Table 3: Network Throughput by Connection [Mbps]

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

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

5 Conclusions

Several recent efforts have been underway to provide high performance host–network interface architectures, including the NAB (network adapter board) [4], the Nectar CAB (communication accelerator board) [3], and the protocol engine (PE) in support of XTP (express transport protocol) [2].

Axon is based on underlying assumptions and tradeoffs that are very different than these other efforts. Specifically, these include the *quasi-reliability* provided by the underlying congram-oriented internet protocol (MCHIP) and subnetworks that make resource reservations and provide guarantees on delay and packet loss, and the much higher data rates of the VHSI. Furthermore, there is a greater emphasis on the integrated design of host architecture, protocols, and operating systems. Finally, the network interface is designed so that no store-and-forward hops are necessary, providing a direct pipeline between the VHSI and memory.

We have presented the design of the Axon host–network interface error and flow control mechanisms. The simulation model has been carefully designed to represent function that can be easily transferred to VLSI design. The simulation results show that the error and flow control have the right performance characteristics. The overhead contributed to latency is minimal, as is the inter-connection interference. Furthermore, the rate control implementation is fair in its delivery of throughput to connections. It should be noted that these results have occurred even with extremely small packet sizes.

References

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

[2] Chesson, Greg, "XTP/PE Design Considerations", *IFIP WG6.1/6.4 Workshop on Protocols for High Speed Networks*, May 1989, reprinted as: Protocol Engines, Inc., PEI 90-4, Santa Barbara, Calif., 1990.

[3] Cooper, Eric C., *et al*, "Protocol Implementation on the Nectar Communication Processor", *SIGCOMM'90 (CCR)*, Vol.20 #4, ACM, New York, Sep. 1990, pp. 135–144.

[4] Kanakia, Hemant and D.R. Cheriton, "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors", *SIGCOMM'88 (CCR)*, Vol.18 #4, ACM, New York, 1988, pp. 175–187.

[5] Mazraani, Tony Y. and G.M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *INFOCOM'90*, Vol.II, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 450–457.

[6] Mazraani, Tony Y., *High Speed Internet Protocols and Resource Management in the Internet*, Wash. U. CS Dept., M.S. thesis, St. Louis, Aug. 1990.

[7] Parulkar, Gurudatta M., "The Next Generation of Internetworking", *ACM SIGCOMM CCR*, Vol.20 #1, ACM, New York, Jan. 1990, pp. 18–43.

[8] Parulkar, Gurudatta M. and J.S. Turner, "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment", *INFOCOM'89*, IEEE Comp.Soc., Wash., D.C., Vol.II, pp. 655–667.

[9] Sterbenz, James P.G. and G.M. Parulkar, "Axon: A High Speed Communication Architecture for Distributed Applications", *INFOCOM'90*, Vol.II, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 415–425.

[10] Sterbenz, James P.G. and G.M. Parulkar, "Axon Network Virtual Storage for High Performance Distributed Applications", *10th ICDCS*, IEEE Comp.Soc., Wash., D.C., June 1990, pp. 484–492.

[11] Sterbenz, James P.G. and G.M. Parulkar, "Axon: Application-Oriented Lightweight Transport Protocol Design", *ICCC'90*, ICCC, Narosa Publishing House, New Dehli, India, Nov. 1990, pp. 379–387.

[12] Sterbenz, James P.G. and G.M. Parulkar, "Axon: Host–Network Interface Architecture for Gigabit Communications", in *Protocols for High Speed Networks, II*, Marjory J. Johnson ed., IFIP, Elsevier (North-Holland), 1991, pp. 211–236.

[13] Sterbenz, James P.G., A. Kantawala, and G.M. Parulkar, *Axon Host–Network Interface Functional Simulation*, Wash. U. Computer and Communications Research Center, tech. report WUCCRC-91-02, St. Louis, May 1990.

parameters in the CSR (t_a, t_i, t_g), as well as count fields to which they can be compared (T_a, T_i, T_g). This is modeled in the Axon simulation by the use of 3 corresponding timers.

The `Rate Control` block (Figure 9) generates events that will trigger the transmission of a page of packets in accordance with the rate parameters. The request FIFO serves to smooth traffic among all connections, so that requests to read a page from the CMP are not made faster then they can be serviced.

In response to a segment transfer request, the active state begins for a connection. This causes the T_a field of the CSR be cleared, and the `a/i` bit to be set to 1 to enable counting. When this field reaches the value in the t_a field, the active state is terminated, and the idle state begins. This causes T_i to be cleared and the `a/i` bit to be reset to 0 to enable counting (and disable T_a counting). T_i is incremented in the same manner until t_i is reached, signaling the end of the cycle and the beginning of a new active state. This continues until the last page has been transmitted.

During the active state, pages may be transmitted. The CSR field t_g holds the number of count cycles between page generation. T_g is cleared at the beginning of each active cycle and is incremented until a match occurs. This causes the generation of a page request, which enters the page FIFO. At the end of each interpage generate cycle, T_g is cleared and begins counting again.

4.1 Performance Implications

There are two significant effects of the rate control scheme to be examined: fairness in throughput among connections and inter-connection interference.

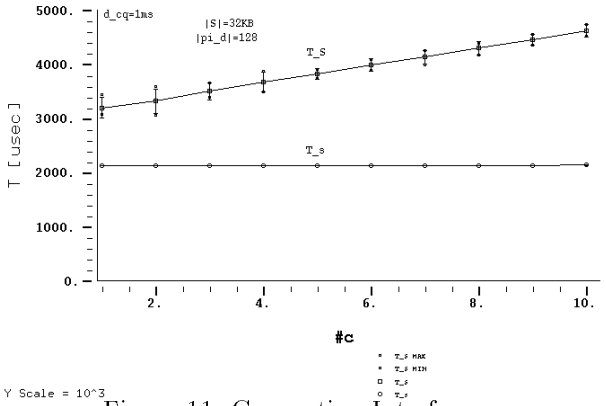

Figure 11: Connection Interference

First, interference between connections will be examined by the effect on the time a process is blocked, as is shown in Figure 11. In this case a number of connections on each host share 90% of a 1Gbps link. The rate specification of each connection is varied so that each process will get an equal share of bandwidth. The network latency is 100μs, and as before packet size is 32 bytes of data with 16 bytes of header and page size is 1 Kbyte. In this experiment segments are somewhat larger, averaging 32 pages.

