Title: AN2: A High-Performance ATM Switch

Authors: Charles P. Thacker and Michael D. Schroeder

Authors' affiliation:

Digital Equipment Corporation
Systems Research Center
130 Lytton Ave.
Palo Alto, CA 94301
thacker@src.dec.com
mds@ src.dec.com

This paper is a draft, submitted for publication in the Digital Technical Journal. It will also appear as a Systems Research Center Research Report.

AN2: A High-Performance ATM Switch

Charles P. Thacker
Michael D. Schroeder
Digital Equipment Corporation
Systems Research Center
Palo Alto, California
March 26, 1995

Abstract

AN2 is a high-performance Asynchronous Transfer Mode (ATM) switch developed at Digital's Systems Research Center. AN2 provides sixteen 800 Mbit/sec. ports, each of which can contain a 4-port 155 Mbit/sec (OC-3) line card, or a single 622 Mbit/sec. (OC-12) line card. AN2 supports both constant bit rate (CBR) and available bit rate (ABR) traffic. It uses a unique hop-by-hop flow control protocol that eliminates cell loss, even in the presence of congestion. Software in the switch provides circuit setup functions, automatic network reconfiguration in the event of hardware failures, and *resilient virtual circuits*, which provide automatically-established connections in a local area ATM network of modest size. AN2 is the prototype for Digital's ATM switch products.

In this paper, we discuss the implementation of the hardware and software of AN2.

1. Introduction

AN2 is an experimental ATM switch developed at Digital's Systems Research Center in Palo Alto, California. An AN2 switch consists of a sixteen-port, synchronous crossbar and up to sixteen *line cards*. Each port can hold either a four-line OC3 (155 Mbit/sec) line card or a one-line OC12 (622 Mbit/sec) card. The bandwidth of the switch is 800 Mb/sec per port, or an aggregate of 12.8 Gbits/sec. Each line card contains an *input unit*, which is responsible for receiving and buffering ATM cells and forwarding them through the crossbar when appropriate, an *output unit* which contains a small amount of FIFO buffering to match the rate of the crossbar to that of the output lines, and a *line control processor* (LCP) which manages the input and output units. Figure 1 shows the data paths of the AN2 switch.

AN2 uses per-virtual-circuit random access buffering in the input units to avoid the head-of-line blocking usually found in fifo-buffered switches. Each port supports up to 32768 (2^{15}) virtual circuits, each of which may be configured for either constant bit rate (CBR), or available bit rate (ABR) service. CBR cells flow through the switch under control of a TDM schedule. Efficient CBR multicast is provided to support services, such as video, which require guaranteed bandwidth and low jitter. ABR virtual circuits compete for the switch capacity not used by CBR traffic, using a distributed arbitration algorithm that ensures high (typically greater than 95%) utilization of the switching fabric. The AN2 switch provides no support for VPs (virtual paths).

ABR virtual circuits may optionally make use of a hop-by-hop, credit-based flow control mechanism that eliminates cell loss under congested conditions. When this mechanism is used, a connected switch (or host adapter) maintains a credit balance for each flow-controlled virtual circuit. Each credit corresponds to an available cell buffer in the downstream switch or host. As each cell is sent, the source decrements its credit balance. If the balance becomes zero, the virtual circuit is *stopped* (i.e., not permitted to send cells). Each time the destination frees a buffer by forwarding a cell, it sends a credit upstream to the source, which increments its credit balance, restarting the VC if it was stopped. The number of buffers (and therefore credits) needed to maintain full rate in the absence of congestion is proportional to the rate of the VC and the round-trip time between the source and the destination. In a local-area network with relatively short links, a few buffers per VC suffice, and non-congested VCs are never stopped.

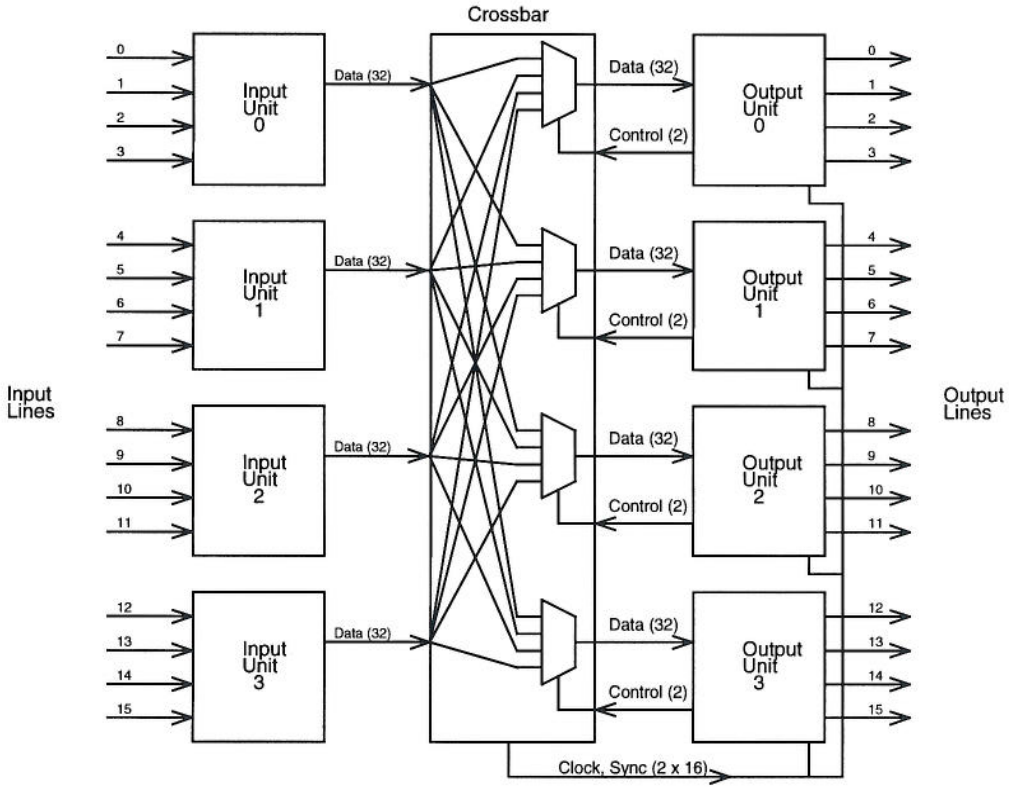

Figure 1: AN2 Data Path
For simplicity, a 4-port switch is shown, although the AN2 switch actually has 16 ports. Input and output line i ($0 <= i < 16$) form a full-duplex connection to another switch or host.

Software running on the LCP in each line card has a number of responsibilities. It establishes virtual circuits by setting up state information in tables used by the input and output units, monitors the integrity of each link, and participates in a network-wide reconfiguration protocol that reroutes virtual circuits in the event of failure. The LCP can send and receive ATM cells through the switch to exchange information with other line cards or network nodes. Each line card also includes a connection to a backplane Ethernet, which provides inter-card communication as well as a connection to an optional external workstation, which may be used for management or debugging. With the exception of ABR multicast traffic, which is forwarded to multiple destinations by the LCP, the LCP has no direct role in forwarding ATM cells.

The remainder of this paper discusses the hardware and software components of the AN2 switch. Section 2 describes the hardware, including the structure and operation of the crossbar, the input and output units, and the LCP. Section 3 describes the software, and Section 4 contains concluding remarks.

2. AN2 Hardware

2.1 Crossbar and Backplane

AN2 line cards plug into a sixteen-slot backplane. The crossbar is housed on three modules which plug into the back side of the backplane, at right angles to the line cards. Each crossbar module is a 12-bit slice of the 16 x 16 switch fabric (there are four unused bits on one module). One of the crossbar modules

generates a 25 MHz system clock which is distributed radially to the sixteen line cards and the three crossbar cards. A second crossbar module generates a *sync* signal, which serves to synchronize the line cards. Sync occurs every 13 clocks, and delineates a *slot*, which is the time required to send a single ATM cell through the switch (a cell consists of one 32-bit word containing the VC and the CLP and PT bits for the cell, plus twelve words containing the 48-byte cell payload). Within the line cards, activities take place at specific times within a slot, whose clock periods are called T0..T12. Every 1024 slots, a two-clock wide sync signal defines the start of a CBR scheduling *frame*. Each input unit contains a scheduling table, indexed by the slot number within a frame, which holds the *virtual circuit identifier* (VCI) of the (CBR) VC that is allowed to send a cell in that slot (schedule entries not allocated to CBR traffic contain zero, which is an illegal VCI). Slots not allocated to CBR traffic are available for ABR traffic.

Each slot, every output unit sends to the crossbar the four-bit number of the input port from which it will receive a cell during the following slot. This information is time-multiplexed on the two control signals shown in Figure 1. At the end of each slot, the crossbar modules reconfigure their multiplexers based on this information. Arbitration for the crossbar is done by a distributed algorithm called *Parallel Iterative Matching* [1,2] that involves both the input and output units. This algorithm requires that each input and output unit exchange *request* and *grant* signals. Each line card has thirty-two dedicated backplane signals for this purpose (a request and a grant from each input unit to every output unit), as shown in Figure 2.

In addition to the request and grant signals, the output units must communicate flow-control information to the input units. Each output unit has a two-bit path to every input unit for this purpose (the *start/stop* signals shown in Figure 2).

The crossbar is implemented with Xilinx field-programmable gate arrays (FPGAs) [3]. These chips must be configured before use. The four bussed control signals shown in figure 2 provide this information during switch initialization. These signals are also used during initialization to conduct a protocol which elects one of the LCPs as the *master*. The master LCP is responsible for configuring the crossbar FPGAs, and for carrying out operations that apply to the switch as a whole (e.g., synchronization between distributed processes running in multiple line cards).

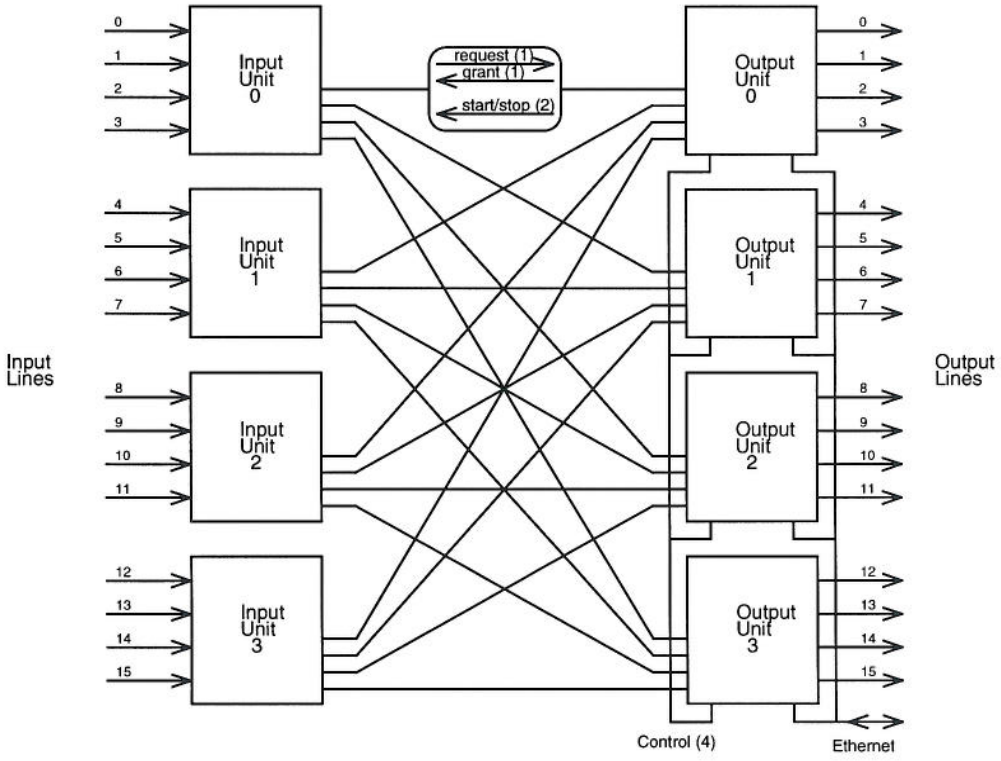

Figure 2: AN2 inter-card backplane control signals (4 ports shown).
Most signals are point-to-point with the exception of four control signals and the Ethernet.

