Report Number: WUCS-89-36

1989-09-01

Axon: A High Speed Communication Architecture for Distributed Applications

James P.G. Sterbenz and Gurudatta M. Parulkar

There are two complementary trends in the computer and communication 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 programs requiring high bandwidth and predictable performance. However, the bottleneck in 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, whose novel aspects include: an integrated design of hardware,...
Read complete abstract on page 2.

Axon: A High Speed Communication Architecture for Distributed Applications

James P.G. Sterbenz and Gurudatta M. Parulkar

Complete Abstract:

There are two complementary trends in the computer and communication 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 programs requiring high bandwidth and predictable performance. However, the bottleneck in 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, whose novel aspects 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.

AXON: A HIGH SPEED COMMUNICATION ARCHITECTURE FOR DISTRIBUTED APPLICATIONS

James P. G. Sterbenz
Gurudatta M. Parulkar

WUCS-89-36

September 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 programs requiring high bandwidth and predictable performance. 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, whose novel aspects 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.

Presented at the 4th IEEE Communications Society Workshop on Computer Communications, Dana Point, California, October-November 1989.

An abridged version of this paper appears in the INFOCOM '90 Proceedings [StPa90b]. Minor revisions April 5, 1990.

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

AXON: A HIGH SPEED COMMUNICATION ARCHITECTURE FOR DISTRIBUTED APPLICATIONS

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

Gurudatta M. Parulkar
guru@flora.wustl.edu
+1 314 889 4621

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) [Pa90]. 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: lack of integration among host architecture, operating system, and communication protocols, performance bottlenecks in the existing end-to-end protocols and their implementation, and almost no support for the shared memory paradigm 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: [1] an integrated design of host and network interface architecture, operating systems, and communication protocols; [2] a network virtual storage facility which includes support for virtual shared memory across a network [StPa89a, StPa90a, StPa90c]; [3] a high performance, lightweight object transport facility which can be used by both message passing and shared memory mechanisms [StPa89b]; [4] a pipelined network interface which can provide a high bandwidth low latency path directly between the VHSI and host memory [St90].

This paper presents an introduction to the Axon architecture, and is organised as follows: Section 2 gives motivation for the investigation of high performance IPC, and sumarises problems with current implementations. Section 3 presents a description of the various Axon architectural components. Section 4 describes other related work.

2. Motivation

This section provides motivation for the Axon architecture by outlining the requirements of communications for future high performance applications. This is followed by a description of the limitations of current architectures in meeting these application requirements.

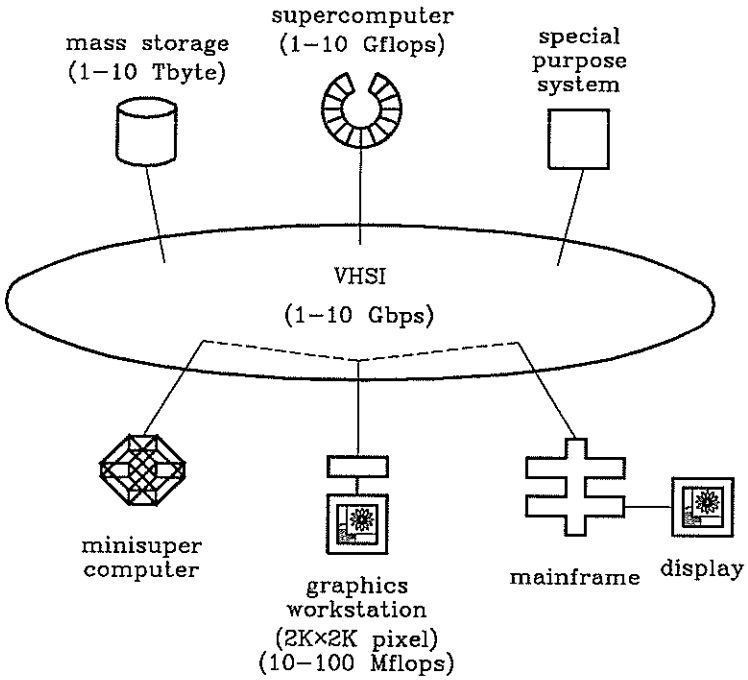

Figure 1: Target Environment

2.1. Target environment

Figure 1 depicts what we view as a typical future scientific and engineering computation environment. 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.

In this scenario, users have high performance graphics workstations with compute engines (such as the Ardent Titan or Stellar GS2000). 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) and to mainframe computers (such as a CDC Cyber 180/990 and IBM 3090). 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) 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. This requires the blocks of data to be delivered with low latency (sub-second), which coupled with the large size of data blocks 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 allow 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 high performance systems. Over the past few years significant progress has been made in the fields of communications and computer architecture, but the major bottleneck remains at the host–network interfaces.

Existing architectures support IPC as follows (Figure 2a): 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 is supported on tightly coupled systems with shared physical memory. Shared memory is not directly supported across a network, and must be mapped onto system level message passing. 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. The problems with this approach are the following:

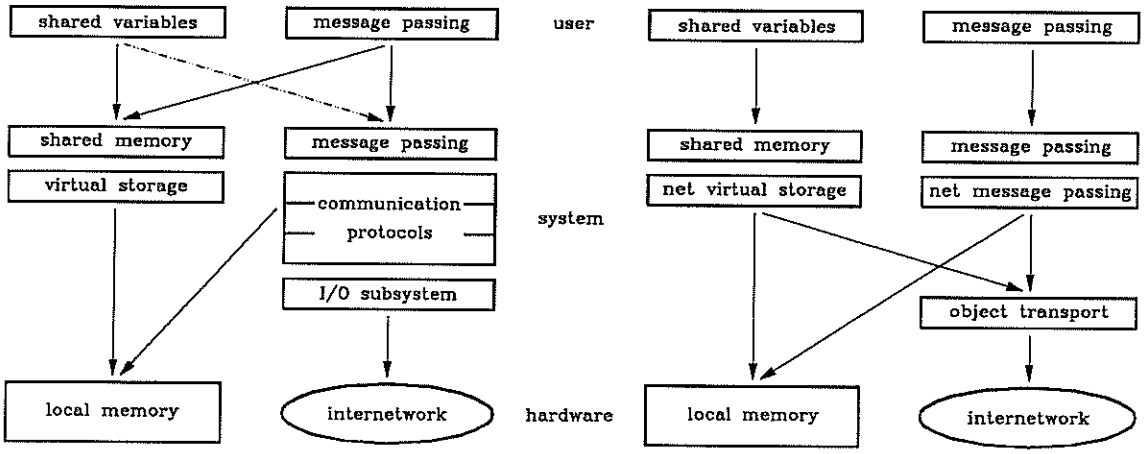

Figure 2: Interprocess Communication Implementation

- 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 modules 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), which results in inefficient control and synchronisation between the network and host (*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.