As expected, the delay for a segment to completely return T_S increases linearly with the number of connections, due to the decreasing fraction of bandwidth received by processes. In addition to the 90% confidence interval bars, the minimum and maximum values are plotted in each case as small dots, indicating that there is little variance among requests.

The T_s (lower) curve indicates little increase with the number of connections, showing little interconnection interference due to the rate control (or other processing). Since pages are transmitted as a single burst of packets at maximum line rate, increasing the number of connections should minimally affect the time the process is blocked waiting for the first page. In particular, the only delay incurred in the CMP (other than the datapath pipeline) is due to the smoothing of the page request FIFO. Additional interference is possible in the host system and CAP (CMP assist processor) and is also reflected in this plot since the entire system is under simulation.

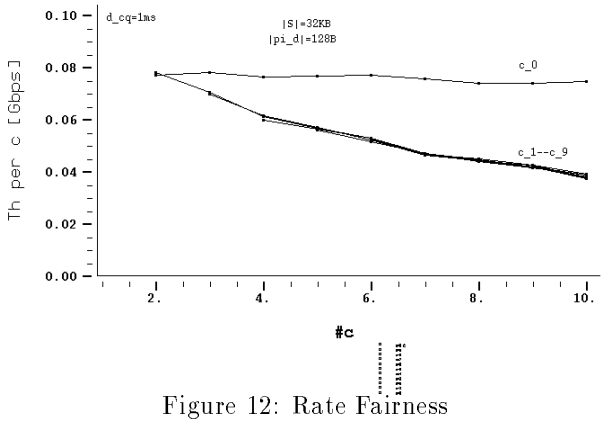

Figure 12: Rate Fairness

For a rate control scheme to be fair, all processes should receive the specified share of bandwidth, and in the case where rate specifications are identical across processes none should receive significantly different service. Figure 12 compares the throughput across connections as they are added. A single connection c_0 was given a peak bandwidth of $\lambda_p = 0.5$Gbps with additional connections evenly sharing an additional 0.5Gbps as they were added. The curve for c_0 is flat as desired, and the curves for the other connections decrease and track one another closely.

Table 3 indicates the network throughput *per* con-

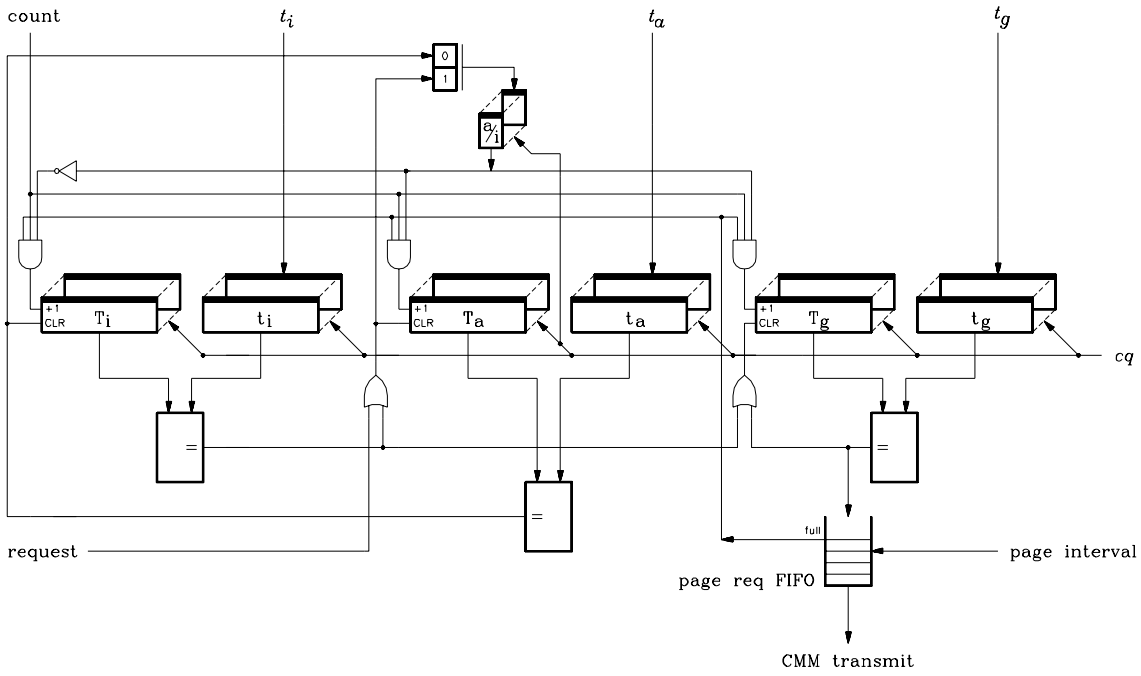

Figure 9: `Rate Control`

the advantage of knowledge by ALTP-OT of the page structure of segments, allowing processes to resume execution more quickly (the height of T_s related to the page size). Note that while the segments simulated are rather small, T_s is independent of segment size. This is the case since T_s is only dependent on the correct arrival of the first page in the segment, and retransmitted pages preempt the primary segment transmission in progress. As segment sizes increase, so will the difference between T_s and T_S, and thus the performance benefits of ALTP-OT.

4 Flow Control

This section describes the flow control strategy and its implementation in detail.

