide", in *Networking on the Sun Workstation*, Sun Microsystems, Mountain View, Calif., 800-1324-03, Rev. B, 1986.

[Su86d] "Network File System Protocol Specification", in *Networking on the Sun Workstation*, Sun Microsystems, Mountain View, Calif., 800-1324-03, Rev. B, 1986.

[TaFa89] Tam, Ming-Chit and David J. Farber, "CapNet – An Alternative Approach to Ultra-High Speed Network", (abstract), submitted to: *ICC '89*,

[TaMu86] Tannenbaum, Andrew S., S.J. Mullender, and Robert van Renesse, "Using Sparse Capabilities in a Distributed Operating System", *Sixth International Conference on Distributed Computer Systems*, IEEE, Washington, D.C. 1986, pp. 558–563.

[Te87] Tevanian, Avadis, Jr., *Architecture-Independent Virtual Memory Management for for Parallel and Distributed Environments: The Mach Approach*, Carnagie-Mellon Department of Computer Science, technical report CMU-CS-88-106, Pittsburgh, Dec. 1987.

[Un87a] *Burroughs Network Architecture (BNA) Version 2 Capabilities Overview*, Unisys (formerly Burroughs), Blue Bell, Penn., 1182318, 1987.

[Un87b] *A Series I/O Subsystem Programming Reference Manual*, Unisys (formerly Burroughs), 1169984, 1987.

[Un88] *Distributed Communications Architecture (DCA) Technical Overview*, Unisys (formerly Sperry Univac), St. Paul, Minn., UP-9676 Rev. 1, 1988.

[WaPo83] Walker, Bruce, Gerald Popek, Robert English, Charles Kline, and Greg Thiel, "The LOCUS Distributed Operating System", *Proceedings of the Ninth Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.17 #5, ACM SIGOPS, New York, 1983, pp. 49–70.

[YoTe87] Young, Michael, Avadis Tevanian, Richard Rashid, David Golub, Jeffrey Eppinger, Jonathan Chew, William Bolosky, David Black, and Robert Baron, "The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System", *Eleventh ACM Symposium on Operating Systems Principles (Operating Systems Review)*, Vol.21 #5, ACM SIGOPS, New York, 1987, pp. 63-76.

[Zh86] Zhang, LiXia, "Why TCP Timers Don't Work Well", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol.16 #3, ACM SIGCOMM, New York, 1986, pp. 397-405.