Since the number of inter-card signals (and the complexity of the crossbar) grows as the square of the number of switch ports, connector pin density limitations probably preclude construction of AN2 switches with more than perhaps 64 ports. The switch ports can be several times faster than the links, so our approach of multiplexing several links onto one switch port can yield switches of substantial size. For switches with a few hundred lines or less, a crossbar provides a simple non-blocking fabric. Since scheduling a crossbar to achieve high throughput is straightforward, AN2 provides an economical, high-performance local area network.

2.2 Line Cards

Each AN2 line card consists of an input unit, an output unit, and a line control processor (LCP). The input and output units are logically independent. There are currently two types of line card. The quad line card, or QLC, provides four full-duplex OC-3 SONET ATM links. The second card was designed at the University of Kansas for the Magic Gigabit Networking Testbed. It provides a single OC12 SONET link (622 Mbits/sec.), and has a somewhat different flow control method more suitable to the wide area. This card is described in [4].

The line card is actually a two-board stack. One board handles the input and output interfaces to the physical medium, and the logic needed to encapsulate and decapsulate ATM cells into a SONET stream. It also contains a limited amount of FIFO buffering to rate-match the input and output lines to the 800 Mbit/sec. bandwidth of the buffers and the crossbar. The main board contains the data buffering for incoming cells, the connections to the crossbar, and the logic to control the transmission of cells. The QLC and the OC12 boards use the same main board, but different line boards. Since a well-defined logical, electrical, and physical interface exists for the line board, other types of physical media could be supplied relatively easily.

2.2.1 QLC Input Unit

The input unit of the QLC consists of a data path and a control section. Figure 3 shows the data path. Each of the four lines terminate in electro-optic receivers and clock recovery circuits operating at 155 Mbits/sec. The serial SONET data streams are processed into a stream of ATM cells by four PMC SUNI chips [5]. These chips provide a fifo-buffered sixteen-bit wide ATM cell stream which may be taken from the chip at up to 400 Mbits/second. The outputs from a pair of chips are time-multiplexed onto two sixteen-bit busses, which pass through the Input FPGA. This chip contains a *quota counter* for each line. The quota counters enforce a limit on the number of cells a line may consume in the main cell buffer. When a cell is passed to the buffer, the quota counter is decremented. When the input unit forwards a cell that originated at a particular input line (or when the input logic discards a cell), the quota counter for the line is incremented. If a line exceeds its quota (i.e., its quota counter becomes zero), subsequent cells from that line are dropped. Note that for CBR and flow-controlled ABR traffic, the quota mechanism is unnecessary, since buffer utilization is controlled by the credits given to each ABR VC on the line, and by the CBR schedule. The quota counters are provided so that switches or hosts that do not use flow control cannot unfairly consume buffers allocated to other lines.

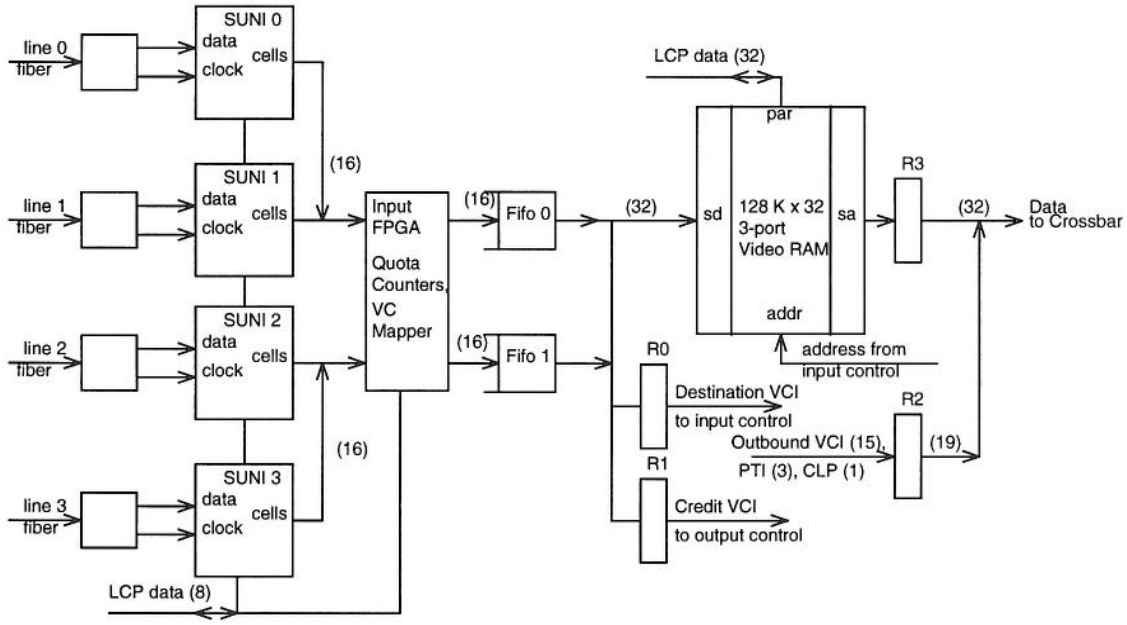

Figure 3: Input unit data path.

The input FPGA also contains logic to modify the VCIs contained in incoming cells. Since the four input lines will use identical virtual circuit identifiers for flows which are logically distinct, the VCIs must be modified before cells from different input lines are commingled. In the ATM header, a VCI can be up to fourteen bits in length (if the VC is flow-controlled, a separate fourteen bit field is used to carry a flow-control credit). Within the switch, VCIs are fifteen-bit quantities. The mapping logic in the input FPGA maps the high-order three bits of the line VCI into a four-bit quantity that is passed to the rest of the switch. At the time the line card is initialized by the LCP, the mapping is established. It can provide one or more lines with a disproportionate share of the VCI space if appropriate. The low-order eleven bits of the VCI are passed through the input FPGA without modification.

From the input FPGA, cells flow into two fifo buffers which match the 400 Mbit/sec rate of the SUNI chips to the 800 Mbit/second rate of the main data buffer. The main data buffer is implemented with triple-port video RAMs, which have two high-speed serial ports in addition to a standard DRAM parallel port used by the LCP. The buffer is implemented with four 128K by 8 VRAM chips (0.5 Megabytes total), and provides approximately ten thousand cell buffers. Each buffer holds only the forty-eight-byte cell payload; The PTI and CLP bits of the cell, which are not modified by the switch, are stored in a separate memory. The cell buffers are managed as a pool by the input control unit.

As a cell passes from the rate-matching fifos to the VRAM, the destination VCI and, if one is present, a flow-control credit VCI are removed from the cell and placed in R0 and R1. The destination VCI is passed to the input control logic, the credit is passed to the output unit (since the output unit maintains credit balances for the switch or host at the far end of the input link).

The assembly and buffering of arriving cells requires essentially no interaction with the input control unit. The input buffer can accept one cell every thirteen-clock slot. If a cell arrives, the input control unit is notified by local control logic in the input FPGA. It uses the destination VCI to do further processing of the cell, as described below.

The final section of the input data path is responsible for delivering cells to the crossbar at the correct time. This activity is not related to the cell arrival activity described above. When it is time to transmit the data contained in a particular VRAM buffer, the input control logic reads the cell into the VRAM serial port

(sa). The VCI which will be used on the cell's next hop (which is in general different from the VCI used when the cell arrived) and the CLP and PTI bits are placed in R2 by the input control logic. The VRAM read is precisely timed so that the cell payload, after passing through buffer register R3, will enter the crossbar on the correct clock cycle. When the correct time arrives, the VCI and PT are transmitted into the crossbar from R2, followed by the twelve words of cell payload from R3.

Input Control Unit

Processing of cells by the input unit involves data structures in three memories, as shown in Figure 4: The *queue ram* contains information about each of the 2^{15} VCs supported by each port, and is indexed by VCI. The *cell ram* contains one entry (a list pointer) for each of the cell buffers in the VRAM.

The four cell header bits that are not stored in the VRAM (the header's PT and CLP bits) are also kept in the cell RAM, as is the schedule table for CBR traffic. There are two CBR schedule tables. At any given time, only one of the tables is used for scheduling. The other is filled with a new schedule when a CBR VC is added to or removed from the schedule. When the new schedule is ready, the master LCP signals all line cards to begin using the new schedule at the start of the next frame. This synchronized switching of the schedules is needed to ensure that no more than one input unit will ever attempt to forward a CBR cell to a single output during a slot.

The 16-entry *port queues* array is kept in an FPGA. Port queue entry *i* consists of a list (i.e., head and tail pointers) of VCs with cells ready to be sent to output port *i*. Each entry also contains a 5-bit set representing the output lines needed by the VC at the head of the port queue (4 lines plus the LCP). The output units have a limited amount of buffering for each line. During each slot, the output units inform the input units whether there is room in these buffers for another cell (this information is time-multiplexed onto the backplane grant wires shown in Figure 2). Before an input unit makes a request to send a cell to an output port, it checks that the line it needs at that port actually has space to hold the cell, and refrains from making a request if it does not.

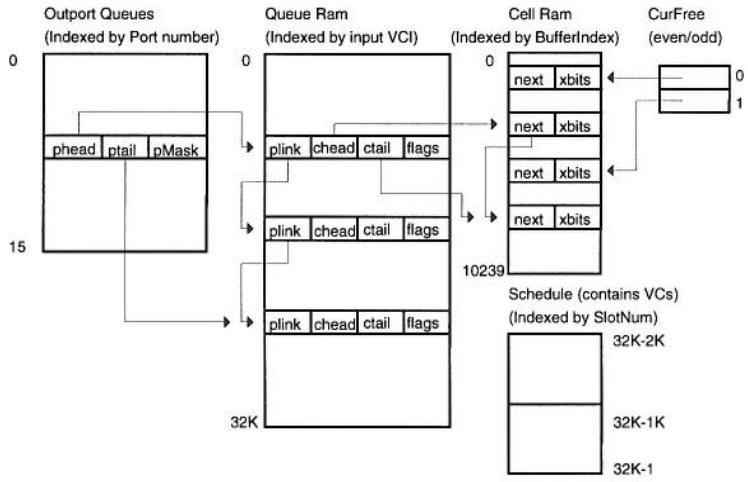

Figure 4: Input Control Unit Data Structures

The input unit's activities are triggered by four events:

1) Arrival of a *cell* on an input line,
2) Arrival of a *grant* from the crossbar arbiter,
3) Arrival of a *stop* indication from an output unit to which the input unit just sent a cell, and
4) Arrival of a *start* indication for some VC from an output unit.

A virtual circuit can be either *stopped* or active depending on whether it has a non zero credit balance. Each credit corresponds to an available buffer at the downstream switch or host. The credit balances are maintained by the output units. The input units maintain a single bit of state, *stopped,* that indicates that the VC is not allowed to send cells. (Note that a VC may also be enabled or disabled by the LCP. Disabled VCs are never placed on a port queue, although if the VC is on a port queue at the time it is disabled, it will send one additional cell before leaving the queue.) The following sections describe each activity in detail.

Cell Arrival

The input unit writes data into the cell VRAM during every slot, even if the data does not correspond to a valid cell. The index of the buffer into which the data is transferred is held in the *curBuf* register. If the data corresponds to a valid cell, curBuf is placed on the cell queue for the VC, and curBuf is loaded from *curFree*, which is the head of a list of free buffers chained through cell ram entries. There are actually two curBuf and curFree registers, since the VRAM is organized as two halves which must be written alternately. If the data does not correspond to a valid cell, then curBuf is not advanced, and the cell buffer is reused to hold the data that will arrive in two slot times.

When a valid cell arrives, its input VCI is used to index the queue ram, and the current buffer is placed at the end of the VC's cell queue (chead and ctail are the queue pointers). If the VC is not already on a port queue and the VC is not stopped, it is placed on the port queue corresponding to the required output port Note that CBR VCs are not placed on port queues, but are transmitted according to a fixed schedule.

Grant Arrival

The input control logic requests a connection through the crossbar whenever there is a cell to transmit. For CBR VCs, transmission is done according to a fixed schedule in the cell RAM. During each slot, the input controller inspects the schedule entry corresponding to the current slot. The schedule contains the VC to which the slot is allocated, or zero if there is no such VC. If a slot is allocated to some VC, the input controller then inspects the VC's entry in the queue ram. If the VC has one or more cells queued and it is enabled and not stopped, the input controller requests the output line (or lines, if this is a multicast VC) needed by that VC. Since the schedules for all line cards in a switch are coordinated by the master LCP, this will be the only request that the output unit receives, and it is guaranteed to grant it. If a slot is needed for CBR traffic, then no ABR requests will be made.