When ALTP-OT opens a connection, it specifies attributes of the connection in terms of a rate specification $\boldsymbol{r}$ consisting of parameters average bandwidth λ_a, peak bandwidth λ_p, and a burst factor B. These parameters can be translated into bandwidth and resource requirements based on a rate between the average and peak specifications. Since the connection is set up end-to-end, all the intermediate systems (packet switches and gateways) can make appropriate buffer and resource reservations [7]. The rate specification is negotiated between ALTP-OT and MCHIP (internetwork/network layers) to ensure that the requested rate does not exceed the capacity of internal network nodes (packet switches, gateways, and subnetworks). As a result, as long as both ends transmit subject to the rate specification, the probability of packet loss due to buffer overruns is very low. Due to the high bandwidth-×-latency product in the VHSI environment, dynamic adjustments to the rate specification should be at most infrequent, reflecting long term changes in application behavior.

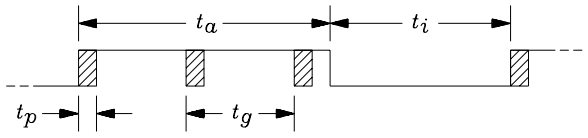

Figure 10: Rate Specification Parameters

The connection attributes can be transformed into time intervals (by the CAP) indicating when packets should be clocked out of memory by the CMP onto the network links: active period t_a, idle period t_i, and interpage generation time t_g, as shown in Figure 10, using the relationships [6]:†

$$t_a = \frac{B8|\pi||p|}{\lambda_p - \lambda_a} \qquad t_i = \frac{B8|\pi||p|}{\lambda_a} \qquad t_g = \frac{8|\pi||p|}{\lambda_p}$$

The three time fields are specified for the `Rate Control` logic. The active time is the length of a burst of page transmissions. The interpage generation time is the interval between the initiation of page transmissions, *i.e.* the inverse of the burst rate during the active time. The idle time is the interval between active states. The hardware implementation is to have fields for these three rate specification

†normalised by the page size $8|\pi||p|$ for per page rate rather than bit rate

packets, offset by the page number. The segment size is $|s| = 10$ pages, and a longer ramp consisting of the entire segment can be seen.

The segment sizes for connections 0, 1, and 2 are $|s| = 10$, 16, and 4 pages, respectively. Each ramp of pages corresponds to a segment on a unique connection. Note that since pages are always transmitted in single multi-packet bursts, the interleaving between connections is at the page level, rather than at the packet level.

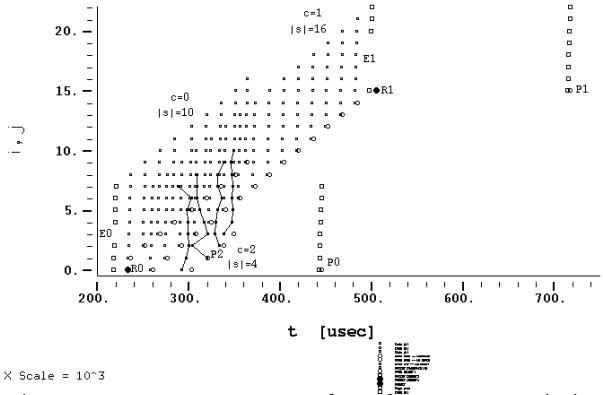

Figure 7: Page Presence and Packet Retransmission

Multiple connections with errors are shown in Figure 7. Three connections are used, with the same segment sizes as in the previous example. The network delay has been increased so that the retransmissions can be clearly seen at the right of the plot; this is why the x-axis begins at 200μs. Each connection has had a different type of error induced by the VHSI model. This plot is similar to Figure 6, except that the packets in pages suffering errors have been emphasised by larger squares.

Connection 0 ($|s| = 10$) contains a single packet loss in the first page. This is indicated by `E0` where the fourth packet is missing. Note that a circle indicating page presence is missing. Instead, when the receipt of the next page begins a retransmission request is made, indicated by the solid dot labeled `R0`. After a network round trip delay, the retransmitted packets are received and page presence marked, indicated by `P0`.

Connection 1 ($|s| = 16$) contains a single burst error of 4 packets dropped in the last page, indicated by `E1`. This causes a single retransmission request to be made for this page, indicated by `R1`. Note that since this is the last page in the segment, it was the last page timer T_p that triggered the retransmission, rather than the arrival of another page. After a round trip network delay, the page returns, indicated by `P1`.

Finally, connection 2 is subjected to enough delay variance that packet missequencing occurs. Instead of the linear ramps of packets elsewhere in the plot, the packets (connected by lines for clarity) can be seen to suffer considerable variance. The sequence within a page does not matter, however, and thus all of the pages are marked present, as shown by the usual page presence circles next to `P2`.

3.2 Performance Implications

A metric of primary concern is the time that a process is blocked waiting for the return of a data segment. The time interval between the process referencing the segment and its complete return to the local host is defined as T_S. More important is the time a process is *blocked*. This is measured by the interval between reference and the return of the *first* page in the segment, since execution can begin when this page is marked present; this is defined as T_s.

