A Perspective on AN2: Local Area Network as Distributed System

Susan S. Owicki

Systems Research Center
Digital Equipment Corporation
130 Lytton Avenue
Palo Alto CA 94301

Abstract

AN2 is a switch-based local area network under development at the Digital Equipment Corporation's Systems Research Center. A network is typically viewed as part of the infrastructure that enables distributed computing. But AN2 and its predecessor AN1 are actually distributed systems in their own right. Hardware and software at the switches operate on local data and cooperate with parallel activities at other switches to manage the network and transmit data efficiently. Network components may fail, so fault tolerant operation is essential. Thus many of the issues and techniques of more traditional distributed systems are relevant for AN2. This paper describes aspects of the AN2 design where the distributed system model is especially appropriate and identifies areas where further work is needed.

1 Introduction

In the past few years, researchers at Digital Equipment Corporation's System Research Center (SRC) have developed two local area networks, AN1 (formerly known as Autonet) and AN2. In both networks, data is transmitted between hosts through a sequence of switches connected by full-duplex links. The switches can be connected in an arbitrary topology; network software detects the connection pattern and determines the paths to be used in routing data between hosts. AN1 has been in operation since early 1990, supporting over 100 workstations at SRC. A prototype version of AN2 should be deployed at SRC in late 1993.

AN1 was designed to provide the same service as ethernet, transmitting variable-length packets between host computers, but with higher bandwidth and high availability. AN1 supports a link bandwidth of 100 Mbit/sec. Aggregate network bandwidth can be much higher, since multiple transmissions can be active simultaneously on different links. The switch is designed to keep latency low. A packet can be routed as soon as its header has been received. In the absence of contention, the first bit of a packet leaves the switch 2 microseconds after it arrives. To provide high availability, fault monitoring hardware and software detect switch and link failures. When a problem is detected, an automatic reconfiguration is triggered to determine new routes between hosts. The automatic reconfiguration is highly effective. A favorite AN1 demo is pulling the plug on an arbitrary switch in SRC's main LAN. The network reconfigures in less than 200 milliseconds, and users see no service interruption.

Figure 1 illustrates a possible AN1 configuration; an AN2 configuration is quite similar. The network consists of hosts and switches connected by links. Each switch has 12 *ports*, each of which may be connected to a host or to the port of another switch. Each host has a controller which serves as its interface to the network. The two grey arrows in the figure represent packets in transit; they illustrate

12th ACM Symposium on Principles on Distributed Computing, Ithaca NY

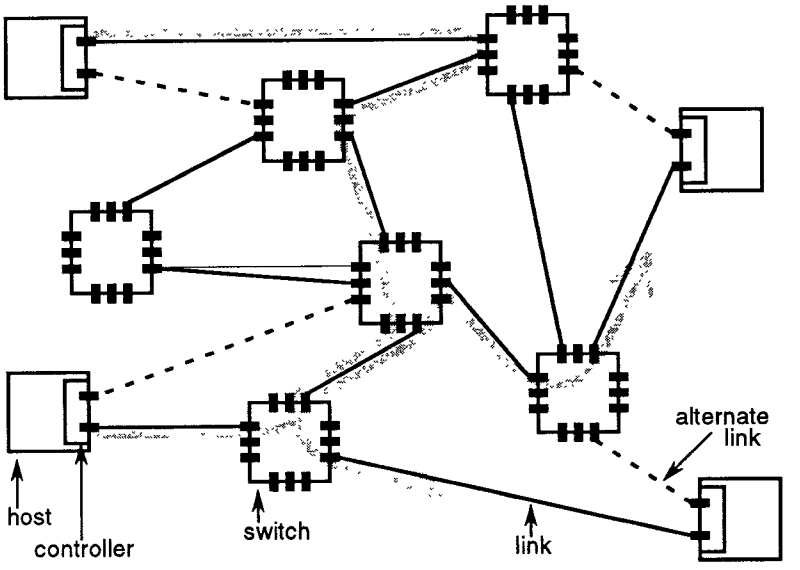

Figure 1: A sample AN1 installation

how packets are routed from one host through a sequence of switches to another host. Redundant connections make the installation fault tolerant. Each host has links to two different switches. Only one link is in active use at any time; the other is an alternate to be used if the first fails. In addition, there are multiple paths between each pair of hosts.

