Aurora at MIT

Final Report on MIT's Participation in the Aurora Gigabit Testbed*

David D. Clark, Henry Houh, and David L. Tennenhouse, editors†

Laboratory for Computer Science
Massachusetts Institute of Technology

The Aurora Gigabit Testbed linked researchers at MIT, the University of Pennsylvania, the University of Arizona, Bellcore and IBM. The Aurora partnership included Nynex, Bell Atlantic and MCI, who investigated issues related to the gigabit transmission facilities which they provided to the research teams.

Aurora activities at MIT were undertaken by two groups: the Telemedia, Networks and Systems (TNS) Group and the Advanced Network Architecture (ANA) Group. Section 1 of this report describes the work performed by the MIT/TNS group, under the direction of David Tennenhouse; Section 2 describes the work of the MIT/ANA Group, under the direction of David Clark; the Appendixes list the publications and students supported by the project.

*This research was supported by the Corporation for National Research Initiatives.

†The authors can be reached at: MIT Lab for Computer Science, 545 Technology Square, Cambridge, MA 02139; Tel: (617) 253-6005; Email: {ddc,hhh,dlt}@lcs.mit.edu

Section 1: TNS Group Activities

1 Introduction

The TNS group's testbed activities included: the development of the VuNet, an ATM-framed local distribution system; research on the design of host/network adapters; and the demonstration of network-based multimedia applications.

Some of our achievements in the testbed program include: the deployment of the VuNet infrastructure (encompassing 8 switches, 11 network-based video appliances, 11 host workstations and a suite of multimedia applications); demonstration of seamless DAN/LAN/WAN internetworking; the development of a number of different host/network adapters; and the investigation of ATM end-working and signaling issues.

Along the way we learned a number of lessons concerning: the distinction between cell switching and ATM layer functions; the role of memory in the design of host interfaces and end-working software; the striping of ATM transmission facilities; and the traffic implications of computation-intensive multimedia applications.

In the following subsections we describe the research objectives, approach, experimental results and lessons learned by members of the TNS group.

2 Research Objectives

The TNS group's activities focused on the following subset of Aurora's research objectives [5]:

- The design and evaluation of ATM-framed local distributions systems
- The design, development and evaluation of alternative host interfaces
- The integration of gigabit switching, transmission and multiplexing systems
- The demonstration and evaluation of new paradigms for distributed applications

In accordance with MIT's portion of the Aurora Statement of Work, we also worked with other interested groups to:

- Develop specifications for the ATM layer and to investigate ATM adaptation

The following paragraphs describe the relationship between these goals and the approach taken in our research.

ATM-framed local distribution

Our work on local distribution focussed on the VuNet, an ATM-based Desk Area Network. **The VuNet**, which was developed as part of the testbed, was deployed in individual offices and laboratories at MIT and a VuNet node was established at U. Penn. To facilitate VuNet / Sunshine integration, a VuNet node was located at Bellcore for a brief period.

Host Interfaces

Within the testbed as a whole, several areas of host interface research were investigated by the Aurora partners and taken together these provide a fairly good map of the *design space*. The MIT/TNS group investigated a portion of this design space, with particular emphasis on the adapter's point of attachment, cost, and use of memory. We developed programmed I/O, DMA-based, and coprocessor-based adapters and collaborated with Bellcore on the design of the Osiris host interface. We also developed the Vidboard, an ATM-based camera that is directly attached to the VuNet.

Gigabit switching, transmission and multiplexing

The MIT/TNS group investigated a number of internetworking and systems integration issues and demonstrated the **seamless ATM-based internetworking** of: the VuNet DAN, the AN2 LAN, and the Sunshine WAN. We also collaborated with Bellcore on the design of an ATM-based cell processor for use in the port controllers of the Sunshine switch.

Distributed Applications

The VuSystem, a separately funded project, is a programming system for the software-based processing of audio and video data. Within the context of Aurora, we demonstrated the operation of VuSystem applications over the VuNet / Aurora infrastructures.

ATM and Adaptation Layer Issues

MIT researchers helped initiate the development of SEAL/AAL5, which has been adopted for a wide range of packet-oriented services (frame relay, IP, etc.). During the course of our research we investigated a number of issues related to **ATM end-working** and an **application-oriented approach to signaling**.

3 Approach

The following sections provide details concerning the approach, design and implementation of key technologies developed within the project.

3.1 Local Distribution Systems - the VuNet

A Desk-Area Network (DAN), as illustrated in Figure 1, is a small local system in which peripherals, such as multimedia devices, are taken out of the workstation and connected directly to the network. Workstations access these devices over the network and coordinate the movement of information streams among the different parts of the system. Aurora's DAN implementation is known as the **VuNet** and the Vidboard, described in section 3.3, is an example of a DAN-enabled peripheral.

The VuNet is a gigabit-per-second desk/local-area ATM network which interconnects general-purpose workstations, network-based multimedia devices and bridges to other networks. The DAN Approach replaces the workstation's haphazard collection of peripheral interfaces and connectors (e.g. SCSI, VGA, EISA) with a generic ATM-based connector that supports a heterogeneous range of peripherals including cameras, displays, disks, etc. We envisage a vigorous market for such peripherals which would be able to operate with any manufacturer's platform. VuNet peripherals are highly programmable and are connected directly to the VuNet, allowing them to be shared among all hosts, including hosts not local to the peripheral's desk area.

With small networks, it is possible to rethink the ways certain network design issues are addressed. Many of these issues also arise in the design of Local-Area Networks (LANs), Metropolitan-Area Networks (MANs), and Wide-Area Networks (WANs), however their characterization within the DAN environment is different than in the LAN or WAN. For example, small desk area networks need not be designed for high utilization as traffic aggregation is not a principal design objective.

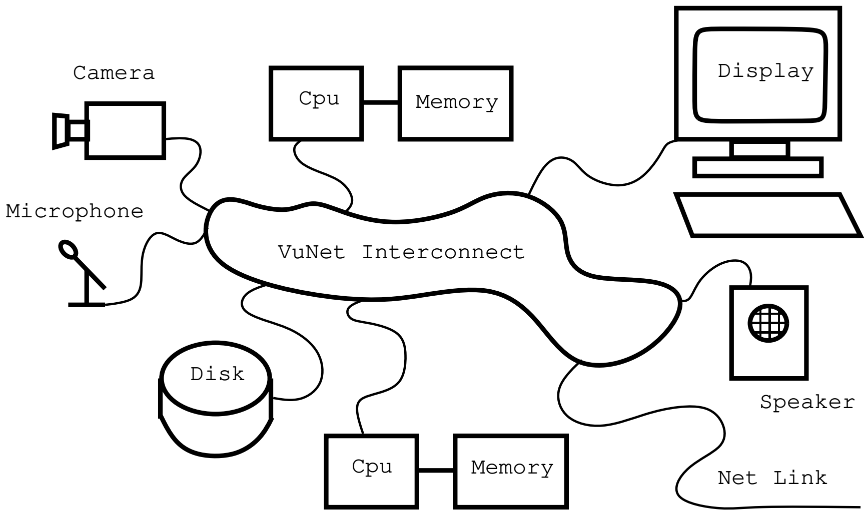

Figure 1: A Desk Area Network where devices are taken out of the workstation and attached directly to the network. The workstations coordinate the flow of data from various devices on the network to other devices.

Larger networks can provide efficient communications because large numbers of hosts share the communications medium. In a small working area environment, there is a smaller number of hosts, making it more difficult to depend on statistics in sharing the aggregate bandwidth necessary for efficient operation. The fact that bandwidth is less expensive over the short distances involved suggests a VuNet design regime that leverages inexpensive, lightly utilized bandwidth.

Similarly, shared wide area components, such as the line card serving a heavily multiplexed link, can be relatively expensive. However, in a small desk area network, careful attention must be paid to the cost per end node. Hence it becomes attractive to trade off functionality for cost in the non-multiplexed components such as the end system switch ports and interfaces. Simple, inexpensive interfaces become an important design objective.

Within Aurora, both regions were explored. Bellcore's Sunshine effort explored the wide area, and the VuNet explored the desk area.

A key goal motivating the design of the VuNet hardware was simplicity. Sophisticated network functions (e.g. multi-cast, back-pressure, support for ATM adaptation layers, service classes) were pushed to the edge of the network and became the responsibility of the clients. We believe that a local environment such as the VuNet can be effectively served by a simple switch fabric having a limited number of access ports and an internal speed that is greater than that of the clients.

The remainder of this section describes the VuNet design and the hardware that was built. The considerations that drove the design were:

Software intensive philosophy: Wherever possible, the VuNet design favors software-based solutions that reduce the complexity of network hardware by moving functionality into the software of the end nodes. An important benefit of this software-intensive approach is the ability to port the system to higher-performance workstations easily. Over the span of the Aurora project, four different hardware platforms were used with two varieties of host interfaces, for a factor of ten difference in performance.

Bursty traffic: Since host processes running within an operating system generate network traffic during their time slices, their generation of data is inherently bursty. Similarly, they consume data in bursts that are not synchronized to the incoming traffic. Accordingly, sources and sinks of video and audio information must be able to handle data in bursts and the DAN and its hosts and peripherals must support bursty traffic.

Asynchronous client interface: Clients should not be slaved to the network clock. Instead, the network ports should support various input and output rates. ATM, with its small cell size and variable cell rates, can accommodate this requirement.

Simple client interface: Fast and slow clients alike should see the network as a place to easily write and read ATM cells. The VuNet uses a modified cell format in which the cell length is increased from 53 to 56 bytes in order to make cells quadword-aligned.

Data transparency: All classes of traffic (video, audio, file transfer, etc.) are treated in a similar manner until they reach the target application, and multiple video streams are easily supported since hardware limitations are not placed on their number.

Interoperability and modularity: Multimedia peripherals such as a video capture board were designed to interface directly to the VuNet rather than to a workstation I/O bus. This allows the peripherals to be shared among all the hosts connected to the VuNet. It also avoids redesigning a peripheral for each vendor's computer platform.

These principles were applied to all aspects of the design of the local ATM distribution system.

3.1.1 The VuNet Cell Switch

In the development of the VuNet we separated the switching of cells (between switch ports) from ATM level functions such as VCI mapping. We started with a simple model of the "ideal" exchange of cells between two hosts that are directly connected to each other over a communications channel of zero length and infinite speed. An important observation is that even this channel requires a memory-based buffer or FIFO of some sort, if only to decouple the clock rates, and ultimately the application burst rates, of the communicating processes.

To satisfy our "ideal" model of cell transfer, the switch fabric, which consisted of bidirectional ports with first-in first-out (FIFO) buffers which fed a crossbar matrix, allowed each host to accept cells at a rate that exceeds the aggregate rate at which cells are presented by all of the attached hosts. Current versions of the VuNet switch have either four or six ports and we clock the crossbar matrix at a rate of 700 Mbps per port. Given the small number of ports and the rate at which our present hosts are able to generate traffic, this provides a close approximation to the model.

The switch provides a straightforward interface that simplifies the design of client hardware. Since the timing of the client side of the port FIFOs is decoupled from the internal timing of the switch matrix, there is no need for clients to synchronize their operation with the internal switch clock, as is often the case with traditional switch designs. Furthermore, both the transmit and receive blocks can be operated concurrently. The FIFOs also serve to buffer cell bursts, an important factor in our simplified network. Finally, the data bus width to the port is selectable, either 32 or 64 bits, allowing a simple mapping to workstation and processor buses. All these factors make it easy to design devices which connect directly to the switch, satisfying the "simple client interface" objective.

The VuNet was designed around a non-standard 56-byte ATM cell, as the switches had 64-bit interfaces. While this presented new issues when bridging to other networks (as discussed in Section 3.4), it greatly simplified the design and complexity of switch and the network-based devices.

3.1.2 Cell Relaying

The preceding section described cell switching among devices that are attached to a single switch. Inter-switch communication can be modeled by a device (or set of devices) that is connected to ports on two different switches. This cell relaying host performs two functions: it copies cells between

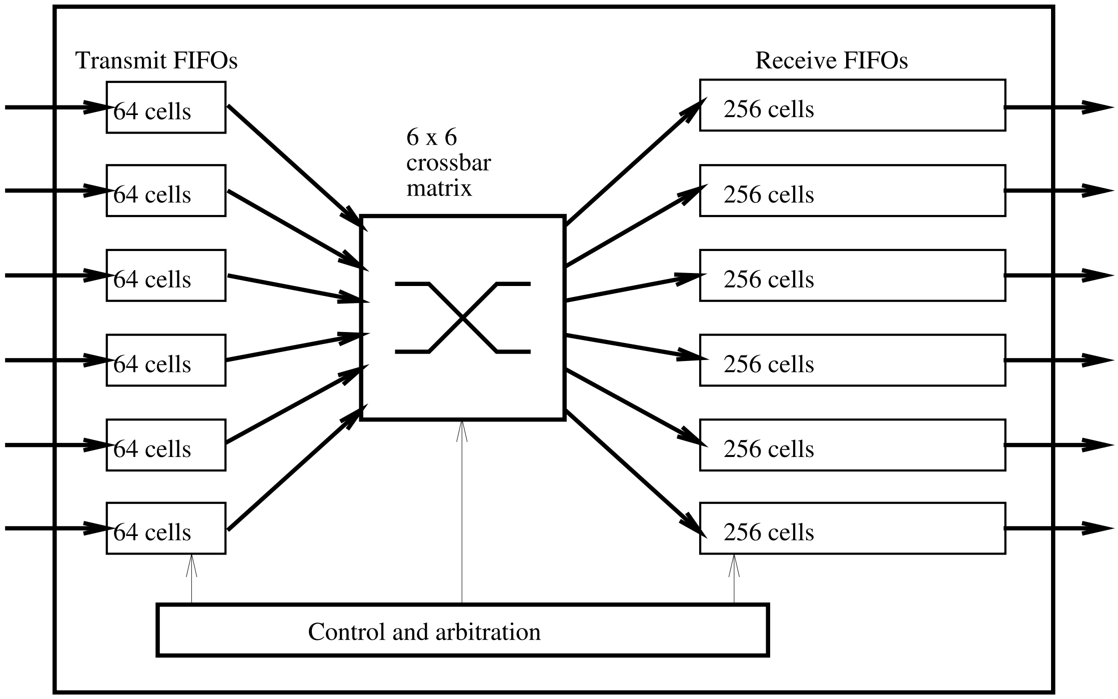

Figure 2: The six port VuNet switch.

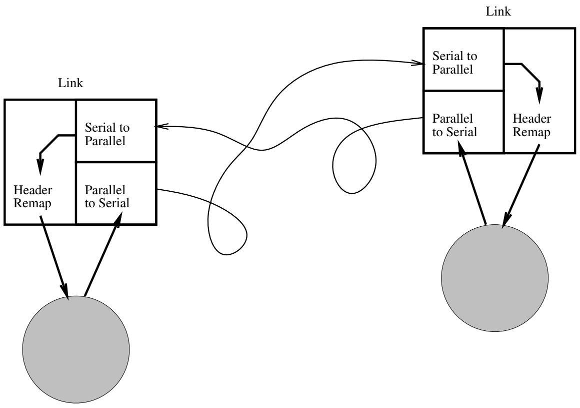

Figure 3: The links in VuNet perform header remapping and next output port lookup for hop-by-hop cell routing.

the switches and it re-maps VCIs. The relay serves as an intermediary with knowledge of the VCI address spaces associated with the switch ports.

The VuNet link implements a minimal set of relaying functions, cell copying and VCI mapping. To support the inter-office separation of switches, it is constructed in two parts that are inter-connected by high speed channels as illustrated in Figure 3. The channels use single mode optics and serial coding based on the HP G-Link chipset and the links presently operate at 500 Mbps. Link processing is table driven, and the management of the link tables is left to the communicating end systems. This will be discussed in further detail in Section 3.5.

The link tables are large enough to remap the entire 64K (16-bit) VCI space to 64K new VCIs and four bits of switch port information. All other bits in the ATM header remain untouched.

The VuNet does not support the Header Error Check (HEC) function on a hop-by-hop basis as we do not believe that HEC is an essential function for desk area or even local area operation. For compatibility purposes a cell source can pre-compute the HEC that is expected by the destination.

Management of the link tables is performed using control cells sent with a reserved ATM VCI. Cells received by the link on these special VCIs cause the link to re-write entries in its header lookup table. They can also cause the link to emit a cell containing table entries, so link tables can be read back.

3.2 Host Interfaces

Many researchers believe that host interface performance can be improved through the off-loading of network protocol processing, i.e., moving functionality from the host to the interface. Within Aurora, several different types of host interfaces were developed to explore this hypothesis. For the VuNet, several types of interfaces were designed. MIT also participated with Bellcore in the design of their Osiris host interface.

Using these various interfaces, different methods of segmentation and reassembly were investigated; this spanned the gamut from specialized on-board host interface hardware to extremely simple host interface hardware combined with software segmentation and reassembly. Different methods of host-interface synchronization were used including interrupt driven handling and device polling. The various interfaces also explored different network points of attachment.

The simplest VuNet interface was a programmed I/O interface in which the processor was responsible for writing data to the interface. Another version supported DMA with variable burst sizes from one ATM cell size and up. In both cases, the segmentation and reassembly was performed by the host processor. An ATM-based coprocessor interface was also designed and fabricated. This interface, which resembled a floating point unit, connected directly to the coprocessor port of the CPU.

Finally, MIT participated in the design of the Bellcore Osiris interface, a custom segmentation/reassembly hardware engine attached via the processor I/O bus. In the Osiris design large packets are transferred to the interface, which segments the packets before transmission into the ATM network.

The following subsections describe the design of these host interfaces.

3.2.1 VuNet Programmed I/O Interfaces

All of the VuNet host interfaces rely on the CPU to perform the ATM segmentation and reassembly (SAR) tasks. Although this approach was computationally intensive, it provided a flexible environment for the study of many host interface issues. Other Aurora partners, working in parallel, investigated other aspects of the design space, especially hardware-based SAR.

The first of VuNet interfaces designed was the VuNet Programmed I/O interface, known as the VupBoard. This host interface served only as a memory-mapped extension of the read port of the switch output FIFOs. Cells were read into memory through 14 reads of the memory address corresponding to the FIFO. Other addresses corresponded to control and status indicators such as the state of the FIFOs.

Similarly, sending cells into the network required making 14 writes to memory, interleaved with writes to the address corresponding to the FIFO write control signals.