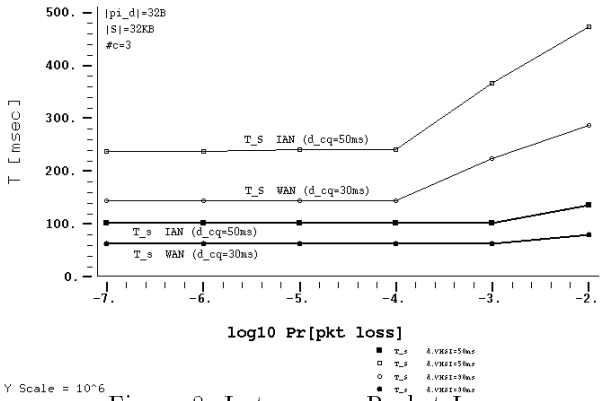

Figure 8: Latency *vs.* Packet Loss

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

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

The latency performance is quite flat, even up to extremely high error rates, which can be attributed to the use of selective retransmission. The T_S (upper) curves indicate the blocking that would occur if a process would have to wait for an entire segment to arrive (as is the case with common general purpose protocols). The T_s (lower) curves indicate

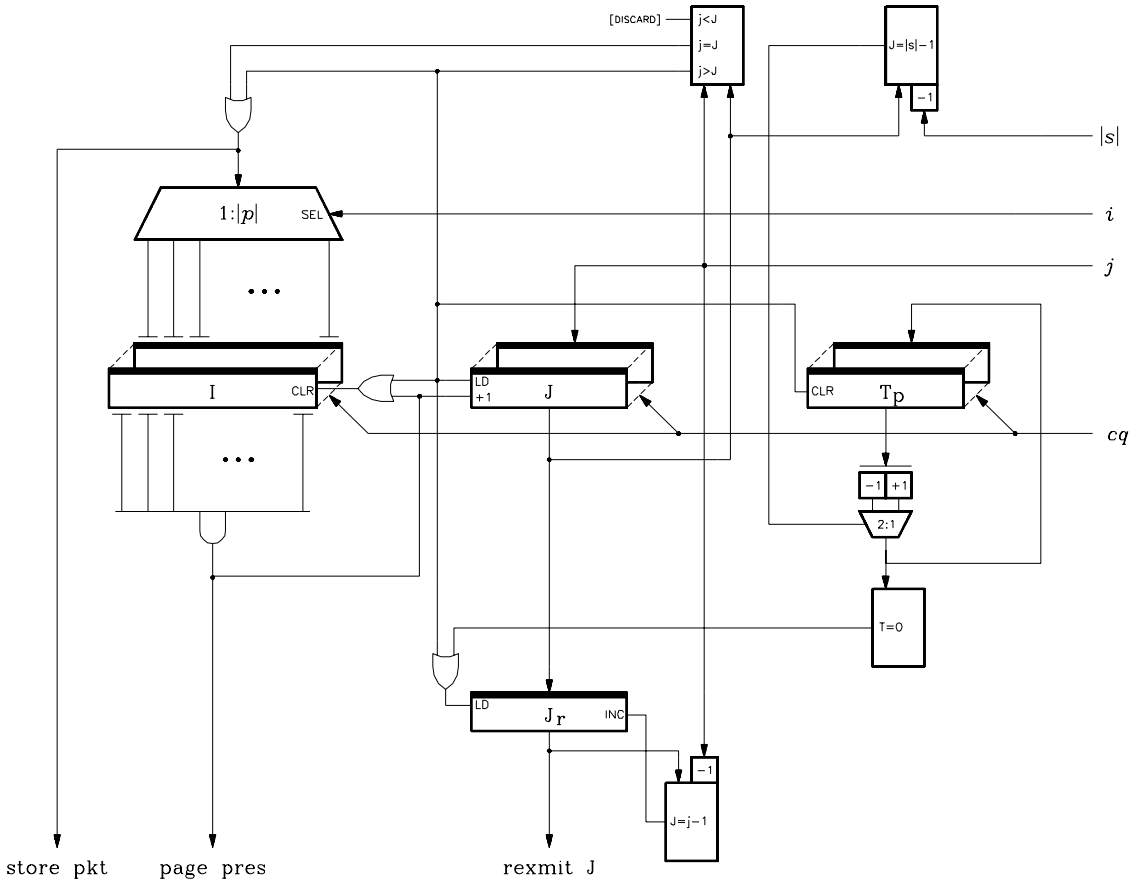

Figure 5: Page Presence

nection. This is done to allow for some delay in the network (either due to interleaving of packets in VHSI switches or due to statistical delays) before a packet is assumed lost. Special treatment is then required for the last packet in a segment.

This is accomplished by running a count for the last page. This is an estimate of the length of the time interval between pages. A value T_p is maintained as a CSR field $\mathtt{T_p}$, and incremented with each CMP counter clock cycle. Each time a new page arrives the value is reset to zero, except for the last page in the segment. The last page is detected by comparing the segment length field $|s|$ from the packet header with the current page index J. When the comparison succeeds, it is an indication that some packet in the last page has arrived. In this case the high order bit is set and T_p is then *decremented* with each clock cycle, until $T_p = 0$. This indicates that twice the time between the previous two pages has expired, and the vector I is checked to see if a retransmission request should be made.

The functioning of the Error control logic is introduced in Figure 6. In this case, three connections are shown The x-axis indicates the time of the events t. The y-axis indicates the packet id. Note that there are no errors in this case.

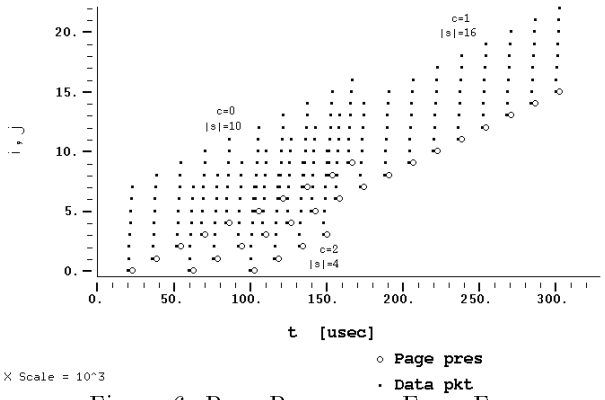

Figure 6: Page Presence – Error Free

Consider one of the three connections, $c = 0$. A request is made by the local host at $t = 0$, and passed through the VHSI. The remote hosts then returns the packets for a segment. Packets are plotted as dots, with the packet id i offset by the page number j. The page size is $|p| = 8$ packets and the segment size $|s| = 10$ pages. Thus the first ramp of packets beginning around 20μs shows packets 0 through 7 for page 0. When all of the packets have been received, the logic indicates page presence, shown by the large circle. Each successive page is shown as a ramp of