- 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.

- 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. 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.

- 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.

3. The Axon Architecture

The Axon architecture provides the support for efficient, high performance (high bandwidth, low latency) IPC across the VHSI.

This section provides an introduction to the Axon architecture. A model to provide an end-to-end high performance data path between applications is first presented. Then, IPC primitives are discussed within the framework of the VHSI environment. Finally, a description of significant Axon architectural components is presented. An example of the interaction of these components is given by describing the transport of a segment across the VHSI.

3.1. 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 high performance applications. This requires that the data path be pipelined to a fine granularity (bit/byte level rather than packet level store-and-forward).

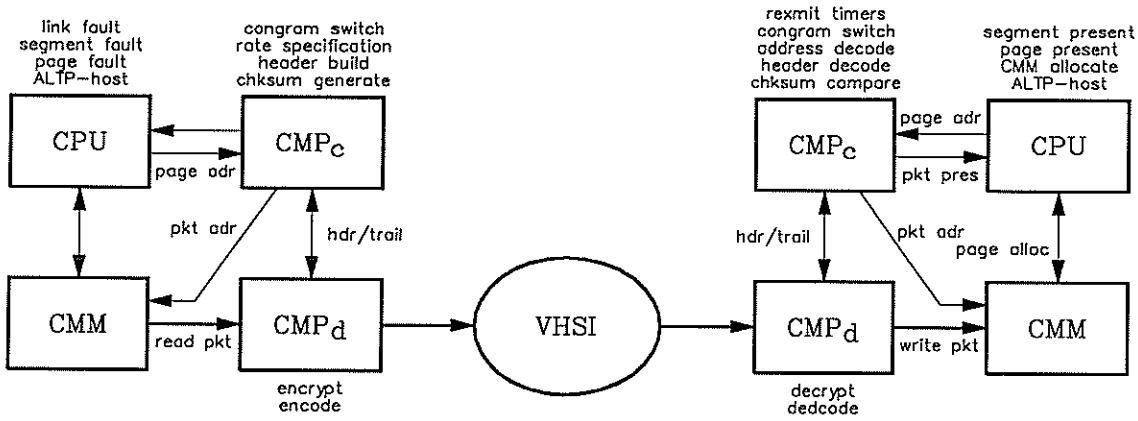

Figure 3: Axon Pipelined Communications Model

The *Axon pipelined communications model*, presented in Figure 3, provides the end-to-end data path between applications with the required IPC functionality. The CPUs use the random access port of a *communications memory module* (CMM), which is similar in concept to a VRAM. Data is transferred through the VHSI using the CMM serial ports. The transport protocol is implemented as an *application-oriented lightweight transport protocol* (ALTP), 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. To avoid any copying of data, the transferred object must be directly mapped into the process address space. This is done in conjunction with the usual address translation mechanisms by *network virtual storage* (NVS). The *communications processor* (CMP) 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 Axon model data pipeline configuration is:

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

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.

3.2. IPC in the Axon architecture

A *logical* view of the Axon protocol hierarchy is presented in Figure 4. 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.

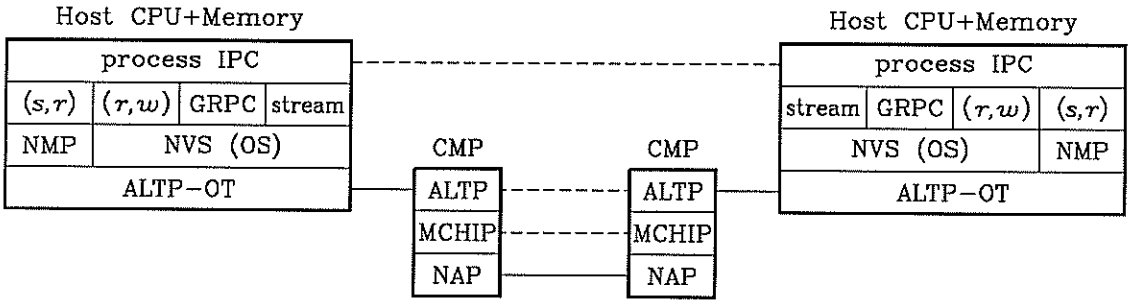

Figure 4: Axon Logical Protocol Hierarchy

IPC is supported with shared variable *read/write* (r, w) and message passing *send/receive* (s, r) primitives. Axon supports a more general form of RPC, in which the code and data segments can be located on arbitrary and independent hosts, with execution specified for an arbitrary host. This is referred to as *generalised remote procedure call* (GRPC). Conventional remote procedure call (RPC) [BiNe84] is thus a restricted form of GRPC. Finally, the special demands of high performance visualisation and imaging applications motivate an additional shared memory based IPC paradigm. Axon provides mechanisms to transfer *segment streams* at high bandwidth with low setup overhead. The performance advantage is that a single ALTP-OT call performs the request for all of the segments, and each segment can be transmitted when ready without the latency of a request. GRPC and segment streaming are described in more detail in [StPa89a].

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, or by the use of segment streaming. Support for message passing IPC is provided by a *network message passing* interface (NMP), which invokes the appropriate message transfer ALTP calls. This is illustrated in Figure 2b.

The transport mechanism is provided by an ALTP (application-oriented lightweight transport protocol) tailored for *object transfer*, called 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) [Pa90, MaPa89], and *network access protocols* (NAP).

[†] A congram combines the desirable features of a datagram with those of a (soft) connection. For the purposes of this paper, it can be thought of a connection with the added attributes of rapid setup and survivability in the presence of network failures.

Three components of the Axon architecture are essential to support IPC: system level IPC (NVS and NMP), transport protocol for IPC object transfer (ALTP-OT), and the host hardware architecture and communications processor (CMP). Each of these will now be discussed to provide an overview to the Axon architecture.

