Xunet 2: Lessons from an Early Wide-Area ATM Testbed

C. R. Kalmanek [1]
S. Keshav [2]
W. T. Marshall [3]
S. P. Morgan [4]
R. C. Restrick [3]

Abstract — This paper is a retrospective on the design of Xunet 2, one of the earliest functional wide-area ATM networks. Work on Xunet 2 began in 1989 and the network, consisting of experimental ATM switches, IP routers, and 45 Mbps transmission lines, has been operational since October 1991. The network serves as a ‘‘laboratory without walls’’ for eight research groups across the United States. While Xunet 2 has only a small number of nodes, it was designed as a prototype of a nationwide ATM network. This paper reviews some of the design decisions and lessons learned in the project and points out the research directions motivated by this work, focusing on the areas of traffic management, ATM switch design, network control, and the implementation of an IP router.

1. Introduction

The Experimental University Network (Xunet) program began in 1986 when AT&T Bell Laboratories formed a research collaboration with the University of California at Berkeley, the University of Illinois at Urbana-Champaign, and the University of Wisconsin at Madison. The collaboration emphasized student research and the sites were linked by a network of Datakit® Virtual Circuit Switches [FRAS83] joined by 1.544 Mbps transmission lines. This network has since become known as Xunet 1. The focus of the Xunet program was broadened in 1989 when AT&T started to plan a high-speed ATM network: Xunet 2. Our research group at AT&T planned to use the new network as a model for a multi-service ATM backbone. In addition, Xunet 2 would be managed as a wide-area systems research laboratory where student researchers could sign up for time to run network experiments. Xunet 2 has been operational since October 1991. Over time, the Xunet program has grown. The network was extended to include Sandia National Laboratories and Lawrence Livermore National Laboratories in January 1993 and Rutgers University in April 1993. The University of Pennsylvania and Columbia University joined the collaboration, although we were never able to fund their connection to the network. In addition, AT&T upgraded some segments of the network to 622 Mbps to support the BLANCA project, one of five ‘‘Gigabit Testbeds’’ in the National Research and Education Network (NREN) initiative [CNRI95].

Xunet 2 consists of ten experimental ATM switches (two in Murray Hill) interconnected with 45 Mbps transmission lines (Figure 1). There is an ATM switch at each user site and switches at three AT&T central office locations: Oakland CA, Chicago IL and Newark NJ. A high performance workstation at each user site acts as an IP router between a local FDDI ring and the ATM network. The routers provide a LAN interconnection service, carrying encapsulated IP packets over ATM virtual circuits. In addition, a native ATM protocol stack on each of the workstations supports end-to-end communication over switched virtual circuits.

Figure 1. Network topology.

Xunet 2 was designed as a small-scale prototype of a nationwide ATM network. Thus, we made decisions that were both pragmatic and scalable to larger networks. Our research focused on three main areas: LAN interconnection over a wide-area ATM backbone, performance issues in supporting multiple types of traffic, and the issues of controlling and managing large-scale ATM networks.

The paper presents our work in four sections: performance and traffic management (Section 2), the design of our experimental switch (Section 3), network control and management (Section 4), and the design of our IP router (Section 5). In each section we present our goals, the architecture that resulted, and the lessons we learned trying to turn vision into reality. We also present the research directions motivated by our experience.

[1] Currently with AT&T Labs - Research. Address correspondence to: crk@research.att.com
[2] Currently with Cornell University Department of Computer Science.
[3] Currently with AT&T Labs - Research.
[4] Currently with Bell Labs/Lucent Technologies.

2. Performance and Traffic Management

2.1. Goals

An important goal in Xunet was understanding how to support multiple traffic types on a shared public ATM network. This section describes the evolution in our thinking about ATM traffic management, and summarizes the studies that reinforce our conclusions. Implementation issues are discussed in later sections.

We wanted to support three service classes, now called available bit rate (ABR), constant bit rate (CBR), and variable bit rate (VBR). ABR service was intended for interconnection of conventional IP-based local area networks, which we foresaw as the initial application of ATM. CBR service would be used for constant bit rate traffic such as voice and circuit emulation, while VBR was studied as a means of carrying variable-bit-rate video traffic.

Our traffic management architecture was guided by three principles. First, *the network exists to serve the needs of its users*. A network designer's role is to provide the mechanisms and policies that satisfy the quality of service desired by users. Second, *network users expect to receive a predictable quality of service* even though facilities are shared. Thus, safeguards are needed to protect against users who misbehave, either maliciously or inadvertently. This suggests that while users may agree to use end-to-end mechanisms which are cooperative, the network cannot assume the correct operation of end-to-end mechanisms: some form of policing is required. Third, *the network must use resources efficiently*, so that high-speed communication is affordable. This precludes, for example, using CBR connections to interconnect local area networks, since this would not provide the tremendous cost saving arising from statistical multiplexing.

2.2. Traffic Management Architecture and ABR Service

Our traffic management architecture has a number of familiar elements. Users request a particular class of service during virtual circuit setup. Once a connection is established, users regulate their traffic within a certain ‘‘user behavior envelope’’ to avoid loss. Schedulers in switches distinguish between virtual circuits of different service classes in order to meet their quality-of-service requirements. Policing insures that users who violate their behavior envelope do not affect the performance seen by other users.

We spent considerable effort understanding ABR service and believe that we have gained some valuable insights.[5] We first focus on three aspects of ABR service: scheduling, buffer management, and flow control, before returning to other service classes.

We believe that ABR service will be used to support computer traffic with widely divergent characteristics and requirements. Some applications will send short messages and want low delay, while others will want to maximize throughput. Since the scheduler at a switch is responsible for allocating bandwidth, and thus controlling delay, it plays a crucial role in providing this service. Fundamentally, in order to avoid congestion, it is necessary to control the offered load. Over the years we have explored a number of mechanisms for doing this. Since these mechanisms operate on individual virtual circuits, we rely on end-to-end flow control to avoid complexity in the center of the network, which operates at high speed and where a large number of virtual circuits are present. In addition, the network protects itself through the use of per-virtual-circuit queueing, round robin scheduling, and intelligent buffer management, all of which act to separate virtual circuits and put limits on or "police" user behavior.

2.2.1. Scheduling

The importance of per-virtual-circuit queueing and round robin scheduling in data networks has been addressed elsewhere [NAGL86, KATA88, DEME89, HAHN91, MORG91], but we briefly summarize the argument. A round robin scheduler is one in which each virtual circuit has its own, logically distinct, data queue and the scheduler serves non-empty data queues in turn. When a link is lightly loaded, there is little difference between round robin and first-come-first-served scheduling. However, when a link is congested, round robin scheduling allocates bandwidth equally among the virtual circuits. Short messages typically see low delay. Provided that sources control their offered load appropriately, no queue will grow very large but if a source consistently sends too much, its queue will overflow and its data will be dropped. Moreover, round robin scheduling provides a more consistent service rate than FCFS. In an FCFS scheduler, the service rate of a virtual circuit is linked to the detailed arrival pattern of every other virtual circuit sharing the link. Even short messages can be subjected to long delays, and a user who sends at a sustained high rate can effectively consume an arbitrary fraction of the network bandwidth. These problems are avoided by using round robin service. Our architecture thus calls for the use of a round robin scheduler for ABR traffic.

2.2.2. Congestion Avoidance

While round robin scheduling shares bandwidth appropriately, it is up to end systems to control the offered load. We have explored several approaches to controlling load: end-to-end window flow control with both static and dynamically adapted windows, as well as adaptive rate control.

The intuition behind the window schemes is that a source never needs a window size larger than its bandwidth-delay product since that is the largest amount of unacknowledged data that must be in transit to achieve the maximum throughput. The research version of the Datakit switch [FRAS83] provided static buffer allocation of a full round trip

[5] The term "ABR" has come to refer to the rate-based flow control framework defined by the ATM Forum [BONO95, FEND96]. We use the term more generically to refer to the service model that we describe in this section, unless the work of the ATM Forum is specifically referred to.

window at the output queue of every switch for each virtual circuit. Thus the total buffer allocation B at an output queue would be

$$B = NW_0 \ , \qquad (1)$$

where N is the total number of virtual circuits that can simultaneously share the queue. In this equation we assume that all virtual circuits have the same round trip delay and are bottlenecked at the same output queue. Thus, W_0 is the end-to-end round-trip window, that is, the bandwidth-delay product, of each virtual circuit.

Static buffer allocation implicitly polices user behavior, since a user sends more than the buffer allocation of data into the network at its own risk. Furthermore, allocation of a full round-trip window to a virtual circuit guarantees that the source will not lose data even if there is momentary congestion while it is transmitting at full speed, since there is always a large enough buffer to accommodate the data in the pipe. However, intuition suggests that static buffer allocation according to (1) uses much more memory than necessary. When the network is congested, the bandwidth available to an individual virtual circuit is reduced, and so is the round trip window needed for it.