While better methods of utilizing the processor were later investigated, this initial interface provided an early development platform that allowed us to explore software segmentation and reassembly of ATM cells and interrupt-driven packet processing. Using the VupBoard we were able to develop and debug device driver software that made the VuNet accessible to applications via the standard UNIX socket services. The driver also made the VuNet accessible via standard IP services, including NFS,

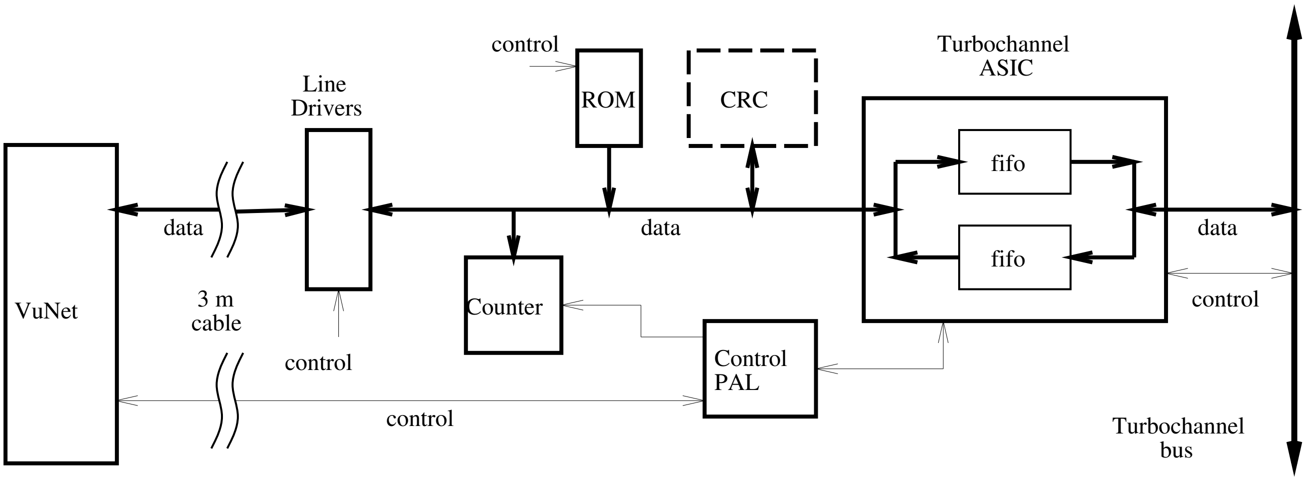

Figure 4: A block diagram of the VudBoard, which consists of 11 chips.

Telnet, FTP, etc. Finally, since the programming model for the VupBoard was similar to (though an order of magnitude slower) that of the cell coprocessor, it provided a vehicle through which we could benchmark software and project the expected performance of a system with a coprocessor interface.

3.2.2 VuNet DMA Interfaces

The next VuNet-specific interface focused on improving the bandwidth and relaxing the temporal coupling between the host CPU and the interface – building interfaces, that are faster, rather than smarter.

Although the design does not use any custom ATM circuitry, it does utilize the TURBOchannel Interface ASIC (TcIA)[1] for the DEC TURBOchannel bus. This bus is used in the DEC 3000 line of Alpha workstations, as well as in the DEC 5000 line of MIPS-based systems. The TcIA performs the DMA request and arbitration, and contains two 60-byte FIFOs for receive and transmit.

As shown in Figure 4, the single chip PAL controls all data movement between the ATM network and TURBOchannel ASIC. Since the data bus from the VudBoard to the VuNet switch is shared by the receive and transmit paths, it arbitrates between reading cells from the switch and writing cells to the switch, with reception given priority over transmission.

The counter is used to implement an optional transmission timer that controls the pace of cells injected into the network.[2] Finally, the ROM provides device identification information. Although it is superfluous to our design, it is included on the interface in order to conform to the TURBOchannel specification.

Various hardware and software configurations allowed experimentation with interrupt-driven versus polled packet-processing. This allowed measurements of packet delivery latency and packet processing/context switching overhead.

3.2.3 Cell Coprocessor Interface

A cell-based coprocessor chip was designed to provide a direct interface between an ATM network and the coprocessor interface of a conventional RISC processor. The combined RISC processor/cell

[1] The TcIA chip has been discontinued and is no longer available.

[2] This could be used as an experimental way for implementing flow control, and is currently used for implementing pacing should a target host not have the ability to receive bursts of data at the full rate which can be transmitted.

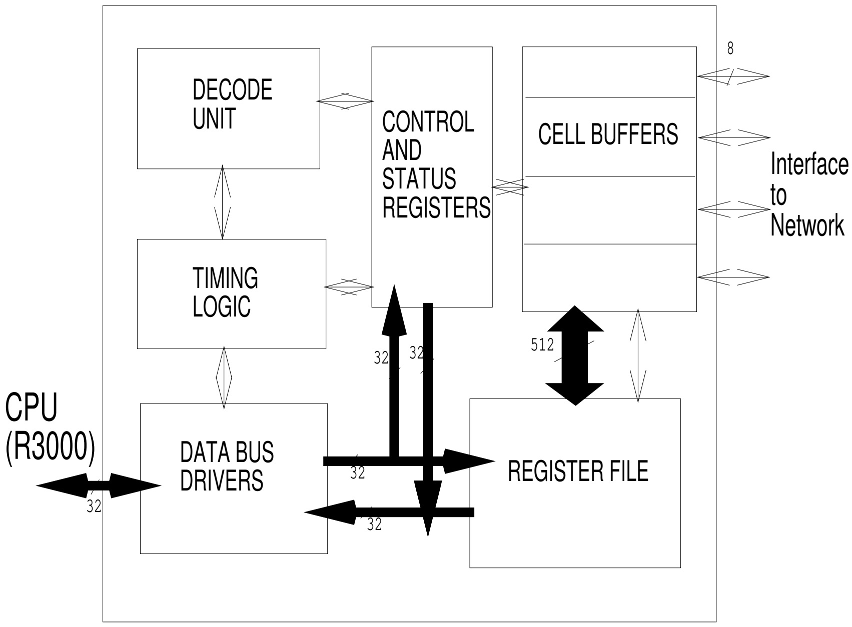

Figure 5: Cell-Based Coprocessor Design

coprocessor complex could form the core of an ATM-compatible workstation or could be used as a stand-alone cell processor, similar in function to Bellcore's cell processing engine. To perform network operations, such as reading and writing cells, the RISC processor executes coprocessor instructions, much the way it performs floating point operations.

¿From a software perspective, this interface was quite similar to our programmed I/O interfaces. However, from simulations, a considerable performance improvement was to be realized – by completely bypassing the memory subsystem and permitting the direct transfer of cell data to/from the CPU's registers.

Figure 5 is a block diagram of the prototype coprocessor which is designed to operate with the 40 Mhz versions of the MIPS R3000 processor. This work was closely aligned with Bellcore's work on a stand-alone cell processing engine. The bus interface, timing logic, instruction decoder, and control/status registers have been substituted for Bellcore's on-chip RISC engine and memory interface. The instruction decoder includes a pipeline follower that tracks the MIPS instruction stream, decoding and sequencing any instructions that pertain to the coprocessor. A large fraction of the design, including the network interface, cell buffers and register file has been directly copied from the Bellcore chip. The savings resulting from the substantial re-use of chip design and layout is a clear demonstration of the benefits of the close collaborative links established within Aurora.

3.2.4 High Performance Host Interface

One of the other interfaces designed within Aurora was the Bellcore Osiris board [7], designed with the assistance of MIT. This was a hybrid system with the ability to scatter ATM cell payloads into host buffers corresponding to VCIs. Though some overhead processing is needed to maintain and set up areas of memory for different packets that come in, no more processing is required than in transferring fully assembled packets from the host interface.

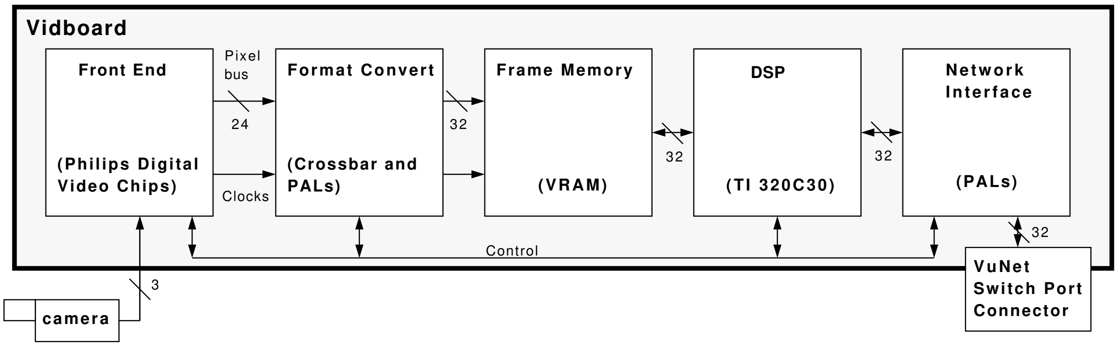

Figure 6: Block diagram of the Vidboard

3.2.5 Summary

Host interface functionality can be categorized into three primary functions: data movement, data processing, and data delivery. Data movement functions include reading data from processor memory and writing the data to the network, and vice versa, while data processing functions include demultiplexing data, performing checksums, packet reassembly, and packet segmentation. Data delivery to the application level often entails working withing the methods used in the operating system, which may necessitate copying from operating system memory space to application memory space.

Functions such as packet segmentation and reassembly (SAR) are some of the more complex functions. The *complex interface* approach would be to perform this processing on the interface itself through the use of customized hardware and/or a dedicated CPU. In a *reduced complexity interface* (RCI) these functions are performed by the host CPU itself, i.e., on the shared and general purpose processing engine of the workstation.[3]

Scattering individual cell payloads into different areas of host memory is an attractive mix between hardware and software SAR. Scatter reassembly is a function that does not require a complex design, yet may save some per packet assembly costs. Essentially the only difference between this hybrid method and outboard packet reassembly is that the bus transfers occur in ATM-sized chunks in the former, and packet-sized chunks in the latter. The trade-off involves a reduction in memory requirements and latency in exchange for an increase in bus arbitration overhead.

3.3 Information Appliances - the Vidboard

Multimedia systems typically consist of workstations that contain a number of multimedia add-in boards for functions such as video and audio capture and playback. Our DAN equivalent, an ATM-based video capture board, known as the Vidboard, was use to explore the properties of a network-based video source.

The Vidboard is based on a front-end frame-memory processor architecture that is capable of generating full-motion video streams having a range of presentation (picture size, color space, etc.) and network (traffic, transport, etc.) characteristics. The architecture is novel in that it also permits the decoupling of video from the real-time constraints of the television world. Through a closed-loop control mechanism, a destination device can dynamically vary the frame rate of a video stream during the course of a session, which allows easier integration of video into the software environment of computer systems.

[3] These concepts are developed further in [10]

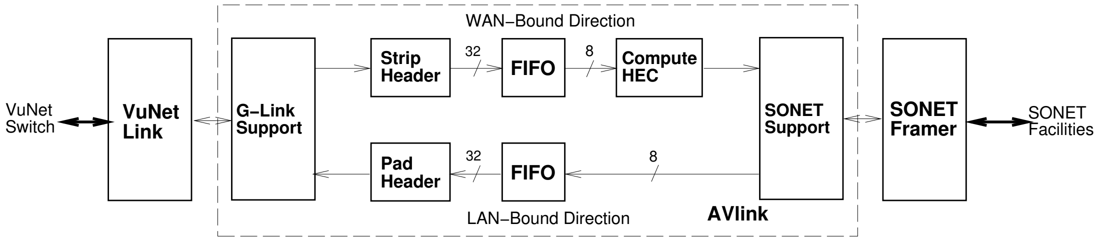

Figure 7: Block diagram of the AVlink

3.4 Seamless ATM – DAN/LAN/WAN Internetworking

One of the important ideas of ATM is that of "seamless" interconnection; that is, the concept of a uniform network service that transcends local and wide areas. Although standardized SONET/ATM represents one approach, ATM-based networks which do not adhere fully to the SONET standards and conventions can still be incorporated into the "seamless" network. The use of the ATM protocol reference model identifies differences between two ATM networks which can then be converged through relatively simple hardware. To investigate these issues we developed two specialized VuNet links, the AVlink and the Zebra.

3.4.1 AVlink

The AVlink is an ATM bridge interconnecting the VuNet and the Aurora testbed wide-area facilities. Differing design considerations for each network have caused a number of architectural differences between them. The AVlink overcomes these differences and provides low latency interconnection between the two networks. Performance numbers and several configurations in which the AVlink has been used are described in a Section 4.1.2.

On the VuNet side, the AVlink is similar to other inter-switch links. On the Aurora side, the AVlink attaches to one of the four available OC-3c channels via Bellcore's STS-3/OC-12 multiplexor. The use of multiple channels in parallel, a technique known as striping, is complicated by the presence of varying end-to-end delays between the channels. This effect, sometimes known as skew, was also investigated in the development of the Bellcore Osiris board.

3.4.2 Zebra – Striping

The Zebra is an ATM bridge interconnecting the VuNet to the AN2 switch, an industrial strength ATM LAN developed by Digital. The AN2 can, in turn, be connected to the Aurora facilities via a four channel OC-3 card and an OC-3/OC-12 multiplexor developed by Bellcore.

In order to formalize the issues associated with network striping, we developed a reference model that distinguishes four degrees of freedom of relevance to striping implementations. These are:

- Striping Topology: How and where the path between two hosts is split into stripes. Possible cases are end-to-end splitting, internal splitting, or some combination of these two.
- Participation of Hosts and Networks: The degree to which the hosts and networks support a striping implementation.
- Striping Layer: The network layer at which the striping occurs. This can range from the physical layer all the way up to the application layer, and determines the size of the striping unit, which is the largest unit of data which is transported intact on a single stripe.

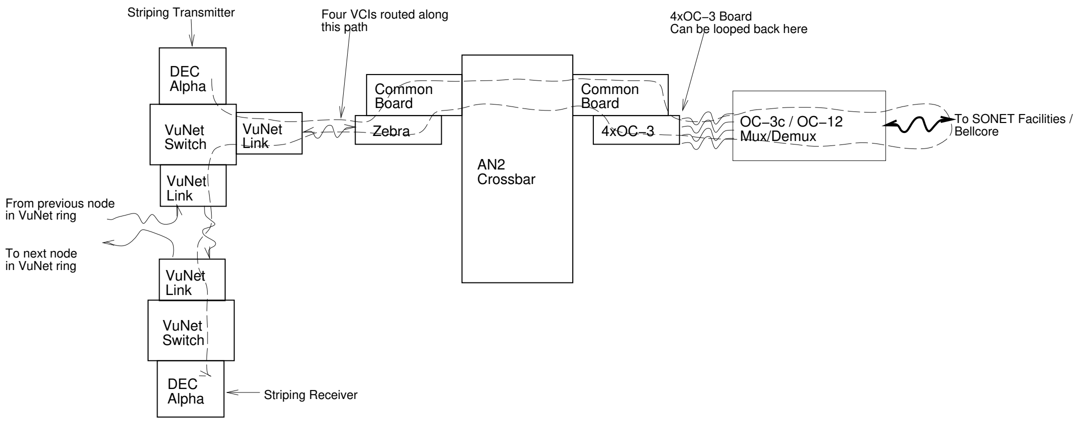

Figure 8: Environment developed for striping experiments.

- Implementation Issues: There are two main problems to overcome when implementing network striping; these are balancing the loads on the stripes, and maintaining synchronization across the stripes so that higher layer data may be correctly reassembled. The implementation issues relate to the various options available for solving these problems and the means by which the characteristics of the striped facilities are made visible to the layer at which striping is being performed.

An environment that interconnected the VuNet DAN, the AN2 LAN, and the Sunshine WAN using the Aurora facilities, was developed (Figure 8) to explore and experiment with the functional space mapped out by the model. As part of this environment, a bridge between the VuNet and AN2 was developed. Known as the Zebra, this board converges the differences between the two networks. The configuration linking these three networks, as well as some lessons learned about network striping, will be presented in a later section.

3.5 ATM End Working

Using the VuNet as a research platform, several aspects of service integration were investigated. First, VuNet network communications were fully integrated into the system, directly via ATM-supported UNIX sockets, and indirectly, via IP-supported interfaces (including UDP and TCP sockets). This allowed all the usual Internet services, e.g., FTP, Telnet, NFS, etc. to be offered over the VuNet.

Also, the Vidboard network based video capture device was used as a platform for real-time traffic generation within the network. Source traffic shaping allowed investigation into real-time traffic interaction with the host device drivers. A closed-loop control mechanism was developed for graceful degradation of real-time traffic bursts.

Investigations into the jitter arising from the middle and upper layers within the end systems of multi-service networks were also conducted.

3.5.1 A UNIX Device Driver with Software SAR

Our initial requirements of data transparency, interoperability and modularity dictated the structure of our network device driver. The driver, as shown in Figure 9, was designed to be organized into several major levels. At the highest level are the application processes, which receive data over the UNIX socket interface. Below this are the kernel side of the socket interface level, the reassembly and protocol processing level, and the buffer and device management level.