3.3. Network Virtual Storage

This section describes the data structures, segment types, and storage management policies for NVS. An operational description is deferred until Section 3.6, which presents an example of object transfer.

NVS extends the typical virtual storage mechanisms to include systems throughout the VHSI. A segmented programming model is used, with underlying paging to facilitate storage management, as in the Multics operating system [Be72, Or72]. NVS provides the ability to easily use the shared variables paradigm across the VHSI. Additionally, segments that are transported across the VHSI are mapped into the address spaces of processes by NVS. This eliminates the need to copy segments from intermediate system buffers into the process address space, resulting in lower latency and system overhead.

Data structures. NVS extensions allow the segments to be addressed when resident on a non-local host. This is accomplished by including a *host id.* field in either the virtual address (*network virtual address*), or 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. As the segment is returned, the appropriate page and segment descriptor presence bits are set, so that program execution can resume with the normal fault recovery mechanisms. The address translation data structures are presented in Figure 5. Address pointers and relative offsets are represented by arrows on solid lines, and other location information (*e.g.* disk *cylinder, track, record*) by arrows on lines with infrequent breaks. The copying of data structures (*e.g.* a segment) or fields (*e.g.* descriptor information) is represented by arrows on dashed lines. The alternate paths for the returning segment *s* labeled RS and AS correspond to the remote placement policies (see below).

The local storage management data structures are extended to allow the addressing of segments on other hosts. This is accomplished by adding a *host id.* field to the *known segment table* (KST), which holds the symbolic segment bindings. This is an index into the *per* process *known host table* (KHT), which holds the symbolic host name to address/path bindings. This binding is resolved by searching the *host address table* (HAT) for each host, which gets its binding by invoking an internet name server, using the *host name database* (HND). There are also tables (not shown in the figure) to assist in *n*-way IPC using multipoint connections. Depending on the method used for network-to-host object mapping, a packet presence bit vector may be in *page descriptor table* (PDT) entries. A TLB provides the typical performance benefits in avoiding table lookup most of the time.

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

Segments have attributes of *read, write, execute,* indicating the type of access allowed. These access bits in the segment descriptor may differ from the (more restrictive) capabilities that individual users possess, or the descriptors of individual processes. Code segments are assumed to be pure

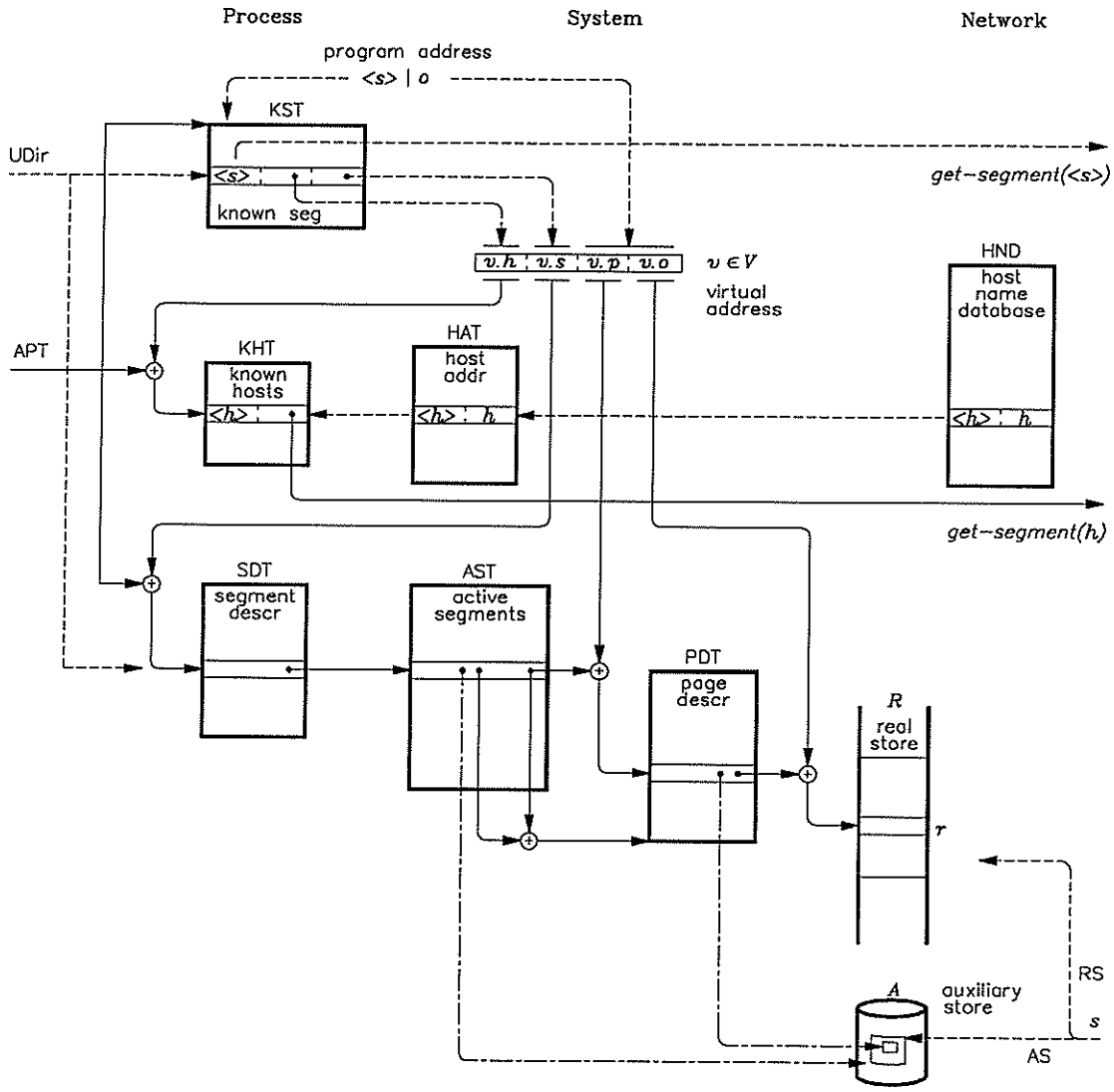

Figure 5: Network Virtual Storage Address Translation

(refreshable), and therefore always have access attributes of execute-only. Data segments may be readable and/or writable.

