Appeared in proceedings of
9th Annual IEEE Workshop on Computer Communications
Duck Key, Florida, October 1994.

Washington University's Gigabit ATM Desk Area Network†

Zubin D. Dittia zubin@dworkin.wustl.edu
Jerome R. Cox, Jr. jrc@hobbs.wustl.edu
Guru M. Parulkar guru@flora.wustl.edu

Department of Computer Science
Washington University in St. Louis
St. Louis, MO 63130.

Abstract

We present the design of an ATM-based interconnect that serves as a high speed packet-switched equivalent of an I/O bus in a workstation/server. By effectively "extending" an external ATM network into the desk-area, the interconnect also serves as a high performance host-network adapter. An OC-12c (622 Mbps) prototype of the interface is being built at Washington University as part of an ARPA-sponsored gigabit local ATM testbed. An OC-24C (1.2 Gbps) version of the adapter is also planned. At the heart of our interface design is a single custom VLSI chip called the *ATM Port Interconnect Controller* (APIC). The various features supported by this chip include: support for streaming continuous media data from I/O devices (e.g., cameras, disk arrays, etc.) to the network or vice-versa, as well as from device-to-device, while bypassing the main system bus; full AAL-5 segmentation and reassembly; connection caching which allows for up to a thousand transmit and a thousand receive connections; single parameter pacing control; a flexible control mechanism which allows full control of the interface either locally or from a remote location; a zero-copy interface to system memory that is achieved through the use of page remapping techniques; support for low latency, high bandwidth applications by allowing the network interface to be controlled from user space; and a highly flexible interrupt scheme.

1 Introduction

The emergence of very high speed networks [1, 18] has opened up a plethora of possibilities for new and exciting distributed applications. But even as the first gigabit networks make their appearance, it is becoming increasingly evident that merely raising raw network bandwidth does not translate to improved performance for end-applications. The bottleneck is easily identified to lie within the end-host (i.e., the workstations or servers that run the end-applications). Within the end-host, a number of problem areas can be identified that contribute to the bottleneck, not the least of which is a lack of integration between the hardware architecture (particularly that of the host-network interface), the operating system, and network communication protocols.

Solving problems inherent in the software architecture or protocols would not help in overcoming limitations imposed by the peak system memory and bus bandwidths, which are often lower than the peak network bandwidth in a gigabit network. By rethinking the host I/O subsystem, and moving processing power closer to network I/O, it is possible to bypass the main system memory and bus altogether. This

† This work was supported in part by the Advanced Research Projects Agency (ARPA), the National Science Foundation (NSF), and an industrial consortium of Ascom Timeplex, Bell Northern Research, Goldstar Information and Communication, NTT, Italtel SIT, NEC America, Southwestern Bell, SynOptics Communications, and Tektronix.

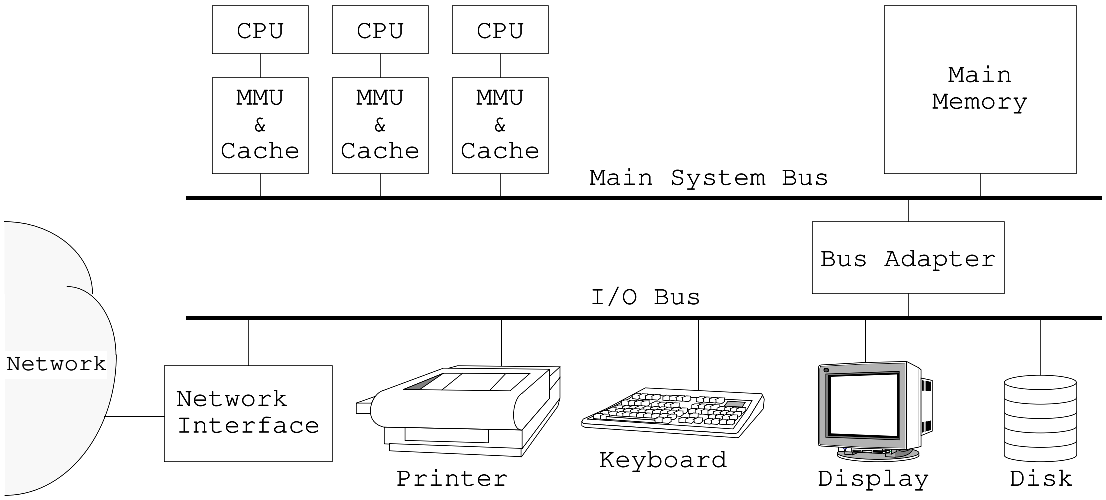

Figure 1: A Typical Workstation Architecture.

approach is especially well-suited for continuous media data (e.g., video, audio, etc.), where the processing is usually of a very specialized nature (e.g., signal processing, compression, encryption, etc.) and needs to be carried out in real-time. For most other types of data, and in particular for discrete media data, it is still important to provide for a high speed pathway to the main system memory.

In this paper, we describe the design of a high performance host-network interface and I/O subsystem that is being developed at Washington University. This design is targeted toward high performance multimedia workstations and servers. A prototype of the interface, which will allow for a sustained data transfer rate of 622 Mbps simultaneously in both directions (into and out from any I/O device or the network), is being built as part of a bigger project (funded by ARPA) involving the setup of a gigabit local ATM testbed at Washington University, and the demonstration of some example applications. We already have an installed and operational 155 Mbps packet switched network developed locally [7], and we are currently in the process of building an ATM switching system [25] that will support data rates of 622 Mbps (SONET) and 2.4 Gbps (G-link) per port. Our primary objective in designing the interface has been to support applications that can effectively utilize these data rates.