the corresponding page. The algorithm for missing packet and page presence detection follows:

```
do                          [initial state]
∀(i) I_{cq,i} ← 0           [packet presence vector]
J_cq ← 0                    [current page index]
T_{p_cq} ← 0                [page interval counter]
count-mode ← 1              [count T_p up]
od

receive(π)                  [receive packet]
cq ← π.cq                   [extract connection/request id]
i ← π.i                     [extract packet number]
j ← π.j                     [extract page number]
if j = J_cq                 [page match]
    do
    write(π)                [store packet in CMM]
    I_{cq,π.i} ← 1          [set packet presence]
    if ∏_{i=0}^{|π|-1} I_{cq,i} = 1   [all packets received]
        do
        present(cq, j)      [indicate page presence]
        J_cq++              [increment page index]
        ∀(i) I_{cq,i} ← 0   [reset packet presence]
        T_p ← 0             [reset page interval count]
        od
    od
if j > J_cq                 [packet from new page]
    do
    write(π)                [store packet in CMM]
    for x = j to J_cq − 1   [current to new page]
        rexmit(cq, x)       [request retransmit]
    J_cq ← j                [set index to new page]
    ∀(i) I_{cq,i} ← 0       [reset packet presence vector]
    I_{cq,π.i} ← 1          [set presence of new packet]
    T_p ← 0                 [reset page interval counter]
    od
if j < J_cq                 [packet form previous page]
    skip                    [discard packet]

if J_cq = |s| − 1           [last page in segment]
    do
    T_p ← 2T_p              [set estimate for last page]
    count-mode ← 0          [count down]
    od

if T_p = 0                  [expire page interval estimate]
    do
    rexmit(cq, J_cq)        [request retransmit]
    od
```

The `Error / Presence` block in the `CMP` performs *per* packet error control. Packets that have been corrupted (bad checksum) are dropped. All other data packets are passed into the `Page Presence` logic (within `Error / Presence`), which is shown in Figure 5. The simulation model is constructed with the same type of low level blocks as in this logic diagram. This is the implementation of the missing packet and page presence detection algorithm, which was specifically designed to result in simple hardware. The logic determines when all of the packets have arrived for a page, signaling the arrival of a complete page or the need to request a retransmission. A very simple mechanism is used to allow easy hardware implementation. The local page index J and packet presence I are maintained in the CSRs (congram state registers) as fields `I` and `J`, corresponding to the packet id (i, j) for each connection/request id combination. When a packet arrives the connection and request ids (c, q) are extracted by `Header Decode` to select the appropriate CSR, as is the packet id (i, j). This is modeled by simulation vectors I and J indexed by a single connection id.

For each packet, the `Page Presence` block is responsible for checking that the expected page number matches $j = J$, and that all bits in the packet vector are one: $\Pi_i I = 1$. If the page number increases before all the packets within the page have arrived, a retransmission request is made for the previous page (and any additional pages to catch up to the newly arriving page using the counter $\mathtt{J_r}$). The detailed state table follows, and corresponds to the cases in the algorithm above.

state		action		output
$\Pi_i I = 0$	$j = J$	$I_i \leftarrow 1$	–	[pkt pres]
$\Pi_i I = 1$	$j = J$	$I \leftarrow 0$	J++	`page pres`
–	$j < J$	–	–	[discard]
–	$j > J$	$I \leftarrow 0$	$J \leftarrow j$	`rexmit`

Packet sequence is irrelevant within a page, since the appropriate bit in the vector I is set for each packet. If packets are missequenced across page boundaries, however, an unnecessary retransmission request is made. This is because the CSR requires a fixed size field for maintaining state on incoming pages. Due to the quasi-reliability of congrams and spacing of pages multiplexed with other connections by `Rate Control` (§4), the probability of packet missequencing across page boundaries is much lower than within a page. It is possible to maintain state for a fixed number of pages. In particular if n page index fields are present in the CSR, false retransmits will not occur unless packets are missequenced across n page boundaries in a given connection. The logic is simpler, however, if only a single page index is maintained, and simulations have justified this approach.

Note that the need for retransmission is detected when the *next* page begins to arrive for a given con-

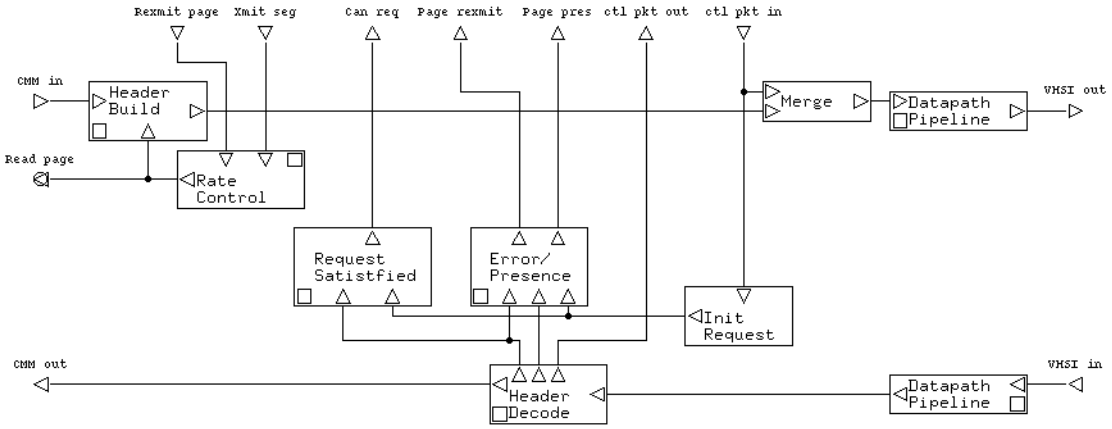

Figure 4: CMP

Error Control. The `Error/Presence` block determines when all of the packets have arrived for a page, signaling the arrival of a complete page or the need to request a retransmission. `Error/Presence` will be described in detail in Section 3.