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

An entirely new policy, the *remote placement policy*, is used to determine where remote segments are placed while being used by the local system. These include real store (RS), auxiliary store (AS), a combination (RAS), or frame buffer (FB) placement, with a number of sub-policy options (swappable, nailed, *etc*). Due to the presence of segments from remote hosts, the conventional replacement policy

is affected. In particular, if RS remote placement is used, an entire segment worth of pages are placed in real store, some of which are not really in the process locality set. This can have significant impact on the availability of real storage, and indicates that the estimation of working sets must consider local and remote segments differently. NVS and its storage management policies are described in more detail in [StPa89a, StPa90a, StPa90c].

3.4. Transport protocol

This section describes the transport protocol for Axon, by first discussing ALTPs in general, and then describing the packet structure, flow control, error control, and retransmission policies used by ALTP-OT.

To address the problems mentioned above with general purpose complex transport protocols, applications using the VHSI are best supported by a set of simple ALTPs (application-oriented lightweight transport protocols) for various classes of applications [StPa89b, PaTu90]. 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 are designed so that their functionality can be split into *critical* and *non-critical* paths. The critical path consists of the data path and routine control functions, which are implemented in VLSI hardware to sustain data rates above 1 Gbps. The non-critical path function consists of everything else, specifically the control that must involve host interaction for connection setup and initiation of object transfer. By optimising the critical path functions, and by processing multiple packets in a single transport level operation, the *per* packet processing is sustained at the full VHSI data rate. For the protocol to be efficiently implemented in hardware, its design must be well integrated with the host architecture and operating system.

ALTPs are optimised to provide the kind of performance guarantees and functionality the specific applications need. The ALTP type used in Axon is designed to support IPC *object transfer*, (especially NVS segments), called ALTP-OT.

Packet structure and format. Information is transferred throughout the internetwork in packets. A structured group of packets corresponding to a single ALTP-OT semantic action is a *superpacket*, consisting of an initial control packet (which may also contain a small amount of 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 format of a data packet is presented in Figure 6.

MCHIP		connid	ALTP		reqid	segment (frame)		page (scanline)		pkt	data	cksum
Ø	type	c	Ø	type	q	$\|g\|$	k	$\|s\|$	j	i		Σ
2		2	2		1	1	1	2	2	1		2

Figure 6: ALTP-OT Data Packet Format

Each data packet corresponds to a fragment π_i of a page p_j of a segment s_k of a segment-group g, which are part of the superpacket σ. In the case of a video-graphics segment a page corresponds to a scanline, a segment to a frame, and a segment group to a complete image. The *connection-id.* and *request-id.* fields of the packet header allows the CMP to associate data packets with connections

set up by the corresponding control packet. Control packets have fields that are dependent on the type of operation.

The benefits of this packet/super-packet hierarchy is that most of the usual *per packet* control processing is only performed *per super-packet* in Axon. 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. 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, resulting in efficient coordination between ALTP-OT and conventional OS operations.

Flow control. ALTP-OT uses rate based flow control. When ALTP-OT opens a connection, it specifies attributes of the connection in terms of parameters such as average and peak bandwidth, and a factor reflecting the burstiness of the transmission. These parameters are used by all the intermediate systems, including various packet switches and gateways, as well as the endpoint hosts that the connection goes through, to make appropriate buffer and resource reservations. The rate specification is negotiated between ALTP-OT and the internetwork/network layers, to ensure that the requested rate does not exceed the capacity of internal network nodes (packet switches, gateways, and subnetworks). Furthermore, any adjustments to the rate specification should be infrequent, based on long term changes in application demands. It is assumed that the internet level below (MCHIP) [MaPa89, Pa90] has the functionality to support connections with specified bandwidth requirements, and furthermore, that the probability of packet loss, errors, and resequencing is very low, which is referred to as *quasi-reliability*.

This results in very simple flow control at the host–network interface, involving clocking packets at the specified rate, and can realistically be designed into the network interface hardware. As long as both ends transmit subject to the rate specification, the probability of packet loss due to buffer overruns is very low. Since the internet level is responsible for resource allocation, ALTPs are not concerned with congestion control, further simplifying the ALTP and network interface. This also means that the error control is decoupled from the flow control, which allows considerable simplification as described below.

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

- duplicate packets are discarded
- corrupted packets are discarded, and retransmission requested based on application need
- missing packets are detected by the expiration of a timer, and retransmission is requested
- packet sequence is irrelevant due to *sequence by placement* (as described below)

Note that due to the orientation of ALTP-OT to this object transfer, the handling of duplicate and out-of-sequence packets is considerably simpler and more efficient than would be the case for a general purpose transport protocol. Since data packets have sufficient header information to indicate the connection and request, and are placed directly into the proper location of target store, the overhead of sequence buffering is not necessary. The simplified error control of ALTP-OT can be efficiently implemented in VLSI hardware.

Retransmission strategies. Several options exist for the retransmission of packets: *location* of retransmission requests, *granularity* of retransmission and timer values, retransmission *fetch policy*, and *preemption* by the retransmission.

location: Requests for retransmission can originate from either the receiving end of the connection (RECV), or from the sending end (SEND). Since the receiving end is best able to estimate when packets should arrive [Cl87a], and since under some fetch policies retransmission may not be requested, the obvious choice is RECV, which is used by ALTP-OT.

granularity: The granularity of retransmission refers to how many missing packet events are accumulated before a request for retransmission is made. In a general purpose transport protocol, retransmission is typically based on selective or cumulative acknowledgement. Due to the knowledge of the super-packet structure of segment (groups) by ALTP-OT, a richer set of options can be explored, that are based on the granularity of the data structure transmitted. Four obvious possibilities for retransmission granularity exist: packet (PKT), page/scanline (PGE), segment/frame (SEG), and segment-group/image (GRP).

fetch policy: The retransmission strategies can also be classified by whether packets are always requested for retransmission, or only if a page is referenced that contains them. These are referred to as fetch policies due to the analogy with OS page fetch policies. In both cases timers will be necessary. If all packets corrupted or missing are retransmitted, this corresponds to *anticipatory retransmission* (AR) thus anticipating the future reference of all missing packets. In this case the timers indicate *when* a packet retransmission request should be made. If the only packets retransmitted are those corrupted or missing which are part of a page actually referenced, the policy is *demand retransmission* (DR), and assumes that a number of packets in the segment will not necessarily ever be referenced. In this case, the timers indicate *how long to wait* before a referenced packet is assumed to be missing, and thus retransmitted.

preemption: Since error control is *in-band*, packets retransmitted use the same connection and allocated bandwidth as the primary data stream. The relative priority of original data and retransmitted packets needs to be considered. The extreme cases are to allow all of the original request to flow before any of the retransmission requests are serviced, resulting in a *non-preemptive* (NP) policy, or to *preempt* (PE) the primary data stream and immediately retransmit.