ABR traffic is handled differently. An ABR VC that is not stopped and which has cells to transmit is on the port queue corresponding to the output port to which that VC needs to send. The presence of a non-null entry on a port queue causes the input controller to generate a request to an output unit, providing that the output unit has space in its line buffer for the cell and there is not a CBR cell to be sent. There may be several such requests from each input unit. The iterative match logic operates during T0..T5, and each of the input units ultimately receives a grant for at most one of the output units it requested.

If an input unit fails to receive a grant from any output unit, it will continue to issue the request(s) in the next slot until a request is finally granted.

The iterative matching logic produces a grant at T5, and the grant arrival process that prepares a cell for transmission operates during T6..T9. If the grant corresponds to a CBR request, the cell to be sent is dequeued from its VC, the VRAM is read to place the cell in the VRAM's output shift register, and the cell buffer is returned to the appropriate free list. If the grant corresponds to an ABR request, the port queue entry corresponding to the granting output port is accessed, and the VC at the head of the queue is removed from the port queue (but the *queued* bit of the VC remains set, so that if another cell arrives, the VC will not be placed on a port queue). The cell at the head of the VC's cell list is dequeued, transmitted, and freed, exactly as for a CBR cell.

When a cell is forwarded through the crossbar, its VCI is placed in the output unit's *ack ring*, so that the output unit can send a credit to the upstream sender. Also, the quota counter for the input line on which the

cell was received is incremented. Finally, the VC is placed in the *penalty box*, where it waits to see if the output unit to which it sent the cell will send a stop. Note that while a VC is in the penalty box, it is not on a port queue, and is therefore unable to make requests for an output unit, even if it has cells to send. This means that a single VC can only obtain half the switch bandwidth, or 400 Mbits/second. This is not a problem for OC3 links, but OC12 links must use more than one VC to obtain full throughput.

The use of a single queue for each port, rather than a queue for each output line also introduces an undesirable blocking effect. Since the VC at the head of a port queue will only make a request for an output port if the output line needed by that VC has room in its rate-matching fifo, if a particular output line is oversubscribed, then traffic for other lines on the same output port can be delayed behind the VC at the head of the port queue. This limitation stems from our use of FPGAs. A denser implementation technology would provide a queue per line, rather than a queue per port. Simulations have shown that with random traffic directed to a single output port by a single input port, the present arrangement reduces line throughput by less than 10%. With multiple input ports directing traffic to a single output port, the effect is substantially less than this.

Stop Arrival

When an output unit receives a cell through the crossbar, it decrements the credit balance for the cell's VC. If the balance becomes zero, the output unit sends a stop indication to the input unit that transmitted the cell. The stop occurs at a fixed time relative to the transmission of the cell through the crossbar, so a single bit of information suffices (the stop is always directed to the VC in the penalty box). The input unit uses the presence or absence of a stop to decide what to do with the VC in the penalty box; If a stop arrives, the VC is marked *stopped* and not *queued*. If a stop does not arrive and if the VC is ABR and it has more cells to send, it is requeued at the tail of the appropriate port queue.

Start Arrival

Stops occur synchronously with cell transmission, but starts occur as a result of the arrival of a credit from the downstream switch or host. When an output unit receives a credit (which is a VC and a count, but the count may be greater than one only in the OC12 line card), it increments the VC's credit balance. If the balance was initially zero, the input unit is known to be stopped, and must be started. On average, an input unit can process only one start per slot, but the peak rate at which the output units deliver starts can be higher, since several output units may need to send a start to a particular input unit during a single slot. Starts are therefore queued at the output units. The output units maintain a queue for each input port containing the VCs that need to be started at that input. An output unit dequeues and transmits the start VCs to the appropriate input units using a flow-controlled serial shift path over the start/stop wires shown in Figure 2. This path can transmit one start per slot from an output unit to an input unit.

When an input unit receives a start VCI, it accesses the corresponding queue ram entry, and clears the *stopped* bit. If the VC is ABR, it is enabled, and it has cells to transmit, it is placed on the appropriate port queue.

Input Control Implementation

The input unit maintains one hundred bits of state for each VC. This state is stored in 32 Kbyte fast static RAMs (the queue RAM), which can be read or written in a single clock tick. The cell RAM, which contains the VC cell queues, the cell free lists, and the CBR schedule, uses similar memories. The port queues occupy a single Xilinx 3190 FPGA. The logic that generates iterative matching requests occupies a second chip (Outmasks), and a third chip (Ramctrl) contains a number of registers. The LCP can read and write all of the state with the exception of a few registers contained in the FPGAs. A few PAL devices are used for logic that must operate before the FPGAs have been configured by the LCP, or which must be faster than the speed available from the FPGAs. Figure 5 shows the input unit control logic.

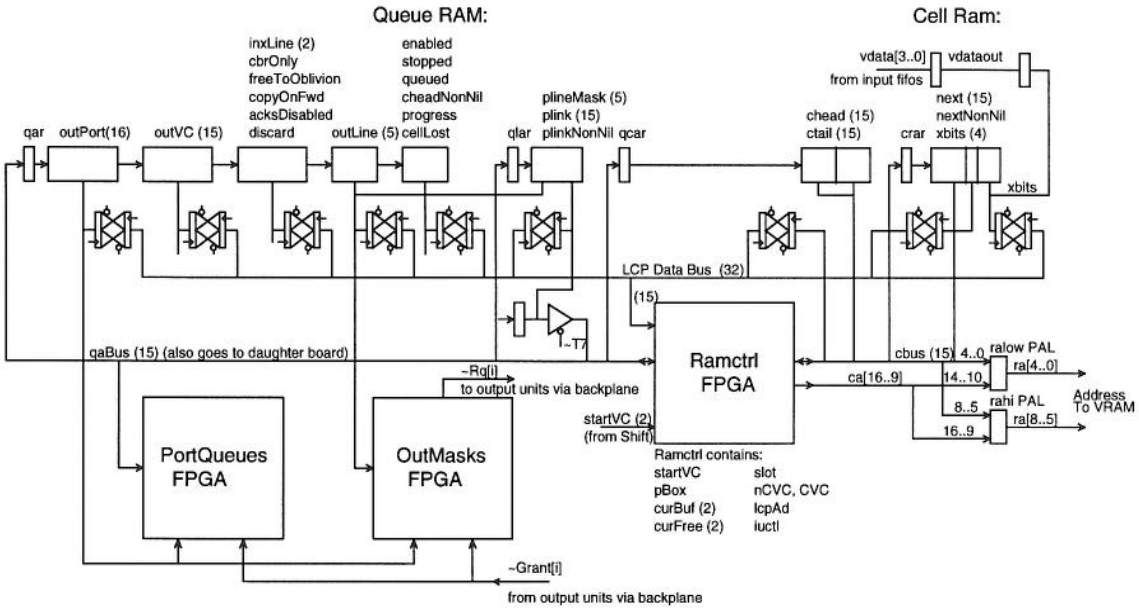

Figure 5: Input Unit Control Logic

The logic was initially described by a program that modeled each of the four activities above, in terms of the required modifications to the state. Each read and write of a memory was then assigned a clock period. The crossbar reconfigures at the end of each slot (at T12), and this time determines when the VRAM must read out the cell payload. In one thirteen-clock slot, three VRAM accesses are needed: One to transfer an arriving cell's payload from the input serial register into the RAM array, one to transfer a departing cell's data from the RAM array into the output serial register, and a third reference which is either used by the LCP to construct a cell for transmission, or is used to refresh the array. The serial read transfer must occur at a specified time within a slot, and this determines the times for the Grant Arrival action. The other actions were assigned to clock cycles such that their accesses to the VRAM and the state memory did not conflict with Grant Arrival.

Thirteen clocks per slot are sufficient to carry out the memory access needed by the four activities described earlier, but they leave no time for the LCP's accesses to the state. The LCP needs to access the queue and cell RAMs during normal operation in order to set up or tear down a virtual circuit, and to queue a cell of its own for transmission. These activities are infrequent, so when the LCP needs to access a memory, it usurps the time that would otherwise be used to process a start from an output unit. Since starts are also infrequent, sharing state accesses between the LCP and start processing results in an insignificant performance reduction.

2.2.2 QLC Output Unit

Like the input unit, the output unit consists of a data path and a control unit. The data path, shown in Figure 6, contains four fifo buffers which match the rate of the crossbar to the (slower) rate of the output lines. It also contains logic to prepare cells for transmission by mapping the 15-bit internal VCI into a smaller VCI used on the link and to add a flow-control credit if appropriate. Flow control credits are taken from the *ack rings*, which are four ring buffers implemented in two 32K by 9-bit fast static RAMs. When the input unit forwards a cell through the crossbar on behalf of a flow-controlled virtual circuit, it places the VCI into the ack ring associated with the input line on which the cell originally arrived. When the output unit is ready to send a cell, it inspects the ack ring for the line (which is half of the full-duplex link pair that includes the original input line), and piggy-backs the credit onto the cell header if the ack ring contains a credit. If a line has a credit to send but no data cells, the output unit fabricates a null cell containing the credit and a destination VCI of zero, and sends it. If there are neither data cells nor credits, the SUNI chip fabricates null cells with zeros in both the destination and credit VCI fields.

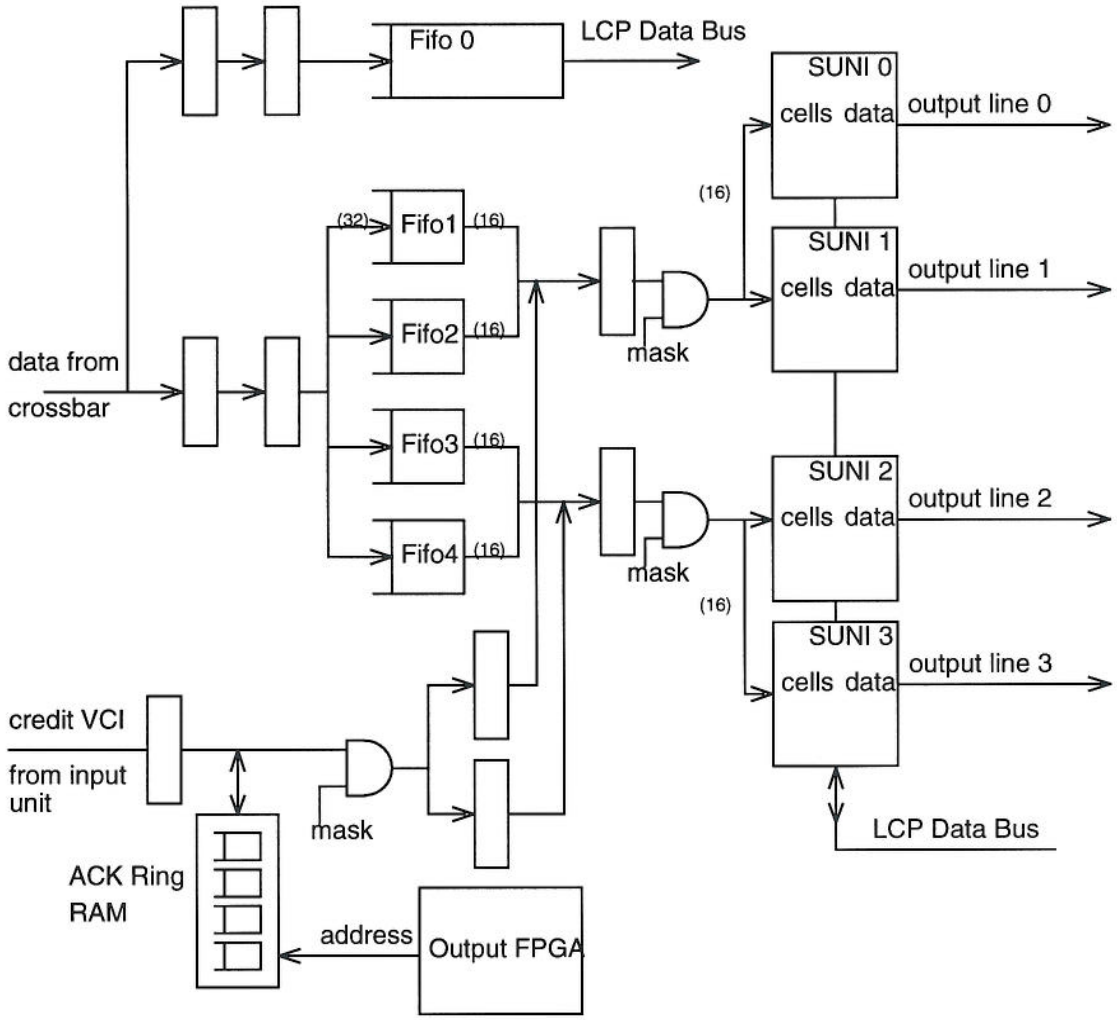

Figure 6: Output unit data path.