In *dynamic window flow control*, virtual circuits start with a small default buffer allocation and set their window size to the same value. A virtual circuit can request a larger buffer allocation from the network. When the network is uncongested, the virtual circuit can potentially send at the full rate of its access line, so the network grants a buffer allocation corresponding to the round trip window at that rate. As the network becomes congested, however, the buffer allocation that is granted is reduced. When virtual circuits become idle, the buffer allocation returns to the default. In the dynamic buffer allocation scheme described by [HAHN93], the total buffer requirement per output port is approximately

$$B = W_0 \log N + C \ , \qquad (2)$$

where C is proportional to the renegotiation interval after which an idle circuit's allocation is returned to the default. For realistic parameter values, Eq. (2) represents an order of magnitude less memory than Eq. (1), but B can still amount to several tens of round-trip windows for each output port.

Dynamic buffer allocation guarantees that a user will never lose cells due to buffer overflow if the user adheres to the current window size granted by the network. On the other hand, dynamic buffer allocation requires a fairly elaborate signaling mechanism between end systems and the network, as well as a scheme for monitoring buffer fill in the switch. Moreover, it requires that the queue handler discard data from a virtual circuit queue when it exceeds the current allocation, which changes dynamically. Thus, it requires a mechanism to coordinate between the state machines which manage the buffers, which necessarily reside in high-speed logic, and a processor which runs the buffer allocation algorithm. The Xunet queue handler supports these mechanisms, but their widespread adoption is unlikely. We prefer to minimize the mechanisms needed in a switch.

Substantially smaller buffers and simpler switch mechanisms are possible if one gives up the requirement that cells are *never* lost due to buffer overflow within the network. ABR traffic can tolerate occasional cell losses. This observation has led us, as it has led others, to prefer rate control rather than window flow control for avoiding congestion of ABR traffic on high-speed wide-area networks.

We described above how a round robin scheduler allocates bandwidth to each virtual circuit sharing a link. The service rate will vary over time as virtual circuits become active and inactive, but suppose a source could determine the service rate of the bottleneck link as a function of time and adapt its sending rate to match this rate. This would avoid congestion and is one of the key ideas behind the *Packet-Pair flow control* scheme [KESH91, KESH95].

With Packet-Pair, a source probes the network by sending packets in back-to-back pairs and measures the spacing of the acknowledgments to estimate the service rate and amount of data buffered at the bottleneck link. The source uses these estimates to adapt its sending rate to keep the bottleneck buffer content close to a desired setpoint. The choice of setpoint permits a tradeoff between queueing delay and link utilization.

Although round robin service allocates bandwidth fairly and separates data from different virtual circuits into different queues, a buffer management policy is needed to prevent an aggressive user from consuming more of the buffer than its fair share. A number of policies are possible, for example, each user could be allocated a static buffer. We have found that a simple and effective policy is to allocate a certain fraction of the total buffer in a queue handler to the ABR service and, when that buffer becomes full, to discard all of the data on the longest virtual circuit data queue. Simulations show that with this policy users who attempt to get more bandwidth at the bottleneck by "cheating" always end up hurting themselves, while users who correctly adapt their sending rate get good service [KESH97]. An example is shown in Figure 2.

Figure 2. Packet-Pair behavior under overload.

In Figure 2, ten statistically identical sources share a trunk whose speed determines the maximum speed of each virtual circuit. All of the sources share the same round-trip delay. Nine sources adapt their sending rate with the Packet-Pair rate control algorithm and one does not. The nominal offered load, which is the sum of the offered loads due to each source individually, varies from 0.5 to 5.0 times the capacity of the bottleneck link. The total buffer space B, measured in terms of round-trip windows, varies from 0.25 to 1.00. When an arriving cell would overflow the buffer, the entire longest queue is discarded. The plots[6] represent the average goodput achieved by Packet-Pair and non-Packet-Pair sources. Perfect

[6] The vertical bars represent approximate 95% confidence intervals of the simulations.

fairness under overload would correspond to a goodput of 0.10 times the capacity of the bottleneck link for each source, in the absence of losses. Figure 2 and other simulations show that Packet-Pair reveals no sign of congestion instability, at least for offered loads up to 5.0, that total goodputs in the 0.8-0.9 range can be achieved with buffering equal to a fraction of a round-trip window per output port, and that non-Packet-Pair sources damage only themselves in the presence of overload.

2.3. Other Traffic Classes

In addition to ABR traffic, it is generally expected that ATM networks will carry continuous-media traffic such as constant bit rate traffic and real-time audio and video. The different types of traffic interact at the scheduler in a switch and through the network's admission control policy. We focus on these topics in this section.

2.3.1. Constant Bit Rate

Constant bit rate traffic consists of cells that are approximately equispaced on entry to the network. As the cells of a given connection pass through the network, they are subject to queueing delay so that the exiting cell stream is no longer equispaced. An elasticity buffer at the destination is used to absorb the delay jitter introduced by the network so that cells can be read out at equispaced intervals. Since the size of the elasticity buffer often needs to be known in advance, the network seeks to control the delay jitter. Both the size of the elasticity buffer and of the switch buffers are of interest.

One approach to controlling delay jitter involves retiming or reshaping of traffic within the network, as in Stop-and-Go Queueing [GOLE91] or Hierarchical Round Robin Scheduling (HRR) [KALM90, SARA94B]. These disciplines allow the network to control the burstiness of the traffic streams exiting a server, which affects the amount of delay jitter that can be introduced by the downstream switch. However, our analytic and simulation studies indicate that it is sufficient for most applications to give CBR traffic priority over other types of traffic, rather than doing reshaping. This simplifies the design of the scheduler, but requires policing at the edge of the network to insure that users don't send faster than their negotiated rate.

Thus, in our traffic management architecture, CBR traffic is given highest priority at a priority scheduler. There is no need to do anything more than FCFS scheduling among CBR connections, although round robin scheduling also works. Peak rate policing is implemented using a leaky bucket, with burst tolerance equal to one cell, at the edge of the network.

Admission control insures that sufficient capacity is available for CBR traffic. However, cell level queueing will occur as a result of the detailed arrival patterns of the input streams. Cell level queueing resulting from the superposition of deterministic streams having the same or different spacings has been extensively studied [ROBE92]. The simplest case occurs when N streams with the same spacing but different phases are multiplexed onto a single link. In an ideal model, the merged arrival process will be periodic, as will the output of the link. In the worst case, the phase relationships of the different streams might cause cells from all N streams to arrive simultaneously, which would result in $N-1$ cells of queueing delay (or cell loss if insufficient buffers are available). However, only a small fraction of the possible phase relationships among the input streams lead to substantial queueing, and ensemble averaging is conventionally used to estimate the probability of queueing delay.

Figure 3: Cell-level delay distributions for CBR traffic.

The ensemble average queue length for a large number of deterministic streams arriving at the output queue of a switch can be approximated by the queue length distribution for an M/D/1 queue with the same utilization [ROBE92]. Figure 3 gives an approximation to the delay distribution for an M/D/1 queue at 90% utilization, as well as the cumulative delay distribution for a series of 10 independent but statistically identical M/D/1 queues, each at 90% utilization. In the figure, $Q(x)$ is the probability that the queue length exceeds x cells. We note that the transmission time for one ATM cell on a 45 Mbps link is about 10 microseconds. Figure 3 shows that the probability that the delay through 10 switches exceeds 2 ms is 7.4×10^{-10}. Buffers roughly 2 ms in length at the output ports of each switch should therefore provide adequately for cell loss rates of 10^{-9}. An elasticity buffer roughly 2 ms in length is also needed at the destination to absorb the delay jitter introduced by the network. For applications for which these buffer sizes and delays are acceptable, there is no need to do more than FCFS scheduling for CBR traffic at a switch. The queueing delay decreases sharply with decreasing utilization.

2.3.2. Variable Bit Rate Video

Current designs for ATM networks include a variable bit rate (VBR) service. It is expected that a user will negotiate a traffic descriptor, typically a leaky bucket, with the network, and the network will agree to meet certain quality-of-service parameters for users whom it admits into the network. The VBR service is intended to be used for traffic that has a well-known long-term average rate, but that periodically generates bursts, hence a leaky bucket involves three parameters: the average or sustainable rate, the peak rate, and the burst tolerance. Since users will presumably not burst simultaneously, the network can statistically multiplex a number of VBR users. However, the choice of the leaky bucket parameters determines the amount of statistical multiplexing that can be achieved for a given cell loss rate, delay variation, and switch buffer pool.

VBR service has been extensively discussed as a vehicle for carrying variable bit rate video traffic. However, measurements [VERB89, RATH93] of both teleconferencing and entertainment video traffic suggest a number of problems with a VBR video service based on leaky buckets.

One problem is that it may be difficult to pick reasonable leaky bucket parameters *a priori* for any particular video service. We found in in-house measurements that two videoconferencing streams produced by the same H.261 codec differed in average rate by a factor of two, depending on whether the speaker was sedate or agitated. In [SARA94B], we supposed that VBR video traffic would be carried through an HRR server, but again we had no way *a priori* of determining the HRR rate. Of course, one might imagine the user picking from a menu of "default" leaky bucket parameters provided by the encoder, since codecs can be designed [REIB92b] to comply with any given leaky-bucket parameters (CBR output is a special case). However, default choices may lead to a video quality that is no better than would be obtained by a CBR service with the same average rate.