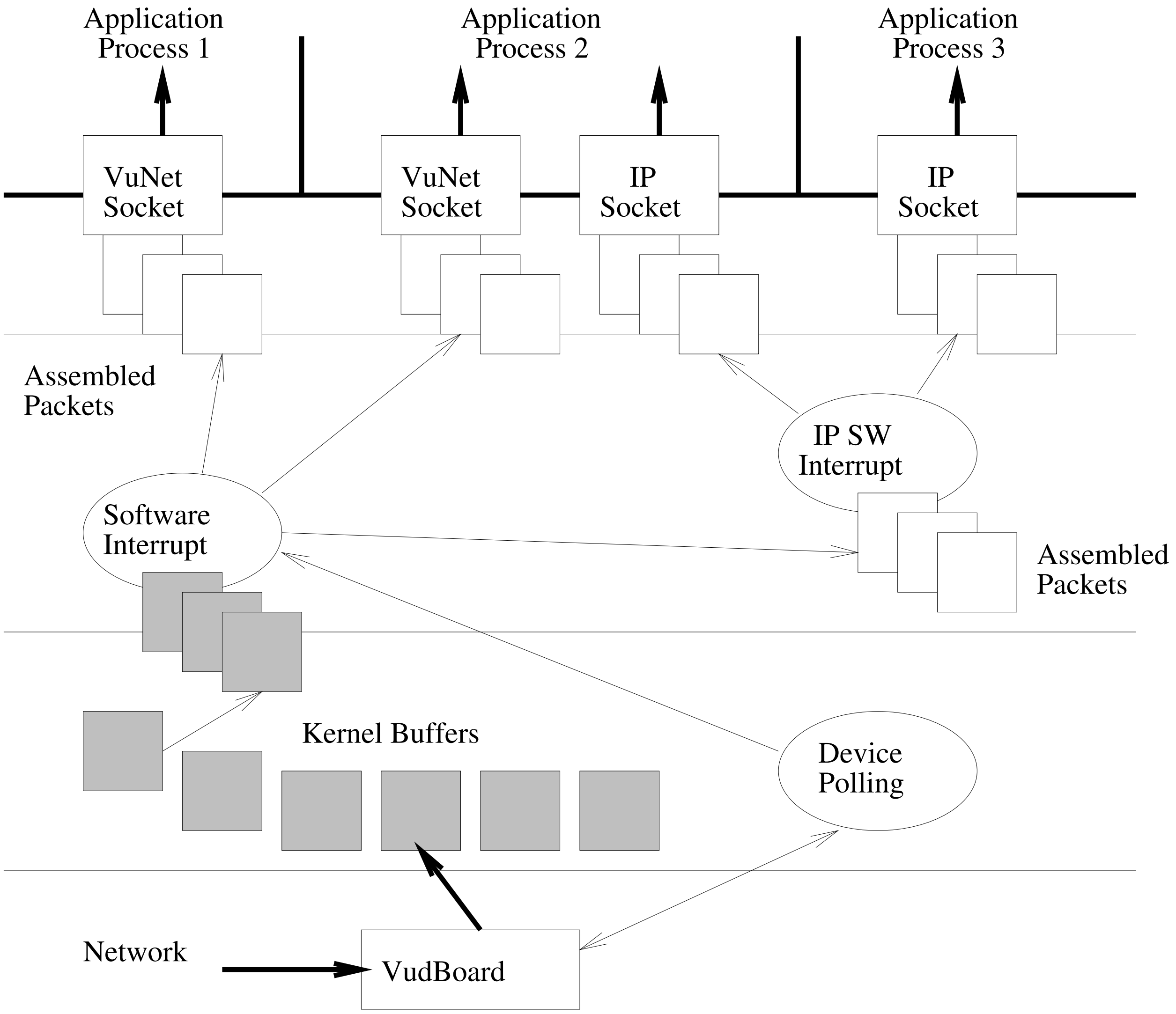

Figure 9: Cells are read from the network at and passed to the software reassembly process at regular polled intervals. These assembled packets are then passed to the IP handler or directly to a VuNet socket.

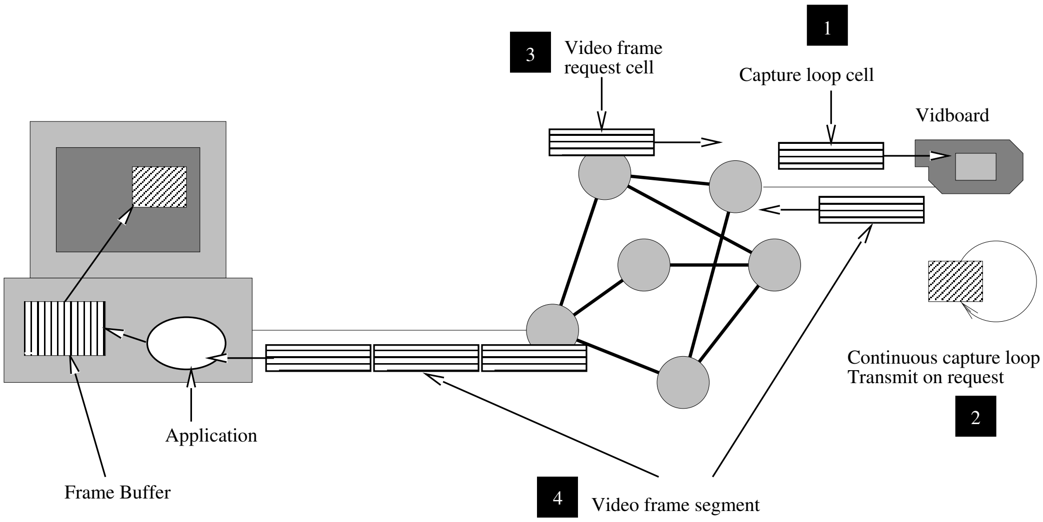

Figure 10: Video source control model: 1) A command cell is sent to the Vidboard, starting it in a 2) continuous video capture loop, where, upon 3) receipt of a frame request cell from workstation, a 4) video frame is delivered.

If cells have been received on the network, the driver temporarily halts the VudBoard in order to point it to a new kernel buffer, freeing up the other buffer to be queued for reassembly. Buffers were allocated and quickly swapped in order to minimize the time the host interface is "deaf." Because of the size of the output buffers on the switch, the driver had to be designed so that cells were not lost while this was occurring. Thus, the VudBoard was restarted before the buffers which contained network data were processed.

The buffers that are queued by the previous level may contain interleaved ATM cells over many difference VCIs. The reassembly process must look up the VCI to determine the type of connection, and reassemble the cell payloads into packets, stripping out the ATM header information and other related information. This is done by copying the data into different buffers reserved for reassembled packets.

Depending on the kind of packet reassembled, packets are passed directly to the VuNet application socket, or may be passed to the IP Packet Handler for additional protocol processing and further demultiplexing. The UNIX socket mechanism is used as the data delivery mechanism for the applications that use our network data. Once the data is delivered to the socket level, it can be copied out to user space on user reads, completing the delivery of data to the application. Integrating UNIX sockets allowed the seamless integration of the network to the application level.

Clearly, our software-based SAR approach involves considerable copying of network data, and this is an issues which we will return to in the *lessons* section of this report.

3.5.2 Vidboard Network Protocols

Hosts in the VuNet are not dedicated to processing and forwarding network data - they are user machines under varying loads. In order to prevent the backlog and loss of cells while hosts are busy with user tasks, a protocol was developed for the use of network based peripherals, such as the Vidboard.

The *command cell* protocol is used in making the Vidboard execute a particular task. The protocol consists of using a single ATM cell, which is formatted into a number of fields, to carry command information to the Vidboard. The command code field indicates the task to be executed. Additional information about the task is carried in the command parameters field.

As shown in Figure 10, cells are usually generated in response to a command cell. Other examples of responses are status information in response to a diagnostic command and a video stream in response to a video data request. The response to a video request has two types of characteristics: video *adaptation layer* and video *traffic shaping*.

For sending video data from the Vidboard to a workstation, we adopted an *adaptation layer* similar to that specified by the ATM Adaptation Layer 5 (AAL5) standard, except that the cyclical redundancy check (CRC) value for error-detection has been replaced by a more easily computed checksum value. In this protocol, scan lines are grouped into packets which are then fragmented into ATM cells. The size of the packet is user-programmable.

The Vidboard is capable of generating video faster than some workstations can process it. In order to avoid overwhelming the workstation network interface with video data, the Vidboard implements *traffic shaping*. Video is sent in small bursts separated by delays, during which the burst is processed by the workstation. The workstation requesting the video sets four parameters in the video request command cell to tune the video stream's traffic characteristics to its needs.

As part of the protocols for controlling the Vidboard from across a network, a scheme was developed which dynamically varied the Vidboard's frame rate during a video session. The Vidboard is placed in a constant video capture loop and is sent requests for frames as they are needed. The application can vary the rate of requests to match its ability to process video. As an example, a video-in-a-window application gracefully halves its rate of requests as a second video-in-a-window is started and the two applications have to share the available machine cycles.

3.6 Application-oriented Signaling

Each application in the VuNet is responsible for opening, maintaining, and closing its own connections. This is done in a "wormhole" fashion by way of ATM control cells embedded in the cell data stream. Applications use library functions that access a shared file and execute an allocation algorithm that prevent nodes from stealing other nodes' VCIs. Background processes can be used to verify link tables and refresh connections when necessary, such as in the case where a link card has been power cycled. In order to maintain table consistency when the network is reconfigured, it is possible to run topology daemons that allow each host to discover the network topology.

3.7 The VuSystem

The VuSystem[14, 15] is a programming system for the software-based processing of audio and video data. The VuSystem was designed to run on ordinary Unix workstations which have no specific support for the manipulation of multimedia data. Because the VuSystem includes an easy to program and extensible in-band processing component, it is uniquely suited to the development of applications that perform intelligent processing of live media.

VuSystem applications [13] combine intelligent media processing with traditional capture and display. Some process live video for more responsive human-computer interaction. Others digest television broadcasts in support of content-based retrieval. Both classes demonstrate the utility of network-based multimedia systems that deliver audio and video data all the way to the application.

Examples of some VuSystem applications include:

The Room Monitor processes video data from a stationary camera in a room. It processes the live video to determine if the room is occupied or empty, and records video frames only when activity is detected above some threshold. It produces a file containing a series of video clips that summarize the activity in the room. A video browser is used to view the segments.

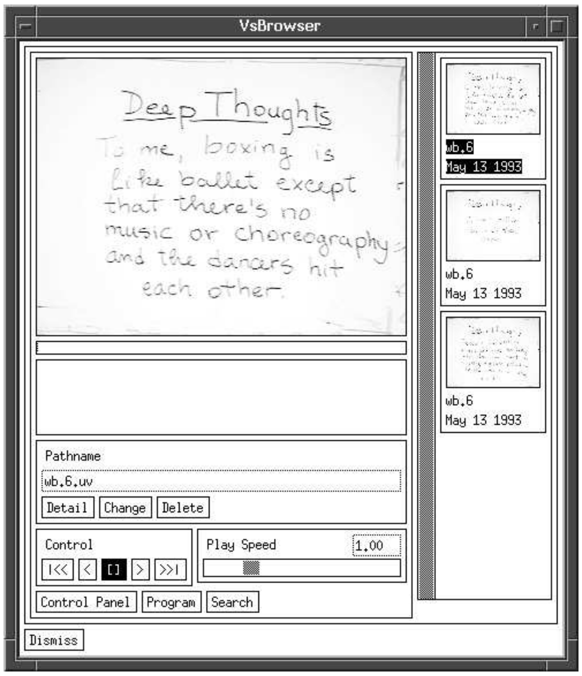

Figure 11: The whiteboard program.

Figure 12: The sports browser.

The Whiteboard Recorder keeps a history of changes to an office whiteboard. It works by taking video from a stationary camera aimed at the whiteboard and filtering it. By following a simple set of rules, the filtering distills the video into a minimum set of images. A browser (Figure 11) can be used to view the saved images.

The News Browser provides interactive access to television news articles. CNN Headline News is automatically partitioned into segments and saved on disk at regular intervals. The stories are viewed with a video browser program. News stories that are closed-captioned can be retrieved based on their content.

The Joke Browser records late-night talk show monologues, and segments them into jokes by processing the closed-captioned text. It extracts information from a recorded monologue through the analysis of closed-captions. In addition to the text of the jokes, the captions contain hints to the presence of audience laughter and applause. A joke parsing module groups captions into jokes.

The Sports Highlight Browser records and segments a nightly sporting news telecast into a set of video clips. Each clip represents highlights of a particular sporting event, and the clips can be browsed as shown in Figure 12. This application generates its annotations through the template matching of scoreboard graphics.

While many VuSystem applications do not require a high speed ATM network in order to function, they were easily layered on top of the VuNet. Performance of the applications using VuNet based devices equaled or exceeded performance using specialized video and audio capture devices.

4 Experiments and Results

4.1 The VuNet

Eight VuNet nodes were deployed at MIT, Bellcore and the University of Pennsylvania over a period of three years. Each VuNet node consisted of a four or six port switch equipped with a link (or AVlink), one to three Vidboards, and one or two host interfaces. The nodes at MIT were typically connected in a ring, with a stub protruding from the ring which connected, through Aurora and two AVlinks, to a similarly-outfitted VuNet node located at Bellcore in Morristown, NJ, then at University of Pennsylvania, as shown if Figure 13. ViewStation applications required no modification to operate over Aurora.

In total, 8 switches, 11 network-based video appliances, 11 host workstations and a suite of multimedia applications were deployed to demonstrate seamless DAN/LAN/WAN internetworking.

The following subsections describe some results achieved with the VuNet hardware, using as the primary example, the Vidboard client connected to the VuNet running the VuSystem software.

4.1.1 Local VuNet Performance

Each switch functioned at a speed of 700 Mbps per port, per direction, for an aggregate capacity of 2.8 Gbps for the four ports switches and 4.2 Gbps for the six port switches. Individual switches were run at over 1.5 Gbps per port, for an aggregate capacity of 9.0 Gbps, though this speed was not used in deployed switches. Within the local ATM ring, the inter-switch links operated at 500 Mbps.

4.1.2 AVlink/Zebra Performance

Seamless interconnection really works. Despite the lack of a common physical layer standard, the integration of the different ATM networks was performed quite easily; cells from the VuNet were

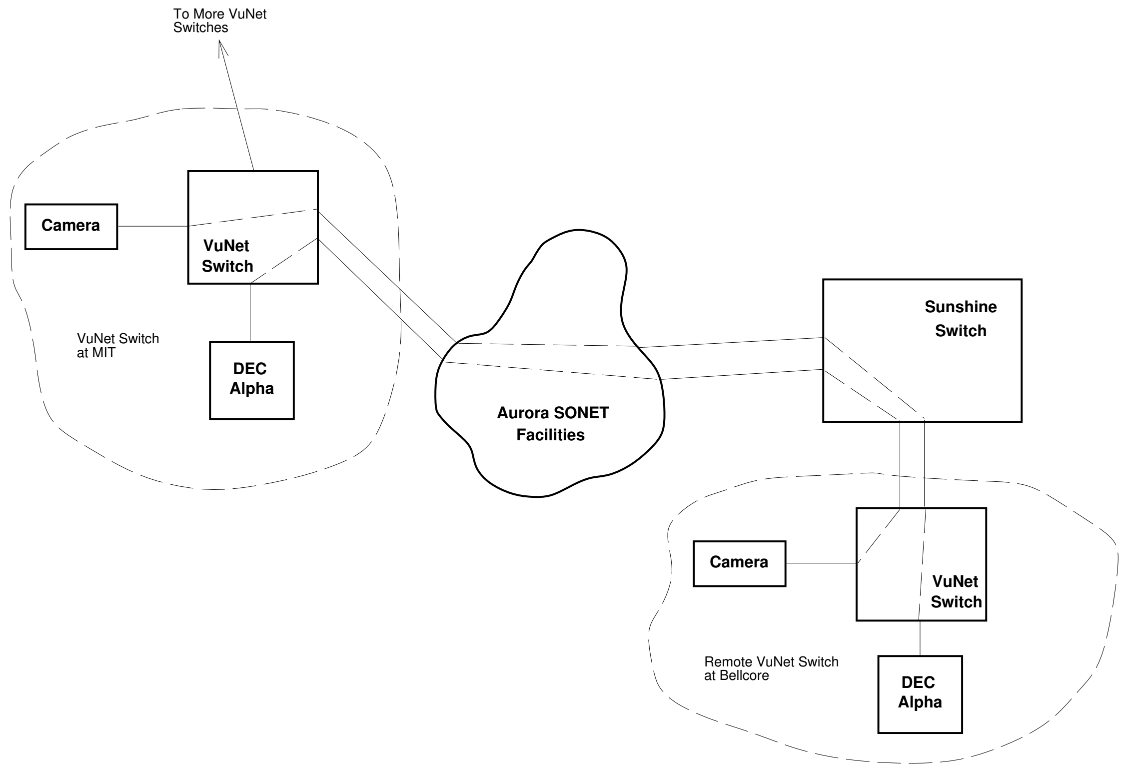

Figure 13: Wide-area configuration.

routed via the Sunshine switch and accepted by the VuNet node on the other end. The only issues then become connection management in the separate conjoined networks. The VuNet used cell-based signaling whereas the Sunshine switch used out-of-band connection management. However, this was bridged by allowing Sunshine to switch the VuNet connection management cells.

In a separate experiment, the AN2 switch was attached to the VuNet as a stub off the main ring. On another port of the AN2 switch was a connection to the Aurora/SONET facilities, using an AN2 line card which generated 4 OC-3c channels and a Bellcore multiplexor board which generated an OC-12 channel from four OC-3c channels and vice-versa. Since only one AN2 switch was available, two hosts on the local VuNet ring were connected using a loopback topology. Software was written for the line card processor which allowed it to support the in-band connection management used by the VuNet. Using this configuration, VuNet hosts were able to set up connections through each of the OC-3c channels, which they could use as stripes. Code could then be written to implement various striping schemes outlined by our reference model.

4.2 Host/Network Adapters

The work on the host adapters spanned four generations of host workstations. Initially, the VuNet hosts were DEC 5000/200 (25 Mhz) and later 5000/240 (40 Mhz) workstations based on the MIPS R3000 processors. During the middle of the project, all the workstations were upgraded to 133 MHz DEC 3000/400 Alpha workstations. At the end of the project, one 200 MHz DEC 3000/800 Alpha workstation was acquired and connected to the VuNet.

4.2.1 VupBoard

Using the VuSystem software, the performance of the VupBoard was measuring using video generated by a Vidboard. The VupBoard driver was a very processor-intensive process as it required making TURBOchannel bus reads for getting data from the network.

While this contributed to the processor load, hence slow interface speed, while reading and writing to the network, the primary limitation of the VupBoard was due not to this, but to the interrupt-driven nature of the board. Interrupt service latencies were not guaranteed in any way, causing data to be potentially lost in the buffers of the switch when the device was not able to be serviced while large amounts of data arrived.

Initially, with the DEC 5000, 10 Mbps of data was delivered to the application level. Upon upgrading to the DEC 3000/400, 15 Mbps of data was delivered to the application. As the VupBoard was phased out soon after introduction of the VudBoard, no numbers are available for the DEC 3000/800.

4.2.2 VudBoard

Using an application which reads and writes data to VuNet sockets along with the standard VuSystem applications, we tested the performance of the current VudBoard.

When a packet is queued for transmission, the packet segmentation occurs before the resulting page can be moved cell-by-cell over the TURBOchannel to the VudBoard. The peak data transmission of a single page occurs at 232 Mbps. However, since we can only DMA one page at a time (due to a bug in the DEC ASIC used), we must process an interrupt for each page to be transmitted. The performance achieved for transmitting many pages is 67 Mbps. On the receive side, 512 Kbit bursts at 232 Mbps can easily by processed by the VudBoard. The bottleneck on the receive side, however, is the host processing.