The total number of strategies is the cross-product of these orthogonal sub-policies: location, granularity, fetch, and preemption. Since ALTP-OT is designed assuming RECV location, the remaining three sub-policies determine the overall strategy, *e.g.* PGE-DRPE indicates retransmit a page worth of packets (PGE) only when the page is referenced (DR), and preempt the primary data stream (PE).

In addition, some combination schemes are supported, such as 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 provides 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. Note that it is the object transfer orientation of ALTP-OT that allows optimisations such as this to be possible, in particular to reduce process blocking by dealing with retransmissions at PGE granularity. ALTP-OT can default to the appropriate strategy using parameters from a particular request. On the other hand, some intelligent IPC applications may wish to explicitly choose the strategy to be used.

Operations. The ALTP-OT requests and operations are listed below. More details on each one can be found in [StPa89b]. The example segment transfer presented in Section 3.6 provides an overview of ALTP-OT operation (for get-segment), as well as relating the operation of NVS.

Connection management	
join-ipc	join or establish multiway IPC connection
respecify-rate	alter rate specification for existing connection
leave-ipc	leave or terminate multiway IPC connection
Object receive	
get-segment	obtain copy of named segment from specified host
acquire-segment	obtain access to named segment for get-page
get-page	obtain copy of page from acquired segment
get-copy	obtain a permanent copy of segment from specified host
get-stream	receive segment stream from specified host/connection
receive-message	receive IPC message
retransmit-packets	request selective packet retransmission
Object transmit	
release-segment	release or return local segment copy (after get/acquire-segment)
release-page	release or return local copy of page (after get-page)
remote-execute	initiate execution of process on specified host
send-copy	send a permanent copy of segment to specified host
send-stream	transmit segment stream to specified host/connection
send-message	send IPC message
invalidate-segment	invalidate segment copy on another host

3.5. Host and network interface architecture

Host architecture. The Axon host architecture is presented in Figure 7. The Axon architecture gives the CMP (communications processor) direct access to memory, by interfacing the CMPs to the back end of a special dual-ported CMM (communications memory module). This is referred to as *memory interface architecture* (MIA dashed box in Figure 7). 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 that of VRAM. If all real storage is not CMM, the physical address space of the system must be partitioned between conventional and communications memory. Note that it is also possible to give 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, referred to as *interconnect interface architecture* (IIA dashed box in Figure 7).

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 CMM (MIA) or the processor–memory interconnect (IIA). More details on Axon host architecture configurations are presented in [St90].

Communications processor (CMP). To perform critical path functions at full VHSI data rate with no packet buffering, the CMP is organised as a pipeline, dynamically reconfigurable based on the ALTP type and options for a particular connection. The CMP block diagram is presented in Figure 8.

The *transmit data pipe* and *receive data pipe* are the main data paths of the CMP, and perform data encryption/decryption and format transformation. Data is clocked out the transmit data pipe from the CMM by the *rate control* logic, which is responsible for adhering to the rate specification for each connection.

