Implementation and Performance of an ATM Host Interface for Workstations

C. Brendan S. Traw and Jonathan M. Smith

Distributed Systems Laboratory, University of Pennsylvania
200 South 33rd St., Philadelphia, PA 19104-6389

ABSTRACT

This brief paper outlines our strategies for providing a hardware and software solution to interfacing hosts to high-performance networks. Our prototype implementation connects an IBM RS/6000 to a SONET-based ATM network carrying data at the OC-3c rate of 155 Mbps. We have measured application-to-network data rates of up to 130 Mbps.

1. Introduction

Despite rapid advances in workstation processor and memory subsystem performance, the next generation of high speed (Gbps), wide area networks threatens to exceed the data management capabilities of the hosts. To assist these hosts, specialized host interfaces are being developed at Penn, Bellcore [4], Carnegie-Mellon/Fore Systems [3], and elsewhere.

The host interface work at Penn has been centered on developing a high-performance host interface for workstation hosts in the AURORA Gigabit Testbed environment [2]. We have chosen to focus on workstations since we believe that they will be the predominant processor class connected to such networks.

1.1. Goals and Design Philosophy

One important outcome of the work is a high-performance host interface for IBM RS/6000 [1] workstations in the AURORA testbed, but our research goals are somewhat more ambitious and far-reaching. In particular, we wanted: (1) a hardware/software architecture which is flexible and allows experimentation with portions of the protocol stack; (2) a focus on architectural solutions to achieve good cost/performance, so our results scale across technology choices; and (3) low absolute cost, so that large-scale replication can be achieved.

We believe that the resulting host interface meets these goals. The design philosophy for our architecture is based on providing a "common denominator" set of services in dedicated hardware. All per cell activities such as CRC creation and verification, segmentation, and reassembly are performed in high density programmable logic. The host is responsible for all higher level activities. This combination meets our goals and provides an excellent balance between performance and flexibility.

Since we last reported on this work [5], we have been carrying this philosophy through to a realization. Here, we update our discussion of the architecture, detail the host software, and present some initial performance results.

2. Hardware

This Host Interface is comprised of two logical sections, each of which occupies a standard sized Micro Channel board in the RS/6000. These two logical sections are the Segmenter and the Reassembler. The brief description of the architecture below documents how the Class 4 adaptation layer is supported by the architecture described previously.

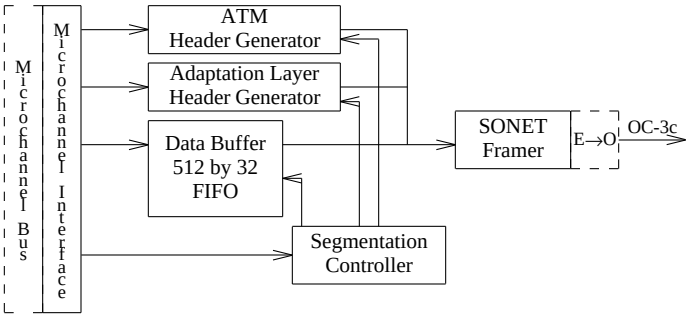

Figure 1: Segmenter

2.1. The Segmenter

A block diagram of the Segmenter is presented in Figure 1. When data is to be transmitted into the network, the virtual circuit identifier (VCI) to be used is loaded into the header generator. A multiplexing identifier (MID) is loaded

into the ATM adaptation layer (AAL) header generator if the data to be transmitted is Class 4. The host then sets up a streaming mode (an optimized form of DMA) transfer to move the data which is to be transmitted from a pinned buffer in host memory to the FIFO buffer on the Segmenter. While this transfer is occurring, the Segmenter produces the header check CRC and formats the control information into the appropriate ATM and AAL header formats. As soon as sufficient data has been placed into the FIFO buffer, the segmentation controller removes the data for the first cell from the FIFO buffer and appends an ATM header, AAL header and AAL trailer. If the cell is carrying Class 4 data, the payload CRC is calculated as the data is moved to the SONET framer and placed in the appropriate field at the end of the cell. This process is repeated until the FIFO buffer is drained.