When data arrives at the host interface, it must contend for the TURBOchannel bus to transfer its data into host memory. Due to bus arbitration, grant latency, and a wait period between successive transfers, the host interface is unable to write to memory at a rate greater than 232 Mb/sec. However, since the host cannot process data at this rate, it cannot be sustained indefinitely. The maximum sustained rate at which data could be transferred across the bus was 131 Mb/sec.

When host processing is factored in and the data is reassembled and delivered to the application level, we find that we can indefinitely sustain a data rate of 100.7 Mb/sec of assembled data packets delivered to the application memory space. The device driver must process and reassemble raw ATM data at 111 Mb/sec in order to sustain this packet data rate to the application level. The actual data transfer rate over the TURBOchannel bus is actually 134 Mb/sec, but this reflects transfer of 64 bytes per ATM cell.

Performance of the VudBoard is summarized in Table 1, along with VupBoard performance numbers. Throughput measurements are made while running the full suite of ViewStation software within the UNIX operating system.

	Data delivered to the application level (Megabits per second)		
Interface	PIO	DMA	DMA/Improved Driver
DEC 5000/200	10.4	15.4	
DEC 3000/400 (133 MHz)	14.7	34.4	66
DEC 3000/800 (200 MHz)			100.7

Table 1: VudBoard performance measured by data delivered to application level.

4.2.3 Coprocessor Simulations

A coprocessor chip for the VuNet was designed and fabricated. Though it was not integrated into a MIPS-based workstation, it was partially debugged to the extent that cell registers could be read and write data using our logic analyzer/signal generator. A full suite of simulations were performed, and the following describes our experiments and results.

To measure the throughput attainable by the coprocessor host interface, we performed timing experiments on a floating point coprocessor. The processes' view of the coprocessor interface essentially consists of coprocessor general purpose registers and coprocessor control registers. These are available on a floating point coprocessor as well, and since the timing of all coprocessors (whether floating point or network interface) with respect to the main processor in the R3000 architecture is the same, the results we obtained can be expected with the network interface cell coprocessor also.

The workstation configurations evaluated were a DECstation 5000/200 and a DECstation 5000/240. We ran a series of experiments in which the **Interface Model**, **Transfer Direction**, **Cell Operations**, and **Processor Speed** were independently varied. The results are summarized in Tables 3 and 4. Some of the simulations present optimistic results that provide an upper boundary on the burst performance that could be achieved by the traditional interfaces. For example, our DMA values capture the data transfer rate between memory and the registers. Thus, we assumed that the DMA transfer has occurred at little interference to the processor, and the rate of this DMA transfer is much higher than the rate at which data can be accessed by the processor.

Name	Function
Vup	Programmed I/O VupBoard interface. For receive routines, this consists of loads from TURBOchannel to processor registers, followed by stores into memory. For transmit, it consists of loads from memory to the processor registers, followed by stores to the TURBOchannel.
Vup1	Same as Vup, except with no header byte swap.
Cop1	Similar to VUP, using the coprocessor approach.
Cop2	For receive, move header from coprocessor to processor registers. Then store body directly from coprocessor register file to memory. For transmit, load header into CPU registers, and load body directly from memory to coprocessor registers.
Cop3	In the receive direction, store entire cell directly into memory. For transmit, load entire cell directly from main memory to coprocessor registers.
Cop4	Same as Cop1, except with no header byte swap.
Cop5	Same as Cop2, except with no header byte swap.
Cop6	Same as Cop3, except with no header byte swap.
DMA	In the receive mode, load data that has already been stored into main memory by DMA. In the transmit mode, store data into main memory to be read by the network through DMA.
DMA1	Same as DMA, except with no header byte swap.

Table 2: Interface models

- As expected, the simulated numbers predicted a throughput higher than observed. In the programmed I/O path, the receive and transmit throughput suggested by the index numbers are 89 Mbps and 166 Mbps. This can largely be accounted for by the fact that we assumed that TURBOchannel writes and reads took 3 and 8 cycles respectively. Some logic analyzer experiments conducted by Chris Lindblad and Dave Tennenhouse indicate that the read performance on the TURBOchannel is 14 Turbo-cycles. This changes our predicted value in the

Direction	Vup	Cop1	Cop2	Cop3	DMA
DECstation 5000/200 (in Mbps)					
Receive	57.2	210.3	292.4	304.4	176.2
Transmit	79.7	104.6	120.7	124.9	96.3
DECstation 5000/240 (in Mbps)					
Receive	39.1	250.5	314.6	368.2	201.8
Transmit	100.4	187.9	226.1	247.2	151.4

Table 3: Peak Throughput (with header byte swap)

Direction	Vup1	Cop4	Cop5	Cop6	DMA1
DECstation 5000/200 (in Mbps)					
Receive	63.1	337.1	512.6	535.8	239.3
Transmit	91.4	127.6	150.3	152.3	116.0
DECstation 5000/240 (in Mbps)					
Receive	40.5	373.6	529.5	559.4	287.4
Transmit	105.2	243.6	299.8	302.5	180.3

Table 4: Peak Throughput (without header byte swap)

receive direction to 48 Mbps, which is much closer to the observed value. Unfortunately, no experimental values are available for TURBOchannel write direction.

- The observed throughput in the *DMA1* case are 290 Mbps and 196 Mbps, in the receive and transmit directions, respectively. These values take into account the *entire* path of data, all the way from the network DMA device, to the correct location in memory. The Bellcore Osiris board [7] runs at 480 Mbps and 384 Mbps. However, these figures are for the direct path between the DMA device and memory, and do not take into account the copying of this data into some final location in memory. Our predicted throughput figures (294 Mbps and 268 Mbps), are limited by the memory subsystem, rather than the throughput between the memory and DMA device. Thus these provide an upper bound on the overall performance achievable, if a DMA board, such as the Bellcore board, were used. Even if the DMA board could transfer at higher rates into memory, the total throughput would be bound by the performance of the memory system.

Direction	Predicted		Observed	
	Path	Rate	Model	Rate
Receive	IO1	89	Vup1	40.9
	IO3	294	DMA1	290.7
	REG1	486	Cop4	379.7
Transmit	IO1	166	Vup1	111.9
	IO3	268	DMA1	196.0
	REG1	358	Cop4	259.7

Table 5: Predicted vs. Observed Throughput (in Mbps)

4.3 Vidboard Performance

The high speed of the VuNet and the properties of its client interface make it so that the traffic characteristics of a data stream injected at one point in the network are preserved as the stream arrives at its destination. This property was exploited to tailor the traffic characteristics of a video stream generated by the Vidboard to the needs of the workstation receiving it.

Our software environment processes multimedia information in small bursts. Video frames are segmented into transmission frames (t-frames) which are transmitted in bursts separated by delays. A pseudo-AAL5 segmentation protocol [16] is used to ship the t-frames. The stream is shaped using the following parameters: number of scan lines per t-frame (LT), number of cells per burst (CB), interburst delay and interframe delay. These parameters are used to shape the video stream to match the burst absorption timing properties of the workstation. These properties are related to the latencies of the interrupts and routines involved in processing a burst of data.

	scan lines per t-frame (480 scan lines = 1 frame)				interburst delay
cells per burst (14 cells = 1 scan line)	4	16	120	480	
	(frames/s)				(us)
14	7.7	8.7	9.1	9.5	150
28	9.1	10.0	10.5	11.7	240
42	9.0	9.5	10.5	11.1	390
56	9.5	10.0	10.5	11.1	510

Table 6: Effects of traffic shaping on frame rate

	Video type	
Picture size	Black and white frames/s (Mbits/s)	Dithered Color frames/s (Mbits/s)
640x480	15.0 (37)	4.3 (10.5)
320x240	30.0 (23)	15.0 (9.2)
160x120	30.0 (4.6)	30.0 (4.6)

Table 7: Single Vidboard to Alpha frame rates.

The effects of traffic shaping on the frame rate performance of a video display application are reported in Table 6. The type of video stream used is 640x480 8-bit monochrome and the workstation used is a DEC 5000/200 using a VupBoard. The table shows the frame rate achieved for different values of the CB and LT parameters. For each pair of values, the delays are found that result in the best frame rate. The results show the intuitive conclusion that larger bursts and t-frames correspond to higher frame rates. Since the workstation takes a single interrupt to read in a burst, larger burst sizes mean less interrupt overhead over the course of a frame. The benefits of larger burst sizes decrease as the bursts become larger because the interrupt overhead becomes insignificant in comparison to the time spent reading the burst during the interrupt. Larger t-frame sizes increase the frame rate because less t-frame processing occurs.

Using the protocols described in Section 3.5.2, experiments have been conducted to determine the video frame rate which could be achieved between one or more Vidboards and an Alpha workstation displaying video with audio. Different types of video experience bottlenecks in different parts of the system: the Vidboard, at any resolution and depth, is limited by the 30 frame per second rate of the camera signal; at full resolution, the relative timing of the current host software and the Vidboard limit the performance of a single stream to less than the full frame rate; eight bit color

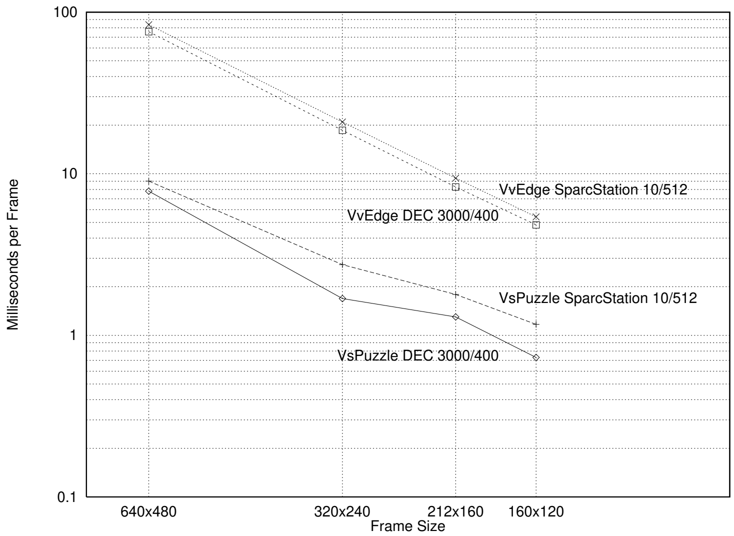

Figure 14: Processing Times of Representative Filter Modules.

frame size	puzzle frames/sec	edge frames/sec
640x480	12	6.67
320x240	30	25
212x160	15	15
160x120	30	30

Table 8: Rates of VuSystem Applications.

video streams are limited by the Vidboard's dithering firmware; and with twenty-four-bit color video, the bottleneck is a host software module that dithers the pixels for display on 8-bit screens.

The aggregate video rate to a single workstation, achieved with concurrently active Vidboards, was about 74 Mb/s. This is equivalent to full resolution (640x480) 8-bit black and white video at 30 frames per second displayed on the host display. The limiting factor was the alpha in that the segmentation, display and other processing required consumed the processor. In the case of color streams, dithering was the limiting factor.

4.4 VuSystem

Our experience with the VuSystem has revealed it to be a good platform for the investigation of concrete ways that computers may become more responsive to their human users. Users of the VuSystem have developed applications that assist users by performing various tasks that require the analysis of live video [13].

The VuSystem has also been used to explore the potential of media processing applications to support content-based retrieval of pre-recorded television broadcasts. VuSystem-based media browsers use textual annotations that represent recognizable events in the video stream. These annotations are analyzed and processed to create higher level representations that may be meaningful to a human user.

We performed a series of throughput experiments to measure the processing rates of the Exercise and Edge filtering modules. The Exercise module walks over each pixel in each video frame to simulate a variety of processing operations. It was chosen as a baseline. The Edge module detects and highlights edges within each frame. It was chosen as an example of an algorithm we use in practice. Both modules were written in C/C++ and not painstakingly optimized; comparable rates may be expected from the straightforward coding of other modules.

Both the DEC 3000/400 and SPARCstation 10/512 were found to be able to process video at up to 15 Mpixels per second, regardless of frame size. They may perform edge detection at better than live rates for 320 x 240 x 8 bit frames. This shows that visual processing of live video in software is viable. We expect further improvements with future advances in workstation technology.

Module	DEC 3000/400 ms/frame (fps)	SPARC 10/512 ms/frame (fps)
Exercise	5.0 (200)	4.5 (220)
Edge	18 (55)	20 (50)
WindowSink	8.3 (120)	10 (100)
FileSource	31 (32)	38 (26)
Vidboard	33 (30)	–

Table 9: System Throughput

The rate at which video may be passed to the X server for display was measured with a WindowSink module. The FileSource module was used to measure how quickly video could be sourced from a local SCSI-II disk. The Vidboard module was used to read live video via a VuNet [11] interface from a Vidboard [1] video capture peripheral. This last module was available only on the DEC 3000/400s. Table 9 shows the observed rates for 320 x 240 x 8 bit frames, and includes rates for the Exercise and Edge modules for comparison.

The WindowSink module was shown to be capable of processing video at over 100 frames per second. Video can be passed to a windowing system, in this case the X Window System with the XShm shared memory extension, and displayed without writing directly to the frame buffer. On the multiprocessor SPARCstation 10/512, the operating system was able to achieve a considerable speed-up over a single processor SPARCstation, presumably by running the X server process in parallel with the video application.

The FileSource and Vidboard modules were observed to process video at approximately 30 frames per second. This is a lower rate than that for the WindowSink and Exercise/Edge modules, but the figures are less conclusive.

The Vidboard result exceeded the frame rate of live video and is primarily bounded by the structure of the network interface. It again serves a lower bound.

The rates of Exercise and Edge filtering modules, from 50-200 frames per second, compared favorably with those of the sources and sinks. We consider this a positive result for our approach, since the incremental cost of processing video does not dominate. Applications such as live video edge detection and display run at approximately 15 frames per second.

VuSystem module-level dispatching is non-preemptive, and depends on the Unix process scheduler for dispatching operations. To verify that this scheduling system is adequate for many perceptual-time processing tasks, we measured the exact time that Timeout member functions run. We instrumented the system to compare the actual time at which a Timeout is called to the time for which it was scheduled. If the VuSystem performs well, the difference between the actual time and the scheduled time will be slight. We recorded this Timeout precision for runs of the vsdemo program. Four runs were taken, each indicating Timeout precision under different system loads, created by running multiple concurrent vsdemo processes.

Figure 15 shows a histogram of the percentage of Timeout calls, as a function of the number of milliseconds after the scheduled time that Timeout was called. It has been truncated at 20ms.

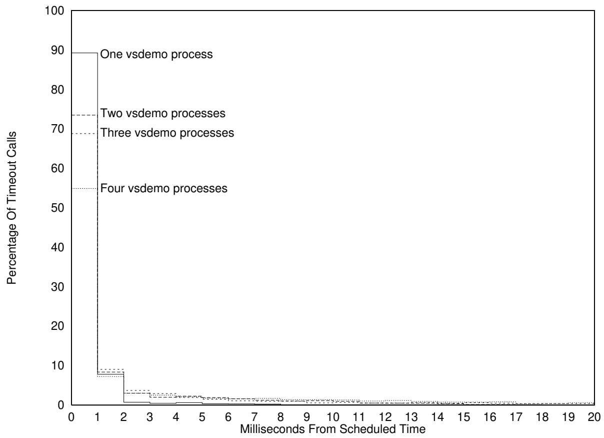

Figure 15: Timeout Precision Histogram.

	one process	two processes	three processes	four processes
5ms	98.9%	89.2%	86.4%	69.8%
10ms	99.9%	96.0%	91.5%	77.6%
15ms	99.9%	98.8%	94.3%	82.8%
20ms	99.9%	99.3%	96.7%	85.7%

Table 10: Timeout Precision.

These measurements were made on the Digital DEC 3000/400, where the precision of the time-of-day clock is one millisecond.

Table 10 indicates the cumulative percentage of Timeout calls made within 5ms, 10ms, 15ms, and 20ms of target, as a function of the number of multimedia processes running on the Digital DEC 3000/400. The chart shows that almost all of Timeout calls were made well within the universally accepted 15 ms of jitter introduced by the "3-2 pulldown" process used to convert motion pictures to NTSC video. The chart also shows that precision of timeout calls gracefully degrades with increased system load.

5 Lessons Learned and Concepts Demonstrated

The TNS group's testbed activities covered the development of the VuNet, an ATM-framed local distribution system; research on the design of host/network adapters; and the demonstration of network-based multimedia applications.

Through deployment of the VuNet infrastructure, the seamless integration of the wide-area Aurora facilities, and the use of the VuSystem software, we were able to demonstrate a wide variety of concepts and draw lessons concerning:

- The distinction between cell switching and ATM layer functions
- The striping of ATM transmission facilities for seamless integration
- The integration of ATM-based Information Appliances
- The host interface design space
- The traffic implications of computation-intensive multimedia applications

5.1 Cell Switching and Desk Area Networking

By simplifying the switch design and integrating certain functions into links and end nodes, we effectively separated the cell switching functions from the ATM level cell relaying functions. This was an important distinction which may prove useful in a number of ways.

Minimal hardware functionality in the switch is an important consideration for a local area ATM network. The switches need not perform complex functions such as those incorporated into the Sunshine switch. In a desk area, only a small number of switch ports need full ATM functionality. For example, serial to parallel conversion and VCI mapping are not required by directly attached devices; they are needed only at switch ports supporting inter-switch links.

Although current ATM standards intertwine cell relaying and ATM level functions, we suggest that the relevant standard bodies (ATM Forum, ITU-T) consider dividing the present ATM layer into cell switching and ATM functions. The former could then be used as the basis of a universal I/O system based on cell switching. Such an approach would unify the haphazard and expensive collection of connectors found on the backs of existing PCs and workstations. While some of the devices attached to such an I/O system might be fully ATM compliant, others, such as keyboards, mice, etc., may only implement the cell level functions. These non-ATM devices could use the features of cell switching without the complexity of VCI mapping, etc.

5.2 Seamless ATM and Striping

The AVlink demonstrated that one advantage of using ATM in both the local- and wide-area networks is the simplicity of interconnecting them. The AVlink only needs to make minor cell changes at the ATM and physical layers, which can be done as the cells pass from VuNet to Aurora. The design of the AVlink allows the LAN/WAN boundary to appear seamless to clients on either side of the interface. Hence, applications which worked within the local area worked just as well over the wide area.