2 Motivation

The current state-of-the-art in workstation architectures (see Figure 1) typically includes: support for multiple processors; a main system bus that interfaces to the processors and main memory, and provides support for snooping based cache coherency protocols; a main memory that is organized as a single DRAM bank; and an I/O bus to which most of the I/O devices, including the network interface, are connected. This type of architecture is not well suited to the class of multimedia applications—the primary limitations arise from the bottlenecks imposed by the two buses[1], and from the fact that the effective memory bandwidth is usually no more than 500 Mbps. To compound the problem, protocol stack implementations in existing operating systems require data to be copied multiple times (once from the network interface to kernel space, and then from kernel space to the application's address space). Each such copy step increases the number of times the data has to traverse the system bus, and also increases the number of times memory is

[1] The effective bandwidth of the fastest buses in use today is no longer much higher than the network bandwidth (of emerging networks). With multiple bus traversals, the available bandwidth is typically much lower than the network bandwidth. Typical high performance workstation effective bus bandwidths range from 500 Mbps to 2 Gbps.

accessed. Furthermore, an application may need to access the data, possibly more than once, in order to perform some computation on the data, or just to copy it to a disk. Add to this the fact that multiple processors working on multiple media streams have to share the same bus, and it becomes apparent why many of the interfaces designed to support high speed networks fail to deliver even a small fraction of the network bandwidth to end applications. Clearly, there is a need to revise the host communication architecture and I/O subsystem, if we are to be able to meet the demands imposed by our choice of target applications.

3 Related Work

Several research groups have attempted to design and implement high speed host-network interfaces over the past few years [2]. One of the earliest efforts include the network adaptor board (NAB) [16], which was especially designed to support the VMTP transport protocol [6]. Important aspects of this interface include clever partitioning of protocol processing between the host CPU and a general purpose on-board microprocessor, and the use of a VRAM to allow concurrent protocol processing and transmission/reception of network data. The VRAM is also used to buffer packets in a store-and-forward manner for resequencing before data can be delivered to the host.

The Nectar communication accelerator board (CAB) [4] is a host-network interface that connects through a 10 Mbyte/sec VME interface to the host system. The CAB's on-board processor runs a small kernel and is responsible for transport protocol processing. The unique feature of this interface is that an extra store-and-forward hop can be avoided by mapping the CAB's on-board data memory into the address space of the node process.

Some of the more recent efforts have been concerned with the design of host-network interfaces for ATM networks [9, 24]. The Bellcore OSIRIS interface [9] was designed for the DecStation 5000, and it connects to the host's TURBOchannel I/O bus on one side and to a 622 Mbps SONET STS-12 link on the other. This interface places all data movement and per-cell operations in custom hardware. With a view to achieving the flexibility necessary for experimentation with different scheduling and segmentation and reassembly (SAR) strategies, the control path functions including SAR are implemented using two on-board processors. The flexibility afforded by these on-board processors has also been exploited to allow the interface to be controllable directly from user space [13]—this results in greatly reduced application-to-application message latencies. We employ a similar technique in our adapter design for user-level device control, but our method improves on that presented in [13] by doing away with the need for special on-board processors or other complex memory management hardware.

The University of Pennsylvania interface [24] was designed for the IBM RS/6000, and it interfaces to a SONET OC-3c link (155 Mbps). It features hardware implementation of the ATM segmentation and reassembly pipeline, and carefully partitions the protocol stack so that all per-cell functions can be carried out using dedicated hardware, while software is responsible for higher-level protocol processing and for handling packet movement to/from host memory. We believe this approach affords maximum flexibility without compromising efficiency.

The Afterburner [8] interface design uses a VRAM to hold packets that have been received over the network. The VRAM is mapped into the host kernel's address space, so protocols can examine headers while the packets are still in the VRAM. Once it is determined where the packet is to be copied (into the application's address space), the copying is done by the CPU. Checksums can be computed in this copy loop. This interface design is highly flexible, scales easily, and can deliver high performance. The Medusa interface [5] is nothing but an Afterburner designed for a 100 Mbps FDDI network.

All of the above interfaces connect to the host I/O bus and thus share the limitations mentioned earlier (see Section 2). These interfaces also require at least one data copy per packet from the interface buffer to application buffer. Finally, these interfaces are targeted for single processor machines and deliver all network data into the main memory; thus, they are not particularly suitable for multimedia multiprocessor workstations.

Washington University's Axon project [22, 23] also represents an attempt at designing a high performance host communication architecture for high bandwidth distributed applications. The proposed architecture allows processes to share their virtual memory address spaces, and when a process attempts to access a segment/page that is not in its main memory, it can be retrieved from a local disk or from a remote machine. The host-network interface design allows network data to be copied directly into the application's address space without any store-and-forward hop, and argues that all per-packet data path protocol processing (including the transport protocol) should be done in hardware. However, recent trends suggest that higher level protocol processing can be left to software for flexibility, without compromising performance. The Axon architecture was simulated but not implemented.