Leaky-bucket descriptors of *unconstrained* video encoder output require a large burst tolerance [REIB92a], unless the average rate is nearly equal to the peak rate, which would not allow for much statistical multiplexing gain. The reason is that encoder output, especially for entertainment video, can have peak-rate bursts lasting for several seconds. VBR video service needs to be able to accommodate such sustained peaks.

There are in principle two approaches to handling large burst tolerances. In the first approach, mean cell loss ratios are kept low by keeping the probability low that enough sources are in burst mode simultaneously to overload the link [WEIN78]. Since few sources are admitted into the network, this approach results in poor statistical multiplexing gain. A different approach increases the number of sources that could be accommodated at a given cell loss ratio. The drawback is that switch buffers of a size comparable with the large burst tolerances of unsmoothed video traffic (potentially tens of megabytes per source for entertainment quality video) would introduce unacceptable delays, at least for real-time video. Moreover, large switch buffers cost.

A final issue is that the amount of statistical multiplexing gain that can be achieved with video sources may be rather small. The video community now appears to expect potential gains in the range from 1.5:1 to 2:1 [REIB92a], but it is hard to estimate this number because there are so many alternative video system configurations, and because there is no easy way to relate subjective judgments of video quality to quantities that can be computed in a network performance analysis.

Our present view is that the need for VBR video service over ATM networks using the "conventional" model, which involves leaky bucket traffic descriptors, is questionable. We will mention briefly some alternatives for video transport in Section 2.4.

2.4. Lessons

We conclude that ATM networks should support CBR and ABR service, and we propose an architecture for doing so. Our notion of an ABR service is similar in spirit to the ATM Forum UBR (Unspecified Bit Rate) service. For this service we are convinced of the importance of per-virtual-circuit queueing and round robin scheduling. The simple buffer management policy of dealing with overflow by discarding all of the data from the longest queue provides protection against misbehaving users. The amount of buffer required at an output queue in a switch depends on the approach to congestion control. It is likely that we can get by with about a round trip window of memory shared among the virtual circuits, while achieving quite low packet loss rates, so long as a well designed congestion control scheme is used. As we shall discuss in the next section, this amount of memory is both practical and affordable, even for 155 Mbps access lines.

CBR service can use either priority FCFS or priority round robin scheduling, with policing at the network's edge. Of course, CBR traffic can also be policed by reshaping traffic at the scheduler in a switch. The decision between these two approaches is one of cost and complexity of implementation. Our view is that the simpler switch scheduler is sufficient.

Congestion avoidance requires end systems to participate in controlling the offered load. We believe that some form of end system rate adaption is most appropriate in wide-area networks since it does not impose onerous requirements for buffering or complexity on switches. In our view, host computers that attach directly to ATM networks should run Packet-Pair or some equivalent rate adaption algorithm. If the network provides explicit rate feedback through the use of Resource Management cells, as specified in the ATM Forum ABR service [BONO95, FEND96], a host would simply use this rate instead of making its own rate estimates. When ATM networks are used to interconnect conventional local area networks, hosts will be running conventional TCP/IP. If routers run a congestion control protocol such as Packet-Pair over a wide area network, our experience suggests that the TCP hosts will see good performance and the network will be used efficiently.

It is not clear to us that a leaky-bucket-controlled VBR service is of much use for traffic that is as bursty as variable bit rate video. Two recent approaches to carrying VBR video are attractive. First, [KANA93, LAKS97] demonstrate that it is possible to achieve good video quality by adapting the sending rate of a video coder in response to network congestion. Our simulations show that video carried using the ABR service, with the coder's sending rate controlled using Packet-Pair, gives a perceptually high quality even in the presence of congestion. [SAFR95] shows how to adapt the sending rate for stored video. Second, [GROS95] describes a scheme based on ‘‘fast’’ renegotiation of a CBR rate. Renegotiation allows the network to extract statistical multiplexing gain on a time scale that is slow compared to the feedback-based rate adaption schemes such as Packet-Pair. Since coded video exhibits sustained periods where the output frame sizes are roughly constant, renegotiation may be well suited to the traffic requirements. By matching network mechanisms to the traffic dynamics, we should be able to get good performance at a low cost.

3. Xunet Switch

3.1. Goals

The design of an ATM switch for Xunet began in 1989 with the goal of creating a flexible platform for experimentation in network performance and control. The switch was designed for a wide area backbone network with 45 Mbps inter-switch trunks and access lines at 1.5 Mbps and 45 Mbps. Though we expected computer traffic to predominate on this network, we wanted the switch to support multiple traffic types effectively. Thus, a major focus of our effort in the design of the switch was on the architecture of the queue handler responsible for cell scheduling and buffer management. This section describes some of the implementation issues.

A second set of goals related to switch administration and maintenance. We planned to run call processing, administration and maintenance software on an external workstation, and wanted a control interface to the switch which was independent of the fabric in order to promote modular design and to allow the switch hardware and the control software to evolve independently. Our previous experience with call processing led us to incorporate consistency checking in the switch software to deal with various error cases. Finally, given the difficulty of maintaining and operating a cross-country network, we thought carefully about how to build switches that could be maintained.

3.2. Switch Architecture

The Xunet switch was designed as a bus-based output queued switch with support for per-virtual-circuit queueing at the output ports, similar to the Datakit switch [FRAS83]. The switch architecture is shown in Figure 4. In the figure, each port controller consists of two cards: a queue handler and a line interface card. The queue handlers communicate with the header translation card via a switch fabric consisting of a set of time-slotted busses on a printed-circuit backplane. When a cell arrives from a transmission line, the line interface card terminates the physical and link layers and transfers the cell to its queue handler. The queue handler buffers the cell and subsequently schedules it to be sent to the header translation card. The queue handler arbitrates with other queue handlers for permission to send the cell on the *contention bus*. When its arbitration succeeds, the queue handler transmits the cell on the contention bus to the header translation card. The header translation card updates the cell header and sends the cell on the *broadcast bus* to the output queue handler(s). The destination output queue handler buffers the cell and schedules it for transmission via the output line interface card.

Figure 4. Switch architecture.

The *maintenance bus* is based on the Signetics I^2C bus and is used for transferring commands and status reports between the various circuit cards in an equipment shelf. Figure 4 also shows that the switch is controlled by an external switch controller. The controller can connect to the switch through a line card, or via an Ethernet connection to the header translation card as shown in the figure.

The queue handlers arbitrate for access to the contention bus using a distributed group arbitration protocol [VERN88]. Arbitration is pipelined, so that queue handlers arbitrate during one time slot for permission to send a cell during the next time slot. During each arbitration cycle, a queue handler that wishes to transmit asserts its slot address on the *arbitration bus*. The protocol insures that all but one competing card withdraws from the competition and the address of the winning card remains on the bus. A traffic priority bit gives CBR traffic priority over other traffic types, and a ‘‘group priority’’ bit insures that within each traffic class the queue handlers share the bus bandwidth fairly.

The contention and broadcast busses are 32 bits wide (plus 4 parity bits) and are clocked at 18.5 MHz. Since a cell transfer takes 14 clock cycles, this gives a payload rate of over 500 Mbps. Our prototype switch has eight ports. Thus, with 45 Mbps access lines, the 500 Mbps bus bandwidth exceeds the aggregate bandwidth of the trunks and queues will only build up on the outputs. Since we operate these busses an order of magnitude faster than the access lines, input queues should occur infrequently even in a switch with a larger number of ports [OIE89].

3.2.1. Header Translation Card

The header translation card serves multiple functions. First, the card does the VCI translation needed for cell switching. In support of this function, a processor on the card audits the translation memory for consistency. Second, the card interfaces the switch controller to the switch, which avoids dedicating a port on the switch for this purpose. Finally, it supports a fabric-independent protocol used by the switch controller to control, administer and maintain the switch. We first focus on header translation.

The header translation card supports virtual circuit switching with a 1 M entry translation memory supporting 64 K virtual circuits for each of 16 ports. The translation memory is indexed by the source port address and source virtual circuit identifier. Each entry contains a destination port, a destination virtual circuit identifier, a cell count (for billing), and some control bits which allow the consistency of the memory to be audited. A processor, called the *monitor* (MON), can read and write the translation memory. The MON also runs an auditing task in the background that we will discuss below.

The header translation card interfaces to the switch controller via a point-to-point Ethernet segment. The switch controller communicates with the *host interface processor* (HIP) over the Ethernet. The HIP does segmentation and reassembly and routes control packets between the Ethernet, ATM fabric, MON, and a *maintenance processor* on the header translation card.

Figure 5 gives a high-level diagram of the data paths in the header translation card. A cell arriving on the contention bus is shifted into a shift register while a translation memory access is initiated. After a one cell delay, the cell with its modified header is shifted out, in most cases onto the broadcast bus. Cells that are destined for an endpoint on the card are shifted into the *input* FIFO, which interfaces to the HIP. A second FIFO, the *spy* FIFO, is used for troubleshooting network problems: it receives cells for connections that have been marked with a special bit in the translation memory. The HIP sends cells to the fabric by writing to the *output* FIFO. After the HIP writes a complete cell to the output FIFO, the card arbitrates with other cards in the switch for access to the contention bus and then sends the cell. The HIP interface to the Ethernet and maintenance processor are also shown.

Figure 5. Header translation card data paths.