The effort to solve the skew problem caused by the SONET facilities highlights striping as an important architectural issue. There is an innovation mismatch between local area networks and the wide area facilities provided by the telephone carriers. Telephone companies will only upgrade their facilities when the aggregate demand of their customers is large enough to recover their investment. Local area networks cost much less to upgrade and maintain than the wide area facilities. Aggressive users of LANs who want a leading edge wide area connection find themselves in a position where they are unable to get the amount of bandwidth they need or want at a cost that is acceptable. This coupling of technology in the two domains is a barrier to innovation and migration to new equipment. Network striping is a method to decouple the selective adoption of technology in the local area from its wholesale deployment in the wide area.

Striping at the ATM layer is particularly attractive as it permits us to implement an end-to-end striping scheme but still derive the low level benefits of using a small, fixed-size striping unit. Striping at a higher layer reduces the range of upper layer protocols and applications that benefit. Striping at lower layers prevents us from implementing an end-to-end solution. Whereas most previous striping efforts have focussed on providing increased bandwidth for all traffic over a single hop, there is a need for end-to-end striping solutions that can provide increased bandwidth to the aggressive users who require it and that are not dependent on the deployment of striping support at nodes within the network. End-to-end methods that are (responsive to but) independent of the characteristics of the underlying physical layer will adapt as the equipment in the network is gradually upgraded.

5.3 ATM-based Information Appliances

A number of important conclusions, concerning the properties of a network-based video capture module, can be drawn from the Vidboard research. The ability to temporally decouple video from its television source combined with a closed-loop control model facilitates the integration of video into the virtual-time computing world. Video service can be adapted to the available system resources and degrades gracefully as resources become scarce. A second conclusion is that a capture module needs to be able to accomplish tasks related to the distributed nature of the environment and not only capture/transmit video. Task examples are network transport, traffic shaping and distributed control.

With the trend towards integration of video into the digital domain, research groups are developing cameras which produce video in a digital format [18] that will come to replace television camera/capture board systems for computer video applications. Similarly, with the trend towards distributed computing, video capture systems which connect directly to a network are becoming desirable. A digital network camera would consist of a lens, a CCD array, and circuitry for network interfacing, distributed control and frame buffering. The Vidboard architecture could provided the basis for this camera of the future.

5.4 The Host Interface Design Space

A thorough approach to host interface design can be taken by answering five questions—the *division of protocol processing* between the host and the interface board, the method of *data transfer*, the *role of memory*, the *point of attachment* to the host's memory hierarchy, and the *synchronization means* by which the processor and interface coordinate their actions. Most of the previous work on host interfacing, including comparative studies of different approaches, has focussed on *one* of these issues at a time. Furthermore, there has been relatively little work concerning the role of memory.

The above questions should be answered within the context of a specific operating environment. Specifically, it is important to consider whether or not the processor is expected to touch all the words of the network data. For example, a file server may move data (via its memory) between its disk and network interfaces without moving the data to or from the processor registers or caches. Similarly, applications receiving network data are often blocked, leaving the processor available for protocol processing, whereas a server might have other clients, not waiting on the incoming data, that could make better use of the available cycles.

Although the TNS group's activities considered many aspects of interface design, we made a conscious effort to avoid the issue of hardware-based segmentation and reassembly(SAR). Other groups within Aurora (Bellcore, Penn, Arizona) were actively investigating this question and we did not want to duplicate their efforts. Accordingly, our results and lessons should be viewed as complementary to theirs, rather than competitive.

5.4.1 Protocol Processing and Data Transfer

Using the VudBoard, we explored the design and use of a simple DMA interface to a gigabit network. The primary functions of the interface and the driver software - are to move, process and deliver data. While our host-based SAR approach involves considerable copying and processing on the part of the host, we find that reduced complexity interfaces (RCI) can deliver good performance at reasonable cost.

Changes to our driver's API, such as Arizona's FBUFs and Penn's buffer strategy, could be used to avoid one of the copies. An important question is whether or not to adopt a hardware-based SAR approach to avoid the other copy. The answer to this question will depend on a number of factors, including cost and the environment in which the host will be used. In many cases, especially server

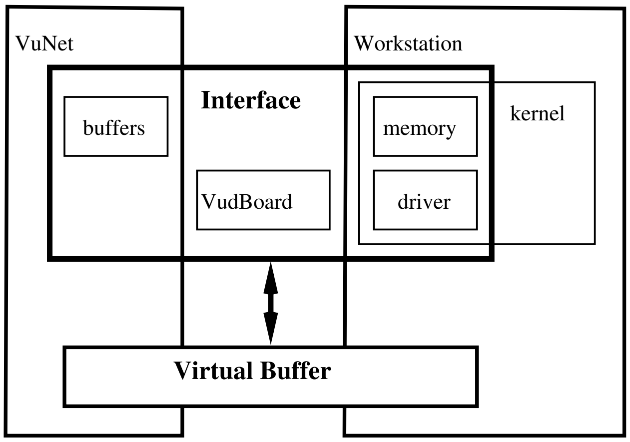

Figure 16: The memory distributed throughout the various parts of the system can be thought of as a large virtual buffer.

environments, we would opt for the reduction in memory copy traffic made possible by our partners' work on hardware-based SAR.

Nonetheless, the VudBoard's simple approach, even without an FBUF-style API, achieves a level of performance that is compatible with commercially available (OC-3) switches. Furthermore, the relative performance of memory subsystems, which had been rather stagnant, is now improving, partly in response to lessons learned within the testbeds. As memory subsystems continue to improve – driving down the cost of the SAR-imposed memory copy – gigabit rate host SAR may become viable, possibly by the time OC-12 switches are widely available.

5.4.2 The Role of Memory

The design of the VudBoard and its associated device driver demonstrates an important lesson concerning the role of memory. Since the buffers of the switch port are limited to a small number (256) of cells, the interface must maintain a nearly continuous flow of cells into host memory. In practice, the switch and host buffers are concatenated together by the interface and can be thought of as one large "virtual buffer," as shown in Figure 16.

The interface and device driver are designed to preclude lapses that would cause cells to be lost at the network boundary. Interestingly enough, it was this consideration rather than raw throughput that drove us towards a DMA based design – one in which the transfer of cells would continue while the processor was performing other tasks. The only time that cell transfer is interrupted is for brief periods, measured in tens of instructions, during which the device driver updates the control registers (especially the buffer pointers) of the interface. Cells arriving during these periods are *absorbed* by the switch buffers.

In retrospect, we are now more sensitive to the roles that various bits and pieces of memory play. The switch output buffers absorb bursts of cells from multiple sources and are also used to absorb cells while the host is performing DMA maintenance; the memory on the interface board (e.g. the FIFO of the TcIA chip) is used to decouple the timing of the host from the network; and the kernel buffers are used to hold large bursts while the host is processing the data. Ultimately, the caches feeding the processor registers decouple the timing and throughput of the processor from that of the memory subsystem.

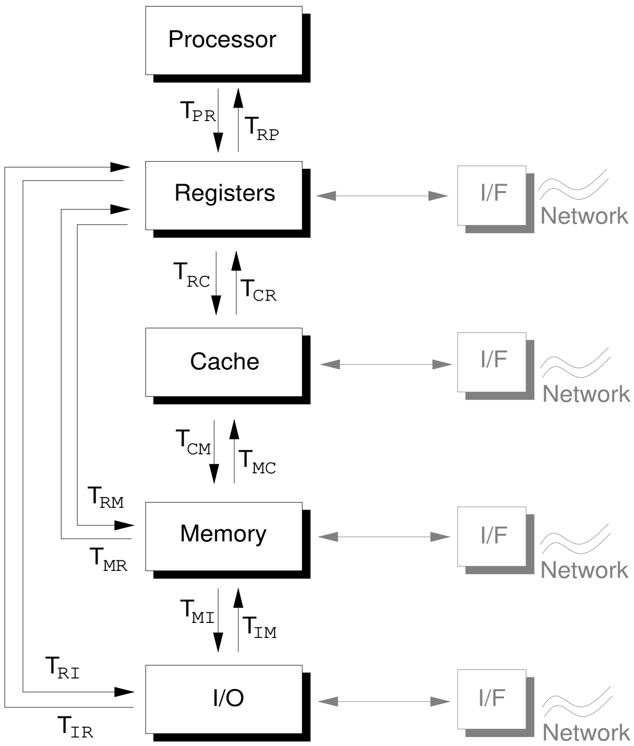

Figure 17: Model Connection Architecture

5.4.3 Network Point of Attachment – Coprocessors and Temporal Decoupling

A study of modern RISC architectures reveals a remarkable similarity in the organization of the various modules of the memory and I/O system. To focus attention on the relatively under-emphasized question of the choice of the point of attachment, we have developed a model of the memory-I/O architecture of a workstation which allows us to identify potential "plug-points" for network interfaces. This "Model Connection Architecture" is shown in Figure 17. The figure reflects the functional relationship of the various modules of the memory hierarchy, and does not necessarily describe the physical placement of the various modules.

In Figure 17, we see the different modules that a network adaptor can emulate, to make the network appear to reside at the different levels of the architecture. Thus, the network can be made to appear to reside in an I/O device of the workstation, in main memory, in cache[4], or in registers. Different issues arise when the network adaptor emulates each level.

Within Aurora, the I/O and register (coprocessor) approaches were investigated. Other projects such as Alewife [2] have investigated pseudo-caches, and yet others have experimented with pseudo-memories.

Although the coprocessor approach appears attractive, we learned that it is only suited to a limited range of environments, such as single task embedded applications. In these environments, the receiving or transmitting application is always active and the processor registers are available to perform timely cell processing. In effect, the processor is synchronized with the cell streams arriving and departing through the coprocessor interface.

In a multi-tasking environment, applications run in timeslices during which bursts of data are generated and/or processed. During these bursts, the application may process data at rates faster than the

[4]Most systems have several levels of cache

network. Between timeslices, the network runs faster than the application. For these non-embedded applications, approaches such as DMA, that leverage the burst absorption properties of the memory subsystem, are more appropriate than a coprocessor.

The broader lesson that we have learned is the importance of temporal decoupling, i.e., allowing various system components to manipulate data at differing rates and granularities. Memory subsystems (including caches) have been carefully designed to absorb bursts and handle speed and granularity mismatches.[5] Therefore, it is our current belief that general purpose interface designs should not bypass the memory subsystem.

5.5 VuSystem Applications - Implications for Network Traffic

Although recent network and multi-media research has paid considerable attention to the support of long-lived streams of continuous traffic, we believe that the network traffic generated by media-intensive applications will be dominated by its available bit rate (ABR) components. Our chain of reasoning is as follows:

- Actual multimedia capture traffic is not continuous, but instead is periodic bursts of cells. Data is captured at a low average rate, but transmitted in packets at the available bit rate. Furthermore, our closed-loop capture approach [1] allows us to adapt to the available bit rate during video capture. This type of feedback has long been used in video displays.
- Intelligent applications might have radically different display traffic patterns. For example, a security monitoring program might only update the display when it recognizes suspect activity. In effect, much of our display traffic is of the X-windows variety.
- Through increased buffering at capture and display, we relax the temporal constraints on data transfer, thereby allowing available bit rate surpluses to offset shortages.
- For every bit transferred over the network for capture and display, there is a *multiplier* effect — many more bits are transferred in support of storage and media-intensive processing. This traffic is totally decoupled from the real world, since our file servers and media processing programs are decoupled from real-time capture and display phenomena, such as sensors and phosphor.

The impact of the technology curve on the final two points is significant. Declining memory prices will lead to larger capture and display buffers. Similarly, exponential growth in processor and network capacity will yield increasingly intelligent applications that have larger multiplier effects. Although some continuous bit rate (CBR) and predictable variable bit rate (VBR) components may remain they will be relatively insignificant.

Table 11 categorizes the network usage for the capture and browsing components of the ViewStation applications we have developed. The table indicates the video data format, network traffic type, bits transferred, subsession duration, and subsession frequency for example application components. For application components that pass data in periodic packet-trains for long periods of time, the number of bits transferred is indicated per second. Otherwise, the number of total bits transferred is indicated. We define a subsession as a single transfer of a video segment. One program session could include many subsessions. For example, a Joke Browser session consists of a subsession for each joke replayed.

Historically, as computer networks have increased in capacity, traffic becomes burstier. Traffic patterns are bimodal, with the bulk of the data transferred in a relatively small number of large bursts. Over time, the size of the large bursts continues to grow, and we expect this pattern to continue. The video server of the future will closely resemble the file server of the present — but the units of transfer will be measured in gigabytes instead of kilobytes and megabytes. Consequently,

[5] The VudBoard design, in which we *pour* network data to / from memory leverages this burst absorption property.

Component	Video Data Format	Network Traffic Type	Bits Transferred	Duration of Subsession	Time Between Subsessions
Room Monitor					
capture	raw	periodic	6 Mb/s	days	-
storage	processed	file	100 Mb	seconds	minutes
retrieval	processed	file	100 Mb	seconds	seconds
display	X protocol	periodic	6 Mb/s	seconds	seconds
Whiteboard Recorder					
capture	raw	periodic	600 Kb/s	days	-
storage	1 frame	file	600 Kb	1 frame	minutes
retrieval	1 frame	file	600 Kb	1 frame	seconds
display	X protocol	interactive	600 Kb	1 frame	seconds
Video Rover					
capture	raw	periodic	6 Mb/s	minutes	minutes
display	X protocol	periodic	6 Mb/s	minutes	minutes
News Browser					
capture	raw	periodic	6 Mb/s	1/2 hour	daily
storage	processed	file	10 Gb	1/2 hour	daily
retrieval	processed	file	500 Mb	minutes	minutes
display	X protocol	periodic	6 Mb/s	minutes	minutes
Sports Highlights					
capture	raw	periodic	6 Mb/s	1/2 hour	daily
storage	processed	file	10 Gb	1/2 hour	daily
retrieval	processed	file	100 Mb	seconds	seconds
display	X protocol	periodic	6 Mb/s	seconds	seconds
Joke Browser					
capture	raw	periodic	6 Mb/s	10 minutes	daily
storage	processed	file	3 Gb	10 minutes	daily
retrieval	processed	file	100 Mb	seconds	seconds
display	X protocol	periodic	6 Mb/s	seconds	seconds

Table 11: The network usage classes for some ViewStation application components.

future networks must support traffic patterns that are at least as bursty as those on present-day LANs, at speeds and burst sizes that are orders of magnitude larger.

The presence of bursty traffic may have significant implications with respect to the distribution of memory buffers across the network, and the presence of short subsessions may affect the design of network signaling systems. If network traffic is fundamentally bursty even at large bit-rates, then high-speed networks must be designed to support higher peaks in load than if the traffic is less bursty. Except for backbone networks that support highly aggregated traffic, gigabit network designers should not plan for high utilization, but instead should accommodate highly variable loads, especially during peak busy hours.

Section 2: ANA Group Activities

1 Introduction

The research of the Advanced Network Architecture group had several high level objectives.

The first was to demonstrate that workstations, not just supercomputers, could be high performance components of a fast network. While supercomputers represent a valid target for experimentation, there are several reasons to concentrate on the workstation. The workstation architecture represents the form of computing that will have the major economic impact on the market. The supercomputer will represent, in any generation, that machine too expensive to produce in quantity. So experimentation that leads to advances in workstation performance are most directly relevant to advancing the state of the art in broadly applicable ways.

The second objective was to show that in order to achieve high performance, it is not necessary to restrict the network to one technology, such as ATM or HIPPI. While much of the attention at gigabit rates has been focused on specific technology options, our experience with IP has taught us that no one technology is likely to dominate the marketplace: today we have competing high speed LAN technology, specifically 100 mb/s Ethernet and ATM, and we have ATM, SMDS, Frame Relay and IP as competing wide area transport offerings from the telecommunications industry. We saw no reason why the gigabit arena should be different. We thus proposed, as a core research goal, to recognize the issues of heterogeneity at high speeds, and to understand if there were critical issues that needed to be addressed. We desired to show that high performance can be achieved in heterogeneous networks with different sorts of technology bases combined within them, in the way the Internet is today built out of different sorts of technologies.

It is interesting to see how expectations have changed since the first proposals for Aurora were drafted. At that time, many people wondered if IP and in particular TCP were suitable for use at gigabit speeds, and many people thought that gigabit communications implied supercomputers. During the time of the project, these perceptions were essentially reversed. It became clear that because most supercomputers had I/O architectures poorly suited to networking and processors targeted heavily towards numerical computation, advanced workstations could execute network protocols as fast or faster than these larger machines. Further, most people now assume that once computers have sufficient processing power and memory bandwidth, IP and TCP will operate at gigabit rates.

This change in perception caused us to alter the specific objectives of our research. For example, at the beginning of the project, we proposed to build a high speed router, both to further our research objectives and because it was not clear that people knew how to structure such a device. Later, it became obvious that the construction of this device would be a waste of time, because commercial products were proceeding toward the target naturally.

2 Research Goals and Approach

To address these top level concerns, we identified a number of specific research areas. We looked at several aspects of heterogeneous computing, including the performance aspects of interconnection. We looked at issues of resource management, in particular support for explicit Quality of Service (QoS) control and approaches to congestion control. We looked at alternative models of protocol modularity, and alternative models of link multiplexing. We looked, to a limited extent, at issues of scale in high speed systems. We participated in the testbed efforts in switch design. Finally, we constructed a number of experimental devices to evaluate our concepts.

2.1 Heterogeneous Interworking

We define heterogeneous network architectures as those which can flexibly incorporate many network and end-system technologies, and which can easily evolve over time as new technologies are developed. Within the Aurora project, our overall goal was to demonstrate that heterogeneous network architectures are effective and appropriate for communication at gigabit rates. We looked at existing paradigms for heterogeneous interconnect, in particular IP packet forwarding, and assessed the performance requirements of this task. We then mapped these performance requirements to possible architectures for a gigabit forwarding element (the gigabit successor to a IP router). Finally, we proposed a novel approach to organizing protocols, an approach we call Application Level Framing, or ALF. We hypothesized that ALF might have better characteristics as an approach for heterogeneous interconnect. We discuss these study areas in the sections below.

In general, heterogeneity is closely aligned with architectural structure and modularity. Systems with modular organization and well-defined interfaces are more flexible and evolvable than those designed as a single, unified entity. However, these otherwise desirable principles may introduce significant performance limitations if not carefully executed. A goal of our work was to carefully differentiate between fundamental performance limits and those created as an artifact of poor structuring, and to identify new structuring abstractions appropriate for use in high-speed networks.