A desk-area network (DAN) is an architecture in which a packet switch serves as a workstation interconnect. The APIC interconnect described in this paper is an example of a DAN architecture. Work on desk-area networks was pioneered at the University of Cambridge [14,17] by M.D.Hayter, I.M.Leslie, D.R.McAuley, and D.L.Tennenhouse, and is now being pursued independently at MIT by D.L.Tennenhouse [15]. Both of these efforts are aimed at using a general switch based interconnect within the host; our approach differs in that we use a daisy-chain topology for the interconnect. While general switch architectures can potentially deliver higher performance than our daisy-chain approach, they do come with some drawbacks, at least in the short term—these are discussed in Section 4.3, following a description of our proposed architecture.

4 Proposed Architecture

A high level schematic of our proposed DAN architecture is shown in Figure 2. At the heart of this architecture is a daisy-chained interconnect comprising of a number of *ATM Port Interconnect Controller* (APIC) chips. Individual chips interface either directly to the main system bus, or through a dual or triple ported memory (in this paper, we shall concentrate only on dual and triple ported memories that are implemented as VRAMs) to an I/O device (both the device and the APIC use the serial port(s) of the VRAM to allow for full rate data transfers). One of the APIC chips on the interconnect is connected to the external ATM network through a link interface. ATM cells arriving from the network pass from one APIC to another until they reach an APIC chip that is directly connected to the destination device (or to the system bus). Similarly, data originating from a source device is passed as an ATM cell stream from the APIC directly connected to that device, down the APIC interconnect towards the link interface, and finally out to the network. It is also possible for two local devices to communicate over the APIC interconnect, without cells ever having to leave the desk area. Thus, for example, a video cell stream originating at a camera could traverse the interconnect in order to reach the local display device. The APIC which directly interfaces to the system bus can be used for traffic originating from, or destined for, the system's main memory. All other traffic movement happens only over the APIC interconnect, thus sparing the main system bus of this burden. In our prototype implementation, each APIC will interface to another APIC over two 622 Mbps[2] links (one is used for incoming traffic, and the other for outgoing traffic). The link to the ATM network switch port also operates at 622 Mbps. Thus, our prototype interface will be capable of supporting an aggregate sustained bidirectional data rate in excess of a gigabit per second. One of the major advantages of allowing full rate bidirectional traffic on the interconnect is that each device can access the network in much the same way as if the device had exclusive access to the network; the bandwidth reservation on the network can, in this case, be thought of as extending into the APIC interconnect, all the way to the device. This advantage would be lost if, for example, the interconnect had a unidirectional ring topology in place of a daisy chained topology.

4.1 Media Processing

It would appear that our design does not allow for any type of processing to be done on continuous media

[2] A 1.2 Gbps (on each link) version of the APIC is also planned.

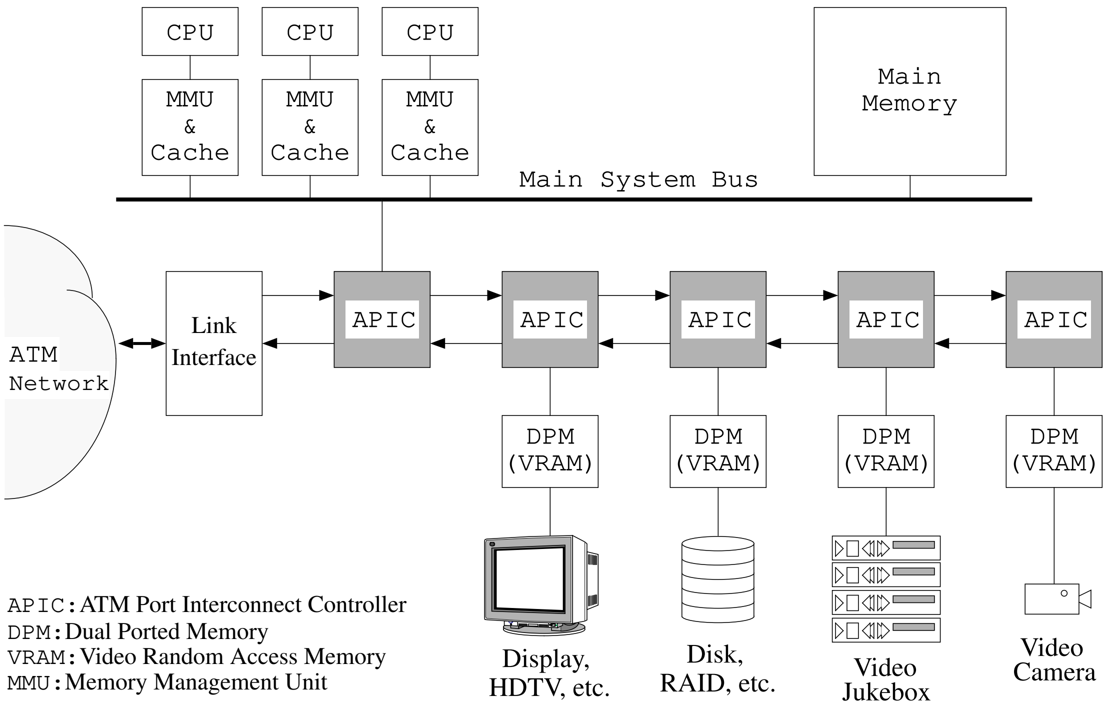

Figure 2: Host-Network Interface Architecture.