Software in the HIP and MON processors implements an abstract model of the switch which hides the details of the header translation card design. The interface is designed to allow switch hardware and control software to evolve independently. The commands fall into two categories: commands to manipulate the translation memory are sent to the MON and commands that provide switch maintenance and administration are sent to the maintenance processor.

We first discuss the header translation commands. These commands allow the switch controller to set the desired state of a connection endpoint, e.g. the endpoint is idle, dialing (connected to the controller), or active (connected to a distant endpoint). The switching path is only set up by the MON when the controller has set both endpoints of the connection to active state. Since switch control software is often written with separate modules for the incoming and outgoing ‘‘ends’’ of the connection, this model allows each module to continue independently: the MON sets up the connection when the two ends ‘‘rendezvous.’’

In our experience, a switching fabric and its embedded fabric control software can be designed to be more reliable than either the commercial computer equipment that we are using for the switch controller or the software that performs call processing. This has a number of implications for building switching systems with high availability. First, the control interface to the MON allows the switch controller to quickly recover connection state information stored in the translation memory. If the switch controller crashes, connections that were in a transient state may be lost, but connections that were in a stable state are still set up through the switch and can be recovered by the controller by sending a `warmboot` command to the header translation card. The MON responds with the switch state information. Second, the MON audits commands that it receives from the controller and audits the translation memory itself for consistency. The MON rejects invalid commands and returns an error indication. In addition, since the translation memory is audited, the switch state can be cleaned up even if the switch controller or one of its software components fails and the switch memory is left in an inconsistent state. The controller can poll the MON for a list of translation memory inconsistencies by sending an `audit` command. The card might respond, say, with a message indicating that an active endpoint is mapped to an idle endpoint, which should be a transient condition.

3.2.2. Maintenance and Administration

The switch and switch control software include a subsystem that detects and isolates faults in switch components, and that supports switch administration. Commands from the switch controller to the header translation card control and collect status from the switch. Our approach to status monitoring is based on three key ideas. Status is *continuously monitored* and filtered by an embedded maintenance processor on each switch element. Status is *collected periodically* by the switch controller, rather than having failures trigger messages. This avoids the possibility of congesting the maintenance subsystem at a time when its proper operation is crucial. Finally, the switch contains a *separate maintenance network*, connected to each of of the maintenance processors, for monitoring and control. This network allows switch elements to be controlled even if their interface to the cell data paths has failed.

On line interfaces, status such as line errors, loss of signal, loss of clock, coding violations and transmission errors are monitored. Data paths within the switch are monitored by including a parity bit with each byte of cell data. When a cell arrives from a transmission line, the line interface card verifies the header error check (and discards it) and generates the parity bit. The cell and parity information flow through the queue handler to the header translation card, where the parity is checked. If a parity error is detected, the cell is dropped and an error is registered. The parity is used to monitor the integrity of the path through the line card, queue handler, backplane and part of the header translation card. By correlating errors with the queue handler that generated them, fault isolation is possible. The data path from the header translation card through the queue handler to the line card is similarly checked. In our experience, parity provides a good indication of the integrity of board level electronics and a reasonably good check on memories.

Each maintenance processor supports configuration commands which remove the card from service or restore the card to service. The switch controller polls the status of every card in the switch periodically by issuing a poll command. Each card responds with a status message that includes error status, hardware type and version information, a serial number and card-specific data. This information can be used to support automatic configuration. Card-specific commands are used for controlling loopbacks, bit-error-rate testing, etc.

3.3. Queue Handler Architecture

A key goal of the queue handler design was to demonstrate per-virtual-circuit queueing at high speeds. We also tried to provide flexibility since we hoped to experiment with different congestion control schemes. Each virtual circuit has a *service class* associated with it that determines its treatment by the scheduler. In addition to ordinary round robin, the scheduler supports several levels of priority, so that virtual circuits in one service class can be given higher priority than those in another service class. The queue handler also supports two variations of round robin scheduling, weighted round robin (WRR) and framed round robin (FRR), which we describe below.

The queue handler implements a buffer management policy which allows the switch controller to limit the length of each per-virtual-circuit queue. Cell arrivals which would exceed this length limit are discarded. This buffer management policy supports the dynamic window flow control scheme described in Section 2. The queue handler also includes specialized hardware for manipulating an eight-bit congestion field that we reserved in our ATM cell header. This feature was motivated by the DECbit scheme [RAMA88], in which routers set a congestion bit to indicate congestion to sources.

A schematic of the queue handler is shown in Figure 6. The card contains a large DRAM array which supports 64 K virtual queues implemented by a high-speed queue control state machine. Each virtual circuit has a receive queue for cells that have arrived from a line card and a transmit queue for cells that are to be sent to a line card. The virtual queues are implemented using linked lists of cells. The queue control logic allocates buffer space dynamically when a cell arrives and frees the buffer space when a cell is transmitted. The "discard longest queue" buffer management policy of Section 2 can be implemented by updating a pointer to the current longest queue during every cell arrival and departure. The operation of the queue control logic is affected by the *queue control table*. The contents of this table can be changed by an onboard processor, known as the *resource manager*, which communicates with the switch controller as described in Section 4.

The DRAM array is multiported, with input and output ports for the line interface card, broadcast and contention busses, and resource manager. Data flows to and from the ports through nine bit-sliced chips that implement a pipeline buffer, shown in the figure. The buffer provides a pair of asynchronous shift registers for each port; each shift register holds one ATM cell. The pair of shift registers allows read and write operations on the DRAM array to be pipelined with cell transfers from the associated port. For example, a cell arriving on the contention bus is shifted into a shift register and a request is posted to the queue control logic for the cell to be written into the DRAM array. The write operation can take place while a second cell is shifted in from the contention bus. In order to support sufficient memory bandwidth, the DRAM array is 252 bits wide and is organized with two half-cells in adjacent locations. An operation to read or write a complete cell is done as two page-mode writes of the DRAM. This takes 280 nanoseconds, which gives an aggregate queue memory bandwidth for 384 bit ATM cell payloads of 1.3 Gbps.

The queue control table has an entry for each virtual queue. Each entry contains pointers to the head and tail of each virtual queue and a 4 bit service class. The table also contains a cell count and a count of the number of AAL5 frames for each virtual queue. The queue control machine contains logic to modify the congestion field in the cell header. This logic overwrites the congestion field in cells that are read out of the queue with a value that is an function of the received congestion field in the cell and a table entry which can be individually set for each virtual circuit by the resource manager.

Figure 6. Queue handler architecture.

The queue handler supports round robin scheduling at multiple priorities in order to support multiple ATM service classes, as shown in Figure 7. In the traffic management architecture in Section 2, CBR traffic is carried at a higher priority than ABR traffic. For each VC at the highest priority with data waiting to be transmitted, the scheduler serves one cell and moves on to the next VC. If no VC at the highest priority has data to send, the scheduler serves VCs at the second highest priority, returning at the end of each cell service time to the highest priority VCs if necessary, and so on. The scheduler supports 16 priorities. The scheduler is implemented using a *control queue* for each level of priority. A control queue contains a list of VCIs with data waiting to be transmitted. The scheduler removes a VCI from the head of the control queue when it is served and returns it to the end of the queue if there is still data waiting to be transmitted.

The scheduler also supports Weighted Round Robin (WRR) and Framed Round Robin (FRR) scheduling. In WRR, each VC conceptually has a weight or *service quantum* associated with it.[7] WRR allows different virtual circuits to receive different proportions of the available bandwidth. When a VCI is removed from the control queue to be served, the VC is eligible to be served for a number of cells equal to the service quantum, although it may be served fewer times if fewer cells are waiting. WRR is equivalent to a scheduling discipline known as Rate-Proportional Processor Scheduling [PARE94] in which the weight given to a VC is proportional to the fraction of link capacity to which it is entitled. FRR is a simplified version of HRR scheduling which, like HRR, insures that no more than a given number of cells are transmitted during an interval known as the *frame time*.

The scheduler provides optional support for the

[7] In the implementation, a weight is associated with each service class rather than each VC, which limits the number of weights that can be used simultaneously.

Figure 7. Priority round robin scheduler.

reassembly of AAL5 frames in order to simplify the design of host adaptors. The queue control machine increments a counter on the arrival of a cell header with a payload type indicating that it is an end of frame. Similarly, it decrements the counter on the cell departure. When frame reassembly is enabled, a queue is only eligible for service when the counter is nonzero.

3.4. Lessons

The Xunet switch has been in service since 1991. By and large, it has met its original goals of providing a flexible, reliable environment for experiments. At the time the switch design began, the ATM standards activity in ITU was only beginning and our architecture required design decisions that would only later be addressed by standards. While details of ATM have changed, it is rewarding that many of the issues we faced and the design decisions we made remain relevant.

We chose to control the switch with an external switch controller rather than integrating the controller into the switching fabric in order to upgrade the controller more easily. However, using an Ethernet for the control port limited the call setup rate that we could accommodate. More typically, the switch controller would use a port on the switch for signaling virtual circuits.

The use of an abstract interface for switch fabric control promotes modular software design. Moreover, our design is intended to support both high reliability and high availability through the built-in auditing mechanisms and the ability for call processing software to recover state from the switch. Switch maintenance is based on continuous status monitoring, periodic status collection, and the use of a separate maintenance network within the switch. This approach to maintenance has allowed us to detect most network problems at their onset, before they affect service.