Request Satisfied. The `Request Satisfied` block determines when *any* packet has arrived in response to a request. A bit in the CSR is used to record if any packet corresponding to the request has returned. When the first packet is recognised, the CAP is notified to cancel the request timer, and the CSR bit is set so that further CAP interrupts will not be made for a given request.

3 Error Control

This section describes the error control strategy and its implementation in detail.

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. This is justified given the assumption of quasi-reliability. To allow for simple error control which can be efficiently implemented in VLSI hardware, the error control scheme is designed for the particular ALTP (application-oriented lightweight transport protocol) using a quasi-reliable connection. The implication to the CMP design is that the error control modules are either designed for a particular ALTP, or when practical built from a generic set of control modules connected and configured appropriately.

The ALTP-OT packet handling for various error conditions are:

- duplicate packets are discarded
- corrupted packets are discarded, and selective retransmission requests are made
- missing packets are detected and selective retransmission requests are made (if the original then arrives, the retransmitted packet is treated as a duplicate and discarded at the receiving end)
- packet arrival sequence is irrelevant (within a given page); packets do not need to be resequenced since they are placed directly into the proper target location – referred to as *sequence by placement*

Note that due to the orientation of ALTP-OT to 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. In particular, the latency and buffer space associated with packet resequencing are not present with ALTP-OT since packets are placed directly into the correct location of application address space with zero store-and-forward operations.

3.1 Strategy and Implementation

Two conditions indicate that a packet retransmission request should be made: a received packet is determined to be corrupted (invalid header or checksum mismatch) or an expected packet is missing. Several policy choices in the error control mechanism can be considered. A detailed investigation of relative merits of various application oriented retransmission strategies is beyond the scope of this paper; a more detailed description of policy choices and related tradeoffs is provided in [11].

The policy implemented is as follows: Retransmission requests are based on receiving timers, since the local end is best able to estimate when packets should arrive. The granularity of selective retransmission is a page of packets, since this is the unit of of data that will cause CPU execution to block, but is still a fraction of an entire segment transmitted. Pages containing a corrupt or missing packet are always fetched, anticipating that they will eventually be referenced by the CPU. Finally, retransmitted packets preempt the primary data stream on the connection, since the local process may be blocked waiting for

in a given page, given the page base address. A page transmission request from the CMP `Rate Control` results in a page burst of packets at full VHSI link rate.

Incoming data packets are sinked, since it is the processing of the CMP, and not the actual destination of the data that is of concern for this paper.

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

It is important to note that the simulation model is constructed to reflect the actual hardware design of the Axon CMP. Since a major thrust of this research is to investigate the implementation of critical path function in hardware, simulation blocks are chosen carefully to represent function easily implementable in VLSI. Thus, high levels of simulation abstraction are not used within the CMP model.

In general, low level simulation primitives accurately reflect the functionality to be expected in a VLSI design library. There are a couple of notable exceptions, and thus the justification for their use and the ability to implement these functions in hardware must be discussed.

The simulation package provides convenient vector data structures which have been used to model the CSRs (congram state registers). State variables that are connection id dependent are modeled as vectors indexed by the connection id c. This is equivalent to associatively addressing a CSR register set by c, but with a model oriented to the simulation package. The operations used on simulation vector elements correspond exactly to hardware register operations in a CMP implementation.

The simulation package also provides timers, which are used by the rate control modules and provide a simple way to model the hardware based timers. These are actually implemented as fields in the CSRs, which are incremented periodically and compared to a terminal value. Thus the timer abstraction replaces an incrementing register field and comparator in each case.

The CMP simulation model is shown in Figure 4. The `Datapath Pipeline` consists of a model of delay based on the number of stages of the pipeline and the CMP clock cycle. Since the current effort is not concerned with the investigation of the datapath functions (such as encryption/decryption and format conversion), these pipelines model only delay. It is merely important to impose a realistic stage length, to allow time for the control functions to take place. Note that the critical *per* packet processing outside of the datapath (error and flow control) *is* of prime concern.

The transmit control blocks consist of `Header Build` and `Rate Control`. The receive control blocks consist of `Header Decode`, `Error/Presence`, and `Request Satisfied`. The `Init Request` block initialises some of the CSR fields related to `Rate Control` and `Error/Presence` when a new request is made for a particular connection. These blocks will now be discussed in greater detail.

Connection/congram multiplexing. *Congram state registers* (CSRs) hold all of the state information for each active connection, to allow rapid control and pipeline configuration changes for multiplexed connections using the CMP. There is a set of CSRs for both the transmitting and receiving side of the CMP. The multiplexing control logic uses the connection and request ids (c, q) to associatively address the appropriate CSR whenever an incoming data packet is received for processing, or when outgoing data packets are constructed. This results in a fast hardware context switch mechanism. Some of the CSR fields have the capability to increment or decrement, providing the functionality for hardware based timers on a *per* connection basis. As mentioned before, the CSRs are modeled by vectors addressed by connection id in the Axon simulation.

Rate Control. The `Rate Control` block generates events that will trigger the transmission of a page of packets for active connections in accordance with the rate parameters, and will be discussed in detail in Section 4.

Header Build. The `Header Build` block is responsible for building and inserting the necessary fields into data packet headers. The connection id c and segment size $|s|$ are passed through from the CAP when the segment transfer is initiated (through `Command Decode` and `Rate Control`). The page number j and packet number i is computed by a counter within `Header Build` as the packets flow through for each page.