streams that do not traverse the main system bus. This is not the case however; media processing on traffic restricted to the APIC interconnect can be done in two ways. The first approach involves embedding a signal processor on the data path between a continuous media device and VRAM. This method suffers from the drawback that the processor cannot be easily shared by multiple devices. However, in some cases this would be the approach of choice, such as, for example, when the media processor is a hardware compression/decompression engine. This ensures that uncompressed data would not have to traverse the interconnect, which is important because the data rate of the uncompressed stream might be a significant fraction of, or more than, the maximum bandwidth of the interconnect. A second approach is to have a special media processor residing as a separate "device" on the interconnect. This method would work especially well when the processing device is an encryption/decryption engine, for example. Data destined for the network could be made to "pass through" the encryption device on its way to the link interface. Similarly, data arriving from the network could be made to pass through this device so the data can be decrypted before being forwarded to the destination I/O device.

4.2 APIC Interconnect vs. I/O Bus

It is easy to see that the APIC interconnect bears a strong resemblance to the traditional notion of an I/O bus. There are several advantages to using I/O buses in addition to CPU-memory buses: I/O buses can be long, can have many types of devices connected to them, and normally follow a bus standard. In contrast, CPU-memory buses are short, cater to very few different types of devices, can change frequently to track changes in technology, and are designed to optimize the performance of the CPU, cache, and memory. It is instructive to contrast our approach with the I/O bus approach with a view to gathering an understanding of why the APIC interconnect can outperform I/O buses in most cases:

- While most of the early I/O buses were asynchronous, many of the new high performance I/O buses are of the synchronous variety (examples include Intel's PCI bus [19], Sun's SBUS, and variants such as the Apple Ring). The APIC interconnect is asynchronous (at the cell level). This allows for high performance and at the same time, the interconnect can span a greater area (i.e., it can be longer).

Furthermore, it is easier to support devices of widely varying characteristics if the I/O interconnect is asynchronous.

- Almost all existing I/O buses are circuit-switched. In contrast, the APIC interconnect is packet switched. Packet switching has been gaining popularity in interconnects used within hosts[20]; however, its use so far in commercial systems has predominantly been confined to the processor-memory interconnect, where it allows for a greater effective bandwidth through the use of split transactions. The primary advantage of using packet switching in our design stems from the fact that we utilize the same packets on the APIC interconnect as those used in the external network. In effect, we are "extending" the external network into the end-host. This allows for devices to be able to stream ATM cells out into the network (or vice-versa) with minimal latency and very little extra hardware.
- An I/O bus is a bus, and therefore it suffers from all the usual drawbacks associated with bus-based architectures; in particular, buses usually scale poorly (to large numbers of devices), they cannot be very long without also becoming unacceptably slow, and only one device can be using the bus at any point of time. Our daisy-chained interconnect topology does not suffer from any of these drawbacks.

By using a standard physical interconnection between APIC chips, an additional advantage of our interconnect becomes apparent—we can make the interconnect longer than would otherwise be possible. For example, our prototype design uses the UTOPIA[3] standard [3] for connecting two APIC chips within a single "box". In going from one box to another within the desk-area, we would switch to an optical medium (622 Mbps SONET). An off-the-shelf chipset can be used to translate from electronic (UTOPIA) to optical (SONET) medium, and vice-versa. The link interface to the ATM network can also be built using this chipset; thus, our entire interconnect architecture requires that we design only one ASIC (the APIC). Finally, note that our architecture does not preclude the use of an I/O bus; in fact, it would be simpler and more cost effective to use an I/O bus for slow devices (such as keyboards, printers, etc.).

4.3 Comparison With General Switch Based DANs

As mentioned earlier, most other DAN based architectures have focused on using a general switch as the desk-area interconnect. While this approach is more general and can deliver higher performance than our APIC daisy-chain approach, it does not come without its drawbacks, at least in the short term. Among these are hardware costs—the APIC solution comes out ahead both in terms of the number of chips that would need to be designed, as well as in the number of chips that would be used in an implementation of comparable size. Furthermore, the APIC scheme provides an incremental cost solution: only as many APICs need to be used as there are devices in the host. An APIC interconnect also has the advantage of being able to span long distances by using optical fiber for connecting widely separated APIC chips. Although not what it was designed for, an APIC interconnect can easily be used as a 622 Mbps optical LAN; this would enable multiple hosts to share a single ATM network link.

4.4 The APIC Chip

The APIC chip, which is central to our design, behaves like a 3×3 ATM switch, two of whose ports are connected to two other APIC chips, and the third port interfaces to external memory (possibly over a shared bus). One of the jobs of the APIC is to take specified data from the memory, create an ATM cell stream using that data, and forward the cell stream to a downstream APIC[4]. In the reverse direction, the APIC is responsible for doing a VPI/VCI virtual channel lookup on every arriving cell to determine if it is destined for the

[3] UTOPIA, which is an acronym for *Universal Test & Operations PHY Interface for ATM*, is a standard proposed by the ATM forum for the data path interface between the ATM layer (performing segmentation and reassembly), and the physical (PHY) layer components of an ATM subsystem. It can support a wide range of speeds: from sub-100 Mbps to 155 Mbps (with an 8-bit wide data path), and up to 622 Mbps (with a 16-bit wide data path).

local device. If so, the cell payload is placed in memory, else it is forwarded to a downstream APIC. In addition to this simple "switching" operation, the APIC implements a host of other features—some of these are described in the next section.