Perhaps the main contribution of the Xunet switch design is in the implementation of the queue handler. The design demonstrates the feasibility of per-virtual-circuit queueing and a priority round robin scheduler at 1-4 Gbps rates in ATM switches. The use of discrete DRAM chips and bit-sliced buffer chips on the queue handler was suitable for a prototype, but clearly too expensive for commercial use. Further work in our group has demonstrated that these ideas can be made cost effective. In 1992, Kanakia proposed Yswitch, which integrated a packet buffer, switch control logic, and multiport serial access memories onto a single VLSI chip. The concept has been developed further in an experimental chip, the ATM Datapath chip, which integrates a buffer of 8 K cells and 16 serial access memories into a single "switch on a chip" controlled by external queue control logic. The ATM Datapath chip supports a wide range of ATM switching applications, from switch line cards to the construction of multistage switches. Since the chip is based on DRAM, it can ride the technology curve, and soon a full round trip window of buffering per port could be provided for roughly the cost of a 16 Mb DRAM chip. We are currently designing a single-chip queue controller for the ATM Datapath, building on the ideas developed in the Xunet queue handler. With the ATM Datapath, this chip will enable very low cost ATM switches to be built.

4. Network Control and Management

4.1. Goals and History

The Xunet control software was designed to support connection control and remote switch maintenance and administration. We designed a simple signaling protocol to establish, maintain, and clear ATM connections. Connection control requires a broad range of associated services, such as name translation, routing, call admission, and resource management. Initially, we needed fairly modest functionality in each of these areas, but we wanted an organization that would allow the software to evolve. We expected areas such as routing and resource management to undergo substantial change over time. In the long term, we hoped to support remote network management from a network management station. These considerations suggested a modular architecture in which pieces could evolve independently.

The organization of software for call processing and switch control has been a focus of our research group for some time. TDK [MCMA81] provided switch control for the Datakit switch and pioneered the notion of a "process-per-line" in which a lightweight process is associated with each half-connection during its transient states. Archos [CAMP92] explored the use of an object-oriented operating system for Datakit switch control. Finally, Milan Jukl in our group developed an early prototype for the Xunet control software which aimed for high performance by structuring the call processing software in a single process executing an event-driven finite state machine. Background and administrative tasks were handled by other processes. Shared memory was used for communication between parts of the system.

Both Jukl’s control system and the system we describe below were designed to run on top of the Silicon Graphics IRIX operating system. We chose to use a commercial operating system to facilitate student research. In addition, with a commercial OS we could use commercial distributed systems software to explore the client-server approach to network control suggested by the ISO-ODP (Open Distributed Processing) community and in the TINA-C standards [TINA95].

Administration and maintenance of an experimental wide-area network pose unique challenges, particularly when hardware and software are still in flux. In addition, a goal of Xunet was to allow research students to use the network as a laboratory, perhaps running their own experimental software. Support for student research required us to develop usage policies and procedures and added an additional source of instability to the network. Since most of the Xunet expertise was in New Jersey, it was important that network administration could be done remotely and we developed practical procedures and simple software that we could count on when all else failed. In Section 4.3 we describe these pragmatic considerations.

4.2. Network Control Architecture

This section describes the software architecture of the Xunet switch controller. Most of the software is designed as a client-server system based on the ANSAware distributed systems environment [ANSA93]. ANSAware supports RPC and includes a run-time environment that runs on several commercial operating systems. It also supports *location transparency* so that servers can be run on any machine in the distributed system. Servers register service offers with a *trader*, and clients bind a service name to the address of a server by contacting the trader.

Software on the switch controller communicates with the embedded processors in the switch itself. This relationship is shown schematically in Figure 8, where the bottom half of the figure represents the switch hardware while the top half of the figure illustrates different modules, such as signaling, routing, etc. that run as separate processes in the switch controller. Most of the illustrated modules run on a single machine in our implementation, although location transparency would allow processes to be remote. For example, if performance requirements dictated the use of a separate machine for routing, that module could be accessed using cross-machine RPC. The trader, an RPC-based network manager or a virtual console are other examples of processes that might be remote.

Figure 8. Switch controller architecture.

Two hardware proxies and a fault management module are tightly coupled to the switch and contain library routines that support communication with the hardware. The queue handler proxy communicates with the resource manager on the queue handler card and presents an RPC interface to other parts of the control system. This proxy is used during connection establishment to set the service class and buffer size limit for a virtual circuit. Similarly, a header translation proxy supports an interface to the header translation card. This interface is invoked by signaling to modify the translation table during connection establishment and is invoked by administration software to restore cards to service during switch initialization. Both of these interfaces could also be used by a ‘‘virtual console’’ or network management system, as shown in the figure. The functionality in the proxies can also be extended, for example, one might wish to provide record locking to synchronize writes to the same record structure.

The fault management module polls each of the elements of switch hardware for status once each second, and does alarm filtering and logging. Alarm thresholds are typically based on a ‘‘k of n’’ rule: an alarm is set when k of the last n polls indicates an error condition, and cleared when j of the last m polls are error free. Determination of the appropriate threshold settings is often based on trial and error, and an OSI agent was developed for the switch which allowed the alarm thresholds to be changed by a remote manager [ANER93]. The fault management software logs status information on each switch and the log files are uploaded to New Jersey over the network each night. The status history for the past 15 minutes can be accessed at a virtual console.

The signaling module supports connection control of simplex virtual circuits. It communicates with a peer module on an adjacent switch or host over a permanent virtual circuit that is established during switch initialization. The signaling module imports services from most of the other modules in the controller. When it receives a connection setup message, it invokes an operation of the routing service to translate the destination address to an output port. It then consults resource management to see if the connection resources can be reserved at that output port, and forwards the connection setup request to the next hop over the signaling permanent virtual circuit. If the setup request is successful, the next hop will eventually allocate the virtual circuit identifier to be used and return this in a connection accept message. The signaling module then contacts the header translation proxy to write the translation table entry in the switch hardware and contacts the queue module proxy to commit the resources for the new virtual circuit. Finally, it forwards the positive response upstream toward the connection originator. Originally, the signaling module was only a client: it did not offer any services. Later, it was modified to allow an OSI agent to collect statistics such as connection blocking from the signaling module [ANER93], and a central network manager used the blocking statistics to control the capacity of virtual paths so as to minimize call blocking and bandwidth requirements [ANER95a].

We designed a simple resource management scheme for CBR bandwidth reservation and ABR static buffer allocation. During the hop-by-hop connection setup described above, a `Reserve` operation is invoked at each switch as a connection request proceeds towards the destination (forward pass). A switch or the destination can reduce the amount of the reservation. When the response returns (reverse pass), a `Commit` operation is invoked. Since the resources committed may be less than those originally reserved, the commit frees the excess. Finally, when the connection is cleared, a `Free` operation is invoked.

The basic routing module supports an operation which takes a destination address and returns the outgoing port number. Routing in Xunet is based on a static routing table, but routing was isolated in a separate module so that the module could evolve to support dynamic routing without any change to signaling.

A skeletal version of the architecture described above was developed and was largely functional in summer 1993. This version had a signaling module, a simple resource management module supporting virtual circuit allocation, a static routing module, a queue handler proxy and a queue handler boot server. The fault management module and virtual console were developed independently and did not make use of ANSAware. At this writing, Xunet supports switched virtual circuits for native ATM virtual circuits, while Xunet's IP service operates over a permanent virtual circuit mesh.

4.3. Practical Considerations

Operating an experimental network poses many pragmatic difficulties, initially because the network itself is changing and later as students begin to use the network. Hardware installation requires the support of on-site personnel, so we funded a system administrator at each site to be "on call" for one quarter of their time. However, this did not eliminate the need for remote maintenance and administration. With experimental hardware and software, we could not count on maintaining the network over its own links, so we connected a statistical multiplexor (stat mux) to a dial-up line at each site as a backup. Using the stat muxes, we can either connect to the consoles of the router, switch controller or header translation card or we can remotely reset them through a relay connected to their hardware reset button. The stat muxes provide a highly reliable but inconvenient interface, so much of the day-to-day administration and maintenance is done over the network. We started off by setting up a PVC mesh network that could be accessed through the header translation card. A daemon on each switch controller allowed us to remotely log in, ship files, etc. from any other switch controller over the PVC mesh. Since the IP service became operational, it has been used for day-to-day adminstration.

Network research is coordinated by allowing students to sign up for use of the network via electronic mail. During their time slot, students can change any of the software that runs the network. Typically, a student creates a private copy of the default network control software and modifies some part of it. To run the modified software, the student changes a single symbolic link in the file system on the router or switch controller and reboots the machine. Students are instructed to restore the default software to its original state when they have finished their experiments. Applications researchers also sign up for the network to insure that the network is running default software during their time slot.

4.4. Lessons

There is relatively little published work on software structure for switch controllers. We make a few observations based on our experience. Modularity is an aim of any software project, and the use of an RPC system for communication among elements of the network control system aided in building open, modular software. We believe that the architecture would have allowed us to evolve the software easily: to support different switches, to add a new signaling or routing protocol, to enhance network management capabilities, etc. This flexibility was demonstrated by the relative ease with which statistics gathering and virtual path management were added to the signaling module. Location transparency also proved to be of surprising utility: debugging a system of programs operating over multiple machines would have been intractable if ANSAware had not made it possible to write location-transparent debugging utilities.