AN2 retains the high availability of AN1. Link bandwidth is higher, at 622 megabits-per-second (155 megabit-per-second links are also provided, e.g. for connecting a host to a switch). In addition, AN2 is compatible with the ATM Forum standard. The most obvious impact of supporting this standard is that the network traffics in *cells* consisting of 48 bytes of data and a 5-byte header. Using fixed-length cells makes it easier to build high-speed switches and support bandwidth reservations, as described below. However, it is more convenient for host software to deal with larger data units, such as the variable-length packets supported by ethernet and AN1. In AN2 a host presents packets to its controller, which disassembles them into cells to transmit to the network. The controller at the receiving host will re-assemble the cells into packets.

AN2 supports two classes of traffic. A guaranteed traffic stream (Continuous Bit Rate stream in ATM terminology) is assured of receiving a specified bandwidth with bounded delay and jitter. It is well suited to transmitting multi-media data. Guaranteed traffic streams require advance setup to reserve the resources to meet their requirements. In contrast, *best-effort* traffic (Variable Bit Rate traffic) requires no setup and receives no guarantee of service. It is transmitted as quickly as network resources allow. File transfers and remote-procedure call are examples of applications where best-effort scheduling is most appropriate.

Routing in AN2 is based on *virtual circuits*. For our purposes here, a virtual circuit represents a stream of cells to be transmitted between a pair of hosts. (There are also multicast virtual circuits, but they will not be discussed here.) Associated with a virtual circuit is a path, i.e., a sequence of switches, through which cells in the stream are routed. There may be multiple virtual circuits connecting a pair of hosts, for example one for best-effort traffic and several for guaranteed streams with separate bandwidth requirements.

The header of each cell contains its virtual circuit id. When a cell for an established circuit arrives at a switch, the virtual circuit id is looked up in a routing table to obtain the output to which the cell

should be transmitted. Transmission from input to output takes place across a 16×16 crossbar. The crossbar operates synchronously, routing up to 16 cells in parallel during each time slot. The entire process is accomplished in hardware, and the cell's delay across the switch is only 2 microseconds, so long as there is no contention for the output.

The choice of a crossbar for the switch's internal fabric reflects a decision about the appropriate size for a switch in a local area network. A number of alternatives have been proposed in the literature [Ahmadi & Denzel 89]. The crossbar has low latency compared to a multi-stage fabric like a banyan, and this is the reason it was chosen for AN2. Crossbars do not scale well, however: their complexity grows as N^2 for an $N \times N$ switch, while a banyan grows as $N \log N$. However, a 16×16 or even 32×32 crossbar has quite acceptable cost, and this size is well-suited to local area networks.

In addition to the crossbar, an AN2 switch contains up to 16 line cards, each connecting the crossbar to one 622 megabit-per-second port or four 155 megabit-per-second ports. (The crossbar has 16 inputs and 16 outputs, and each line card connects to one input and one output, to support the full-duplex links.) The line card contains a processor, buffers for incoming cells, memory for routing tables, logic for buffer and crossbar management, and optical devices for one fast port or four slow ones.

Although most cells are processed completely by hardware, software running on the line-card processors handles exceptional situations, such as faults, network reconfiguration, and virtual circuit setup. In the initial deployment of AN2, planned for late 1993, the software will be very similar to that in AN1. Later versions of the software will take advantage of AN2 features to provide greater functionality and performance. A number of proposed extensions are mentioned below; some are well understood, while others require further research.