Header Decode. The `Header Decode` block is responsible for extracting and decoding the appropriate fields from packet headers. The type of the packet is first determined from the corresponding header bit. Control packets are passed to the CAP for further processing. Data packet headers have the connection id c extracted to perform the CSR switch, simulated by using c as the index to corresponding vectors. The packet id (j, k) and segment length $|s|$ are extracted for use by the `Page Presence` block of the CMP. The base page address field in the CSR is then used in conjunction with the packet id to form the target CMM address, where the packet is written.

Data packets pass between the VHSI and the CMM, through the CMP. Connections between the CPU and CAP as well as between the CAP and CMP are used for control only. In particular, control packets pass between the VHSI and the CAP, through the CMP, with some other control signals connecting the CMP and CAP. Connections between the CPU and CAP consist only of control synchronisation for CPU requests and CAP interrupts.

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

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

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

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

The CPU sends requests to the CAP, such as for a remote segment fault. The CAP then builds the appropriate control packet and passes this to the CMP. The CAP also decodes incoming control packets from the CMP and takes the appropriate action. This may involve interrupting the CPU, for example when a link fault is required in response to a get-segment control packet received. Each of the CAP functions will now be described (with instruction cycle count given).

Request issuance (94 cycles): The CAP builds all control packets to be passed through the CMP to the VHSI. A get-segment control packet is built in response to the CPU issuing a remote segment fault. This request control packet is then passed to the CMP to be sent to the destination host. Note that other requests would be modeled in a similar manner.

In response to missing packets detected by the CMPa retransmit-packets control packet is built for the corresponding page, which is passed to the CMP for transmission.

Request retransmission timer (17 cycles): A request from the CPU also causes the initiation of the `Request Retransmit Timer`. If the CMP does not detect an associated return packet within the timeout interval, the request will be reissued and the timer restarted.

Page presence (2 cycles): In response to page presence detected by the CMP, the CPU is interrupted and passed the page number and connection id.

Control packet decode (33 cycles): All control packets received by the CMP are passed immediately to the CAP. In the case of an incoming get-segment request, the segment name and authentication are stripped from the control packet body and passed to the CPU for normal link fault resolution.

In the case that a retransmit-packets control packet has been received, the information to retransmit the necessary pages is passed to the CMP.

Return segment (19 cycles): When the link fault has been resolved, the CAP sets up the necessary state. The CMP is then passed the control information to begin the segment return.

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

A delay models the access time to start up the read from the sequential output port. The data packet length field is inserted in the packet for use by simulator probes in computing throughput. Individual read operations take place for a sequence of packets

`stamp` and `corrupt`. `Time stamp` is used to carry information on packet creation necessary for the simulator to compute latencies. `Corrupt` is set to indicate that the VHSI has introduced bit errors to the packet. The `control packet` and `data packet` data structures are children of type `packet`. The `control packet` data structure models the ALTP-OT control packets, including the operation type and segment id requested.

Table 2: `Data Packet` Structure

Field	Type	Documentation
(packet)	PKT	
|g|	INT	group size
k	INT	segment #
|s|	INT	segment size
j	INT	page #
i	INT	packet #
|pi_ d|	INT	data length
rexmit	INT	rexmited pkt

The `data packet` data structure is shown in Table 2. Note that all fields in the parent `packet` structure are inherited; these are indicated by the first line (packet) and not fully listed in the table. The standard data packet fields are present, as is a field indicating the data length to be used by some of the statistics gathering simulation probes. Each data packet is part of a larger structure, called a *superpacket*. While the host is involved with *per* superpacket operations (such as the transfer of a data segment), only the network interface is concerned with *per* packet processing (such as the handling of data packets within the superpacket). Each data packet is self-describing, consisting of information indicating the packet id within a segment group (ijk), as well as information on the length of variable size structure ($|g|$ $|s|$). This results in significant savings in CMP hardware complexity, allowing decisions on destination to be made based only on the header of each incoming packet, regardless of the sequence or multiplexing. Additionally, a bit indicates if the packet is an original or a retransmission.

2.3 Axon Model

The Axon system level simulation consists of two hosts connected by the VHSI, as shown in Figure 2.

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

Figure 2: `Axon` System Level Simulation

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

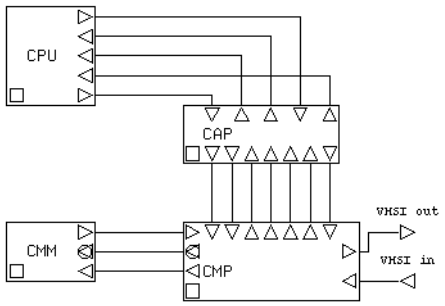

Figure 3: Axon `Host`

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

The Axon host model is presented in Figure 3. Each host contains a `CPU`, `CAP` (CMP assist processor), `CMP` (communications processor – the network interface chip), and `CMM` (communications memory module). The local host is responsible for generating requests based on the address reference trace model. The remote host receives requests and returns segments based on the request.

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

It is important to note the nature of the interfaces among the various components of an Axon host.

*performance internet protocol** (MCHIP) [5], and *network access protocols*.

This paper describes the Axon network interface from the perspective of its simulation, and in particular the implementation of error and flow control in hardware. Section 2 provides background on the simulation package that has been used to explore these mechanisms, and gives a brief overview of the Axon architecture as implemented by the simulator. Section 3 describes the error control mechanism in detail, presents the hardware design, and shows functional and performance simulation results. Section 4 describes the rate control scheme, along with its implementation and simulation. Section 5 discusses related work and gives concluding remarks.

2 Axon Simulation

This section describes the Axon architecture in the context of its simulation. First, a brief introduction to the BONeS™ simulation package [1] will be given to facilitate interpretation of the block diagrams in this paper. Then, the Axon data structures will be described. Finally, the Axon model will be described in a top-down manner, beginning with the highest level.