As an aid to understanding the real problems of heterogeneous interconnect, we coded a high performance IP forwarder, in order to understand its performance implications, both fundamental and structural. We measured its performance, and used software tools to assess where the performance issues lay.

It has been suggested that any use at all of operating system or similar modular abstractions inside the high-speed forwarding path of a router will introduce unmanageable overhead, and instead one must implement the forwarding software as stand-alone code or in hardware. To understand this issue, we explored one operating system, the x-kernel [12], which is specialized for the implementation of network protocols, but at the same time attempts to provide an abstract interface to key system services such as buffer management.

2.2 QoS

An overarching research objective of our group has been the development of models and mechanisms for the support for Quality of Service. This term is generally taken to mean the capability of a network to offer multiple, controlled levels of service, and of an application to request from the network service which explicitly offers the needed values of key performance parameters such as bandwidth, delay, and loss rate. Our core work in this area includes the study of application requirements vis a vis traditional network service models, the development of new service models that the network can present to the application, the development, simulation and implementation of new algorithms which implement these service models through queue management and packet dropping, and the development related protocols for setting up resource reservations and for admission control.

Our work in this area is broadly based, combining activities within the Aurora context, collaborative efforts within the ARPA-funded DARTnet testbed, and within our local research group. This broad perspective has enabled us to identify and focus on solutions applicable to a wide range of networking environments. Our work has led to proposed standards for providing this capability in the IP protocol family, and in part to the current approach to adaptive bit rate service in the ATM forum.

Within Aurora, our objectives have been the following. First, we wished to determine if any fundamental, as opposed to implementation, issues arose from the high speeds involved. Second, since ATM is a key technology for emerging broadband networks, we wished to apply our overall results specifically to the ATM context.

A key issue arose with respect to ATM. While ATM carried the promise of sophisticated QoS support, the standards at the time did not realize this capability. Rather, they defined an interface to the capability from higher layers, without giving any guidance as to how ATM network elements might actually support it. This implied the need to develop a framework for management of bandwidth within ATM. We approached this by adapting our more general framework to the specific requirements of ATM We then studied our adaptation through simulation, and through the use of a programmable output port controller board for the Bellcore Sunshine ATM switch, which allowed us to demonstrate cell-level traffic management in the ATM portion of the Aurora network.

High speeds imply that the data forwarding path in the router or switch must process a large number of packets or cells per second; at 622 mb/s the cell rate for an ATM link is over one million cells per second. This implies that any cell-level processing must be simple and efficiently implementable. We thus undertook the specific project of tuning our basic QoS control algorithms, which support a rather sophisticated model for allocation of network bandwidth, for the specific case of ATM forwarding. Our approach was to evaluate the detailed performance characteristics of the algorithm, and to propose approximations to the exact algorithm that may be more efficiently implemented.

2.3 Congestion Control

A topic closely related to support for QoS is the problem of congestion control at high speeds, especially in ATM networks. Two general approaches to this problem are available; hop-by-hop control and end-to-end control. Our work focused on the end-to-end approach, which is used in the Internet today.

Congestion control in the IP protocol family is handled primarily by TCP[1]. TCP contains an algorithm for rate adaptation, based on packet loss as a feedback signal. Early experiments with TCP over ATM, performed at a number of sites, demonstrated that improper design of the output buffers in the switch were causing cell losses, which interacted with the TCP algorithms to cause very poor performance. For this reason, we studied the congestion control and avoidance algorithms for TCP, which might be implemented in the host running the TCP, or perhaps even in the network. Our approach was based on simulation using highly detailed models of a current best-practice TCP and of the Sunshine ATM switch.

Current congestion control algorithms in TCP hunt for the correct sending rate; increasing their sending rate until they encounter an indication of congestion (a lost packet) and then backing off, after which they begin again to increase their rate. The current backoff algorithm can reduce the achieved sending rate to one packet each round trip, which is such a reduction that it can effectively prevent effective use of any high speed link. We thus explored a number of modifications to the current implementation practice for TCP, with the goal of improving its effective performance over links of high delay and bandwidth. These were evaluated using our network simulator and real tests over portions of the testbed and the Internet.

2.4 Alf

Application Level Framing, or ALF, is a proposal to restructure the modularity of network protocols. There are a number of objectives for this remodularization, but the one relevant to the gigabit environment was the hypothesis that the breaking of the data into transmission units would be more efficient with ALF than with traditional packet switching. In particular, for ATM, the application data unit, which might be substantially larger than a packet, could be directly broken into cells rather than being first broken into packets.

[1] This is becoming more and more unacceptable as UDP-based real-time data traffic grows to be a significant percentage of total internet traffic.

These architectural ideas were first proposed by us in 1990 [4]. We hypothesized a number of advantages, one of which related to speed. We thought it might be possible to build a faster router forwarding ALF elements than forwarding IP packets. The reason for this is that the ALF approach appeared to reduce the number of routing and scheduling decisions needed to achieve the same data throughput rate.

To better understand the real issues in this approach we reduced the conceptual framework of ALF to a specific technical proposal, which included details such as header formats and forwarding algorithms.

To validate this specification, we coded the core of ALF, including the operating system support, in the Unix operating system. The first implementation was not targeted towards an aggressive performance target, but allowed us to understand the issues of state maintenance, the structure of demultiplexing and so on.

Our experience with this implementation allowed us to construct a performance profile for the ALF forwarding process, which we then mapped on the hardware requirements for a high performance forwarder.

2.5 Alternatives for Link Multiplexing

A key issue for ALF is the problem of access latency at the switch. The transmission of a large application data unit over a slow link might delay the sending of a subsequent unit beyond acceptable limits, especially in the case of data flows requiring bounded real-time delivery QoS. It is thus necessary to break the ADU up into link-layer multiplexing elements, such as packets or cells. We explored an alternative, transmission of integral ADUs, but with pre-emption of the transmission if a higher priority data unit arrives. This approach permits us to build a system with the transmission latency of ATM cells, but which has none of the disadvantages of the small cells. We explored this approach as a way of better understanding the intrinsic issues in interleaving traffic with different QoS requirements and different data unit sizes. We proposed a specific switch design based on pre-emption, and performed detailed simulations on this design. By detailed simulation of links and switches, we uncovered a number of interesting issues, including a data clumping phenomenon that arises when small and large data units are mixed.

2.6 Large Scale Distributed Systems

While the major focus of the testbed was on speed, we explored issues of scale and decentralization. We discussed with Bellcore the design of the signaling system proposed for the Sunshine switches, called Expanse. We installed the package from Bellcore called Touring Machine, both to evaluate the possibility of using it as an application platform for Aurora, and to understand its basic control framework.

2.7 Switch Design

While our top-level interests are architectural, architecture must be validated by an understanding of the technology issues that arise in reducing architecture to practice. We participated in the design discussions for the Sunshine switch, both to assist in assuring that the switch would be suitable for the range of intended experiments and to learn about the interplay of design issues in developing the key network components.

We also developed, as a part of the MIT network simulator, a component modeling the Sunshine switch, that could form the basis of detailed comparisons between the simulated and real environment.

2.8 Prototyping Platforms

A core component of our experimental program is the development of forwarding and end-node protocol software, both to understand the structural issues in reducing architectures to practice and to understand where the performance bottlenecks lie. This objective required the selection and development of a hardware and software environment for these experiments.

The focus of the testbed generally was on workstations and high-end RdesktopsS rather than on supercomputers. While we understood that workstations might have intrinsic performance limits that would preclude operation at a gigabit rates, we felt that workstations were more representative of the eventual context for high performance networks, that workstations presented a more obvious and less convoluted memory and I/O architecture, and that workstations were likely to offer a more open and malleable programming environment than a supercomputer.

For this reason, our focus in developing a software testbed environment focused on the issues of operating systems and network interfaces to machines of the workstation class.

3 Experiments and Results

3.1 Heterogeneous Network

We implemented a fully functional IP packet forwarder with in the x-kernel protocol implementation framework. We then analyzed the performance of this implementation and developed several modifications to the x-kernel framework which removed artificial performance limitations. Our goal in this experiment was to understand and measure the ratio of required functionality to overhead in this implementation, and by extension in other well-tuned protocol implementation frameworks. Understanding this ratio offers an approach to determining whether high-performance routing elements can be implemented within a framework, or must be implemented completely Rfrom scratchS on raw hardware.

The implementation of IP in the x-kernel was first tested in the context of a null network interface, an interface that required no instructions and thus imposed no overhead on the processing. Running on a Mips 3000 processor (a 33 MHz machine), the code was capable of forwarding approximately 30,000 packets a second, and faster RISC processors continue to yield faster forwarding rates. The instructions executed were about equally divided between the IP forwarding code itself, and the x-kernel. This level of performance is consistent with what other careful implementations of IP have demonstrated on RISC processors. The major overhead of the x-kernel is the creation of a thread to handle the forwarding of each packet. However, the consequence of this approach to kernel organization, the factor of two in performance to run the x-kernel, is probably not acceptable for any context except an experimental platform.

We next proposed and implemented some extensions to the x-kernel to reduce the overhead of thread and message buffer management. These modifications allowed us to increase packet forwarding rate of our experimental system to approximately 55,000 packets per second. After these modifications, approximately 60 percent of the CPU cycles were used to perform core forwarding requirements, while 40 percent reflected overhead. The additional increase in performance is due to improved instruction scheduling and memory cache utilization.

These experiments gave an upper bound on the IP forwarding performance of our experimental hardware platform's CPU and memory subsystems. Note that they intentionally elided the cost of managing hardware interfaces.

We then extended our forwarding code to drive Ethernet (AMD RLanceS controller), ATM (Bellcore ROsirisS interface constructed for Aurora) and MIT's FLORB interface. Detailed performance results for the experimental interfaces are given below. With our implementation of the ethernet interface driver as the only hardware interface, the experimental forwarder operated at approximately 3000 packets per second. This number is lower than the figure suggested by the raw capabilities of the hardware. We determined that the packet rate was limited by the workstations I/O subsystem and bus arbitration scheme.

The core costs of processing a packet at the IP level are minimal. While it is difficult to extrapolate from specific implementations to a general conclusion, some rough estimates are helpful in getting a perspective on the requirements. Our code audits suggest that IP forwarding can be coded in from 50 to 200 RISC instructions. The important processing steps, with costs in our context, are the following.

- Checking header fields for correctness – 15 inst.
- Computer header checksum – 15 inst.
- Decrement TTL field – 5 inst.
- Look up route in cache – 25 inst.
- Add local network header to packet – 20 inst.

One in three of these instructions touches memory, either the packet or the forwarding data structures. If we assume one cycle per instruction plus one cycle extra for a memory reference, the total cycle count to process a packet is 107. A 100 mHz processor could thus process IP headers at a little less than 1,000,000 packets per second. At this rate, the minimum packet size that would allow a 622 mb/s link to be fully utilized is 83 bytes. Based on this sort of analysis, it is reasonable to conclude that a practical gigabit IP forwarder can be constructed using a software forwarding engine based on a general-purpose RISC CPU.

Our assessment of the processing overheads of the IP and ALF forwarder (see below) led to the conclusion that the burden of the IP processing was not the dominant cost of forwarding. In our initial experiments, described above, the cost of managing hardware was substantially greater than the processing cost at the IP level, as shown by the greatly reduced packet rate when the hardware interface was added to the system. Analysis of the code required to drive other common commercial network interfaces suggests that this is a generally accurate conclusion.

Even with specialized operating systems and device drivers, we found that the cost of the device drivers for traditionally designed interfaces dominates the processing costs at the IP level. Adding to the cost of the device driver the cost of queue management for QoS scheduling, we conclude that the overhead of the device driver is the key to high-performance processing. One does more to speed up packet processing by changing the way one controls the I/O interface than by changing IP.

3.2 High Performance Flow Forwarder

On the basis of the above analysis, we proposed a design for a high performance packet or flow forwarder. While the time-frame of the project did not permit us to implement this device, we believe that the original design approach, first proposed in June of 1991, remains correct. This design, which is illustrated in Figure 1, involves a processor to manage each device, a processor to execute the forwarding code for each input path, and a processor to perform the background control tasks such as route computation and network management services. We estimated that given the processing rates of advanced RISC processors, this architecture could support reasonable packet processing rates for gigabit links.

In addition to the packet processing limits, a forwarder must have sufficient memory and bus bandwidth to pass the data at gigabit rates, and sufficient decouplingJbetween the modules outlined above to allow the modules to function in parallel. To deal with the data rates implied by this architecture, we proposed two designs. In one, there is a buffer at the middle of the data path, with a data moving element attaching each network device. In the other design, especially suited to a

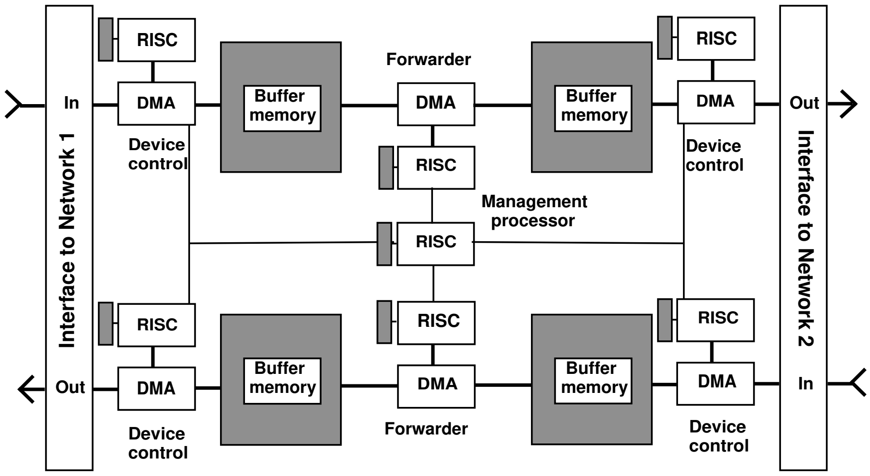

Figure 1: Block Diagram of High-Performance Flow Forwarder

two-port device, there is a efficient data mover at the center of the data path, which cross-connects the buffers on each side. This latter design seems to offer some advantages, and was used in the development of the FLORB interface, discussed below.

3.3 Memory Architecture of Workstations

We performed a number of tests on a (then) state of the art workstation (Digital Equipment Corp. Model 3000/400) to characterize latency and throughput across the I/O interface and the memory bus, and CPU caches. Our intention was to understand the workstation's performance adequately enough that we could later relate network performance to these fundamental properties of the hardware implementation. Our goal was to differentiate between limits imposed by the hardware and limits imposed by poor structuring of software, and to be able to determine how fast a particular network-intensive application, protocol or driver should run.

The rather obvious model holds. When data arrives from the network, it is placed in memory. Then the application picks it up. There will thus be a minimum of two memory cycles as a part of receiving data. (Sending is the same). Protocol overhead will add to this count of memory cycles. So to attach (usefully) a gigabit network to a host, we need a host with a memory bandwidth of 2 gigabits/second. Very few workstation-class computers have that memory bandwidth even today, and thus end-to-end gigabit applications were not literally achievable in the time frame of this research. In time, these speeds will naturally come. What we showed is that one can push up against this limit in practice. One actually can build implementations that run at the speed of memory. For traditional workstations (some parallel and supercomputers are special cases) since memory is the expensive component, and processors seem to get faster than memory does, we do not anticipate a sudden protocol processing bottleneck. Memory bandwidth to move the data will continue to be the fundamental throughput limit.

3.4 QoS

A major thrust of our research into QoS management was to investigate theoretical and implementation issues related to resource management and control algorithms for an Integrated Services Packet Switch (ISPS). Our Aurora objective in this area was to show that the processing necessary to enforce QoS scheduling in switches could be done at high speed. To accomplish this, we developed algorithms that completely realized our proposed model, characterized their performance, and then developed approximations that were more efficient to execute but still close in functionality. We tested these in simulation and in the Sunshine switch to verify their performance and correctness in various situations.

The ATM framework, which includes explicit circuit setup, can in principle support a very sophisticated range of QoS management capabilities. However, in practice the first products and the first standards depended only on admission controls at connection setup time, together with a very simple queue discipline such as first come, first served. Early simulations performed at Bellcore suggested that in real-world conditions (traffic with realistic levels of burstyness), this simple approach would support only very low levels of link utilization before failing. We thus concluded that we should develop and propose a cell queuing scheme to improve link loading in ATM networks.

Working with Bellcore, we determined that we would propose to the ATM Forum as a first scheme a very simple weighted fair queuing (WFQ) mechanism. After some discussion, we found that the overhead and complexity of even this simple scheme was a major source of concern to implementors. With our collaborators at Bellcore we responded to this concern in a number of ways, including the development of the high-efficiency approximate HWFQ algorithm described below, and development of the second generation Sunshine Output Controller Card. Given the concern with the performance of a simple WFQ scheme, we recognized that the complexity of our full scheme, which required a number of queues, would be even more problematic. Oumar Ndiaye and Christopher Lefelhocz investigated one solution to remove the complexity.

As part of his master's work [17], Ndiaye laid out the groundwork of the link sharing algorithm, developed a more efficient algorithm, and simulated the new algorithm. We briefly mention each of these areas below. In our full model for link bandwidth allocation, the bandwidth a customer requires can be thought of as a service. Since a customer may also be a provider of service, the service can be broken down into several sub-services. A service that is broken into sub-services is called a service class. A service that is not broken into sub-service is called a real service. A tree structure that shows the sharing of a link's bandwidth among customers can be generated. This tree structure has service classes as internal nodes in the tree and real services as leaf nodes in the tree.

Ndiaye identified two goals to the sharing of a link's bandwidth :

- At any active period of the link (when there are cells to transmit), each active customer is guaranteed a share of the link no less than the portion of the link bandwidth it owns.
- For any sub-service, the amount of link resource at its disposal that it is not using is distributed to all its sibling sub-services that need more link usage than the amount they are entitled to. This distribution is done proportionally to the rates of the sub-services.