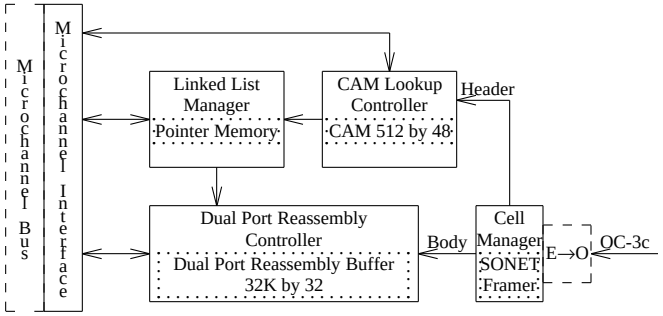

Figure 2: Reassembler

2.2. The Reassembler

The Reassembler is composed of four major subsections (Figure 2) which operate in parallel to form a cell processing pipeline.

The cell manager verifies the integrity of the header and payload (if the cell is carrying Class 4 data) of cells received from the network by the SONET framer. It extracts the VCI from the header and the MID and length from the AAL header. We currently ignore the 4-bit sequence number in the Class 4 adaptation layer as we believe it is insufficient to provide a reliable means for cell loss detection. The body of the cell is placed in a FIFO buffer for later movement in the dual port reassembly buffer.

The CAM lookup controller manages two CAMs which provide lookup support for a total of 256 simultaneous virtual connections and the reassembly of 256 datagrams. The host is able to flush undesired virtual circuits and datagrams from the reassembler through the CAM lookup controller.

A reference resulting from the CAM lookup operation is passed to the linked list manager (LLM). The LLM, as its name suggests, establishes and maintains a linked list data structure for each of the virtual circuit and datagrams that is being received. Data received from the network is placed at the end of the appropriate list while the host reads data from the beginning of the list.

The LLM allocates space in the reassembly buffer for data coming into the reassembler from the network and passes the location to the dual port reassembly controller. The cell body which was placed into the FIFO by the cell manager is removed and written into the reassembly buffer.

The host is able to read data from a particular virtual circuit or datagram by specifying a list reference to the LLM which determines where in the reassembly buffer the data is stored. The location is passed to the reassembly buffer controller which removes the data from the buffer for transfer into host memory over the Micro Channel.

3. Software

The current host interface support software consists of an AIX character-special device driver. We used AIX's capability to support dynamically-loadable device drivers; this allowed us to work despite the unavailability of kernel source code. The driver is configured into the system at boot time when the device is detected. The host interface presents a unique device identifier when probed, and this identifier is used to gather descriptive information (including driver routines) from a system object database. Configuration includes allocating addresses for use by the device; the device uses these addresses for its control registers and to support streaming mode transfers.

The interface is initialized when the device special file */dev/host{n}* is first opened (*n* is a small integer, 0 on our test system). Initialization consists of probing the device at a distinguished address which causes it to be reset, as well as performing various set-up operations for the device driver software. The operations currently include pinning the driver software's pages into real memory and allocating two 64K-byte contiguous buffers which are also pinned. After initialization, the device and driver are ready for operation; while routines for all appropriate AIX calls (e.g., *read(), ioctl(), etc.*) are provided, only *write()* is currently fully supported. The code fragment in the Appendix illustrates how a programmer would access the device; this particular fragment is taken from the measurement apparatus we used for the data of Section 4.2.

When the *write()* call is invoked on the device, data is copied from the user address space into one of the 64K buffers. When a status flag indicates the device is inactive, a streaming mode transfer is set up by initializing a number of translation control words (TCWs) in both the RS/6000 and in the Micro Channel's I/O Channel Controller (IOCC). The TCWs allow both the device and the CPU to have apparently contiguous access to scattered pages of real memory. This is illustrated in Figure 3.

After the TCWs and other state are set up, the device is presented with the data size and buffer's address, and the transfer begins. At this point, the driver marks the other buffer inactive and returns control to the user process. This combination of a hardware-provided state flag and double-

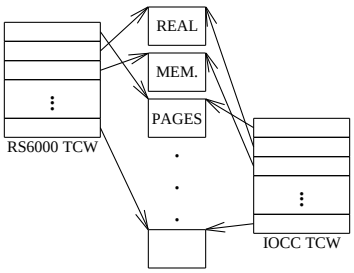

Figure 3: Illustration of TCW usage

buffering permits overlapped operation of the host interface and the host processing unit.