5 APIC Features

In addition to mundane features such as segmentation and reassembly which can be found in almost any ATM host network interface design, the APIC includes a number of unique and novel features which make it highly comprehensive and flexible. The list of features is extensive: multipoint and loopback support; full AAL-5 segmentation and reassembly; support for sending and receiving raw (AAL-0) cells; connection caching which allows for up to a thousand transmit and a thousand receive connections; single parameter pacing control; cell batching for improved performance; a flexible control mechanism which allows full control of the interface either locally or from a remote location; a zero-copy interface to system memory that is achieved through the use of page-remapping techniques; support for low latency, high bandwidth applications by allowing the network interface to be controlled from user space; low latency bypass mechanisms for latency-sensitive applications; a highly flexible interrupt scheme; and, automatic connection demultiplexing combined with support for efficient OS handling of interrupts through the use of an innovative threaded connection strategy.

To describe all of the above-mentioned features would require much more space than we have available here, so we have chosen to concentrate on only a few of them in this paper. In particular, we focus here on the following aspects of the APIC design: multipoint connections and loopback; AAL-5 segmentation and reassembly; connection caching; pacing; batching; and control of the APIC. A detailed description of all the features as well as detailed specifications for the chip can be found in the *APIC System Architecture Document* [10].

5.1 Multipoint Connections and Loopback

If we think of the APIC as a switch with three input ports and three output ports, and if we allow an incoming cell to be forwarded to any combination of output ports, then we are in effect providing a multipoint capability. Such a feature is useful to have if, for example, we wanted a video stream from a camera to be viewed on the local display device, as well as sent out over the network to a remote host. The number of combinations of output ports to which an incoming cell can be forwarded is 8 (this includes the case where the cell is not forwarded to any port, and is discarded). Thus, three bits (on a per connection basis) are sufficient to support multipoint connections. Note that loopback within the APIC comes free with this extension.

5.2 AAL-5 Segmentation and Reassembly

Since the (ATM) educated reader will be familiar with AAL-5, and with segmentation and reassembly functions using AAL-5, we shall not dwell on them here. Suffice it to say that the APIC does not do store-and-forward of entire AAL-5 frames; instead, it periodically reads cells from appropriate locations in external memory (be it the system's main memory or a device VRAM) based on the current position in an AAL frame (for the connection in question), and sends them out into the network. On the receive side also, the APIC reconstructs the original frame in external memory without storing an entire frame in a local memory. Full AAL-5 functionality, with frames of up to 64 Kbytes and a 32-bit hardware CRC, is implemented.

---

[4] We refer to *downstream* and *upstream* APICs several times in this paper; while no such distinction can be made out of context, we typically use these terms when discussing a specific APIC chip, to refer to its two neighboring APIC chips. In the context of a cell arriving at the APIC chip that is the focus of the discussion, we refer to the originating neighbor as the *upstream APIC*. On the other hand, when a cell is forwarded from the APIC in question to another APIC, then the latter will be referred to as the *downstream APIC*.

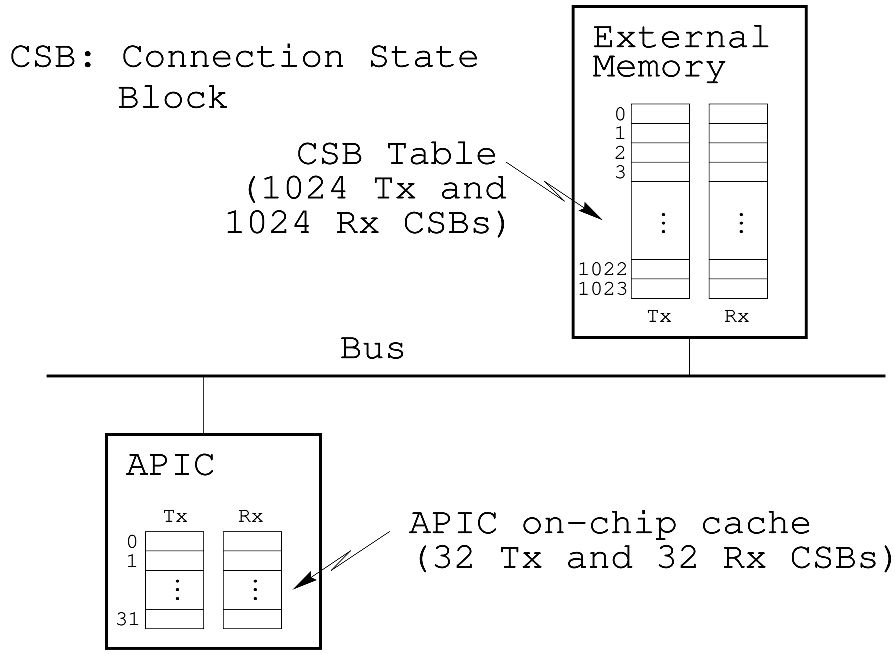

Figure 3: Connection Caching in APIC

5.3 Connection Caching

For each connection, the APIC has to maintain considerable state information which includes how far it has got reading or writing to the current user buffer, where it can find the next buffer, the partially computed CRC for the current frame, where it is in the current frame (start, middle, or end), etc. All of this information is stored in a *Connection State Block* (CSB). The prototype APIC's CSB contains 32 bytes of information (per connection). To get an estimate of the total on-chip storage requirements for all connections, we need to have an estimate of the number of connections that will be supported. A reasonable maximum number of connections for a typical host, which also happens to be the maximum number of connections actually supported by a single prototype APIC, is 1024 transmit and 1024 receive connections. Thus, with 32 bytes per CSB, the APIC would have to have as much as 64 Kbytes of on-chip storage dedicated to connection state information alone. This is clearly impractical; with 0.7 micron technology, 64 Kbytes of SRAM can take anywhere from 250 to 300 square millimeters of silicon. In comparison, technology and cost constraints limit the entire APIC chip to 200 square millimeters.