The AND gates shown in figure 6 are provided to undo the effects of the VCI mapping done at the input unit (i.e., they zero the high-order VCI bits). This logic and the ring buffer pointers for the ACK rings are contained in the output FPGA.

Piggy-backing credits in the headers of outgoing data cells provides a low-latency mechanism for their delivery. Other credit-based flow-control schemes [4,6] batch several credits into special cells on a well-known VC. These schemes are appropriate in wide area networks that modify the high-order header bits, but they have two disadvantages: First, they introduce extra delay in the delivery of credits, since the switch must accumulate several credits before sending any of them. This means that each flow-controlled virtual circuit must have more buffers than are strictly necessary if it is to operate at full rate. Second, they are more complex, since the output unit must know how to assemble cells (with their associated checksums), and the upstream switch must know how to disassemble and parse credit cells. AN2 flow control is more efficient in its use of buffers, and less complex to implement.

Output Control Unit

The output control unit, shown in Figure 7, is considerably simpler than the input control unit. The state of each virtual circuit is held in fast static RAM (the *credit RAM*), as in the input unit. The LCP can read and write these memories via the bidirectional registered transceivers shown in the figure. The output control

unit includes an FPGA (Grant) responsible for arbitration. This chip operates autonomously, deciding which request to accept, and supplying setup information to the crossbar. The only connection to the rest of the output control logic is a signal indicating that a cell will arrive through the crossbar in the next slot, and logic to send a stop to the input unit that sent the cell if the VC must be stopped. Since the backplane grant wires are only used to transmit iterative matching grants during eight of the thirteen clock periods in each slot, the 'half-full' flags from the output unit's rate-matching fifos are time-multiplexed onto this wire and transmitted to the input units during the remaining five clock periods (one clock for each output line fifo, plus one clock for the fifo that holds cells directed to the LCP). The input units will refrain from making a request for an output port if the rate-matching fifo needed at the port is more than half full.

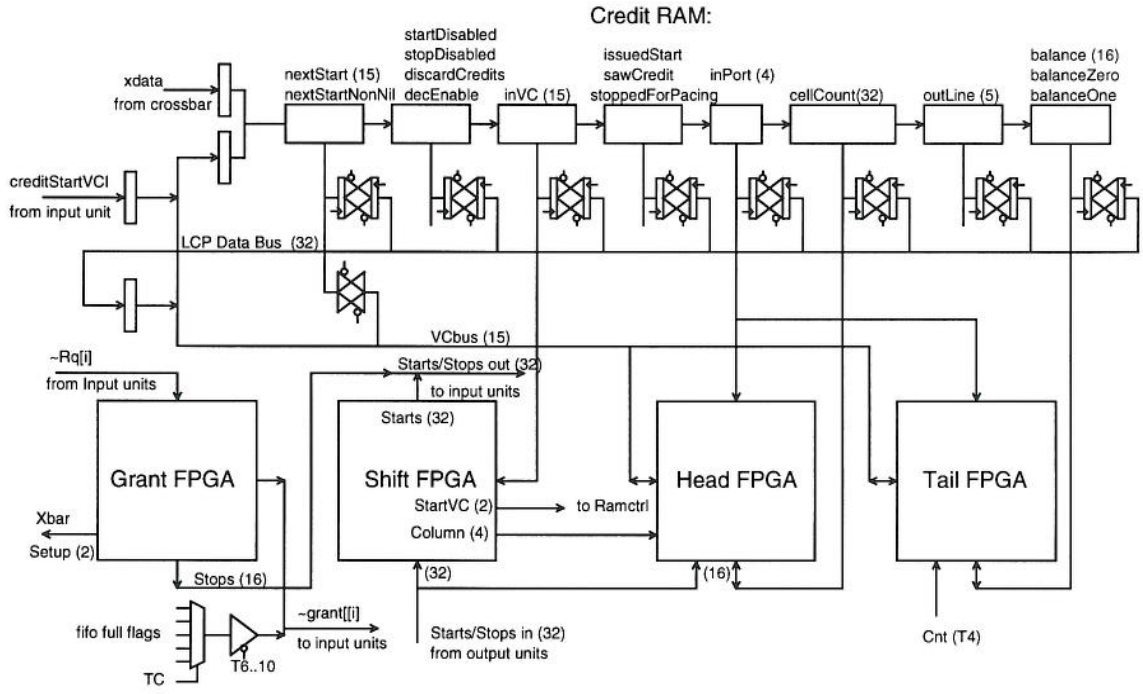

Figure 7: Output Control Unit

The rest of the output control unit has two functions:

1. When a cell arrives from the crossbar, the output control unit decrements the flow-control credit balance (if the cell is on a flow-controlled VC), and sends a *stop* to the input unit that sent the cell if the credit balance becomes equal to zero. The stop is transmitted back to the input unit that sent the cell over the start/stop backplane wires. At this time, the output unit also determines the output lines that should receive the cell, and enables the appropriate fifo. It also increments the thirty-two bit *cellCount* field of the VC, which is provided for monitoring and accounting purposes. The *stopDisabled* bit in the credit RAM disables the transmission of stops. The *decEnable* bit enables the decrement of the credit balance.

2. The output control unit must process flow-control credits that arrive from the input unit or from the LCP. A credit arrives from the input unit when a downstream switch or host frees a cell buffer. For flow-control purposes, the LCP appears as a downstream entity. When the LCP removes a cell from its fifo, it must increment the credit balance for that cell's VC. If the increment of the credit balance makes the new balance positive, then the appropriate input unit must be *started* (the input unit's VC was stopped when it forwarded a cell that decremented the credit balance to zero). The LCP fifo can never overflow, since its 'nearly full' flag is used to inhibit input unit requests, but if the total number of credits available to VCs destined for the

LCP exceeds the size of the fifo (157 cells), momentary traffic overloads can degrade the throughput to the output links on the same line card.

The transmission of a start from an output unit to an input is tricky, requiring three FPGAs for its implementation. Since an input unit can process only one start per slot, starts are queued at the output unit, and are sent to the input units only as rapidly as the input units can accept them. The Head and Tail FPGAs contain the (sixteen) head and tail pointers (VCIs) for a list of virtual circuits that have pending starts for a particular input unit. The list is threaded through the *nextStart* field of the creditRam. When a credit for a VC arrives at the output unit, and that credit causes the VC's balance to become nonzero, the VC is appended to the start queue for its input port. The transmission of a start VCI to an input unit is done serially, two bits at a time, over the start/stop backplane wires. The Shift FPGA contains a bank of sixteen shift registers that transmit the VCIs. The Head chip contains sixteen full/empty bits for the shift registers. Every slot time, Head inspects the start queues and the full/empty flags. If it finds an empty shift register and a nonempty start queue for a particular input port, it accesses the *inVC* field of the credit RAM (which contains the VCI as it is known at the input unit), loads that VCI into the appropriate shift register, and sets the full flag. As each shift register is loaded, a *bid* bit that is transmitted with the VCI is also set. This bit notifies the receiving input unit that the VCI is valid. The shift registers recirculate once per slot, until the input unit accepts the start. The start/stop wires are received at the Shift chip (the receiving circuitry is logically part of the input unit). When the input unit is able to accept a start, the Shift chip transmits the VCI to the input unit's Ramctrl chip over the *startVC* wires, where it is buffered to await further processing. The Shift chip also transmits a *bid accept* signal back to the output unit that sent the start. This signal also travels on the start/stop wires. When it arrives at the output unit that originally sent the start, it clears the shift register's bid bit and the associated full flag. The Head chip can then send another start to that input port.

The input unit accepts one start per slot unless the LCP uses the cycles normally used for start processing. When this occurs, the input control logic notifies the Shift chip, which withholds the bid accept signal.

The start/stop path between output and input units is tortuous, and it might appear that doing the credit accounting at the input unit would be simpler. Unfortunately, most of the mechanism of the present scheme (e.g., the VCI translation and the flow-controlled transmission between output and input units) would still be necessary. It would also be necessary to send credit VCIs from output ports to input ports as rapidly as credits arrived. Additional latency in the credit-delivery path would mean that a VC would require more buffering to operate at full rate in the absence of congestion, so this approach would make the latency of this path important. In the present scheme, in the absence of congestion, stops and starts are never generated at all. When congestion occurs, it is less important that VCs be started quickly, since they will simply add more traffic, so the performance of the start path need not be optimized.

2.2.3 Line Control Processor

The LCP is an LSI Logic LR33000, a RISC processor with a performance of approximately 20 mips. The LCP has 8 Mbytes of DRAM, 2 Mbytes of flash EPROM, 0.5 Mbytes of bootstrap EPROM, an Ethernet controller, and a serial line interface for debugging.

The LCP can read and write all the input and output unit memories. Special write operations are provided to implement functions (e.g., placing a VC on a port queue, or modifying a credit balance) that must be done atomically.

The LCP contains facilities to allow it to send and receive ATM cells. To transmit, the LCP assembles one or more cells in the VRAM, using buffers that are not part of the hardware's buffer pool. It then sets up the VC in the queue and cell RAMs. Finally, it *enables* the VC, which causes the hardware to place it on the appropriate port queue. The *freeToOblivion* bit in the VC state is set, so that the hardware will not place the cell buffers on the hardware's free list as they are transmitted. The LCP can then poll the queue RAM entry to determine when the cells have been transmitted by the hardware.

Reception is considerably simpler. One of the possible destinations for cells leaving the crossbar is a large fifo, which the LCP can read. When this fifo is nonempty, the LCP is interrupted, and it copies the cells from the fifo to the DRAM for further processing. The fifo can hold 157 cells, and uses the same mechanism as the output unit rate-matching fifos to ensure that it never overflows.

The LCP may appear overpowered for its application, but we have found that a powerful processor with a relatively large amount of memory makes the software development task much easier. Most of the complexity of the AN2 is in its software, so minimizing the effort needed for software development is critical. Having a powerful processor will also allow us considerable latitude for future experimentation.

3. AN2 Switch Software

The AN2 switch software controls the operation of the switch but normally does not directly forward client cells; Cell forwarding is done by the switch hardware under the control of tables set up by the software. The software treats the switch hardware as a distributed system in which each line card is a node; communication among these nodes is by the Ethernet and the switch crossbar. Working together, the software on the line cards in a switch is responsible for initializing the hardware, monitoring the status of the hardware and connected lines, setting up and tearing down circuits, and determining the best routes through a network of interconnected switches.

The software is constructed above a real-time kernel, DECelx[7], a copy of which runs on the LCP on each of the line cards. The kernel provides scheduling, resource allocation, and communication facilities, including a TCP/IP protocol stack. It also provides a remote debugging facility, a shell, and a dynamic module loader, which are particularly useful for hardware and software debugging. The development environment includes a C compiler and linker that runs on DS5000 workstations.

Much of the behavior of the AN2 switch, including the setup and tear down of switched and permanent virtual circuits (SVCs and PVCs), and the SNMP management interface, is dictated by standards [8]. In addition to these standard capabilities, AN2 provides a number of extensions. Flow-controlled ABR virtual circuits are available when a connected host or switch is equipped to participate in the flow-control protocol. These circuits have the attractive property that, in the absence of hardware failure, ATM cells are never discarded by the network. Resilient Virtual Circuits (RVCs) are provided that connect each host to every other host in a multi-switch network of moderate size. RVCs are established by the switch and allow hosts to use extant datagram protocols in a LAN-emulation mode without requiring circuit setup. Routing for RVCs is based on a reconfiguration protocol similar to that of the earlier Autonet [9] that automatically detects changes in the physical configuration and quickly reroutes circuits to bypass failed components and utilize new components.

In an AN2 switch one of the LCPs becomes the master and assumes special duties. At initialization the LCP that occupies the lowest-numbered backplane slot is elected the master. The master communicates with external hosts and switches using SNMP, Q93B, and other protocols over certain well-know virtual circuits. It communicates with the line cards within the switch using a simple request-response protocol (RRP) over inter-card virtual circuits.

3.1 Communication Software

Figure 8 shows the communication protocol stacks in a running AN2 switch. The structure illustrated is replicated on each line card, although many of the protocols are exercised only in the master line card.