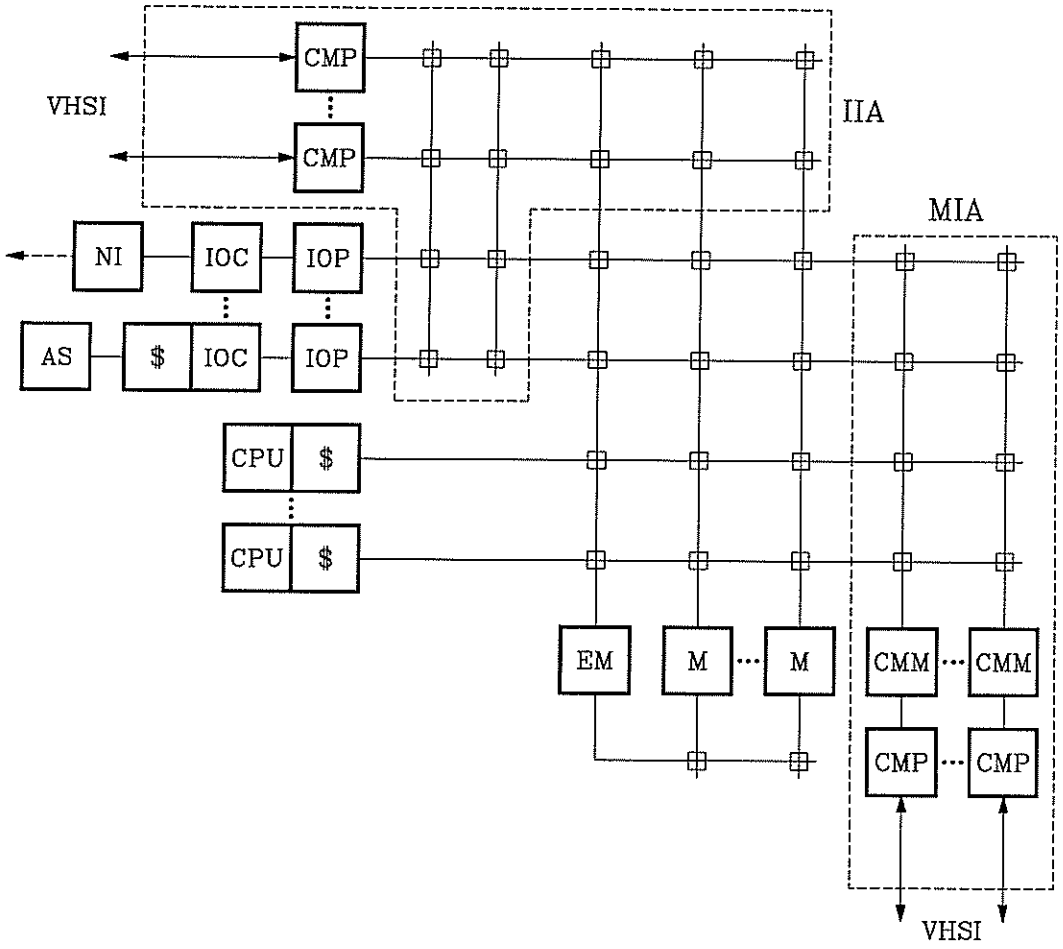

Figure 7: Axon Host Architecture

Connection multiplexing is handled by the *mux control* logic, with the *congram state registers* containing the state information for each connection/congram, allowing a fast hardware context switch of the CMP based on the connection id of each packet. The arrival of each packet is tracked by the *packet presence logic* which is responsible for determining when entire pages and segments have arrived, so that the appropriate PDT (page descriptor table) and SDT (segment descriptor table) presence bits can be set and the host program dispatched. The *error control* logic is responsible for recording missing/corrupted packets, and generating the appropriate retransmission requests.

Associated with the transmit data pipe, the *header build* logic constructs the appropriate header information from a template in the CMM, and inserts the proper packet identifiers (cq) and index (ijk). The *checksum generate* logic generates the checksum as the packet passes through the pipeline, and inserts it into the packet trailer. Associated with the receive data pipe, the *header decode* logic decodes the header to determine the connection and request ids (cq) for CMP configuration. It also determines the packet address in CMM from the packet index (ijk) and the base address of the page from the corresponding congram state register. The *checksum check* logic sums the packet as it passes through the pipe, and compares against the checksum field in the packet trailer. If the packet has been corrupted, it is discarded by clearing the appropriate packet presence state. Greater detail on the CMP design is presented in [St90].

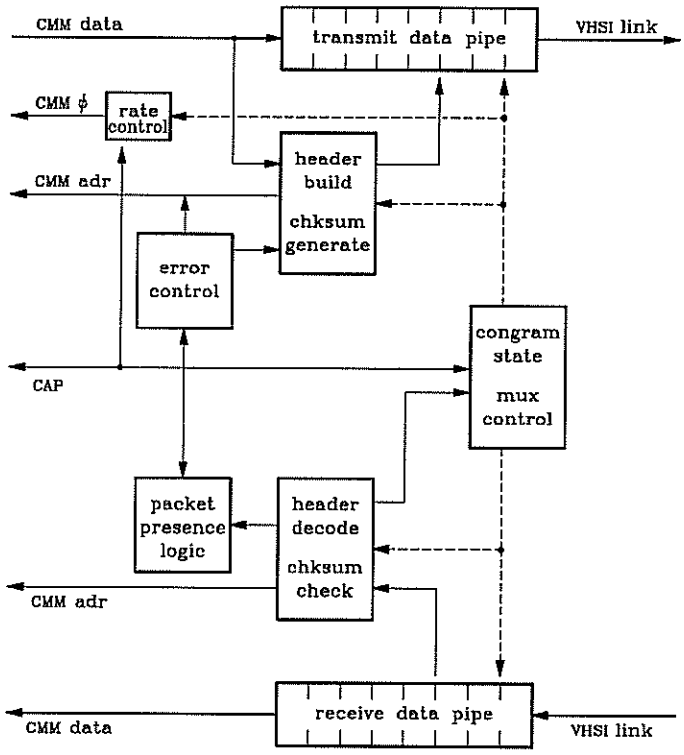

Figure 8: CMP Block Diagram

3.6. Summary of Axon object transfer

The operation and relationship of components in the Axon architecture will be introduced by the description of a segment transfer. Explicit references to Figure 9 in this discussion are enclosed in brackets: [[ ]]. Figure 5 can be consulted for the relationship between NVS data structures. Note that certain assumptions and policy choices have been made for clarity in this discussion.

An executing process has associated with it a virtual address space, which is a subset of the segments available to the user which owns the process. When a process refers to a remote segment, either explicitly by name, or *via* a GRPC, the appropriate segments must be transported from the desired system. The segment is located, either by an explicit reference to the segment and host name, or by resolution of the host name associated with the segment capability stored in the user context directory (UDir). The first time a segment is referred to symbolically, a *link fault* resolves the name and location, and adds the segment binding to the KST (known segment table), and host name binding to the KHT (known host table) for the process. This allows further *symbolic* references to avoid the overhead of searching the user context for segment attributes. In addition, an entry is added to the process SDT (segment descriptor table), which contains the process specific attributes of the segment. An entry is added to the system AST (active segment table), which contains the attributes of the segment common to all processes sharing the segment, if the segment is not already in use by another process. The mechanism for sharing is to have the SDT entries of multiple processes pointing to a single AST descriptor, which refers to a single instantiation of the segment. When a remote segment transfer is necessary, the transport mechanism is accomplished by ALTP-OT.

The critical path function of ALTP-OT is implemented in the CMP hardware [[ALTP-critical]], and includes the data path and routine control functions (error and flow control). The non-critical