The solution to this dilemma emerges from the realization that not all of the 1024 transmit and 1024 receive connections would be simultaneously *active*[5]; indeed, the total number of active connections is usually quite small. In our campus LAN environment, even a busy file server that serves 20 to 30 desktop workstations was observed to have no more than ten TCP connections simultaneously sending or receiving data (over a 1 second period of observation). With the advent of applications that utilize continuous media (which can be thought of as using "permanently active connections"), this number will presumably increase, but this increase can be approximated by the total number of open (and active) continuous media connections we expect will enter or leave a host. Given the bandwidth requirements and the potential rate of consumption of information for typical continuous media streams (video, audio, and animated graphics), it is unlikely that this number will be much higher than 10 or 20. Of course, large-scale media servers can

[5] We define "active" connections as those that are involved in sending or receiving data during a sample observation interval. Reasonable values of the observation interval that can be used to decide if a connection is active range from a few milliseconds to a few seconds.

have many more, and in such cases, multiple APICs would be needed to support the desired number of active connections.

In summary, it suffices if the APIC can support a few tens of *active* connections, but the total number of open connections that should be supported needs to be much higher. An active connection usually has its CSB in an on-chip memory. The CSB for connections that have remained passive for a long time may eventually get "swapped out" into external (and cheap) DRAM or VRAM memory, with the CSB for a connection that has newly become active replacing it. This scheme of *Connection Caching* allows the APIC to support a large number of total connections in an inexpensive way, and is illustrated in Figure 3. The prototype APIC design includes support for 32 cached CSBs for transmit and 32 for receive connections.

When a cell is received on an open connection and the CSB for the connection is not in the APIC cache, then it needs to be fetched from the external memory. A candidate has to be chosen for replacement; the most intuitive one, and one that we have verified through simulation as suitable, is the *Least Recently Active* (LRA)[6] connection. This is defined as a cached connection on which a cell has *not* been received for the longest duration of time.

On transmit connections, all caching decisions are left to the host processor or controlling entity; the assumption here is that the sender of the data knows whether it currently has data to send (active) or not (passive). Thus, the APIC control command interface includes commands for loading a specified CSB (identified by a number between 0 and 1023) into a specified on-chip cache entry (identified by a number between 0 and 31), and similarly for saving cache entries into the external memory.

A note on the cost of connection caching is in order. For receive connections, on a cache miss the APIC has to save the CSB for the LRA connection and load the CSB for the connection that just became active. The time for this replacement operation depends on the time necessary to arbitrate for and gain access to the memory bus; for the prototype APIC, the overhead excluding the bus arbitration overhead is close to 2 cell times (a cell time or *cell cycle* is 0.68 µsec at 622 Mbps). Using this estimate, and assuming the probability of a cache miss is p, it is easy to see that the effective bandwidth (as a percentage of the maximum) is given by $1 - \frac{2}{3}p$. This demonstrates that caching is not as expensive as it may seem; even if every arriving cell results in a cache miss (something that is almost impossible, given that we are using an LRA replacement policy), we would still get an effective bandwidth that is a third of the maximum.

5.4 Pacing

ATM networks usually do not provide any feedback to the sender of a cell regarding successful transmission. Given that switch cell buffers are usually restricted to a few hundreds of cells, it would be suicidal to send a big burst of cells back-to-back into the network. Such bursts can cause overflowing of switch buffers, network congestion, and swamping of a slow receiver that is not prepared to handle the high rate of cell arrivals.

To work around this problem, many network interfaces send cells belonging to the same connection spaced out over time, and allow cells from other connections to occupy interleaving cell slots. This operation is commonly referred to as *pacing*. The APIC does pacing based on a single bandwidth parameter that specifies the inter-cell gap. In other words, it sends out cells from the same connection spaced out evenly (periodically) over time. This is not to say, however, that the APIC supports only CBR (continuous bit rate) kind of traffic. While cells within a burst are sent out at a constant rate which is the peak rate for the connection (and is different for different connections), the bursts themselves are scheduled by transmitting applications at arbitrary times and so the overall bit rate can be variable; this is illustrated in Figure 4.

The APIC employs a novel algorithm for pacing (the authors would like to thank J. Andrew Finger-

[6] If a cached CSB is referenced every time a cell is received on the corresponding connection, then the LRA connection can easily be identified by keeping track of the *Least Recently Used* (LRU) cache entry.