The LCP's Ethernet hardware is used by a standard IP-based protocol stack, shown on the right-side of the figure. The Ethernet provides an out-of-band management (OBM) path, meaning management via a channel other than the ATM lines of the network. The typical facilities of a Unix workstation, such as Telnet access to a command interpreter, file transfer, and debugging are available. In addition one Telnet port connects to

an interactive agent on the master line card that allows the user to setup and tear down permanent virtual circuits. Another connects to an SNMP agent for the switch. The barrier synchronization protocol shown in Figure 8 is used during switch initialization and will be described later.

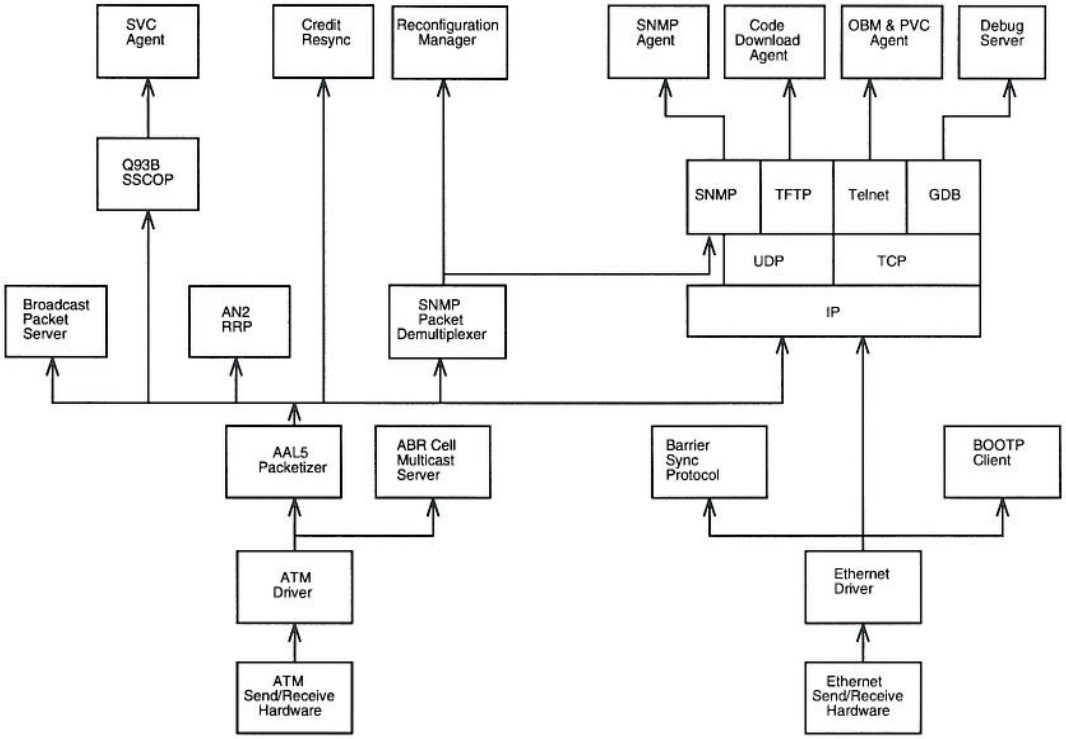

Figure 8: AN2 Communication Protocol Stacks

The ATM send/receive hardware allows the LCP on each line card to send and receive ATM cells via the crossbar. The driver copies incoming cells to the LCP's memory from the hardware receive FIFO and copies outgoing cells from the LCP's memory to the transmit cell buffers provided by the hardware. At the level of the port driver incoming circuits are split into two categories. First, for circuits that are designated as carrying AAL5 packets to the LCP, arriving cells are passed to the AAL5 packetizer module where they are reassembled into packets in MBufs; when a complete AAL5 packet is available the MBuf is passed to the appropriate higher level protocol module. Second, for circuits that are managed at the cell level, the individual cells are passed to the appropriate higher level protocol module (of which there is only one example so far, the ABR Cell Multicast Server). In each case the protocol module is chosen based on the circuit's VCI. Circuits carrying host-to-host traffic normally bypass the LCP and are forwarded directly by the hardware.

3.1.1 VCI Terminology

As described earlier, the AN2 switch hardware performs a mapping between the VCI values used in cells on a line and the values used within the switch. We refer to the former as *line VCIs* and the latter as *port VCIs*. We distinguish two uses of port VCIs: input VCIs correspond to cells received from one of a port's lines or from the LCP; output VCIs correspond to cells moving through the crossbar to an output line or to the LCP.

Line VCIs from different incoming lines on a line card map to different ranges of input VCIs; output VCIs are truncated by the mapper for the outgoing lines. Port VCIs to and from the LCP are not mapped. Typically line VCIs are 11 to 14 bits, depending on the equipment at both ends of a line. Port VCIs in AN2 are 15 bits.

At a switch a *connection* establishes the relationship between the VCIs used on adjacent lines of a circuit. The input port converts the input VCI to an output VCI when it forwards a cell on a connection as follows:

port i	xbar	port k
lineVCI1 -> inVCI -> outVCI	>>>>	outVCI -> lineVCI2

For a duplex circuit the pattern is repeated, mutatis mutandi, to provide connections in both directions, with the corresponding input and output VCI in each port usually having the same values:

port i	xbar	port k
lineVCI1 -> inVCI -> outVCI	>>>>	outVCI -> lineVCI2
lineVCI1 <- outVCI	<<<<	outVCI <- inVCI <- lineVCI2

For circuits to or from an LCP we omit the mapping to line VCIs. Since an LCP can send or receive cells only to and from the crossbar, a circuit from a line to an LCP, even to the LCP on the same port, must be connected at the input port to an output VCI that passes through the crossbar.

3.1.2 Internal Circuits

A switch is automatically provisioned by its software with various internal circuits to and from the LCPs. Each set of related internal circuits corresponds to one of the protocol modules shown at the layer above the AAL5 packetizer module in Figure 8. All of these internal circuits transport AAL5 packets and are demultiplexed via a table that associates the VCI with a protocol module.

RRP Circuits

At each LCP, sixteen VCIs are established to carry request-response protocol (RRP) messages. At each LCP, VCI R + p (R is a constant) is the circuit to the LCP at port p. For example, the LCP at port 5 would send and receive on VCI R+14 to perform RRP actions at the LCP on port 14. The LCP at port 14 would receive and send on port VCI R+5 for the reverse communication. RRP is a simple remote procedure call mechanism that can pass a single C struct as a parameter, uses tokens to match response messages with request messages, and provides "exactly once" call semantics. Packets arriving on the RRP circuits at an LCP are passed up to the AN2 RRP module in Figure 8.

Credit Resync Circuits

AN2's flow control scheme uses a well-known circuit on each line to occasionally resynchronize the flow control credits. At each LCP, four VCs, one per head on that line card, are setup for this purpose. These circuits are used only if the remote end of a line supports flow control. Packets arriving on the credit resync circuits at a LCP are passed up to the Credit Resync module in Figure 8.

SNMP Circuits

The ATM Forum specifies that line VCI 16 is to be used for SNMP communication with a switch. At the master LCP of an AN2 switch, 64 distinguished VCIs, one for each of the four possible heads on the sixteen line cards, are setup for this purpose. The SNMP packets on these circuits are further divided into *SNMP actions* and *CONFIG actions* by the module labeled SNMP Packet Demultiplexer, based on the contents of a type byte in the SNMP header. The CONFIG actions are used by the automatic reconfiguration machinery in AN2. Given that a single well-known circuit on each line turns into 64 different VCIs at the master LCP, this additional layer of demultiplexing into subcircuits is employed to conserve VCI values and to avoid arbitrarily choosing some other line VCI value for the reconfiguration messages. The SNMP actions are passed up to the SNMP Agent in Figure 8. The CONFIG actions are passed to the Reconfiguration Manager

module. CONFIG actions will appear on a line only if both ends agree in the line startup protocol to handle them.

SVC Setup Circuits

The ATM Forum specifies that VCI 5 is for setting up switched virtual circuits (SVCs). At the master LCP of an AN2 switch, 64 distinguished VCIs terminate SVC setup circuits. Packets arriving on the SVC setup circuits at the master LCP are passed up to the SVC Agent module in Figure 8.

LAN-Emulation Broadcast Circuits

As part of the LAN-emulation functionality of AN2 it is necessary to provide a broadcast channel. From the point of view of a host node, the broadcast channel is the well-known line VCI values B_{out} and B_{in}. When a host transmits an AAL5 packet on the line VCI B_{out}, that packet will arrive on line VCI B_{in} at every host attached to the network. (B_{out} could have the same value as B_{in}, but making them different is safer because it prevents an arriving broadcast cell that is inadvertently retransmitted by a host from circulating forever.)

LAN-emulation broadcast is implemented by choosing a single LCP of a single switch in the LAN to be the *broadcast server*, which is the module labeled Broadcast Packet Server in Figure 8. At this LCP, one VCI per host terminates the point-to-point circuit known by VCI B_{out} at each host.

In the reverse direction the VCI B_{in} identifies a unidirectional CBR multicast circuit setup as a tree from its root at the broadcast server to leaves at all hosts. Whenever the broadcast server has received a complete packet from some host node via the inbound point-to-point circuit, the server then transmits that packet on the multicast circuit. Because the broadcast server transmits all the cells of a particular packet together on the multicast circuit, a single such circuit suffices as the LAN-emulation broadcast channel for the entire network. The broadcast VCIs on a host-to-switch line are used only if the host supports LAN emulation.

The choice of the LCP to be the broadcast server is made by the automatic network reconfiguration mechanism, which also sets up the inbound point-to-point and outbound multicast circuits to correspond with the choice.

ABR Multicast Circuits

So far all of the internal circuits we have discussed have been AAL5 packet circuits. Each LCP also handles one kind of cell-at-a-time circuit. Recall that the AN2 switch hardware is not able to schedule the one-to-many cell forwarding required at branch points in ABR multicast circuits. In order to provide this class of service the incoming trunk of such a branch point terminates at the LCP of the local line card. A cell that arrives on such a circuit is passed individually to the ABR Cell Multicast Server in Figure 8. This module determines the set of outgoing circuits that constitute the associated branches and transmits the cell separately to each of them. The set of circuits which receive this treatment at any given LCP depends on which ABR multicast circuits have been setup in the network.

ABR multicast and LAN-emulation broadcast are the only examples in AN2 of host-to-host data cells being directly handled by the switch software.

3.2 Control Software

Figure 9 presents the structure of the control software for an AN2 switch. The modules at the tops of the various protocol stacks in Figure 8 also appear in Figure 9. The overlap illustrates the correspondence between the communications and control views of the software. Figure 8 shows how the packets or cells are sent and received by these modules. Figure 9 shows how an arriving packet or cell is acted upon.

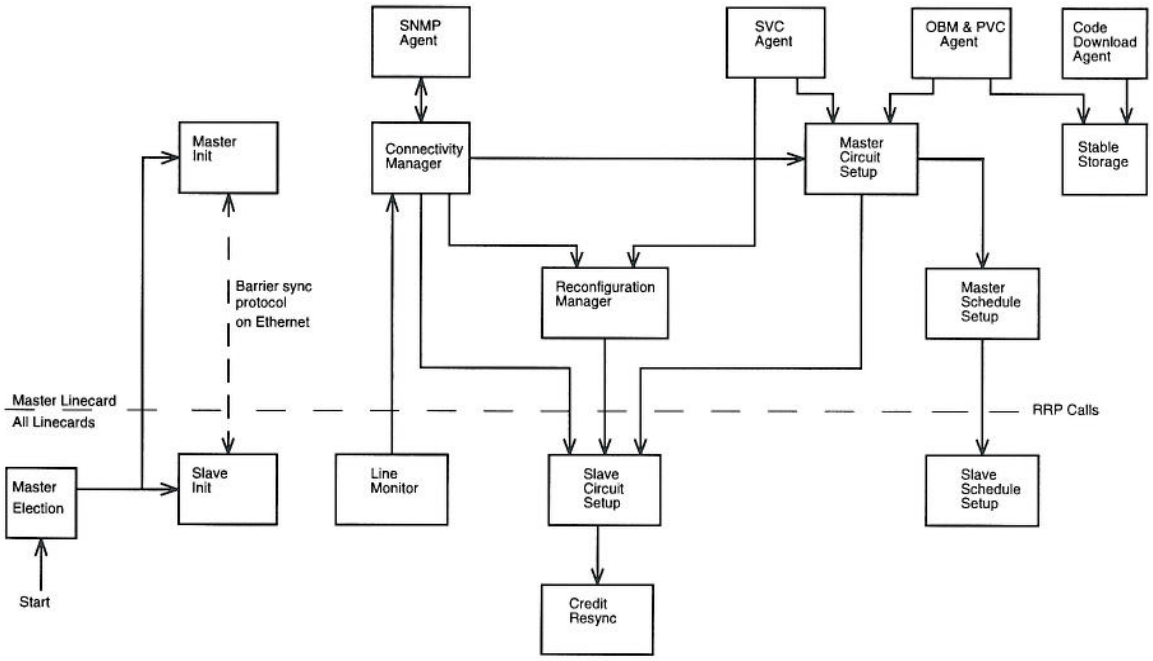

Figure 9: AN2 Switch Control Software

The horizontal line across the bottom third of Figure 9 separates actions on the master LCP of a switch from actions that can occur on all LCPs in a switch, including the master. When control passes across this boundary it usually is via an RRP call. That is, the calling module sends an RRP packet on the RRP VCI to the target slave LCP, which reacts by performing the indicated procedure call and then returning the result in a responding RRP packet. RRP is the second way in which the communications and control diagrams fit together. While most RRP calls are from master to slave, slave to slave RRP calls do occur, as will be pointed out below.