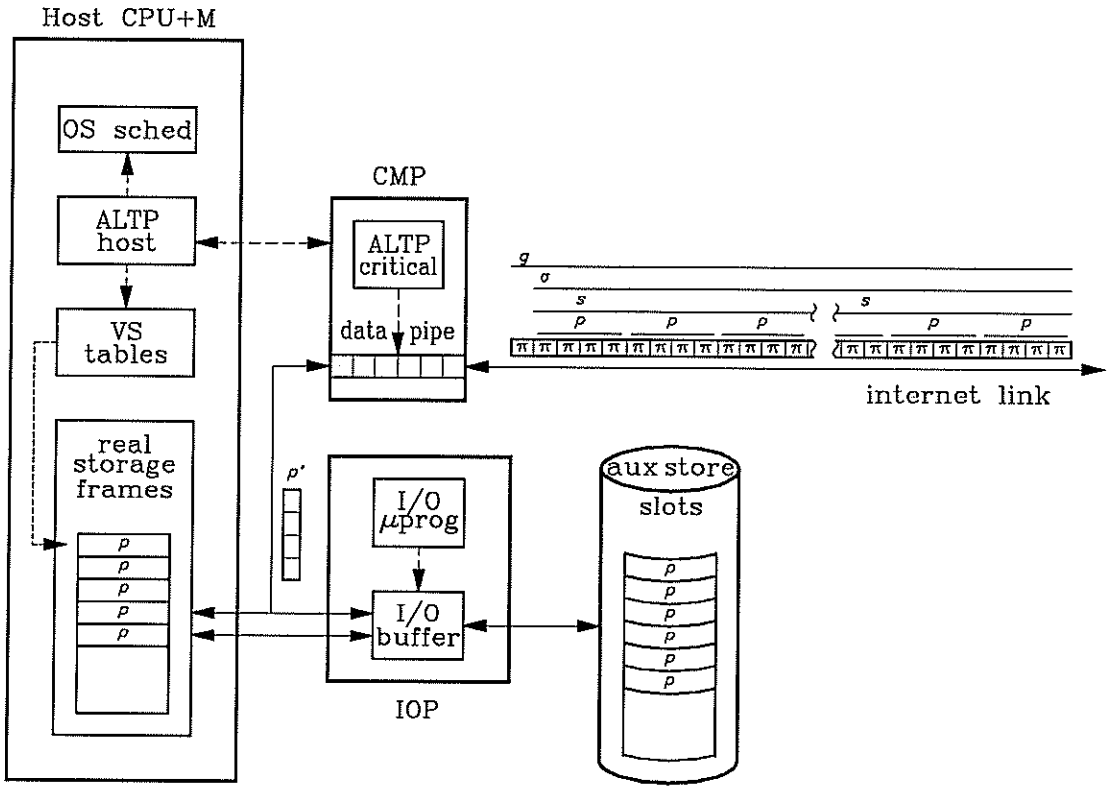

Figure 9: Interaction of Axon Components

part resides in the systems software on the host (or CMP assist processor [St90]), and is tightly integrated with the host architecture and operating system [[ALTP-host]]. In particular, the host portion of ALTP-OT must have direct access to operating system services such as the scheduler [[OS-sched]] through lightweight system calls, and be able to manipulate virtual storage management data structures [[VS-tables]].

The remote transfer is initiated by an ALTP-OT operation such as get-segment, which retrieves a segment from a remote host for local use. This requires a connection between the two hosts, thus ALTP-OT issues an open call to MCHIP which establishes the connection if not already present from a previous call. In addition, the CMP data pipeline is configured appropriately for the connection. ALTP-OT then sends the get-segment control packet out the VHSI link interface and through the internet, using the established connection.

At the remote end, the CMP receives and decodes the control packet at the internet link interface, and passes it to the host operating system. The normal mechanisms for locating the segment and authenticating the request are used. When the segment is found, locks are set (if necessary), and a copy of the segment is returned to the requesting host in a super-packet along the same connection. The data packets consist of fragments from each page of the segment, with an integral number of packets *per* page. Note that if multiple segments are defined within a segment access group, all of them are returned in a single super-packet. Thus the unit of structure is a superpacket $[\![\sigma]\!]$ consisting of a segment group $[\![g]\!]$ of segments $[\![s]\!]$ of pages $[\![p]\!]$ of packets $[\![\pi]\!]$.

At the local end, storage has been allocated for the returning segment(s), based on the estimated segment size $|\hat{s}|$ and remote segment placement policy in use (either [[real store frames]] or [[aux store slots]]). The data packets contain the actual segment size $|s|$, allowing adjustments to be made in

the estimated storage allocation. The header of each data packet also indicates the packet and page (and segment) number (i, j, k), as well as the connection and request ids. Since the connection has been established, the CMP pipeline configured, and storage allocated, packets are placed directly in storage according to the remote placement policy; no buffering of the data by the CMP takes place, the order of packet arrival is not significant (*sequence by placement*), and there is no involvement of MCHIP or the host software portion of ALTP-OT. The structure of data between the CMP and target memory is the page $[\![p']\!]$. Note that the peer-to-peer connection between ALTPs is *physical*, without the strict calling and data copying involved in the OSI or other layered models, and there is none of the overhead associated with multiple packet encapsulation/decapsulation between layers.

When certain events occur, the CMP issues a signal to the host software portion of ALTP-OT. For example, each time all of the packets of a given page have been received, the presence bit in the PDT (page descriptor table) must be set, and a lightweight system call must indicate to the low level scheduler that the process can be dispatched, as in the standard page fault recovery mechanism. When the entire segment has been received, the presence bit in the AST (active segment table) is set, and the ALTP-OT connection idles until the process ends, or an explicit leave-ipc is issued.

4. Related Work

This section briefly describes related projects and research efforts.

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, FaFe73, FaHe70], and the Newhall ring [MaPe75], but these both were in the LAN context.

There has been some research on exploring the shared memory paradigm for IPC over the network, exemplified by Memnet and LOCUS. LOCUS [PoWa85, WaPo83], a UNIX variant based on a distributed file system, has had UNIX System V communication primitives added, specifically, *messages*, *semaphores*, and *shared-memory* [Fl86], with current work extending this support to provide the *shared-memory* across a network [Fl87].

In the case of Memnet [De88, DeSe88], processes communicate across a ring LAN by reading and writing into shared memory. Memnet's emphasis has been on studying cacheing 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. The CapNet project [TaFa89] is extending the Memnet work in directions complementary to Axon, but with different emphasis. The Apollo DOMAIN [LeLe83] 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, and Ra. The Ivy [Li86, LiHu90], 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].

A segmented, paged virtual store was first implemented by Multics [Be72, Or72] on a GE-645 and the IBM 360/67 running TSS/360 [Co65, Le65]. 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 [CDC84] and Prime 50 Primos [AuLa83]. 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 [ScGa89] under MVS/ESA.

Additionally, systems that provide a segmented paged virtual store include the IBM AS/400 [IBM88] and System/38 CPF [IBM78], AT&T 3B series [HeKu83], Intel iAPX432 [In81] i486 [In89b] and 80960 [In88], and Motorola 68030 [Mo87].