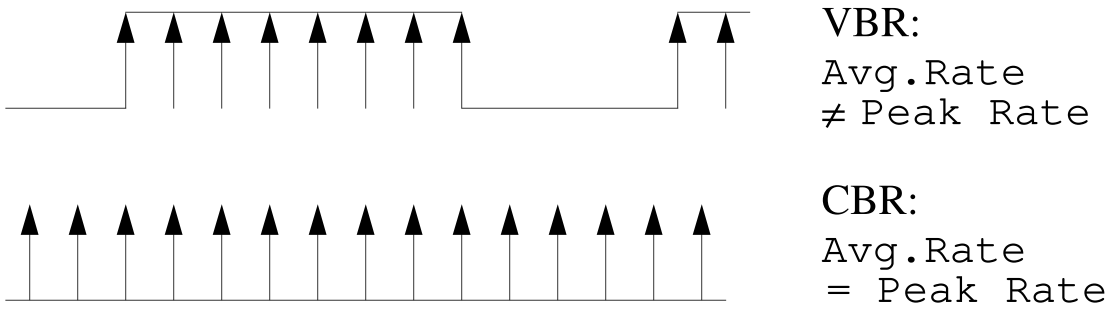

Figure 4: VBR and CBR Traffic

Initialize:

```
    a = 0
```

On each cell cycle:

```
    if (a + m ≥ n) send a cell;
    a = (a + m) mod n.
```

Figure 5: Pacing Algorithm

hut, also of Washington University, for the original idea for this algorithm). In the general case, the algorithm allows the pacing bandwidth for a connection to be any fraction m/n of the total link bandwidth, where m and n are integers and $m \leq n$. The APIC maintains an accumulator a on a per connection basis, which is incremented every cell cycle by m (modulo n). This effectively makes the value of a equivalent to a cell clock that is running at m times the regular cell clock. Thus, by sending a cell whenever a wraps around (in modulo n arithmetic), we effectively send a cell once every n/m cell cycles—this means that the effective (peak) bandwidth of the connection is m/n times the link bandwidth. The algorithm is outlined in Figure 5; note that the "if" condition is satisfied every time there is a carry out from the addition in the following line. This suggests an implementation with a modulo-n adder.

If we fix the value of n, then varying m over a range from 0 to n allows the connection pacing bandwidth to vary from 0 to the link rate in multiples of a fixed "base" rate. The prototype APIC has a link rate of 622 Mbps, and n is fixed at 38880, which implies a base rate of 16 Kbps. All transmit CSBs contain a 16 bit m value (ranging from 0 to 38880) which, when multiplied by the base rate of 16 Kbps, yields the pacing rate for the connection.

5.5 Batching

When interfacing to a continuous media device through a VRAM, the APIC inputs and/or outputs cell contents to the serial access memory (SAM) in the VRAM over its serial port. In order to achieve high efficiency over the serial port without sacrificing bandwidth on the random access port of the VRAM, it is important to make SAM transfers as large as possible. This ensures that there will be a minimal number of wasteful transfer cycles between the SAM and the random access memory (in the VRAM). Typically, the size of the SAM in a 64-bit wide organization of a VRAM is around 512 bytes. With only 48 bytes of payload in an ATM cell, it clearly wouldn't do if we were to waste one transfer cycle on a per cell basis, with the SAM filled to only a small portion of its capacity. The APIC overcomes this problem through the use of *batching*: on transmit, the APIC reads batches of up to 8 cells in a single read operation. Similarly, on receive, the APIC lets cells accumulate and then writes batches of up to 8 cells (all from the same connection) into external memory. Batching also has the potential to reduce bus arbitration overhead and improve overall bus efficiency—this is because a single bus transaction can now be used to move a larger unit of data, and consequently fewer such transactions would be required.

5.6 Controlling the APIC

The APIC can be controlled by two methods: either through memory mapped I/O over the APIC bus interface, or through the use of control cells (perhaps sent by another APIC). Any operation that can be performed through memory mapped I/O can also be done using a control cell and vice versa. All control cell operations result in the return of a response cell. Furthermore, care has been taken to ensure that loss of control or response cells, followed by the retransmission of the control cell by the sending entity, will not cause any problems.

While most devices including network interfaces are controlled using memory mapped I/O, the concept of remote control through the use of control cells is new. The major advantage of this approach is that a local control processor is not required at each of the APICs in an APIC interconnect; indeed, it does not seem worthwhile to dedicate control processors for devices such as cameras, although it may make sense to do so in some cases with devices such as disk arrays. Full remote configurability cannot be achieved however, unless the remote controlling process can also control the media device and read or write to/from the external memory or VRAM connected to the APIC. To this end, the APIC provides for two special control cells that do not change the APIC'S internal state, but can be used to command the APIC to read or write data (specified in the cell) to external memory. With this extension, a remote device can be controlled as follows (assume that memory mapped I/O is used to control the remote device): the APIC connected to the remote device can be made to read or write the device registers simply by sending to it read/write control cells containing the appropriate (memory mapped I/O) addresses.

6 Conclusions

We presented the design of a high performance interconnect that serves both as a high speed replacement for the I/O bus in a multimedia workstation or server, as well as an efficient host-network interface. We also described a few of the features of our interface; a detailed treatment of all the features, as well as a description of the internal design of the APIC chip and its interaction with the operating system software can be found in [10,11].