While this architecture supports overlapped operation, the copying between user and kernel address spaces is a major impediment to high-performance operation. The provision for TCWs in the IOCC allows large contiguous transfers directly to and from the address space of an AIX user process. We have a prototype device driver which supports such transfers, which we expect to be stable by January, 1992. Short tests have shown improved performance over what we report in Section 4.

Overlapped operation from user address spaces is somewhat trickier than from copies kept in kernel buffers, due to the risks inherent in concurrent access to shared state by the device and the process. Two obvious approaches are: (1) blocking the process until streaming is complete, and (2) trusting the process to not access the data (e.g., the process could do its own double-buffering). The first approach prevents a single process from using the hardware's capability for overlapped operation. This seems unwise, since most applications use the CPU to transform data which travels to and from the network. The second approach assumes too much, and could cause crashes with inconsistent kernel data. A third approach is to force the process to block (cease execution) when it accesses a ''busy'' buffer. In this way, ''well-behaved'' processes can achieve maximum overlap, while AIX is protected from the indiscretions of ''poorly-behaved'' processes. This is easily accomplished by tagging the active buffers TCW entries with ''fault-on-write''; the process is then blocked until the streaming transfer is complete and the page fault can be resolved. This combines the good features and removes the complications of the other two schemes, and is the approach we are currently pursuing.

4. Performance Measurements

4.1. Hardware

As of November 1991, the segmenter has been prototyped except for the AAL header generator; the reassembler is more than half done. At this stage, performance measurements of the hardware and software have been made:

- Header generation in the Segmenter requires 5 clock cycles (250 ns).
- In the Reassembler, the cell manager is able to verify cell integrity (CRC check) and extract the control fields in one cell time (2.6 μs).
- The longest per cell operation performed by the CAM lookup Controller requires 11 clock cycles (550 ns).
- From simulation results, it appears that the longest per cell linked list operation requires 12 clock cycles (600 ns).

Since multiple cell managers can be used in parallel, it would appear that the bottleneck of the reassembler pipeline is the LLM. Thus, there is a cell processing latency of about 4 μs and an overall bandwidth of about 700 Mbps.

By far the worst bottleneck in the hardware portion of the system is the Micro Channel bus. Utilizing the fastest mode of data transfer on the bus, 32 bit streaming, we have achieved a sustained transfer rate of 130 Mbps with one version of our device driver (which unfortunately crashes the system intermittently; a more conservative driver was used for the measurements of Table I). A typical transfer cycle on the RS/6000 model 320 would be as follows:

- 1.6 μs to transfer sixteen 32 bit words
- 2.0 μs to reload the buffers in the RS/6000's I/O Channel Controller (IOCC).

These measurements were made using an HP 16500A logic analysis mainframe with 10 nanosecond resolution. This duty cycle would suggest that the maximum obtainable bandwidth of 320's Micro Channel's bus is a little under 142 Mbps. We expect that later models of the RS/6000 will contain an improved IOCC to allow higher bus performance.

4.2. Software

Using the program outlined in the Appendix, we ran a script which varied the buffer size and number of repetitions of the *write()* call necessary to write 67.1 megabits to the interface. The script was run on a lightly-loaded IBM RS/6000 Model 320. Benchmarking by an unrelated process connected through an Ethernet connection noted little or no performance degradation, even when competing for I/O resources (e.g., a several megabyte FTP).

Buffer Size	Elapsed Time	Bandwidth (Mbps)
1K	2.88	23.3
2K	1.65	40.7
4K	1.03	65.1
8K	0.76	88.3
16K	0.71	94.5
32K	0.65	103.2
64K	0.58	115.7

Table I: Results for 67.1 Megabit transfers

Each of the seven tests in the script represent about 67

million bits worth of transfer, so that 1K byte transfers are giving about 23.3 Mbits/sec, and 64K byte transfers are giving about 115 Mbits/sec. These tests were of short duration, the clock is relatively imprecise, and not all of the variables were controllable. However, these experiments are repeatable to the accuracy given, lending credence to the measurements.