To support these properties, our original link sharing algorithm used a Hierarchical Weighted Fair Queuing(HWFQ) mechanism. Weighted Fair Queuing is a scheduling discipline that classifies each incoming packet into a service class, and then maintains a sorted list of classes, based on a computed departure time for that class given its service allocation. Hierarchical WFQ consists of a WFQ mechanism at each non-leaf node of the tree. Each leaf node contains a queue of cells to be sent. For each non-leaf node, weights are assigned in proportion to the amount of service requested by each child. The entire mechanism can be thought of as a WFQ of WFQ's. This mechanism was found to be accurate in preserving the properties stated above. However, the complexity was large since the number of different WFQ's involved in the sending/receiving of a cell was proportional to the depth of the tree. We performed a study to determine where the costs were in our algorithm, by actual audit of path lengths, and determined that the major cost was entering the data structure representing its service class into the ordered list representing the WFQ at each level during the queuing stage. This complexity is particularly troublesome in an ATM network where the number of cells to send/receive is large.

Bellcore had developed a sequencer chip [3] which performed WFQ sequencing in hardware, but that chip only implemented one ordered queue, while our full algorithm depended on a hierarchy of queues. This complexity made it impossible to implement our scheme directly using that chip.

We concluded that a flat, single-level queue structure, in addition to matching the particulars of the Bellcore sequencer chip, was likely to lead to efficient implementation in a number of cases For this reason, we sought an algorithmic approximation to our hierarchy of queues that depended on a single queue.

Ndiaye proposed a way to approximate the behavior of the HWFQ by mapping the weights of the real services into a single flat WFQ. To preserve property 2, periodically a separate algorithm was run which determined which classes in the HWFQ were active and recalculated the mapping from service classes to real services in the single WFQ. Whether a real service was active was based on whether it had cells waiting to be sent.

This simplified the complexity of the send/receive algorithm. Given that the sorting portion of the WFQ mechanism was implemented in hardware, the receiving of a cell was estimated to take 55 instructions and the sending of a cell to take 70 instructions. These were estimates since the algorithm was tested through simulation rather than on a physical output port of a switch.

In simulating the new algorithm, Ndiaye varied the period of update of the mapped weights. It was found that for smaller periods, the results were almost the same as the original algorithm. For larger periods, the results were a good approximations to the original algorithm. Thus it is believed that for larger sharing trees where the updating algorithm takes longer, the performance of the simpler algorithm will be a good approximation to the performance of the complex algorithm.

While Ndiaye's work was being completed, an ATM Output Port Controller (OPC) was being developed for the Bellcore Sunshine switch, which contained the features necessary to test out the complexity of the algorithm. Lefelhocz took the algorithm developed by Ndiaye and implemented it on the new output port controller. The intent was to learn more about the coding of the algorithm while showing that such an algorithm was fit for use in an ATM environment. Also it was discovered that the updating algorithm could be done in a simple pipelined manner. These results are described below.

The processor used on the OPC for implementation of our proposed algorithm was an Intel 960CA 33 MHz microprocessor. Given the bandwidth of the output port, the number of instructions which can be executed in one cell time is approximately 80. Simplifying Ndiaye's algorithm to the extreme it was found that C code compiled into assembly took 32 instructions to receive a cell and 46 instructions to send a cell. Since during periods of congestion, two cells can arrive at the OPC during the time one cell leaves, the receive algorithm is run twice in the worst case. Thus, the processing cycle would take a total of 110 instructions to complete. Simple analysis of the compiled assembly revealed 7 instructions could be removed from the receive algorithm and 6 instructions from the send algorithm. This resulted in a instruction count of 88 instructions only slightly above the ceiling of 80. It is believed with only slightly faster processors, this algorithm could run at full rate.

It was hoped that the same microprocessor could be used for both the send/receive algorithm and for the background task to recompute the weights based on the current activity of the classes. The speed of the processor used did not permit this. However, two options exist for making the weight recomputation algorithm run as a "background process". The first is to add another microprocessor. The mapping algorithm can run independently of the send/receive algorithm. The second is to use a faster microprocessor. Since the number of instructions is only slightly larger than the current speed, a processor which is twice as fast would be capable of running the mapping and send/receive algorithm and meet the cell time performance.

Finally, in the process of developing the mapping algorithm for the i960, Lefelhocz found that the weight recomputation algorithm proposed by Ndiaye could be improved so that it computes a new set of weights for each tree traversal. The original proposal required two distinct phases to check for activity and to recalculate the rates. The former phase is a post-order tree traversal. The later phase is a pre-order tree traversal. The two phases cannot be combined. However, the phases can be pipelined such that the first rate mapping occurs after the second tree traversal, the second rate mapping occurs after the third tree traversal, etc. In this way, the algorithm is pipelined to constantly update the mapped weights of real services.

3.5 Congestion Control

The current algorithms for TCP congestion control are rather complex, and depend on specific details of the design. Prior experiments in our simulator have been limited by the ability of the simulator's TCP component to represent all these details. To remedy this limitation, we began our work on TCP congestion control by adding to our simulator a very detailed representation of the TCP congestion control mechanisms. Our approach was to modify our simulator so that we could incorporate into it code directly derived from a real TCP implementation, so that the behavior of the simulator and the TCP in the real network were based on the same implementation. This eliminated any concerns as to whether the simulation component was a proper representation of the real world.

Based on this code, we explored a number issues in the operation of TCP, to understand better how it might be tuned to perform better over long delay high bandwidth circuits.

We identified two major effects that limit performance. First, the so-called long timeout event, in which a lost packet is detected only when a retransmission timer at the source expires, represents a degradation of performance whose impact becomes much more important as the speed of the link increases. Thus, it becomes increasingly important to avoid this class of event by finding alternative techniques for detecting packet losses. Second, reducing the window size to one after a loss event slows the throughput to such a degree that reasonable throughputs are not regained for much too long a period of time. Thus, a less conservative backoff algorithm should be employed when it can be justified.

A MS thesis by Janey Hoe [9] proposed algorithms to achieve these goals. To respond to a higher percentage of lost packets without invoking retransmission timeouts, she proposed an extension to the currently used "fast retransmit" scheme. In the current scheme, the sender concludes that there has been a lost packet when three duplicate acknowledgments have been received, since duplicate ACKs are sent by the receiver when arriving packets do not come in a continuous sequence. Her modification was to extend this idea to incorporate a fast recovery mode. Fast recovery mode is initiated whenever a fast retransmit trigger event (three duplicate acks) occurs. Once fast recovery mode is initiated, additional packets are retransmitted whenever an acknowledgment is received that acknowledges fewer than the known number of outstanding packets. This modification is very effective in distinguishing between lost packets and other sequences of ACKs that reflect normal operation, and allows rapid recovery from an instance of multiple packets being lost within a single round-trip time; a circumstance significantly more common in networks with a high bandwidth-delay product.

Hoe's second proposal involves allowing a larger window size after a loss event, but explicitly limiting the size of a burst of packets that a sender can emit at any one time. By limiting the burst size, it seems possible to be more liberal in setting the window size after a congestion slowdown. This allows the TCP to keep the pipelined flow of data active, while still providing protection against repeatedly triggering episodes of congestion.

3.6 ALF

Our assessment of the code implementing the ALF forwarder and end-node is that there is a slight increase in overhead in processing ALF in a forwarder. This increase is offset by a slight simplification of the processing in an end-node. The increase in the forwarder is caused by the fact that while the forwarder must now deal with forwarding at the ADU level rather than the cell or packet level, it is still necessary to recognize and deal with the transmission multiplexing unit to some extent. So there is some aspect of packet processing together with the higher level ADU processing.

In exchange for this increase in cost, there are substantial simplifications in the processing at the end-node. The benefit is not a reduction in code path lengths, but in scheduling and system overhead. In the case of ATM networks, for example, there is no intermediate processing of packets, which will reduce the interrupt rate and process scheduling rate of the system. More importantly, the ALF framework defers virtually all processing of messages until the application can actually act on the data present. This allows the operating system scheduler to efficiently manage processes which are using the network, and eliminates the unfortunate effects of "layered multiplexing."

ALF and ILP were first proposed in the context of a specific alternative architecture to the IP and TCP architecture. As the ideas became more widely understood, they instead served as a model to reason about designing and coding protocols and applications in the context of IP. An example is the work on Light Weight Sessions [8]. At this time, ALF is best thought of as a new model for thinking about how to use existing protocols and mechanisms.

3.7 Alternatives for Link Multiplexing

One of the system level objectives for ALF was to use the native multiplexing mode of the network, for example packet or cell switching, in an efficient manner. However, one could ask the question in another way and inquire, once the application has broken the data into ADUs, what the most efficient network multiplexing mode would be. Cell switching offers the advantage of very low latency in admitting high priority data into the network, since the network can be reallocated at the end of every cell. The penalty for this is the cost and complexity in the switch controller of making a scheduling decision for every cell. Since cells are small, the processing rate is high. One could make the data units larger, as in a classical packet switch, and thus reduce the header processing rate, but the longer latencies waiting for a packet to complete transmission may cause undesirable delays for high priority traffic.

Chris Lefelhocz, in a MS thesis, explored a new scheme which avoids this tradeoff by constructing a network based on pre-emption. The basic mode of operation is that an entire ADU is scheduled for transmission over a network link, but the ADU can be pre-empted as needed if a higher priority ADU arrives. The required processing rate in the switch controller is thus reduced; it corresponds to ADU arrivals rather then cell or packet arrivals. At the same time, the latency to forward a high-priority ADU is minimized. We hypothesized that this architecture might provide the best of both previous approaches: better than the low latency of the cell architecture but lower processing demands than the packet architecture.

We developed a detailed simulation of such a switch, and explored its operation with a variety of traffic loads and switch configurations. In the abstract, the basic scheme seems to offer the expected benefits. With a realistic set of estimates for ADU sizes, a system with less than one tenth the processing power of a cell switch seems to offer effective operation with the same latencies. The detailed results are presented in LCS-TR-621 (in publication).

In practice a new scheme such as this is not likely to completely displace either the classical packet switching schemes or the rapidly advancing ATM schemes. The results of this study, in addition to suggesting a new type of link-level technology, can be extrapolated to show how an ATM switch, for example, might be implemented more efficiently. If the default behavior of an ATM switch was to send a continuous flow of cells from one VC, unless a cell from a flow of higher priority were to arrive, this algorithm might reduce the expected processing cost for each cell.

The simulations described above identified one very important issue, which applies generally to packet, cell and pre-emption systems. The issue is the phenomenon which we have seen and described as "ack compression" in packet systems [20]. In a pre-emption system, if data packets and acknowledgments are at the same priority level, then a number of acks can accumulate behind a large

ADU, leading to a burst of acks being later delivered to a end-node, which in turn results in a burst of subsequent data packets. One solution to this is to assign acks a higher priority, which solves the problem, both in the packet and the pre-emption case. However, this solution requires the switch to distinguish acks from data packets, which is not consistent with most layering architectures. The alternative of giving all small packets a higher priority leads to other anomalies, in which users can obtain preferential treatment by using the network in very inappropriate ways. (Small packets represent a very inefficient use of the network.)

3.8 Development of Workstations Environment

As part of our development of a software environments suitable for forwarder and end-node networking research, we undertook a number of development projects to obtain a useful operating system for a current generation workstation. We imported an implementation of Mach, and explored its use in this context, but concluded that its internal structure did not naturally lead to the most efficient and straightforward implementation of network code. Our current target for an open system (free of license restrictions) is one of the variants of the openly available Unix BSD 4.4 derivatives. We have worked with the community to support these packages, and have imported and incorporated into our computing environment such open systems on Digital's Alpha-based workstations, Intel-based PC's and Sun workstations. This hardware independence allows us both to adopt new technology as it appears and to compare the effects of different hardware designs on the performance of highly tuned scheduling and forwarding algorithms.

Another of our projects involved the importation and evaluation of the x-kernel, an framework for the implementation of network protocols done at the University of Arizona. We performed a number of projects with the x-kernel. We implemented an IP forwarder and a version of our queue management for QoS to evaluate the architectural features and the performance. We identified no architectural problems inserting these features into the system, but, as noted above, the cost of the x-kernel task abstraction may be somewhat too high to permit the system to be used as a basis for a specialized situation such as a packet forwarder.

3.9 FLORB

As part of developing our prototype platform for implementing forwarders and related software, we developed a network interface called a Florb (a contraction of Flow to Orbit interface). The interface was first designed to interface to the IBM Planet/Orbit technology, but in fact is a very general 32 bit high speed interface suited to a wide variety of network devices. Our prototyping platform at the time was the DEC workstation, and the Florb interfaces to the Turbochannel interface for that family of machines.

A block diagram of the Florb is shown in Figure 2. The novel aspect of the Florb is its architecture for moving data, which we believe is suited for a wide variety of high speed applications. At the center of the Florb is a CPU, fast SRAM, and high speed data bus, which connects all the relevant inputs and outputs of the interface card. At the periphery of the card are a number of retiming FIFO's which allow the various interfaces to operate with some degree of asynchronicity. Data movement between the FIFOs and connected devices (the host I/O bus, the network or any other device) may utilize a DMA controller or an external FIFO clock, as required. The data is moved from external devices into FIFOs, and the FIFOs are in turn connected to this central bus.

Transfers of data across the bus are controlled by a RISC processor (in our implementation, an Intel 80960). However, the way the processor controls the bus is somewhat unusual. Rather than issuing read and write instructions to move data to and from the various FIFOs, individual address lines of the processor are wired directly to the read and write ports of the FIFOs. Thus, to read from a particular FIFO, the CPU simply issues any instruction which loads the external data bus onto its onboard registers, with the particular memory bit set on.

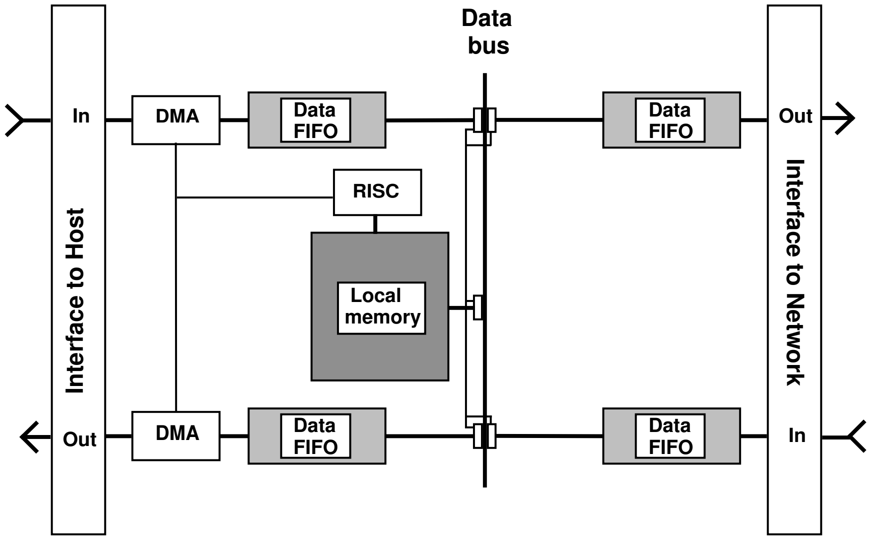

Figure 2: Block Diagram of FLORB Host-Network Interface

One instruction, by setting several of the address lines, can cause a number of reads and writes to occur simultaneously. Multiple reads in general would represent an error, but one read and multiple writes has the effect of putting a value on the bus and taking it off at a number of other points. Thus, in one instruction, one could read a byte from the host, write it to the network, and wrap it back to the host.

The processor is thus capable of being used in two modes; one a normal RISC processor, and the other a micro-sequencer. It can switch between these two modes on an instruction by instruction basis, based on which regions of the address space are referenced.

The key to the performance of the FLORB is that each data transfer can be accomplished in one processor cycle. Our implementation of the Florb uses a modest 25 MHz clock, which implies a peak data transfer rate of 800 mb/s. More aggressive processors with higher external clock rates could drive a bus at even higher speeds.

The performance of this design is achieved with great simplicity. The core FLORB hardware consists of a single circuit board approximately 4 inches by 5 inches in size. An additional card of the same size may be mounted in a piggy-back fashion to hold the components required for a specific network interface, such as an ATM physical layer driver.

We conducted a number of experiments with the FLORB. Our initial goal was to determine the validity of the concept and demonstrate its performance capabilities. This was accomplished by connecting two FLORB's back to back to form a point-to-point network link. To verify correct operation of the devices, we also developed device drivers for Digital's OSF/1 operating system which treated the devices as standard network interfaces.

The performance initially achieved with this arrangement was lower than expected. We found that the link operated at a throughput of approximately 280 mb/s. At this speed, the FLORB's processor was operating at approximately 50 percent of full capacity. Further, the poor performance could not be explained by known limitations of the workstation hardware.

To study this problem further, we characterized the performance of the OSF/1 network protocol implementation itself by implementing a RnullS network interface which generated and accepted packets without actually touching hardware. Using this method, we discovered that the overhead of the OSF/1 IP protocol implementation was substantially higher than expected.

We next eliminated the use of OSF/1 and drove the FLORB devices directly from our x-kernel based

IP router. In this configuration we achieved an overall effective throughput of approximately 400 mb/s for a mix of IP traffic created by tcplib [6]; a library which artificially synthesizes network traffic based on statistics from real networks. At this processing rate, the FLORB's cpu's were operating at approximately 65 percent of capacity.

To determine the bottleneck limiting performance to this figure, we profiled our forwarder's operation using the Alpha's hardware cycle counters. We learned from this experiment that our performance was limited by I/O bus arbitration and memory contention effects within the workstation, and could not be further improved.

We conducted one further experiment with the FLORB in this configuration, varying the traffic mix to cover a range of packet sizes. We determined that as we moved to smaller packets the per-packet handling overhead never became high enough to exceed the capacity of the FLORB's cpu, although the overall throughput of the system fell significantly with very small packets.

The development of the FLORB demonstrated two points. One was that the control interface to an I/O device need not be awkward and slow. As noted above, processing overheads in packet forwarders are not typically dominated by IP level processing steps, but by the overhead of controlling the associated I/O devices. The FLORB was designed to demonstrate that device control can be efficient. In our tests, the efficiency of the overall device handler was limited by the workstation's bus arbitration, showing that the FLORB design itself was not a limiting factor.

The second point it proved is that a general purpose RISC chip can be the basis of a very high performance network interface. The FLORB incorporated a Intel 960, but used this processor in a novel way. There has been a continuing debate as to whether it is worth designing and using special chips for I/O control. Such a chip can have special functions on it, which could conceivably lead to higher performance. On the other it is a more specialized product with less anticipated total demand, a chip vendor cannot afford to put as much effort into such a chip as into the next generation RISC processor. So by the time the special I/O control chip comes out, the RISC chips then appearing can go just as fast. Our design, by showing a creative way to exploit a RISC chip for micro-control, lends support to the thesis that special control chips are not worth the effort to build.