The AN2 switch software is divided into two parts. The first part, the boot code, resides in the boot ROM of each line card. The boot code consists mainly of the DECelx operating system, but also includes the module labeled Master Election in Figure 9. The second part, the switch application code, resides in a flash memory on each line card. The boot code is responsible for loading the application code, which includes all the other modules in Figure 9.

We start with a high level tour of the control structure. This will be followed by a more detailed consideration of various aspects.

3.2.1 Startup and Master Election

When a line card is powered up or reset it begins executing the boot code from the local boot ROM. Since AN2 does not support addition or removal of line cards while power is applied, power up occurs within a window of at most a few milliseconds at all line cards in a chassis. When a line card is booted it knows its local Ethernet UID from a UID ROM on the board and its slot number in the chassis from pins in its backplane connector.

Master election allows the line cards in the chassis to learn one another's identity and appoint one of themselves to be the leader. For this purpose the crossbar is not available, since it will operate only after its internal FPGAs are configured by exactly one of the line cards. Performing this FPGA configuration is one of the reasons to elect a master. The Ethernet is functional, but not useful for master election, since no line card knows the UIDs of any others in the same chassis, there is no chassis UID, and it is possible for a switch's Ethernet segment to be connected to other switches and to the building LAN.

Master election is done by temporarily hijacking two of the backplane wires that are normally used to configure the crossbar FPGAs. These wires visit all the line cards in a single chassis and are arranged electrically so each LCP can generate and sense the signal level. Signals from multiple line cards are wire-ORed. In the first part of the protocol one wire is used to define 16 time intervals, each corresponding to one of the 16 backplane slots. A card will assert a signal on the other wire when its slot number's turn comes. The first turn occupied with a signal thus corresponds to the lowest numbered occupied slot, which line card becomes the master. Since all line cards see all turns, all agree on the master. In the second part of the protocol the master, acting as the sole writer of the backplane wires, signals its Ethernet UID to all line cards in the chassis. Master election concludes with each line card sending its own Ethernet UID to the newly elected master over the Ethernet and the master returning the full set of slot-to-UID correspondences to each line card.

3.2.2 Application Code Versions

The majority of the application code that initializes and operates the switch is stored in the flash ROM of each line card. In customer sites we expect that this flash-resident code will be quite stable and that almost all line cards will almost always contain the same code version. While debugging and developing the code, however, and occasionally when a new version must be installed at a customer site, it is necessary to be able to update the flash code and to deal with different versions being resident in the multiple line cards of one switch.

As the final step of the boot code, each line card verifies with a checksum that uncorrupted application code exists in its flash memory and reports the code version to the master using the Ethernet. In addition the master attempts to contact a BOOTP server on the Ethernet to discover the version of the application code that may be available there. If a newer version is available via BOOTP then the master downloads it into the its flash memory using TFTP. The master now tells all line cards which ones have the newest version of the applications code. Any that have an older version retrieve the newest version from some other line card in the switch over the Ethernet

The result of this phase of startup is that all line cards have the newest available version of the application code stored in their flash memory. This version coordination prevents the possibility of different line cards attempting to operate using different, possibly incompatible, versions of the application code. Changes to the boot code, of course, must be made by physically replacing the boot ROM on each line card.

3.2.3 Initialization Synchronization

During switch initialization certain actions must be completed at all line cards before the next action can be started at any. For this purpose AN2 uses a barrier synchronization protocol on the Ethernet. Each line card reports with a message to the master that it is waiting at barrier *N*; when the master hears that all line cards are so waiting it tells each to pass barrier *N*. The wait and pass messages can communicate initialization data. For example, the barrier 0 pass message tells all line cards certain configuration parameters that have been read from the master's flash memory. The barriers and the actions preceding each are as follows:

Barriers	Actions
	Load main board FPGAs
0	
	If master then load crossbar FPGAs
1	
	Initialize main board queue structures
2	
	Reset the main board
3	
	Initialize the cell buffer free list
4	
	Load line board FPGAs Initialize line hardware Start deadman's handle
5	
	Initialize per VCI mutexes Initialize LCP's ATM driver
6	
	Build internal circuit and PVCs Startup agent threads
7	

Between barriers 6 and 7 the Master and Slave Init modules shown in Figure 9 start threads in the remaining modules.

Once barrier 7 has been passed then line cards may send and receive cells via the crossbar – to other line cards, to themselves, and to the external lines. Because of the barrier synchronization mechanism no line card will try to use the crossbar before the crossbar FPGAs and all the line card FPGAs that interface to the crossbar are configured. Startup of a multicard switch in which all line cards have the current application version takes several seconds. This could be decreased by tightening various time-outs.

3.2.4 Deadman's Handle

The AN2 software can be optionally configured so that if the LCP on any line card hangs then all line cards in the switch will receive a signal to reboot themselves. The purpose of this mechanism is to get the switch back into service as quickly as possible if the LCP on some line card stops executing instructions. The hardware mechanism underlying this facility is a 1.6 second interval timer on each line card that, if not refreshed before it runs down, will reboot the LCP. Refreshing is done by a thread that is scheduled about twice a second. This thread also will reboot the local LCP if a signal is present on the global crossbar FPGA reset wire (one of the wires used in master election). Since signaling on this wire is part of the reboot sequence, any LCP rebooting will cause all to reboot within about half a second. We plan to add monitoring and history facilities that will allow any line card to trigger a reboot, leaving out a non-responsive line card from the new switch configuration.

3.2.6 Circuit Allocation and Initialization

The line VCI values for each head on a switch are partitioned according to the kind of circuit that may be setup. *Well-known circuits* have been defined by the ATM Forum to associate particular line VCI values with particular functions. The range of line VCI values reserved for well-known circuits is 0 through 31. AN2 currently implements VCI 0 for the null circuit, VCI 5 for SVC signaling, and VCI 16 for SNMP. *Permanent virtual circuits* are setup by human management via the OBM path. PVCs are remembered in the flash memory of the master line card and will be reestablished each time the switch is booted. *Switched*

virtual circuits are setup dynamically using signaling on VCI 5. *Resilient virtual circuits* are setup by the automatic reconfiguration mechanism to provide a path, used as part of LAN-emulation, between every pair of hosts in the network. Finally, *internal virtual circuits* are setup by the AN2 software for communication to and from the LCPs.

When a switch is booted the defined well-known, permanent, and some of the internal virtual circuits are unconditionally setup and activated. Thus, a host can use these circuits without further interactions with the switch.

3.2.7 Monitoring Lines and Connectivity

Once switch initialization is completed the Line Monitor reports the existence of each head to the Connectivity Manager, which in turn then asks the Line Monitor to watch them. In the case of 155 Mb/sec. SONET lines, the monitor first causes SONET carrier to start being transmitted on the outbound fiber and then waits for the presence of SONET carrier on the inbound fiber. Incoming carrier means that some device is active at the other end of the line, which fact the monitor reports to the Connectivity Manager.

The next step is for the Connectivity Manager to determine what sort of device is present at the other end of the line. This line startup handshake is performed by each end doing an SNMP get of the capability variables of the other end. For hosts that do not support the SNMP startup protocol in their drivers, capabilities can also be communicated using a simple exchange of fixed format packets encapsulated as CONFIG actions on VCI 16. This method is used when the initial SNMP response indicates that the remote end supports CONFIG actions on VCI 16, or when a CONFIG action is received.

The Connectivity Manager will periodically transmit the SNMP get request once it is informed of incoming carrier, until a reply is received, backing off the frequency of transmission if no answer if forthcoming. The reply tells the Connectivity Manager the capabilities of the device at the other end of the line. Until a reply is received the remote device is treated as a minimum-function host for which well-known circuits and PVCs will work.

Receiving the capabilities of the remote device allows the Connectivity Manager to report "line up" to Master Circuit Setup, which in turn can tell the SVC Agent to begin the SNMP-based address registration protocol that supports SVC circuit setup. If the capabilities indicate that the remote device supports the additional variables in the Digital MIB extension, the Connectivity Manager also gets these to determine support for flow control, LAN-emulation via resilient virtual circuits, and the CONFIG actions subcircuit on VCI 16. Digital MIB variables also allow a host to learn from a switch various useful details about VCI assignment, credit management, and LAN-emulation, the uses of which will be discussed later.

The Connectivity Manager in an AN2 switch, upon discovering that a remote host supports flow control, will cause the Credit Resync module to turn on flow control for the line. Upon discovering that a remote host supports RVCs it will cause the Slave Circuit Setup module to activate the LAN-emulation circuits on the line to that host.

The Connectivity Manager, upon discovering that the line connects to a remote switch that responds on the CONFIG actions subcircuit, must arrange to include this new switch in the active topology of the network. For this purpose it instigates a network reconfiguration as describe in the next section.

The Line Monitor module also continuously watches for the disappearance of SONET carrier on a previously active line. Disappearing lines are reported to the Connectivity Manager. For a disappearing host, flow control and LAN-emulation are turned off. For a disappearing switch, a reconfiguration is started. In either case the SVC Agent is notified (via master Circuit Setup) so it can begin to tear down any switched virtual circuits that traverse the line.

3.2.8 Reconfiguring the Network

The reconfiguration mechanism in AN2 is very similar to that used in Autonet [9], which has been in service as the building LAN at SRC for several years. The objective of this mechanism is to discover and distribute the switch-to-switch topology of the operational network very quickly after each change due to failure, removal, repair, or addition of lines and/or switches. For the 30-switch Autonet service LAN, the reconfiguration algorithm takes about 150 milliseconds to completely reroute all communications, fast enough that high level protocols are not disrupted. For AN2 we expect to achieve higher performance, even though the introduction of circuits complicates the computation, because the AN2 switch processors are much faster than the Autonet processors.

Whenever the Connectivity Manager determines that the switch-to-switch topology of the network has changed it tells the Reconfiguration Manager to perform a network reconfiguration. The first phase of this operation is to build a propagation order spanning tree of the switch-to-switch topology with the initiating switch at the root. An earlier paper on reconfiguration in Autonet [11] describes the details of this algorithm, including the mechanisms to deal with multiple overlapping reconfigurations triggered by different roots and to determine when the spanning tree is complete.

The reconfiguration messages are carried to neighboring master LCPs as AAL5 packets on the CONFIG action subcircuit (identified by having SNMP-packet data byte 0 = 0x55) of VCI 16, the well-known SNMP circuit that is always available on any line. The algorithm collects the network topology along with the spanning tree. Once the spanning tree is completely known by the root switch, it sends the complete topology down the tree to all switches. The topology collection and distribution thus is a distributed computation performed by the Reconfiguration Manager module in the master LCP of every switch in the network. In addition to supporting RVC routing, the topology description is made available to the SVC Agent as a basis for routing SVC's.

At the conclusion of the topology collection and distribution phase, the Reconfiguration Manager at the master LCP of each switch knows the complete switch-to-switch topology of the network. Its job is now to set up the forwarding tables in the line cards of its switch to established the circuits needed for LAN emulation. Two sorts of circuits are needed. First, the network provides a point-to-point RVC between every pair of non-switch heads in the network. By "non-switch head" we mean a head that is not currently connected to a line to a responsive switch that implements the reconfiguration protocol. Second, the network provides the LAN-emulation broadcast circuits.