4.3. Conclusions

It's clear from Table I that software is the limiting factor to system performance. Larger block sizes let the hardware stream effectively, and smaller sizes force the AIX system to context-switch frequently. This can be seen by examining the relative performance gain for each doubling in block size. The performance is almost doubled as block size is increased from 1K bytes to 2K bytes, but the increase from 32K to 64K gives only a 10 percent gain.

For many sources of traffic, the 64K byte blocks, and hence the performance figures, seem unrealistic. We are looking at device driver strategies which can give us good performance with smaller block sizes, perhaps by optimizing the device driver strategy for stream-startup.

To the best of our knowledge, this is the fastest measured software/hardware combination for ATM, and we intend to continue tuning so that we can focus on the right issues for our OC-12 (622 Mbps) follow-on. Our approach of pursuing architectural solutions, such as concurrent operation (as in the parallelism in the header processing pipeline), allows us to take advantage of improvements in technology which would allow higher clock speeds. We have found (from using the software and logic analyzer concurrently) that a major factor limiting hardware performance is the Micro Channel bus. This is due to the delay induced in fetching data from the Micro Channel IOCC. It is not entirely clear (as of November 1991) why this is happening: it may be software structuring or limitations inherent in the IOCC and its relationship to system memory. We hope to be able to report a precise characterization of the problem and its solution in 1-2 months.

5. Notes and Acknowledgments

We would like to thank Bruce S. Davie for his detailed and constructive criticisms of this work and its presentation in this paper.

AURORA is a joint research effort undertaken by Bell Atlantic, Bellcore, IBM Research, MIT, MCI, NYNEX, and Penn. AURORA is sponsored as part of the NSF/DARPA Sponsored Gigabit Testbed Initiative through the Corporation for National Research Initiatives. NSF and DARPA provide funds to the University participants in AURORA. Bellcore is providing support to MIT and Penn through the DAWN project. IBM has supported this effort by providing RS/6000 workstations. The Hewlett-Packard Company has supported this effort through donations of laboratory test equipment.

RS/6000, AIX and Micro Channel are trademarks of IBM.

6. References

[1] H. B. Bakoglu, G. F. Grohoski, and R. K. Montoye, ''The IBM RISC System/6000 processor: Hardware overview,'' *IBM Journal of Research and Development* **34**(1), pp. 12-22 (January, 1990).

[2] David D. Clark, Bruce S. Davie, David J. Farber, Inder S. Gopal, Bharath K. Kadaba, W. David Sincoskie, Jonathan M. Smith, and David L. Tennenhouse, ''The AURORA Gigabit Testbed,'' *Computer Networks and ISDN Systems* **25**(6), (to appear) (January 1993).

[3] Eric Cooper, Onat Menzilcioglu, Robert Sansom, and Francois Bitz, ''Host Interface Design for ATM LANs,'' in *Proceedings, 16th Conference on Local Computer Networks*, Minneapolis, MN (October 14-17, 1991), pp. 247-258.

[4] Bruce S. Davie, ''A Host-Network Interface Architecture for ATM,'' in *Proceedings, SIGCOMM 1991*, Zurich, SWITZERLAND (September 4-6, 1991), pp. 307-315.

[5] C. Brendan S. Traw and Jonathan M. Smith, ''A High-Performance Host Interface for ATM Networks,'' in *Proceedings, SIGCOMM 1991*, Zurich, SWITZERLAND (September 4-6, 1991), pp. 317-325.

7. Appendix: Experimental Apparatus

```
/*
 *  testwr.c - main block
 * (no declarations or set-up shown)
 */

if ((fd = open("/dev/host0", O_WRONLY)) == -1)
{
   perror("Couldn't open dd");
   exit(-1);
}

gettimeofday( &tv1, &tz );

for(i=0; i<repeats ; i++)
{
   if (write(fd, buf, count ) == -1)
      perror("write failure");
}

gettimeofday( &tv2, &tz );

clock = elapsed( tv2, tv1 );

printf( "elapsed time: %d microseconds\n",
         clock );
```