The simulation of the Axon architecture serves two purposes. First, it is a verification of design and the ability to implement key mechanisms in hardware. In particular, the simulation model of the CMP (communications processor) uses modules that correspond to functions available in a VLSI cell library. Secondly, the simulation provides a platform to evaluate design options and tradeoffs before a prototype system is built.

2.1 Simulation Package

The package used for the simulation of the Axon architecture is BONeS. It is a hierarchical discrete-event simulator, which provides for the construction of models through the definition of data structures and graphical construction of block diagrams. These data structures flow through and are modified by simulation blocks.

Data structures are defined in a hierarchical fashion, with children inheriting all the fields of the parent structure. The top level structure is of type `TRIGGER`, and is frequently used as a typeless signal for triggering the operation of blocks. A number of conventional data structures are provided with the package, such as `ARRAY`s, `VECTOR`s, and `SCALAR`s, all of which can be of type `INTEGER` or `REAL`. The Axon model has defined the additional data structures necessary to represent the different packet types.

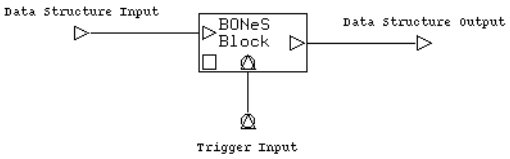

Figure 1: BONeS block

A block can either be a primitive module consisting of C source code or be constructed from other blocks. Blocks are represented by rectangles containing the name of the block (or other assigned label for clarity), as shown in Figure 1. Inputs and outputs are represented by triangles that point into or out of the block, to which signal connections are made. All inputs and outputs have an associated type, which must match (or be a child of) the type of data structures flowing through.

2.2 Data Structures

The Axon simulation requires the addition of packet data structures to the conventional structures provided in the BONeS library. The Axon data structures will be described and presented in a format similar to that used by the data structure editor of the BONeS package. Data structures always inherit all fields of their parent structure type, but these are not fully enumerated in the following tables. The `packet` structure contains the fields common to both control and data packets, as well as auxiliary simulation fields. Note that simulation fields do not affect the performance results of the simulations, since parameters specifying the actual packet header and data lengths are used by the simulator for computations of latency and throughput.

The `packet` fields (Table 1) correspond to the standard ALTP-OT packet fields [11], except for `time`

Table 1: `Packet` Structure

Field	Type	Documentation
time stamp	REAL	time stamp
ALTP c/d	INT	type
c	INT	connection id
q	INT	request id
corrupt	INT	corrupt data

*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.

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

HARDWARE BASED ERROR AND FLOW CONTROL IN THE AXON GIGABIT HOST–NETWORK INTERFACE[§]

James P. G. Sterbenz[†]
jpgs@watson.vnet.ibm.com
+1 203 783 4380

Anshul Kantawala[‡]
anshul@grad1.cis.upenn.edu
+1 215 898 9040

Milind M. Buddhikot[*]
milind@dworkin.wustl.edu
+1 314 935 4203

Gurudatta M. Parulkar[*]
guru@flora.wustl.edu
+1 314 935 4621

ABSTRACT

We have proposed a new architecture called **Axon** that meets the challenges of delivering high performance network bandwidth directly to applications. Its pipelined network interface must perform critical *per* packet processing in hardware as packets flow through the pipeline, without imposing any store-and-forward buffering of packets. This requires the design of error and flow control mechanisms to be simple enough for implementation in the network interface hardware, while providing functionality required by applications.

This paper describes the implementation of the Axon host–network interface, and in particular the hardware design of the critical *per* packet processing with emphasis on error and flow control. An extensive simulation model of the network interface hardware has been used to determine the feasibility and performance of hardware implementation of these functions.

1 Introduction

A new communication architecture called **Axon** [9] has been proposed for distributed systems. The primary goal of the Axon architecture is to support a high performance data path delivering high network 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 on loosely coupled systems [10]; (3) a high performance, lightweight object transport facility which can be used by both message passing and shared memory mechanisms [11]; (4) a pipelined network interface which can provide a high bandwidth low latency path directly between the network and host memory [12].

It is assumed that the underlying *very high speed internetwork* (VHSI) [7] is *quasi-reliable*, *i.e.* the probability of errors is low enough that protocols are *success-oriented*, designed with errors as the exceptional case (but they still must be handled). The probability of bit errors is extremely low due to the reliability of fiber optic links and modern fast packet switches; the probability of packet loss and missequencing is low due to the connection-oriented substrate providing resource reservation. These assumptions lead to a number of simplifications in the transport protocol design and its efficient high performance implementation, as described in this paper.

At the transport level, the VHSI model is best supported by a set of simple application-oriented lightweight transport protocols for various classes of applications [8]. These transport protocols can have their *critical path* functions implemented in the VLSI *communications processor* (CMP). The critical path consists of the data path and routine *per* packet processing allowing data to flow at VHSI rate once a transport operation has begun. It is thus important to devise error and flow control mechanisms that are simple enough to be hardware based in the network interface.

The transport protocol that is used by Axon is designed to support the transfer of data segments, referred to as *application-oriented lightweight transport protocol* for *object transfer* (ALTP-OT) [11]. ALTP-OT uses rate based flow control, and efficient end-to-end error control, optimised to include only what is necessary for object transfer.

The underlying internet/network layers of function are provided by a *multipoint congram-oriented high*

[*]Computer & Communications Research Center and Dept. of Computer Science, Bryan 405 – Box 1115, Washington University, One Brookings Drive, St. Louis MO 63130-4899

[†]IBM HPCC MS93 D34C/58W, 472 Wheelers Farms Rd., Milford CT; research performed at Washington University.

[‡]Distributed Systems Lab and Dept. of Computer and Information Sci., Univ. of Penn., Philadelphia PA, 19104-6389

[§]This work is supported in part by Bellcore, BNR, DEC, IBM, Italtel SIT, NEC, NTT, Southwestern Bell, SBC TRI, Synoptics, and NSF grant DCI-8600947