We are currently in the process of refining our design. We hope to completely specify the design by December 1994, and we shall subsequently undertake the prototyping of the APIC chip using a VHDL toolset. We plan to have the first chips ready in the fall of 1995, and these will be used to prototype APIC interconnect based interfaces for a few target workstations and servers that will include I/O devices such as video cameras, high resolution displays, video storage, and possibly a video wall. This work is part of a bigger effort sponsored by ARPA (slated for completion in 1996) that is aimed at the design and deployment of a gigabit local ATM testbed at Washington University.

References

[1] "Broadband Switching: architectures, protocols, design, and analysis", edited by Chris Dhas, Vijaya K. Konangi, and M. Sreetharan, *IEEE Computer Society Press*, 1991.

[2] "IEEE Journal on Selected Areas in Communications, Vol. 11 No. 2 — Special issue on High-Speed Computer/Network Interfaces", edited by J.M. Smith, E.C. Cooper, B.S. Davie, I.M. Leslie, Y. Ofek, and R.W. Watson, February 1993.

[3] "An ATM PHY Data Path Interface: Draft Version 0.5", an ATM Forum Document Draft, May 1993.

[4] Arnould, Emmanuel *et al*; "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers", *ASPLOS-III (ACM SIGOPS Operating Systems Review)*, Vol. 23, New York, April 1989, pp. 205-216.

[5] Banks, David; and Prudence, Michael; "A High-Performance Network Architecture for a PA-RISC Workstation", *IEEE JSAC*, Vol. 11 No. 2, February 1993, pp. 191-202.

[6] Cheriton, David R.; "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *Proc. ACM SIGCOMM '86*, Vol. 16 No. 3, 1986, pp. 406-415.

[7] Cox Jr., Jerome R.; Gaddis, M.; and Turner, Jonathan S.; "Project Zeus: Design of a Broadband Network and its Application on a University Campus", *IEEE Network*, Vol. 7 No. 2, March 1993, pp. 20-30.

[8] Dalton, C.; Watson, G.; Banks, D.; Calamvokis, C.; Edwards, A.; and Lumley, J.; "Afterburner", *IEEE Network*, Vol. 7 No. 4, July 1993, pp. 36-43.

[9] Davie, Bruce S.; "The Architecture and Implementation of a High-Speed Host Interface", *IEEE JSAC*, Vol. 11 No. 2, February 1993, pp. 228-239.

[10] Dittia, Zubin D.; Cox, Jerome R.; and Parulkar, Guru M.; "A Gigabit ATM Desk Area Network: System Architecture Document for the ATM Port Intercoonnect Controller (APIC)—Draft Version 1.0", August 1994.

[11] Dittia, Zubin D.; Cox, Jerome R.; and Parulkar, Guru M; "Catching Up With the Networks: Host I/O at Gigabit Rates", *Techinical Report WUCS-94-11*, Washington University in St. Louis, April 1994.

[12] Druschel, Peter; and Peterson, Larry L.; "Fbufs: A High-Bandwidth Cross-Domain Transfer Facility", *Proc. 14th Symposium on Operating System Principles*, Asheville, NC, Dec. 1993, pp. 189-202.

[13] Druschel, Peter; Peterson, Larry L.; and Davie, Bruce S.; "Experiences with a High-Speed Network Adaptor: A Software Perspective", *Proc. ACM SIGCOMM '94*, Vol.24, No. 4, 1994, pp. 2-13.

[14] Hayter, Mark D.; and McAuley, Derek R.; "The Desk Area Network", *Operating Systems Review*, Vol. 25 No. 4, October 1991.

[15] Houh, Henry H..; Adam, Joel F.; Ismert, Michael; Lindblad, Christopher J.; and Tennenhouse, David L.; "The VuNet Desk Area Network: Architecture, Implementation, and Experience", to appear in *IEEE JSAC*, First Quarter, 1995.

[16] Kanakia, Hemant; and Cheriton, David R.; "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors", *Proc. ACM SIGCOMM '88*, Vol. 18 No. 4, 1988, pp. 175-187.

[17] Leslie, Ian M.; McAuley, Derek R.; and Tennenhouse, David L.; "ATM Everywhere?", *IEEE Network*, Vol. 7 No. 2, March 1993, pp. 40-46.

[18] Partridge, Craig; "Gigabit Networking", *Addison Wesley*, 1994.

[19] "Peripheral Component Interconnect (PCI), Revision 1.0 Specification", *Intel Corporation*, June 22, 1993.

[20] "SPARCcenter 2000 Architecture and Implementation", Technical White Paper—Draft Edition of 11/18/92, Sun Microsystems Inc., November 1992.

[21] "SPARC MBUS Interface Specification: Revision 1.2", SPARC International, April 1991.

[22] Sterbenz, James; and Parulkar, Guru; "Axon: Network Virtual Storage Design for High Performance Distributed Applications", *Proc. 10th IEEE Conf. on Distributed Computing Systems*, May 1990.

[23] Sterbenz, James; and Parulkar, Guru; "Axon: Host-Network Interface Architecture for Gigabit Communication", *Protocols for High Speed Networking*, Elsevier (North Holland), 1991.

[24] Traw, C. Brendan S.; and Smith, Jonathan S.; "Hardware/Software Organization of a High-Performance ATM Host Interface", *IEEE JSAC*, Vol. 11 No. 2, February 1993, pp. 240-253.

[25] Turner, Jonathan S.; "An Optimal Non-Blocking Multicast Virtual Circuit Switch", *Technical Report WUCS-93-30*, Department of Computer Science, Washington University in St. Louis, MO, 1993.