The RVCs are based on the idea of *home VCIs*. An ordering of the switches is established based on the Ethernet UID of the master line card. From this switch ordering a head ordering is computed that covers all heads in the network. The head ordering is then used to assign a unique home VCI to each head. The home VCIs are chosen sequentially starting from the lowest numbered line VCI than can be used for RVCs. On a host-to-switch line a home VCI identifies the circuit for communicating with the corresponding remote host. For example, if a host with a home VCI of 713 wants to send AAL5 packets to the host with a home VCI of 797, then host 713 sends the cells of the packets on outbound line VCI 797 and host 797 will receive the cells on inbound line VCI 713. The reverse circuit has 797 sending on 713 and 713 receiving on 797. This works for every pair of hosts in the network with substitution of appropriate home VCI values.

A host learns its home VCI value from the line startup packet (which will be repeated when a reconfiguration occurs that changes the assignment of home VCIs). A host discovers the home VCI of another host by sending a "home VCI ARP" packet, containing the target Ethernet UID plus the requester's Ethernet UID and homeVCI, on the LAN-emulation broadcast circuit. The recipient, upon matching the target Ethernet UID, generates a "home VCI ARP response" packet, directed to the requester's home VCI, that reports to target's home VCI.

The Reconfiguration Manager must set up circuit segments on the lines of the network to implement the needed RVCs. By having each switch route the RVCs with the same algorithm, starting from the same network topology, it is possible to do this in a coordinated way without further communication among the switches.

The setup of the LAN-emulation broadcast circuit was described in section 3.1.2. Each switch autonomously performs its part of the setup using the same routing algorithm and topology, so a coordinated broadcast distribution tree results. On host-to-switch lines the incoming and outgoing broadcast circuits use the line VCIs B_{in} and B_{out}, which values are communicated to the host at line startup. On the switch-to-switch lines, VCI values for the circuits from hosts to the broadcast server are chosen from the RVC range are assigned as necessary. The broadcast distribution circuit from the server to all hosts uses the line VCI B_{out} on all lines. This circuit is setup as a CBR point-to-multipoint circuit with a scheduled bandwidth of 10 Mb/s so broadcast cell forwarding can be done by the switch hardware.

RVC-based LAN emulation is applicable to moderate sized networks. The scaling limitation is the number of VCIs available for routing RVCs on the switch-to-switch lines. In the worst-case topology for a network with n host heads, $(n/2)^2$ RVCs could be routed over the one switch-to-switch line. Given 13-bit VCIs on such lines, n is limited to around 180 hosts. Assignment of a home VCI to each host is less of a scaling limitation since the number needed is proportional to n, not n^2. Most topologies allow considerably more that 180 hosts and for any network that encounters the limit it can be increased by adding the appropriate switch-to-switch lines to the topology. Within its scale limitations, however, RVC-based LAN emulation provides high performance and availability. It responds quickly to changes in network topology and provides all needed point-to-point circuits with no setup delay. The ATM Forum is defining a standard method for LAN emulation based on SVCs and AN2 will implement it. The RVC-based LAN emulation provides a functionality and performance benchmark for that standard in all areas except scaling to very large networks. RVC-based LAN emulation will likely coexist with SVC-based LAN emulation in Digital ATM LANs, giving customers the RVC option in appropriate configurations.

3.2.9 Connection Setup

Connection setup in AN2 means adjusting the contents of the data structures in the port input and output control units that control cell forwarding. The input circuit table, stored in the port's queue RAM, is indexed by input VCI and tells the hardware where and when to forward a cell. The output circuit table, stored in the port's credit RAM, is indexed by output VCI and tells the hardware which line to use, which port a cell came from, and contains the flow control counters and flags. The software visible fields of the input and output circuit table entries are as follows:

Input Circuit Entry for a VCI

1 BIT enabled	T: can forward queued cells F: will not forward queued cells
1 BIT discard	T: incoming cells are discarded F: incoming cells are queued
1 BIT acksdisabled	T: send no acks on forwarding F: send ack when a cell is forwarded
1 BIT copyonfwd	T: disable circuit after forwarding & don't dequeue cell F: normal forwarding
1 BIT freetooblivion	T: don't put dequeued cell on free list (for LCP output VCI) F: dequeue to free list
1 BIT cbronly	T: is CBR F: is ABR
15 BITS outvci	Output VCI for next hop
5 BITS outports	Set of ports for next hop; singleton for ABR. Cells go to port "i" if (outports & (1 << i)) is non-zero. "i" = 0 is LCP.
3 BITS outhead	For CBR, ignored For ABR, head needed at output port (including LCP = 0)
2 BITS ackhead	For CBR, ignored For ABR, head (excluding LCP) for acks

Output Circuit Entry for a VCI

16 BITS balance	credit balance
1 BIT startdisabled	T: generate no starts F: ok to queue starts for this VCI
1 BIT stopdisabled	T: generate no stops F: ok to generate stops for this VCI
1 BIT discardcredits	T: discard incoming credits F: use incoming credits to increment balance
4 BITS inport	Input port that forwards to this VCI on this output port
15 BITS invci	the input VCI that is forwarded here
5 BITS outheads	Set of heads to transmit on; singleton for ABR. Cells go to head "i" if (outheads & (1 << i)) is non-zero. i = 0 is LCP.

The Slave Circuit Setup (SCS) module on each line card is responsible for adjusting the tables at that port to configure the needed connections. The Master Circuit Setup (MCS) module, running on the master line card only, coordinates the circuit tables at all ports of the switch. MCS maintains a global allocation map of the VCIs at all ports in order to be able to quickly assign VCIs to new circuits and makes RRP calls to the instances of SCS to manipulate the local in and out circuit tables.

Switch hardware directly supports simplex, point-to-point connections for both CBR and ABR. By pairing of point-to-point connections that are between the same end points and VCIs, but in opposite directions, we provide duplex point-to-point connections. The hardware also directly supports simplex, point-to-multipoint connections for CBR. In all these cases the end point may be any line or the LCP. In addition, as described earlier, by having the LCP forward cells from an incoming point-to-point circuit to multiple outgoing point-to-point circuits we provide a software implementation of simplex, point-to-multipoint, ABR connections.

MCS provides its services to the PVC Agent, the SVC Agent, and the SNMP Agent. The first manages the permanent virtual circuits at the switch, creating and destroying them at the command of a human manager using the OBM interface. PVCs may be saved in the flash memory of the master line card to be re-established each time the switch boots. The SVC Agent implements the ATM Forum protocols for dynamically providing switched virtual circuits when demanded by hosts attached to the network. A multi-hop circuit from a source to a destination requires action by the SVC Agent at each switch along the path and communication among the SVC Agents in hosts and switches via line VCI 16. The ATM Forum has not yet finalized the protocols for switch-to-switch interaction for configuring SVCs. Circuits may also be setup via SNMP. The Reconfiguration Manager uses Slave Circuit Setup directly to create and destroy the resilient virtual circuits and the broadcast circuit described earlier that provide LAN emulation.

To avoid conflict over VCI values MCS defines ranges of VCI values for each line to be used for the different purposes. The ranges supported are well-known, permanent, switched, resilient, and internal. MCS will choose a free value from the appropriate range on behalf of the client, or let the client suggest a specific value to use but check that it lies within the appropriate range and is free. Because the AN2 switch translates an input VCI to the connected output VCI at the input port, all output branches of a CBR point-to-multipoint connection will have the same output VCI. This hardware constraint means that, when setting up such a connection, the output VCI must be reserved on all potential output lines. Since the circuit setup protocols allow adding branches after the fact, this requirement means reserving the output VCI on all lines when a CBR point-to-multipoint trunk is allocated. The same requirement does not apply to ABR point-to-multipoint connections, since the LCP implements these by iterative forwarding to multiple output VCIs, which may be different on different lines.

The typical pattern of actions for setting up a new (simplex) connections is that MCS allocates or validates the input and output VCIs to use, then makes an RRP call to SCS at the input line card, which sets up the in circuit state and in turn makes an RRP call to SCS at the output line card to set up the out circuit state. For a duplex connections this pattern is repeated for the opposite direction. In the duplex case the obvious optimizations are planned but not yet implemented.

The details of connection setup in AN2 are somewhat intricate because of the interaction of making connections with flow control and with CBR scheduling. These interaction are discussed in the following sections.

3.2.10 Flow Control and Credit Resynchronization

As discussed earlier, AN2 implements a per line, per circuit, credit-based flow control mechanism as a way to keep congestion from leading to cell loss and to share smoothly the bandwidth on congested lines. Flow control applies only to ABR circuits. It is not needed for CBR circuits where the pre-computed schedule eliminates these problems. AN2 flow control is activated on a line only if both ends indicate in the line startup protocol that they support flow control.

The ATM Forum has not finished specifying a standard mechanism for controlling the rate of cell loss and smoothly sharing the bandwidth on congested lines. It is working out the details for a scheme based on rate control. We believe that rate control will not work satisfactorily in LAN use of ATM networks, where the load is extremely bursty. The AN2 flow control mechanism demonstrates a simple scheme that does eliminate cell loss due to line congestion in such an environment and does provide smooth sharing of line bandwidth among ABR circuits. AN2 will implement the standard rate control scheme as well.

The mechanisms of AN2 flow control have been covered in Section 2. Here we considered activation and deactivation of flow control on a line and the closely allied subject of credit resynchronization.

The flow control mechanism keeps track of the cell buffers used up at the downstream port by a single direction circuit. Initially, the credit balance at the upstream port accounts for all available buffers, since they are all free and no cells or acks for this circuit are in transit. The credit balance is decremented each time a cell is transmitted from the upstream port. The corresponding increment occurs only after the cell has arrived at the downstream port, been forwarded from there, and had its ack make it back to the upstream port. We can think of a credit from the balance accompanying the cell from the transmitter, over the line, and through the buffers; and then accompanying the ack back though the queues in the downstream port and over the line to the balance again in the upstream port. The invariant of the credit mechanism for a single circuit on a single line is that the total number of credits, when accounted for in all these places, is conserved.

In order to start using flow control on a circuit we need to know that no credits are represented by outstanding cells or acks. Then we can set the credit balance to the desired value and know that it will be conserved. Subsequently, any error that causes a cell or ack to be lost will break the credit invariant of the circuit. If the number of credits available becomes fewer than the number needed to cover the round trip delay then the circuit will no longer be able to use the line at the full rate. If all credits are lost then the circuit will stop. A corrupted cell on the line can cause a cell and/or an ack to be lost, which can lose credits. (Fortunately it is much harder to gain credits, since both destination and ack VCIs are carried in a cell header protected by a checksum.) To re-establish the invariant we need a way to assure that the credit balance accounts for all credits. This is the purpose of the credit resync protocol. It is used both to activate flow control and to re-establish the invariant in the suspicion that credits have been lost.

AN2's credit resync protocol is simple to understand and implement. The upstream end of the one-way circuit to be resynced stops sending cells and transmits a *sync request* packet for the circuit to the LCP at the downstream end using the credit resync VCI. The downstream LCP, upon receiving the request, waits until the circuit's local buffers are empty and any associated acks have cleared the local ack queues, enables the sending of acks for the circuit if not already enabled, and then sends a *sync ack* packet back to the upstream LCP using the credit resync VCI. Upon receiving the matching *sync ack* the upstream LCP sets the desired credit balance and then allows cell transmission on the VCI. Note that a *sync request* has the effect of activating ack transmission from the downstream port, so it can be used to turn on flow control for a circuit.

In theory the use of flow control in the two directions on a line is independent. In practice we want to use it in both directions or in neither direction. To facilitate activating flow control in both direction at once the resync protocol has a third operation, the *sync both* packet, that acts like a *sync request* except that it also causes the downstream port to issue a *sync request* in the other direction on the same circuit. For efficiency, *sync request* and *sync both* actually can be applied to a range of VCI values, which range is carried in the packet. *sync request* and *sync both* packets carry a nonce token value which is returned in the matching *sync ack*.

To be effective, when activated flow-control should be used for all circuits in both directions on a line. There are a couple of exceptions, however. The credit resync VCI is never flow controlled, to spare us the complexity of having to resync the credits on the credit resync circuit. And the SNMP well-known circuit is not flow controlled, since it is used in deciding whether to activate flow control. To allow special circuits such as these to avoid flow control, part if a circuit's software state information in each port will specify either "no flow control" or "line flow control"; the latter, which is the default, means to flow control the circuit when the line supports flow control.

Some ATM designers express concern that our method of credit resync stops a circuit. Our experience is that the stoppage is for such a short time that the higher-level protocols on these ABR circuits, and their human users, are not disrupted. Non-stopping methods of credit resync are possible, but are more complex and seem to require hardware assistance.