The use of a commercial operating system for the controllers had distinct advantages. Students were able to run small experiments with a minimum of investment learning the system. Moreover, in running the network, we made frequent use of the general-purpose software available on the controllers. For example, we used remote login programs for remote debugging, used the file system for release management, and used electronic mail as a primitive mechanism for distributing log files.

There were some problems with the architecture. The heavy use of RPC requires a large number of context switches to establish a virtual circuit — a severe penalty when building a controller on top of a heavyweight operating system. A production call processing system requiring high throughput and low latency would need to be carefully tuned to minimize system call overhead. Slower time scale functions such as administration, maintenance, logging and boot service place different requirements on the operating system than call processing does, and an environment that supports both lightweight and heavyweight processes seems attractive.

While most routine network maintenance and administration can be done using the network itself, there will always be cases where a ‘‘back door’’ is needed to recover from failures. In Xunet, the back door is based on the telephone network and stat muxes. Our policies for managing use of the network are simple and reasonably effective. Nonetheless, there are rough edges. There is an inherent conflict between doing network research and simply using the network. Users are not accustomed to scheduling their network use and even friendly users lose interest quickly if the network fails to meet their expectations. In Xunet, the biggest cause of network down-time is that a student doing network research has failed to follow the instructions for restoring the default software to operation.

5. Xunet Router

5.1. Goals

The initial focus of our endsystem work was support for IP encapsulation over ATM virtual circuits for a high quality LAN interconnection service. The performance goal was an end-to-end throughput from a single router of at least 75% of a DS3 line in an otherwise unloaded network. We planned to have routers at the edge of the network participate in congestion control so that the wide-area network would offer a service with little or no congestion loss.

A second goal was efficient use of network resources. Routers would be interconnected using ABR virtual circuits and would benefit from statistical multiplexing within the network. The desire for efficiency also led us to develop a frame-oriented adaption layer similar to AAL5 that used transmission bandwidth more efficiently than the AAL4 standard of the time.

A third goal was flexibility for experimentation. The implementation uses high-performance Silicon Graphics Power Series workstations as routers, with a home-brew

ATM interface card interfacing to the VME bus. Using a workstation-based router provided the greatest amount of flexibility for experimentation and gave us, fortuitously, an environment in which we had hosts directly attached to an ATM network. We therefore got early experience with host computers that use both IP encapsulation and a native ATM protocol stack.

5.2. IP Service Architecture

Our implementation of an IP service on Xunet is similar to the *Classical Model* of IP over ATM [RFC1577]. Hosts or routers which attach to Xunet are part of a logical IP subnetwork. When an IP packet arrives from a local area network, the router forwards it over an ATM virtual circuit to the egress router, as illustrated by the dashed line in Figure 9. If no virtual circuit exists, signaling software sets one up. The path across the ATM network is considered a single IP hop.

Figure 9. IP service architecture.

IP packets can be multiplexed over virtual circuits in a number of different ways, with some differences in end-to-end performance. For example, all packets from a router to a given egress router might use the same virtual circuit. Another possibility is that packets for a given destination host or source-destination host pair might use the same virtual circuit. Multiplexing different traffic types and different end-to-end sessions on a virtual circuit means that a packet that expects low delay, such as a `telnet` session, may suffer queueing delays behind a large block because the round robin scheduler is not scheduling individual sessions. These issues were studied in conjunction with Xunet in [CACE92, SARA94A].

In order to provide a good quality LAN interconnection service, end systems and/or routers must avoid inducing congestion on the wide area network. Using simulation, we explored end-to-end performance when TCP's end-to-end congestion control scheme (TCP-Tahoe) is used in conjunction with an "edge-to-edge" congestion control scheme between routers. The router-based flow control approach, based on the dynamic window flow control scheme of Section 2 [HAHN93], insures that no packets are lost in the wide-area network due to congestion. As a result, packet losses only occur in the router or local area network. Losses can occur at the input router, but our work showed that the throughput is high with appropriate choice of parameters. Recent work on TCP performance in ATM networks has suggested Early Packet Discard policies [ROMA94, TURN96] which discard an entire TCP packet if a queue length threshold is exceeded, indicating congestion. However, this work relies only on end-to-end flow control and does not explore use of congestion control between routers at the edges of the ATM network.

When we began work on Xunet, AAL4 had been proposed for data service on ATM and we investigated the efficiency of IP encapsulation using this adaption layer. Cáceres [CACE92] collected packet traces from our Internet gateway and we used the histogram of packet sizes to estimate efficiency. In some cases, IP encapsulation in AAL4 resulted in link utilizations as low as 65%. As a result of this study, we defined a payload type in the ATM cell header to mark the end of an IP packet and designed a frame-oriented adaption layer called AALX [KALM92] that eliminated the per-cell overhead of AAL4. Eliminating the per-cell overhead improved the efficiency for the same mix of packets from 65% to 85%, since many small packets would now fit in two cells rather than three. Both of these ideas have since been incorporated in standards.

In developing our architecture, we planned to develop a route server in the backbone to collect routing information from the routers and to control routing policy by determining what routing information to distribute. The ATM ARP server in the classical model would have been implemented by the route server. However, administrative issues prevented us from propagating routes among the sites and the routing tables are administrated statically rather than by a route server. Each site has an ‘‘experimental’’ FDDI ring that is attached to the Xunet router at that site. The routers run the `gated` routing daemon which is configured to allow machines on any of the FDDI networks to communicate using the Xunet IP service.

5.3. ATM Adaptor

The design of an ATM host adaptor involves a set of tradeoffs that depend on the cost and performance targets, host architecture, operating system, and workload. ATM introduces two new problems to adaptor design: reassembly of small cells into packets on the receive side, and possibly the need for cell-level transmission scheduling on the transmit side. Packet reassembly can be done in the switch, in adaptor memory, or in host memory. Our design is unique in taking advantage of per-virtual-circuit queueing in the switch to reassemble packets. Queue handlers that communicate with routers have the reassembly option enabled as described in Section 3. The scheduler delays the service of a virtual circuit queue until a cell with an end-of-frame payload type has arrived. Once service is started, it serves the virtual circuit until the end-of-frame cell is reached. The adaptor card receives a string of cells from the same virtual circuit and uses AALX or AAL5 to verify that the packet is correct. This approach eliminates the need to manage multiple reassembly buffers on the host adaptor or the need to manage partial state for the AAL5 CRC computation.

The adaptor design [BERE93] is based on a RISC CPU, the AT&T Hobbit, acting as an intelligent DMA engine (Figure 10). Fiber transceivers operating at 200 Mbps interface to the link encoder/decoders which then feed a 16 KB receive fifo buffer and a 4 KB transmit fifo buffer. The fifos provide an elastic store to smooth traffic flow to and from host memory. The elastic store fifos appear as a memory-mapped register in Hobbit address space; transfers to and from host memory are done by the Hobbit under program control.

Generation and checking of the AAL5 CRC is performed by a pair of field programmable gate arrays which snoop on the Hobbit address and data bus. The Hobbit accesses cell headers and payload using a different address offset so that the CRC is only updated during the transfer of payload words. In the figure, the dashed line illustrates how a transfer to host memory updates the CRC check logic in passing.

Figure 10. Host adaptor.

Hobbit accesses to host memory operate as follows. The host initializes a receive and transmit ring buffer in a region of memory called the communication area. Each ring entry is a chain of pbufs, which contain a physical and virtual address pointer to a host mbuf, as well as an ownership field. The state of the ownership field can only be changed by the owner. Initially, all receive ring entries are owned by the Hobbit and all transmit ring entries are owned by the host. When data arrives from the network, the Hobbit copies the payload into host memory. When the packet end is reached, the Hobbit changes the buffer ownership to host, possibly posts a receive interrupt to the host, and moves on to the next entry in the receive ring. The host unlinks the mbuf chain from the ring, links in a fresh chain of buffers, and gives the entry back to the Hobbit for a new packet. The transmit side works similarly.

5.4. Protocol Stack

The protocol stack shown in Figure 11 includes a hardware driver, the orc[8] driver, to interface to the adaptor. Our software organization puts performance-critical code in the kernel, primarily the code that is called during packet handling, while code that is not performance critical is kept in user space. We partitioned the functionality between user space and kernel so that we could do experiments that involved signaling [ANER95a] or multiplexing policy without touching the kernel. The remainder of this section focuses on the kernel code.

Figure 11. Router kernel implementation.

Consider a packet which arrives on FDDI that is to be routed over Xunet. The FDDI driver appends the packet to the IP receive queue and issues a software interrupt to the IP input handler, which routes the packet and passes a pointer to the packet into the Xunet interface code if_xunet. This routine determines whether the packet should be forwarded over an existing virtual circit or if a virtual circuit needs to be established. The kernel maintains a cache mapping IP destination addresses to ATM virtual circuit identifiers in the address resolution module and if there is a cache hit, the packet is forwarded over that virtual circuit. If there is a cache miss, address resolution calls up into a user-level IP connection server which contacts user-level signaling software to set up a virtual circuit to the appropriate destination. Once the circuit is set up, the IP connection server loads the cache with the mapping from IP packet header information to virtual circuit. The address resolution module also maintains an activity timer for each virtual circuit. When a connection has been inactive for one second, it calls up to the IP connection server. The IP connection server can choose whether to clear the connection [SARA94A], in keeping with the idea that policy decisions are kept in user space.