4 Lessons Learned

4.1 Performance and Architecture of Packet Forwarding

We observed that in a packet forwarder (a router or flow forwarder) the device driver and queue management algorithms constitute the major source of overhead. This is a critical observation, because it suggests that there are limited opportunities for speeding up a packet forwarder by parallel processing. The device driver seems to be intrinsically serial. It involves device registers and packet queues that must be updated automatically, and per-packet operations that must be done at each arrival and departure time. Since the network hardware sends packets in a serial manner, the resulting timing of the device code is serial. One can use a separate processor for each device, and one or more for the IP processing, but one cannot speed up the processing by massive parallel execution of the code. Given that the cost of IP is already not the dominating cost, making it parallel will not be strongly effective.

This line of reasoning suggests the following approach to forwarder design. Instead of a focus on silicon for the forwarding code (e.g. IP in hardware) there should be a focus on very efficient hardware for network interfaces, and for key processing steps in the queue management, for example the cell sequencer chip from Bellcore described elsewhere in this report. Even hardware support for the allocation and freeing of buffers can have a significant impact on the processing overhead. Our overall conclusion is thus that while a massive parallel forwarder is not an effective approach, that selective hardware support combined with a state of the art RISC processor is an effective way to build a high performance router. This is a very desirable conclusion, since we argue very strongly that to support flexible evolution of the network services, it is critical that the parts of the algorithm that reflect the details of the offered service be coded in a flexible manner, for example in software, so that the packet forwarder can evolve as the service definition does.

There is an evolutionary path which forwarders and switches will follow, both in the current product offerings and in the higher speed context. The first generation packet forwarders – gateways or routers – combined in one element the format conversion implied in moving from one network type to another and the cross-connect capability of switching packets between a multiplicity of interfaces. This was reasonable, as a very cost effective way to build a router at current mainstream product speeds is to put the packets into a central shared memory buffer. However, at higher speeds, memory is not as effective a cross-connect architecture as parallel switch fabric designs, because the single shared communication point (the shared memory) must operate at very high speeds. However, with a switch fabric at the core of the switch, it is difficult to utilize a single processor to perform the forwarding decisions necessary at all ports of the switch. The architecture thus more naturally becomes a separate forwarding processor for each switch port.

In this more evolved architecture, the format conversion is performed in each of the switch port controllers, and the traditional router becomes a two-port device connecting the line to the switch port. This simplification of the router architecture makes it much easier to build a cost-effective high performance router element. Unfortunately it also introduces some limitations, in that it becomes much harder to implement global resource management algorithms. In the shared-memory switch these algorithms can be implemented on a single processor, while in processor-per-port design they must be implemented in a distributed manner. Managing this and other tradeoffs implicit in the processor-per-port design is a topic requiring further exploration.

4.2 The Fundamental Limit of Memory Bandwidth

One fact that came up repeatedly in this project is that there was widely varying expectations among members of the project as to how fast a computer of a given design would be able to deal with data coming across a network. A number of possible performance limits were postulated as the real bottleneck, including the design or implementation of the network protocols. What we concluded, based on a number of experiments and calibrations, is that in most cases, the real determinant of achieved throughput is the bandwidth of the computer memory, and the effectiveness of the I/O connection into that memory.

In general, the bandwidth of the memory must be at least twice, and more practically, four times the speed of the desired overall throughput. Looking at arriving data (departing data is similar) there are at least two memory cycles required to move the data from the network: the network controller must write the data to memory, and the application code must pick it up. Protocol processing may add perhaps another two cycles, to move the data between system and user buffers and to compute checksums. Creative design can eliminate one or more of these cycles, but the memory bandwidth still remains the basic limit to achieved throughput.

On the other hand, with careful design of system software, these limits can actually be reached. While overhead in the system and the protocols can be a practical cause of lost performance, these issues can be sidestepped if the effort is justified.

This balance of limitations will continue to hold into the future. We do not anticipate some reverse in the current situation, in which processing overheads become more of a bottleneck, and memory less. Current trends in silicon design are making processors continuously faster, while memory chips are getting bigger, rather than faster. The speed of a memory architecture can, of course, be increased by use of wider memory busses, cache architectures, and so on. But these approaches add more cost than a faster processor, and thus we can expect, for future machines, that as memory speeds are increased, the processor speeds will increase to match.

This analysis does not address one key issue, which is the potential for a mis-design in the way the network is connected into the memory. Most memory designs today focus on providing a high speed transfer path between the memory and the processor. This objective can, in simple cases, lead to designs in which network interfaces cannot interact with the memory at highest speeds. Both we and other members of the project spent a substantial amount of time dealing with this issue as a practical impediment to progress. However, we conclude that this issue is not fundamental, and could be resolved as an engineering issue in the design of future generations of workstations if the objective is warranted.

4.3 The Flexibility of Software

One means to increase speed is to cast the critical processes in hardware, in the hope of making them go faster, or of going faster at a lower cost. While this approach is technically feasible, it can essentially lead to a disaster if the desire for speed is not balanced by the need for flexibility.

ATM is a good example of this tradeoff. ATM cells are small, and must thus be processed at high rates in order to service a high speed link. At 622 mB/s, ATM cells arrive at over a million per second. The concept, when ATM was first proposed, is that the actual processing of the cells would be a very simple act that could thus be performed in hardware. Only the higher level control functions, such as opening connections, would need to be performed in software.

This assumption has not been totally justified so far, and has led to some of the growing pains of ATM. For example, some of the QoS work done here cannot be implemented without changing the actual cell processing in the switch. If that process is cast in hardware, there is no way to upgrade an existing ATM switch to support explicit QoS. In contrast, in packet switches that perform these steps in software, this migration is just a new software load.

Data networks have evolved rapidly over the past several years, in part because the field is young and new approaches emerge, and in part because customer expectations for service evolve, as for example the desire for multi-media networking. This evolution calls for products with flexibility. Part of our research was to show that high speed packet processing could be done in software. Our conclusion is that indeed, packet switches that employ general purpose processors for forwarding are reasonable. Our research suggested two factors that can contribute to a high performance, cost effective network device, whether packet switch or host interface. The first is to engineer the actual network interface hardware so that it does not represent a high overhead to control. Some current controller chips require as many instructions to control as the packet processing computation.

The second point is less obvious. There are indeed simple, basic building block operations in the forwarding of a packet that could be cast in silicon, and would result in increased performance at reduced cost. However, there may not yet be enough understanding to let us say with confidence what these modules are, and getting the design wrong can have a disastrous effect on the final flexibility and performance of the resulting device.

Perhaps the most simple building block that could be cast in hardware is allocation and freeing of buffers. Hardware support for a free list of buffers would represent a useful component. The Bellcore sequencer chip represents a more ambitious step in this direction, in that it implements a list of buffers sorted by hardware according to departure time. This chip allowed us to realize an ATM output port controller that could implement a WFQ cell scheduler at OC-3 rates, and indeed which could approximate a hierarchical WFQ scheme at essentially this speed. However, the interface to the chip, in retrospect, could itself have been lower overhead, due to issues of dealing in software with the possibility of field overflow. Simple issues such as this must be resolved as ideas for hardware building blocks mature.

4.4 Speed May Not Be The Hardest Problem

We concluded that the issue of speed alone represented only the obvious problem of efficient implementation and adequate bandwidth. What we showed in our research is that it is possible to build devices that perform well enough to hit the fundamental limits of memory and link bandwidth. We did not see any issues that suggested that this would change in the short term future. The more complex issue of high delay-bandwidth product paths may represent more significant problems, as issues of flow and congestion control need to be rethought. This matter is not a simple one of speed, however, but an issue of speed coupled with scale. It seems quite practical to implement a transport protocol such as TCP on a suitable hardware platform and expect it to go at a gigabit. The interesting question is what will happen when many copies of this implementation contend for bandwidth over a shared link, and deal with issues of congestion. These testbeds were not large enough to reveal any of these issues, which for the moment must be dealt with through the somewhat artificial means of simulation.

4.5 The Importance of a Suitable Experimental Platform

Especially at high speeds, attention to detail and careful design is required for success. It only takes one bottleneck to throughput to limit the overall experiment. This situation meant that it was necessary, in almost all of the experimental situations, to tune all parts of the system for good performance, and to understand what the root causes were of apparent performance problems.

This is work that, while highly necessary, is not directly related to the research question at hand. It is thus important to structure the project such that the difficulty of this work does not come to dominate the whole project. We identified two success factors in this desire. First, the experimental platforms should be shared among as many as possible of the participants, so that experience could be shared. Second, proprietary aspects of the experimental apparatus should be minimized, so that time is not wasted trying to deduce how a hidden part of the system is influencing the overall performance.

5 Conclusions

The gigabit objective, in retrospect, should not be thought of as an actual numerical goal to which performance should be pushed. While it may not have been obvious at the beginning of the project, the gigabit objective will be achieved in due course as processors and related devices become faster. Rather, the objective of the project was properly to push performance forward to the extent possible, both to understand what the fundamental limits were and to excite people to the prospects of achieving higher performance in practice. Other parts of the testbeds looked at different issues, such as speculation on how applications might change if these bandwidths were available. What we concluded that we can indeed build systems with high performance, and that we understand what the fundamental limits to performance are, and how to reach those limits. Those limits are not absolute numbers, but derive from the capabilities of key system components, most particularly memory bandwidth. When we have a workstation with a memory bandwidth somewhere between two and four gigabits per second, we will have a workstation that can be networked at a gigabit. Depending on how the I/O architecture is constructed, we are not far away from that objective today.

References

[1] Adam, J., "The Vidboard: A Video Capture and Processing Peripheral for a Distributed Multimedia System," Proceedings of the 1993 ACM International Conference on Multimedia, pp. 113-120, ACM Press, August 1993.

[2] Agarwal, A. et al. "The MIT Alewife Machine: Architecture and Performance," *Proceedings of ISCA '95.*

[3] Chao, H., *Architecture Design for Regulating and Scheduling User's Traffic in ATM Networks*, Appeared in Proceedings of ACM SigComm, October 1992.

[4] Clark, D., and Tennenhouse, D., *Architectural Considerations for a New Generation of Protocols*, Appeared in Proceedings of ACM SigComm, September 1990.

[5] CNRI, "Gigabit Testbed Initiative Program Plan," Corporation for National Resarch Initiatives, November, 1990.

[6] Danzig, P., et al., *An Empirical Workload Model for Driving Wide-area TCP/IP Network Simulators*, Appeared in Internetworking Research and Experience, March 1992.

[7] Davie, B., "The Architecture and Implementation of a High-Speed Host Interface," *IEEE Journal of Selected Areas in Communications*, February 1993, 11 (2), pp. 228-239.

[8] Floyd, S., et al, *A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing*, Appeared in Proceedings of ACM SigComm, September 1995.

[9] Hoe, J., *Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes*, MIT Masters Thesis, May 1995.

[10] Houh, H. and Tennenhouse, D., "Reducing the Complexity of ATM Host Interfaces," Proceedings of Hot Interconnects II, Stanford CA, August 11-12, 1994.

[11] Houh, H., Adam, J., Ismert, M., Lindblad, C. and Tennenhouse, D. The VuNet desk area network: Architecture, implementation, and experience. *IEEE Journal on Selected Areas in Communications*, 13(4):710–721, 1995.

[12] Hutchinson, N., and Peterson, L., *The x-Kernel: An Architrecture for Implementing Network Protocols*", Appeared in IEEE Transactions on Software Engineering, January 1991.

[13] Lindblad, C., Wetherall, D., Stasior, W., Adam, J., Houh, H., Ismert, M., Bacher, D., Phillips, B. and Tennenhouse, D. ViewStation Applications: Implications for Network Traffic. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[14] Lindblad, C., Wetherall, D. and Tennenhouse, D., "The VuSystem: A Programming System for Visual Processing of Digital Video," *Proceedings of ACM Multimedia 94*, October 1994.

[15] Lindblad, C., "A Programming System for the Dynamic Manipulation of Temporally Sensitive Data," MIT/LCS/TR-637, MIT Laboratory for Computer Science, Cambridge, MA, August 1994.

[16] Lyon, T., "Simple and Efficient Adaptation Layer (SEAL)," *ANSI T1S1.5/91-292*, August 1991.

[17] Ndiaye, O., *An Efficient Implementation of an Hierarchical Weighted Fair Queue Packet Scheduler*, MIT Lab for Computer Science TM 509, June 1994.

[18] Nikoh, H. and Kuwajima, T., "The Full Digital Video Camera System and Simulation of its Essential Parameters," *Proceedings of the IEEE International Conference on Consumer Electronics*, 1989, pp. 48-49.

[19] Tennenhouse, D., et al, "The ViewStation: A Software-Intensive Approach to Media Processing and Distribution," *ACM Multimedia Systems Journal*, vol. 3, No. 3, July 1995.

[20] Zhang, L., and Shenker, S., *Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic*, Appeared in Proceedings of ACM SigComm, September 1991.

Appendix A: List of Publications

1. Adam J., "The Vidboard: A Video Capture and Processing Peripheral for the ViewStation System," Master's Thesis, MIT, September 1992.

2. Adam J., "The Vidboard: A Video Capture and Processing Peripheral for a Distributed Multimedia System," Proceedings of the 1993 ACM International Conference on Multimedia, pp. 113-120, ACM Press, August 1993.

3. Adam J., Houh H., Ismert M., and Tennenhouse D., "A Network Architecture for Distributed Multimedia Systems," Proceedings of the International Conference on Multimedia Computing and Systems, May 1994.

4. Adam J., Houh H., Ismert M., and Tennenhouse D., "Media-Intensive Data Communications in a 'desk-area' Network," *IEEE Communications*, August 1994.

5. Adam J., Houh H., and Tennenhouse D., "Experience with the VuNet: A Network Achitecture for a Distributed Multimedia System," *The IEEE 18th Annual Conference of Local Computer Networks*, Minneapolis, MN, September 1993, pp 70-76.

6. Adam J. and Tennenhouse D., "The Vidboard: A Video Capture and Processing Peripheral for a Distributed Multimedia System," *ACM Multimedia Systems Journal*, vol. 2, No. 2, April 1994.

7. Bauer M., "Self-Framing Packets in the ATM Adaptation Layer," Bachelor's Thesis, MIT, May 1992.

8. Charny A., Clark D., and Jain, R., "Congestion Control With Explicit Rate Indication", *Proceedings of the ICC Conference*, June 1995.

9. Clark, D., Davie, B., Farber, D., Gopal, I., Kadaba, B., Sincoskie, D., Smith, J., and D. Tennenhouse, "An Overview of the AURORA Gigabit Testbed", *INFOCOM '92*, Florence, Italy, May 1992, Pages 0569-0581.

10. Clark, D., Davie, B., Farber, D., Gopal, I., Kadaba, B., Sincoskie, D., Smith, J., and D. Tennenhouse, "The AURORA Gigabit Testbed," *Computer Networks and ISDN Systems*, vol 25, No. 6, January 1993.

11. Clark D., Shenker S., and Zhang L., "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proceedings of SigComm 1992 Conference*, ACM, August 1992.

12. Clark D., and Tennenhouse D., "Architectural Considerations for a New Generation of Protocols," *SigComm Symposium*, ACM, September 1990.

13. DeAddio M., "TCP/IP UNIX EAluation in a High-bandwidth ATM Network," Master's Thesis, MIT, May 1992.

14. Gautam N., "Host Interfacing: A Coprocessor Approach," Master's Thesis, MIT, January 1993.

15. Heybey A., "Video Coding and the Application Level Framing Protocol Architecture," MIT Lab for Computer Science TR #542, June 1992.

16. Hoe J., "Start-up Dynamics of TCP's Congestion Control and Avoidance," MIT Masters Thesis, May 1995.

17. Houh H., Adam, J., Ismert, M., Lindblad, C. and Tennenhouse D., "The VuNet Desk Area Network: Architecture, Implementation and Experience," *IEEE Journal on Selected Areas in Communication*, vol. 13, No. 4, May 1995.

18. Houh H. and Tennenhouse D., "Reducing the Complexity of ATM Host Interfaces," Proceedings of Hot Interconnects II, Stanford CA, August 11-12, 1994.

19. Ismert M., "The AVlink: An ATM Bridge between the VuNet and Sunshine," Bachelor's Thesis, MIT, May 1993.

20. Ismert M., "ATM Network Striping," Master's Thesis, MIT, February 1995.

21. Lefelhocz C., "Investigation of a Preemptive Network Architecture," MIT Lab for Computer Science TR #621

22. Lindblad C., "VuSystem Performance Measurements," Proceedings of NOSSDAV 95, Durham, NH, April 1995.

23. Lindblad C., and Tennenhouse D., "The VuSystem and its Implications for the OS Designer," Technical Note, Laboratory for Computer Science, MIT, March 1995.

24. Lindblad C., Wetherall D., Stasio W., Adam J., Houh H., Ismert M., Bacher D., Phillips B., and Tennenhouse D., "ViewStation Applications: Implications for Network Traffic," *IEEE Journal on Selected Areas in Communication*, vol. 13, No. 5, June 1995.

25. Martin D., "The Design of a Tranceiver Chip for Broadband ISDN ATM Cells," Master's Thesis, MIT, June 1991.

26. Ndiaye O., "An Efficient Implementation of an Hierarchical Weighted Fair Queue Packet Scheduler," MIT Lab for Computer Science TM #509, June 1994.

27. Sun K., "ATM Adaptation Protocols for MPEG Video: An Experimental Study," Master's Thesis, MIT, May 1993.

28. Tamashunas B., "Supporting Service Classes in ATM Networks," Master's Thesis, MIT, May 1992.

29. Tennenhouse D., Adam J., Houh H., Ismert M., Lindblad C., Stasior W., Wetherall D., Bacher D., and Chang T. "A Software-Oriented Approach to the Design of Media Processing Environments," Proceedings of the 1994 International Conference on Multimedia Computing and Systems, May 1994.

Appendix B: List of MIT Student Supported by the Aurora Project

- Adam, Joel
- Bose, Vanu
- Dukach, Semyon
- Guatam, Nikhil C.
- Heybey, Andrew T.
- Hirschfeld, Rafael
- Houh, Henry H.
- Ismert, Michael
- Lindblad, Christopher J.
- Martin, David
- Maw, David
- McGraw, Janet
- Shepard, Timothy
- Troxel, Gregory D.
- Wetherall, David
- Yip, Patrick