Several protocols have been proposed for use in higher performance versions of the DOD Internet. These include VMTP [Ch86a, Ch88] and NETBLT [Cl87a, Cl87b]. VMTP is designed as a general purpose transport protocol, with emphasis on RPC 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, as well as 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) [Ch86b, ChEi88]. While the goals for XTP are similar to those for ALTPs in VHSI, there are also some significant differences. The XTP approach is to streamline existing protocols mechanisms and packet formats for pipeline processing, and implement each step in the pipeline using a customised VLSI processor.

The underlying assumptions and trade-offs that these proposed protocols are based on are very different than the VHSI model. Specifically, these assumptions include the quasi-reliability provided by an underlying connection-oriented internet protocol (MCHIP) [Pa90, MaPa89], and data rates that are several orders of magnitude greater than these proposed protocols assume. More detail concerning the incompatibility of current and proposed protocols extended to the VHSI environment, and the justification of ALTPs has been discussed in [BhSt88].

5. Conclusions

We have proposed a new host communication architecture for the distributed systems called Axon, which can support IPC with high throughput and low latency across the VHSI. The significant features of Axon are the network virtual storage facility, which includes support for virtual shared memory on loosely coupled systems, a high performance object transport facility which can be used by both message passing and shared memory mechanisms, and a pipelined network interface. Our emphasis in the design of Axon has been to provide a direct data path between communicating applications, using an integrated design of host architecture, operating systems, and communication protocols.

In this paper we have presented the overall design of Axon and its main components. Work is in progress on analytical and simulation models to evaluate these tradeoffs more rigorously, on a detailed design of the communication processor, and a prototype implementation of the architecture.

References

[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, GIT-ICS-87/35, Atlanta, 1987.

[AuLa83] August, Martha and Sarah Lamb, *PRIME 50 Series Technical Summary*, Prime Corp., Framingham, Mass., rev 19.1, DOC6904-191, 1st ed., 1983.

[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 P.G. Sterbenz, and Gurudatta M. Parulkar, *Comments on Proposed Transport Protocols*, Washington University Department of Computer Science, technical report WUCS-88-30, St. Louis, Oct. 1988.

[BiNe84] Birrell, A. and B. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, Vol.2 #1, ACM, New York, Feb. 1984, pp. 39–59.

[CDC84] *System Architecture: Cyber 180 Systems*, Control Data Corp., Minneapolis, 204 137, 1984.

[Ch86a] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM'86 Symposium (Computer Communication Review)*, Vol.16 #3, ACM, New York, 1986, pp. 406-415.

[Ch86b] Chesson, Greg, "Protocol Engine Design", *Proceeding of the Usenix Conference*, 1986.

[Ch88] Cheriton, David, "VMTP: Versatile Message Transaction Protocol", DARPA – Information Processing Techniques Office, RFC-1045, Arlington Va., Feb. 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 Barbera, Calif., 1988.

[Cl87a] Clark, David D., Mark L. Lambert, and LiXia Zhang, "NETBLT: A High Throughput Transport Protocol", *SIGCOMM'87 Symposium (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 – 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.

[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, Delaware, May 1988.

[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 (Computer Communication Review)*, Vol.18 #4, ACM, New York, 1988, pp. 165–174.

[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*, 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.

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

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

[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.

[IBM78] *IBM System/38 Technical Developments*, IBM, Rochester, Minn., G580-0237, 1978.

[IBM88] *IBM Application System/400 Technology*, IBM, Rochester, Minn., SA21-9540-0, 1988.

[In81] *Introduction to the iAPX 432 Architecture*, Intel Corp., 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.

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

[In89b] *i486 Processor Programmer's Reference Manual*, Intel Corporation, Santa Clara, Calif., 240486-001, 1989.

[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.

[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 of Communication*, Vol. SAC-1 #5, IEEE, New York, Nov. 1983, pp. 842–856.

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

[LiHu90] Li, Kai, and P. Hudak, *Memory Coherence in Shared Virtual Memory Systems, ACM Transactions on Computer Systems*, Vol.7 #4, ACM, New York, Nov. 1990, pp. 321–359.

[LiSc89] Li, Kai and Richard Schaefer, *An Operating System Transforming a Hypercube into a Shared-Memory Machine*, Department of Computer Science, Princeton University, technical report 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.

[MaPa89] Mazraani, Tony Y. and Gurudatta M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, abridged from: Washington University Department of Computer Science, technical report WUCS-89-20, St. Louis, Aug. 1989.

[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, New York, 1975, pp. 65–73.

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

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

[Pa90] Parulkar, Gurudatta M., "The Next Generation of Internetworking", *Computer Communication Review*, Vol.20 #1, ACM SIGCOMM, New York, Jan. 1990, pp. 18–43, also: Washington University Department of Computer Science, technical report WUCS-89-19, St. Louis, May 1989.

[PaTu90] Parulkar, Gurudatta M. and Jonathan S. Turner, "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment", *IEEE Network*, Vol.4 #2, IEEE, New York, March 1990, pp. 19–27, also: *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 Department of Computer Science, technical report WUCS-88-7, St. Louis, 1988.

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

[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.

[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 Corp., Armonk, New York, 1989, pp. 15–38.

[StPa89a] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: Network Virtual Storage Design*, Washington University Department of Computer Science, technical report WUCS-89-13, St. Louis, May 1989.

[StPa89b] Sterbenz, James P.G. and Gurudatta M. Parulkar, *Axon: Application-Oriented Lightweight Transport Protocol Design*, Washington University Department of Computer Science, technical report WUCS-89-14, St. Louis, Sept. 1989.

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

[StPa90a] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon: Network Virtual Storage Design", *Computer Communication Review*, Vol.20 #2, ACM SIGCOMM, New York, April 1990, pp. 50–65.

[StPa90b] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon: A High-Speed Communication Architecture for Distributed Applications", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, pp. 484–492.

[StPa90c] Sterbenz, James P.G. and Gurudatta M. Parulkar, "Axon Network Virtual Storage for High Performance Distributed Applications", *10th International Conference on Distributed Computing Systems*, IEEE, Washington D.C., June 1990.

[TaFa89] Tam, Ming-Chit and David J. Farber, "CapNet – An Alternative Approach to Ultra-High Speed Network", *International Conference on Communications '90*, IEEE Communications Society, Piscataway, New Jersey, April 1990.

[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, New York, 1983, pp. 49–70.