The interface presented by the orc driver to higher layers of the protocol stack does not depend on IP and Figure 11 shows a native ATM protocol stack. The orc driver does protocol demultiplexing using the virtual circuit identifier when receiving from the network. Application programs can communicate using virtual circuits directly via a slight modification to the socket library [AHU96].

5.5. Lessons

The Xunet router has been in service since April 1993. The simple architectural model that we used for IP over ATM service, in which routers attached to the ATM network are part of a single logical IP subnet, was independently developed in the IETF and is now quite widely deployed. As for our implementation, the Xunet router achieved respectable performance for the time. User-to-user performance between a pair of FDDI hosts running the ttcp benchmark over Xunet through the Xunet router was measured at 35 Mbps. Moreover, the router and architecture gave rise to a number of research projects.

Implementation of packet reassembly in an ATM switch simplifies the work of an adaptor card, although with VLSI support for segmentation and reassembly, this may be of diminishing importance. Another issue is that reassembly increases the delay variance for CBR streams by reducing the number of scheduler preemption points. However, reassembly remains relevant to the design of ATM hubs interfacing to conventional local area networks. In an ATM hub, packet reassembly is typically implemented on a line card, but reuse of the switch buffer itself for packet reassembly would provide an significant cost saving.

With or without switch-based packet reassembly, it is clear that RISC processors are capable of performing ATM reassembly at quite respectable speeds. With hardware assist for the AAL5 CRC, ATM packet reassembly takes less than 100 RISC instructions. On a 60 MIPS machine, this is approximately 1.6 microseconds, whereas a cell is transmitted every 2.7 microseconds at 155 Mbps. Even if VLSI segmentation and reassembly engines dominate for high-performance host adaptors, there is a role for a processor-based approach to segmentation and reassembly at the low end and in embedded controllers where a single processor can support the network and application functions. We recently applied our understanding to the development of Euphony [ONU96], a MIPS R3000-based processor with support for ATM and digital signaling processing.

Our work on congestion avoidance explored whether the performance of TCP could be improved by using flow control

[8] Orcs are a warlike people in J. R. R. Tolkein's *The Hobbit.*

between routers at the edges of the ATM network. This model differs from the conventional model of an IP router, which simply forwards packets. The interaction between TCP flow control and a separate flow control protocol between ATM-attached routers, such as the ATM Forum ABR protocol, remains an interesting research topic suggested by our work.

The impact of the congestion control scheme on host adaptor design is also important. It is commonly expected that CBR service will require hosts to implement a fine-grained cell transmission schedule, and adaptors can inexpensively implement a transmission schedule for this service using a variation of the HRR scheduler described in [KALM90]. In our view, however, an ABR service should be designed to allow a range of options on host adaptors. The simplest and potentially most cost effective approach would give the host responsibility for transmission scheduling of one KByte or larger "transmission blocks." While Packet-Pair flow control allows this, the rate-based proposals [BONO95, FEND96] in the ATM Forum require a more complex and more expensive host adaptor with finer-grained scheduling and algorithms to modify the schedule based on feedback or an explicit rate allocation.

The software organization on the router seems as valid today as when it was done. Signaling and policy code should be in user space. This differs from commercial implementations today, but should be the direction for the future.

6. Conclusions

Since the Xunet program began, interest in ATM has experienced dramatic growth. In 1991, the ATM Forum was founded by four companies interested in developing ATM products for use in the local area. At this writing, the ATM Forum has more than 700 member companies. There are solid technical reasons behind this enthusiasm. ATM offers a common approach to multiplexing and switching that can operate over a wide performance range. Moreover, ATM has the potential to offer good performance to a broad class of applications since switches maintain connection state and can offer a quality of service appropriate to individual connections. There is presently commercial interest in using ATM for both facility switching as well as statistical multiplexing in the wide area, as a switch-based local area networking technology, and as a distribution technology for local access. In addition to these commercial applications, there is fertile research exploring the use of ATM for wireless communication, in desk area networks and as the basis for networks in the home. In light of all this activity, it is often hard to keep sight of fundamental principles. This paper has attempted to glean some principles from the work we’ve done over the years.

In traffic management, there is clearly a need for a CBR service in ATM to support circuit emulation, voice and video traffic. The principles here are well established and involve a user traffic descriptor, call admission, priority at a switch scheduler, and user policing. Recent efforts in the ATM Forum to define the ABR and UBR service classes recognize the desire by computer users to get "as much as possible" from the network, while also recognizing the need to avoid congestion. We are concerned that explicit rate allocation complicates switches and that fine grain transmission scheduling complicates host adaptors. In our view, round robin scheduling for ABR traffic achieves a fair sharing of bandwidth among active users and an appropriate buffer management policy protects one user from another. Congestion avoidance is a matter of controlling the offered load using an end-system rate adaption scheme such as Packet-Pair. However, even if explicit rate adaption is used, round robin scheduling will still be important as a means of providing protection among users.

In the past, per-virtual-circuit queueing and round-trip sized buffers have been considered to be prohibitively expensive. It now appears that this approach may be commercially viable, at least for 155 Mbps lines and for switches in the 1-4 Gbps range, through the use of large scale integrated circuits incorporating memory and logic. This will go a long way towards providing a good quality ABR service.

We believe that the simple output-queued architecture of the Xunet switch remains appropriate to the design of low-cost ATM multiplexers and switches. Regardless of the switch architecture, a well-designed maintenance subsystem can make even inexpensive switches easy to maintain. The data paths in a switch can be error checked and continuously monitored, and the maintenance subsystem can support automatic configuration and hardware fault detection. Switches that are designed to provide high availability can use embedded processors to do audits of internal tables.

The ATM community has only begun to scratch the surface of network control and management. Our work suggests the possibility of an open, modular call processing architecture. Such an architecture might allow a single software system to control different vendors’ switches and might allow protocols like signaling and routing to evolve independently. Whether such an architecture is used or not, it is clear that building switch controllers with high throughput and reliability will require a serious committment to software architecture work. The standards for end-system signaling and inter-switch routing have evolved to be much more complex than the simple protocols used in Xunet. It is clear that such complex protocols are not needed everywhere. For example, a desk area or home network might use a lightweight signaling protocol, with a concise encoding and good support for the "common case," along the lines of what we did in Xunet.

Although there has been a great deal of work on support for IP traffic on ATM local area networks, a number of issues remain. For example, it should be possible for IP routers connected over ATM networks to make use of ABR flow control to avoid packet losses. This would make IP service over ATM networks more predictable than in the current Internet. Currently, hosts with ATM support typically run an IP stack over ATM. This is necessary when communicating with IP-only hosts, but the IP layer could be avoided when both hosts are attached to ATM. Hosts can easily support a dual stack, including both IP and a native ATM stack that allows applications to take advantage of ATM quality-of-service guarantees. The ATM stack would also improve performance by eliminating the redundant IP network layer processing. However, there is significant inertia to overcome in bringing this about.

Much has been written about host adaptor design issues in the last few years. It is difficult to make generalizations since the solutions depend strongly on the specific system and goals. We do expect that embedded processors will be used for ATM protocol processing in a wide range of applications. At the low end, embedded processors can handle ATM segmentation and reassembly directly with a small amount of hardware support for the AAL5 CRC.

Acknowledgments

Xunet would not have been possible without the efforts of many people. A. G. Fraser and R. L. Snowden deserve the credit for creating the Xunet program. Fraser played a major role in setting the technical direction and managing the project. A talented group of hardware designers, headed by J. H. Carran and J. A. Grandle at AT&T's Columbus Works designed and built the Xunet switches, transforming our changing requirements into working hardware. P. E. Parseghian deployed the network and managed its operations: without her, it is hard to imagine how Xunet would have come into being. M. J. Dixon proposed the network control architecture and wrote much of the code. A. D. Berenbaum and A. Iyengar were responsible for our ATM host adaptor and device driver. A. E. Kaplan worked on our DS3 and 622 Mbps line cards. J. H. Condon consulted on hardware issues. R. Sethi, E. K. Grimmelmann and G. S. Subramanian provided funding and moral support as well as running interference for us.

Xunet would also not have been possible without the active participation of our friends at the universities and research labs in the Xunet student program. We came together at the annual Student Meeting: a technical conference consisting of papers presented by students to their peers. We are deeply indebted to this large extended community for many, many fun and productive hours.

Epilogue

The Xunet 2 network was officially decommissioned in February 1996, after this paper was submitted for review. We left the paper in the present tense as was appropriate at the time of writing.

References

[AHU96] R. Ahuja, S. Keshav and H. Saran, "Design, Implementation, and Performance of a Native-mode ATM Transport Protocol," Proc. IEEE INFOCOM'96, March 1996, pp. 206-214.