Interactions with Connection Setup

Ideally, we would like to view the flow control state of a VCI on a line as the local concern of the upstream output port and the downstream input port. In this view the flow control state would be independent of the connections for the VCI to other lines; the VCI could be disconnected and reconnected without interacting with the flow control machinery. While this model holds at the downstream end, it cannot hold at the upstream end. The credit and ack machinery at the circuit's upstream output port must communicate with the connected input port in the same switch. In AN2 this communication is to a hardware *stopped bit* for each VCI at the input port: the *stopped bit* is true when the corresponding credit balance in the connected output port is zero, thus preventing the input port from forwarding further cells on the circuit. Something like the *stopped bit* is necessary and complicates the local model of flow control state.

The functioning of the *stopped bit* was described earlier in the hardware section. To review, when a credit balance goes from positive to zero a *stop* is communicated to the input port synchronously, over dedicated backplane wires. When a credit balance goes from zero to positive, however, a *start* must be communicated asynchronously via a pipelined channel with variable delay, since the *start* is the result of an ack arriving at the output port from the line, an event that is uncorrelated with cells being forwarded from the connected input port to this output port through the crossbar. Even though a *start* can take many cells times to arrive at the input port (although it usually arrives within one or two cell times), the correct relationship between the credit balance and the *stopped bit* is maintained by the hardware for VCIs that are connected. When the software reconnects VCIs, however, it must make sure that the *stopped bit* properly reflects the credit balance for the new connection. Similar concerns apply to stopping a circuit for credit resynchronization and turning flow control on and off on a line. These actions further complicated because the software cannot see *starts* that are in transit.

We have explored a variety of techniques for synchronizing the input port *stopped* bits with output port credit balances. The current choice works as follows. Connection setup proceeds in two steps: first a step at the input port and then a step at the output port. The initial state of an input VCI before connection setup is known as *grounded*. In the grounded state the input VCI forwards cells to a reserved output VCI that discards all cells and never generates *stop* or *start*, and the input VCI is not stopped. The initial state of an output VCI before connection setup is also known as grounded. A grounded output VCI never generates *stop* or *start*, discards all cells, but maintains a credit balance and accepts any credits that may arrive.

The connection setup step at the input port takes a grounded input VCI, sets its *stopped bit*, and adjusts the circuit state so that it will forward cells to the desired output port and VCI once the input VCI receives a *start*. However, nothing done to this point results in the input VCI receiving a *start*, so the input VCI will remain stopped and forward no cells. After the step at the input port is completed, the connection setup step at the output port takes a grounded output VCI and adjusts the circuit state so that the output VCI will send any *starts* it might generate back to the desired input port and VCI. However the output VCI remains in the state where it does not yet generate any *starts*.

At this point we have the connection almost complete, except that the input is stopped and the output is not generating *starts* or *stops*. Next we perform a sequence of operations at the output port to turn on the generation of *stops* and *starts*. This sequence results in one of three cases: (1) the balance at the output VCI is zero and no *start* has been sent to the input VCI; (2) the balance at the output VCI is greater than zero and a *start* has been sent to the input VCI; (3) the balance at the output VCI is greater than zero and no *start* has been sent to the input VCI. Note that in cases 1 and 2, no further actions need to be taken by the connection setup, because the *stopped bit* for the input VCI correctly reflects the output balance. In case 3, the input VCI needs to be started but no *start* will come from the output VCI, even if more credits arrive. In this case we *start* the input VCI manually.

The actual sequence of operations at the output port is in detail as follows. We zero a flag that is set by the hardware whenever the output VCI generates a *start*. Then we enable the generation of *stops* and then the generation of *starts*. Then we examine the output balance to see if it is zero. Then we examine the hardware flag to see if the output VCI has generated any *starts* since we reset the flag. Now we have enough information to determine which of the three result cases obtains. If a *start* has been generated, then case 2 obtains. Otherwise, no *start* was generated at least until after we examined the balance, and we must discriminate between cases 1 and 3. If the balance was zero, then case 1 obtains. Otherwise case 3 obtains, and we manually *start* the input VCI. by forging a *start*. Forging a start requires its own intricate sequence of operations which we do not cover here.

These details are included to give a flavor of the intricate actions often required where the abstractions of the switch software meet the reality of the switch hardware. There are many other similar examples in AN2, which we will not cover.

3.2.11 Schedule Setup

The schedule for CBR traffic at a switch is global to all line cards, since it must provide a conflict-free matching of input and output ports for each of the 1024 slots in a schedule frame. The Master Schedule Setup (MSS) module is responsible for computing a new schedule whenever the CBR demands on the switch change. In order for the switch to operate according to a particular schedule, however, that schedule must be distributed to all line cards. Once MSS has computed a new schedule it makes RRP calls to Local Schedule Setup (LCS) on each line card to deliver the needed slice of the schedule. LCS installs the new schedule slice in the unused half of the schedule memory and then returns. Once MSS has received a return from each line card it needs to cause all line cards to synchronously start using the new schedule. This schedule swap is implemented by using a special feature of the hardware – a wire to all line cards that causes them to swap schedule memory halves at the end of the next frame if a signal is present. MSS asserts a signal on the wire, being careful not to do so too close in time to the end of the current frame so as to give all line cards a chance to notice, and all line cards adopt the new schedule synchronously at the end of the current frame.

AN2 uses an innovative algorithm to compute the schedule. This algorithm is able to schedule 100% of the line bandwidth. It also produces a so called "balanced" schedule, in which the slots for each input port, output port, output head, and output circuit are evenly distributed over the frame. Even distribution minimizes the amount of speed matching buffers needed in the output ports, minimizes latency within the switch for each CBR circuit, and reduces the variance of latency for CBR circuits. Another paper [12] describes the schedule balancing algorithm. For our purposes here it is sufficient to note that schedule calculation can take a second or so for a fully loaded schedule. We don't want to limit the rate of schedule

changes to one circuit per second, so the AN2 software batches schedule changes. The first change is acted on immediately. Other changes that arrive while a schedule computation is underway are queued, to be acted upon as the next batch.

Scheduling point-to-multipoint circuits is problematical. All output ports for branches of such a circuit must be matched to the same input port in each of the circuit's schedule slots. So far, to accomplish this, the algorithm we have developed requires us to schedule every point-to-multipoint circuit as though it had branches at every output port of the switch. Thus, even if such a circuit uses only a few output ports, all appear occupied for that slot. Other CBR circuits cannot used the free output ports. Of course ABR circuits can use this free capacity.

Each AN2 switch operates according to its own internal clock. The switches in a network thus will have slightly different clock rates. Since the CBR schedule is advanced according to the local clock it is possible that the schedules on different switches will operate at slightly different rates. If a "fast" switch were forwarding cells for some CBR circuit to a "slow" switch at the downstream end of the line and the circuit were completely using all slots allocated to it, then the buffers at the slow switch would eventually overflow. To prevent this possibility we arrange for the first switch along a circuit's path allocate one fewer slot to the circuit than the others on the path. One out of 1024 more than compensates for any expected differences in clock speed. Thus later switches will never receive CBR cells at a rate higher than they can handle. The host adapter is also setup to deliver cells at a slightly slower maximum rate than the first switch, to prevent cells loss for that circuit at the first switch. But if the host misbehaves and overruns the first switch only that host is hurt, since the buffering that is overflowed is private to that host's line.

Interactions with Connection Setup

Batching of schedule changes generates some extra complexity for MSS in setting up CBR circuits. The AN2 hardware does not enforce buffer quota limits for individual CBR circuits. A incoming line from another switch can carry circuits for many different hosts. If one such CBR circuit used more than its share of buffers then others could find all CBR buffering for the line full and be forced to drop cells. If a CBR circuit were able to receive and buffer incoming cells before the schedule for it were installed, then this possibility could occur: cells would be buffered but could not be forwarded. To prevent buffer overflow MCS is careful to initially connect a CBR circuit with the *discard* bit of the in circuit state set. Only after the schedule containing the slots for the circuit is installed will MCS cause the *discard* bit to be cleared. This requirement causes an extra RRP call from MCS to SCS compared to the actions for setting up ABR circuits. But MCS does not want its client to have to wait for the potentially time consuming schedule computation to complete. Rather MCS will return to its client once the VCI's are assigned and quick checks verify that sufficient bandwidth is available. Thus it is possible that the client will call to change the allocation or disconnect the circuit before the scheduling is completed. To guard against such asynchronous requests causing confusion, MCS will delay these later client requests until the schedule computations associated with earlier requests are completed and the circuit state at the line cards involved is up to date.

3.3 Remarks on AN2 Software

So far about 10 person-years have been invested in the switch software. Its size and complexity will increase as the specifications of SNMP management, SVC circuit setup, and line rate control are finalized and the implementation of these functions completed.

Supplying AN2 line cards with highly capable processors and adequate memory has been a good method of reducing the capacity risk in the face of this functional uncertainty. The DECelx kernel has been stable, largely bug-free, and has offered adequate performance. Its general-purpose loading, debugging, communications facilities have smoothed software development.

From an operational perspective, the hardware feature we would most like to revisit it the absence of any chassis-centric stable storage for configuration data. Having all stable storage on line cards makes it

difficult to insulate network managers from unintended side effects when swapping line cards. The next version the switch will remedy this problem.

4. Conclusions

AN2 is quite different from most ATM switches that have appeared in the literature (see [10] for an excellent overview). Rather than using a high speed switching fabric with output buffering, we use a simple input buffered crossbar operating at wire speed. By segregating the buffers into per-output-port queues, we eliminate the undesirable head of line blocking effect that usually occurs in input buffered switches.

Parallel Iterative Matching provides an efficient method of scheduling traffic through the crossbar, and eliminates the need for a self-routing fabric. The use of credit-based hop-by-hop flow control for each ABR virtual circuit makes admission control, traffic shaping, and traffic policing unnecessary. Using flow control, it is possible to offer loads substantially in excess of the network's capacity with no risk of data loss. This is particularly valuable in a data network with long lived connections and bursty traffic. The use of time division multiplexing for CBR traffic provides guaranteed bandwidth and low jitter for applications that require it.

AN2's automatic reconfiguration makes it possible to construct networks that are robust in the presence of hardware failures. Resilient virtual circuits and LAN emulation will ease the transition from packet switched to circuit switched networks, by making it unnecessary for applications to be aware of the new properties of the underlying network.

While adhering to emerging ATM standards, AN2 adds capabilities and features that enhance the performance, predictability, and robustness of local-area ATM networks.

Acknowledgments

A number of individuals contributed to the design of the AN2 hardware. Greg Waters, Mark Levesque and Hal Murray implemented the QLC. Hans Eberle, Chuck Jerian, and Didier Roncin implemented many of the FPGAs. The physical design of the printed circuit boards and packaging were done by Jim Collias and Steve Jeske. The software team included Eric Muller, Ricardo Sanchez, Tom Rodeheffer, and Mike Burrows from SRC, plus Kovaali Kumar, Brendan Hannigan, Aleksey Romanov, and Martin Griesmer of Digital's Networks Engineering Group.

References:

[1] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High Speed Switch Scheduling for Local Area Networks. In *Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 98-110, Boston Massachusetts, ACM and IEEE Computer Society, October, 1992.

[2] Thacker, Charles P. United States Patent 5,267,235: Method and Apparatus for Resource Arbitration. November 30, 1993.

[3] The Programmable Logic Data Book. Xilinx Corp., San Jose California, 1993.

[4] G. Minden, J. Evans, D. Petr, and V. Frost. An ATM WAN/LAN Gateway Architecture. In *Proc. 2nd IEEE Symposium on High Performance Distributed Computing*, pp. 136-143. July, 1993

[5] PM5345 Saturn User Network Interface. PMC - Sierra, Inc., Burnaby BC Canada, 1993

[6] H.T. Kung and R. Morris, "Credit-Based Flow Control for ATM Networks", *IEEE Network Magazine*, March 1995.

[7] DECelx Programming Guide. Order number AA-PMRUB-TE Digital Equipment Corporation, Maynard, Mass., 1992

[8] ATM User-Network Interface Specification Version 3.0. The ATM Forum.

[9] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. P. Thacker . "Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links" IEEE Journal on Selected Areas in Communications, Oct 1991

[10] Partridge, Craig *Gigabit Networking*. Addison-Wesley Publishing Company, Reading, Mass., 1994

[11] T. Rodeheffer and M. Schroeder, "Automatic Reconfiguration in Autonet", Proc. 13th ACM Symp. on Oper. Sys. Principles, p. 183 - 187, 1991.

[12] J.Saxe and T. Rodeheffer, "Smooth Scheduling in a Cell-Based Switching Network", in preparation.