[ANER93] N. G. Aneroussis, C. R. Kalmanek, V. E. Kelly, "Implementing OSI Management Facilities on the Xunet ATM Testbed", Proc. 4th IFIP/IEEE Workshop on Distributed Systems: Operations and Management, October 1993.

[ANER95a] N. G. Aneroussis and A. A. Lazar, "Managing Virtual Paths on Xunet III: Architecture, Experimental Platform, and Performance," Proc. IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, May 1995.

[ANE95b] N. G. Aneroussis, A. A. Lazar and D. E. Pendarakis, "Taming Xunet III," ACM Computer Communications Review, Vol. 25, No. 3, pp. 44-65, October 1995.

[ANSA93] "ANSAware 4.1: Application Programming in ANSAware," Architecture Projects Management Limited, Poseidon House, Castle Park, CB3 0RD, Cambridge UK, February 1993.

[BERE93] A. Berenbaum, J. Dixon, A. Iyengar, and S. Keshav, "A Flexible ATM Host-Interface for Xunet II," IEEE Network, Vol. 7, No. 4, July 1993, pp. 18-23.

[BONO95] F. Bonomi and K. W. Fendick, "The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service," IEEE Network, Vol. 9, No. 2, March-April 1995, pp. 35-39.

[CACE92] R. Cáceres, "Multiplexing Traffic at the Entrance to Wide-Area Networks," PhD thesis, U. C. Berkeley, December 1992 (Report No. UCB/CSD 92/717).

[CAMP92] R. H. Campbell et al., "Control Software for Virtual-Circuit Switches: Call Processing", Future Tendencies in Computer Science, Control and Applied Mathematics, Lecture Notes in Computer Science 653, Springer-Verlag, Berlin 1992, pp. 175-186.

[CNRI95] On-line reference to the Gigabit Testbed Initiative at http://www.cnri.reston.va.us/

[DEME89] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," Proc. ACM SIGCOMM, September 1989, pp. 1-12; also Journal of Internetworking Research and Experience, Vol. 1, No. 1, September 1990, pp. 3-26.

[FEND96] K. W. Fendick, "Evolution of Controls for the Available Bit Rate Service," IEEE Communications, Vol. 34, No. 11, November 1996, pp. 35-39.

[FRAS83] A. G. Fraser, "Towards a Universal Data Transport System", IEEE Journal on Selected Areas in Communications, Vol. 1, No. 5, November 1983, pp. 803-816.

[GOLE91] S. J. Golestani, "Congestion-Free Communication in High-Speed Packet Networks," IEEE

Transactions on Communications, Vol. 39, No. 12, December 1991, pp. 1802-1812.

[GROS95] M. Grossglauser, S. Keshav, and D. Tse, "RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic," Proc. ACM SIGCOMM ’95, Boston, August 1995.

[HAHN91] E. L. Hahne, "Round-Robin Scheduling for Max-Min Fairness in Data Networks", IEEE Journal on Selected Areas in Communications, Vol. 9, No. 7, September 1991, pp. 1024-1039.

[HAHN93] E. L. Hahne, C. R. Kalmanek, and S. P. Morgan, "Dynamic Window Flow Control on a High-Speed, Wide-Area Data Network", Computer Networks and ISDN Systems, Vol. 26, No. 1, September 1993, pp. 29-41.

[KALM90] C. R. Kalmanek, H. Kanakia, and S. Keshav, "Rate Controlled Servers for Very High-Speed Networks," GLOBECOM ’90, San Diego, California, December 1990, pp. 12-20.

[KALM92] C. R. Kalmanek, B. Lyles, and W. T. Marshall, "Proposal for a Robust *SEAL* Protocol," Contribution to ANSI T1S1.5, Chicago, May 1992.

[KANA93] H. Kanakia, P. P. Mishra, and A. R. Reibman, "An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport," Computer Communication Review, Vol 23, No. 4, October 1993, pp 20-31.

[KATA88] M. G. H. Katavenis, "Fast Switching and Fair Control of Congested Flow in Broadband Networks," IEEE Journal on Selected Areas in Communications, Vol. SAC-5, October 1987, pp. 1315-1326.

[KESH91] S. Keshav, "A Control-Theoretic Approach to Flow Control," Proc. ACM SIGCOMM ’91, September 1991, pp. 3-15.

[KESH95] S. Keshav, "Packet-Pair Flow Control", submitted to ACM Transactions on Computer Systems.

[KESH97] S. Keshav and S. P. Morgan, "SMART Retransmission: Performance with Overload and Random Losses," Proc. IEEE INFOCOM’97, April 1997, to appear.

[LAKS97] T. V. Lakshman, P. P. Mishra and K. K. Ramakrishnan, "Transporting Compressed Video over ATM Networks with Explicit Rate Feedback Control," Proc. IEEE INFOCOM’97, April 1997, to appear.

[MCMA81] L. E. McMahon, "An Experimental Software Organization for a Laboratory Data Switch," Proc. ICC’81, IEEE International Conference on Communications, Vol. 2, 1981, pp. 25.4.1-25.4.4.

[MORG91] S. P. Morgan, "Queueing Disciplines and Passive Congestion Control in Byte-Stream Networks", IEEE Transactions on Communications, Vol. 39, No. 7, July 1991, pp. 1097-1106.

[NAGL86] J. B. Nagle, "On Packet Switches with Infinite Storage," IEEE Transactions on Communications, Vol. COM-35, No. 4, April 1987, pp. 435-438.

[OIE89] Y. Oie, M. Murata, K. Kubota and H. Miyahara, "Effect of Speedup in Nonblocking Packet Switches," IEEE International Conference on Communications, June 1989, pp. 410-414.

[ONU97] P. Onufryk, "Euphony: An Embedded RISC Processor for Low-Cost ATM Networking and Signal Processing," Proc. Supercon 97 - Digital Communications Design Conference, Santa Clara, CA, January 1997. Also see http://www.research.att.com/˜pzo/euphony.html.

[PARE94] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," IEEE/ACM Transactions on Networking, Vol. 2, No. 2, April 1994, pp. 137-150.

[RAMA88] K. K. Ramakrishnan and R. Jain, ‘‘A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer,’’ Proc. ACM SIGCOMM, August 1988, pp. 303-313.

[RATH93] E. P. Rathgeb, "Policing of Realistic VBR Video Traffic in an ATM Network," International Journal of Digital and Analog Communications Systems, Vol. 6, 1993, pp. 213-226.

[REIB92a] A. R. Reibman and A. W. Berger, "Traffic Descriptors for VBR Video Teleconferencing over ATM Networks," GLOBECOM ’92, Orlando, Florida, December 1992, pp. 314-319; also IEEE/ACM Transactions on Networking, Vol. 3, No. 3, June 1995, pp. 329-339.

[REIB92b] A. R. Reibman and B. G. Haskell, "Constraints on Variable Bit-Rate Video for ATM Networks," IEEE Transactions on Circuits and Systems for Video Technology, Vol 2, No. 4, December 1992, pp. 361-372.

[RFC1577] M. Laubach, "Classical IP and ARP over ATM," Internet Network Working Group Request for Comments #1577, January 1994.

[ROBE92] J. W. Roberts, editor, "Performance Evaluation and Design of Multiservice Networks," COST 224 Final Report, Commission of the European Communities, Brussels (1992), pp. 111-147.

[ROMA94] A. Romanov and S. Floyd, "Dynamics of TCP Traffic over ATM Networks," Proc. ACM SIGCOMM ’94, London, August 31 - September 2, 1994, pp. 79-88.

[SAFR95] R. Safranek, C. Kalmanek and R. Garg, "Methods for Matching Compressed Video to ATM Networks," Proc. IEEE Workshop on Information Theory, Multiple Access and Queueing Theory, St. Louis, MO, April 1995.

[SARA94A] H. Saran and S. Keshav, "An Empirical Evaluation of Virtual Circuit Holding Times in IP," Proc. IEEE INFOCOM’94, June 1994.

[SARA94B] H. Saran, S. Keshav and C. R. Kalmanek, "A Scheduling Discipline and Admission Control Policy for Xunet 2," ACM Multimedia Systems Journal, Vol. 2, No. 3, September 1994.

[TINA95] M. Chapman and S. Montesi, "Overall Concepts and Principles of TINA," Document Label TB_MDC.018_1.0_94, February 1995, available from http://www.tinac.com/.

[TURN96] J. S. Turner, "Maintaining High Throughput During Overload in ATM Switches," Proc. IEEE INFOCOM’96, March 1996, pp. 287-295.

[VERB89] W. Verbiest and L. Pinnoo, "A Variable Bit Rate Video Coder for Asynchronous Transfer Mode Networks," IEEE Journal on Selected Areas in Communications, Vol. 7, No. 5, June 1989, pp. 761-770.

[VERN88] M. K. Vernon and U. Manber, "Distributed Round-Robin and First-Come-First-Serve Protocols and Their Application to Multiprocessor Bus Arbitration," 15th IEEE International Symposium on Computer Architecture, May 30 - June 2, 1988, pp. 269-277.

[WEIN78] C. J. Weinstein, "Fractional Speech Loss and Talker Activity Model for TASI and for Packet-Switched Speech," IEEE Transactions on Communications, Vol. COM-26, No. 8, August 1978, pp. 1253-1257.