Both AN1 and AN2 can have many of the characteristics of distributed systems, including parallel computation, asynchronous operation, fault-tolerance, and data locality. This is most obvious when one considers the switches as nodes in a distributed system. The switches must cooperate to route messages during normal operation, to monitor links for errors, and to reconfigure the network in response to failure. In addition, the internal operation of the switch is distributed in nature: the line cards operate in parallel and have limited bandwidth for communication with each other. The remainder of this paper discusses some of the distributed algorithms used within a switch and between switches. Section 2 describes how the switches cooperate to monitor for errors and to obtain routing information during network configuration. Sections 3 and 4 discuss the internal operations of the switch while handling best-effort and guaranteed traffic, respectively. (Some cooperation between switches is also required with guaranteed traffic.) Finally, Section 5 presents the inter-switch techniques used for flow-control and deadlock prevention.

Metanet [Cidon & Y.Ofek 90, Ofek & Yung 90] is a network design with many of the same goals as AN1. However, it is based on an underlying ring network, and the resulting structure is quite different.

2 Configuration and Routing

A cell entering AN2 is routed from switch to switch until it reaches its destination. The cell reaches a switch through a port connected to one of the switch's 16 line cards. The line card contains a routing table that maps the cell's virtual circuit id to the port on which the cell should leave the switch. In networks with a fixed topology, like hypercubes or banyans, routing can be "wired in" to the switches. In AN2, since the topology is arbitrary, the routing tables must be built dynamically.

The first stage in generating routing tables is topology acquisition. A distributed *reconfiguration algorithm* is run to detect the current topology and communicate it to each switch. Reconfiguration is triggered when a switch is booted, or when any switch detects a change in the state of its inter-switch connections, such as failure, recovery, or creation of a new connection. State changes in host links do not trigger reconfiguration. Automatic reconfiguration simplifies network administration and provides high availability, so long as there are enough redundant connections that a failure does not partition the network.

After topology acquisition, virtual circuits can be set up and appropriate entries placed in the routing tables. The mechanisms for circuit setup are different for guaranteed and best-effort traffic; they are discussed below. Both reconfiguration and circuit setup involve software running on the line card processors.

The reconfiguration algorithm for the first release of AN2 is much the same as in AN1. Initially, each node knows the identity of its neighbors; this information can be obtained by sending a query out each port. At the end of execution, each node knows the full network topology. The technique, resembles other distributed algorithms for acquiring global state information. The first algorithm presented assumes that a reconfiguration is not triggered while another reconfiguration is still under way. A subsequent modification handles overlapping reconfigurations.

The algorithm consists of three phases.

1. In the *propagation phase*, a spanning tree is built as follows. The initiator of the reconfiguration, that is, the switch that detected a state change, becomes the root; it invites each of its neighbors to join the tree. A node invited to join the tree accepts the first invitation it receives, becoming the child of the inviter. It then invites all its other neighbors to join the tree. Any invitations received after the first are declined. Each invitation is acknowledged with an indication of whether it was accepted or declined. At the end of this phase, each switch knows its parent and children.

2. In the *collection phase*, topology information is passed up the tree to the root. At the end of this phase, the root knows the complete topology.

3. In the *distribution phase*, the complete topology is passed down the tree. At the end of this phase, each switch knows the full topology.

The tree produced in this way is a *propagation-order* spanning tree. In the worst case, the tree could be linear, and there would be no parallelism during execution of the algorithm. It has been observed in practice, however, that the first invitation a switch receives usually comes from one of the set of neighbors closest to the root. Thus the tree obtained is usually very close to a breadth-first tree, yielding high parallelism.

With the algorithm above, if a second reconfiguration was triggered before the first completed, switches might end up disagreeing about the state of links, and so about the network's topology. To ensure that the results are consistent when configurations overlap, each reconfiguration message is tagged with an *epoch* number and the id of the initiating switch. Each switch maintains a copy of the largest tag it has seen, where the ordering is based first on epoch number and then on switch id. When a switch initiates a configuration, it uses an epoch number one greater than the one in its stored tag. When a switch receives an invitation to join a configuration tree, it ignores it unless the message tag is larger than its currently stored value. In that case, it aborts its activity in the earlier configuration and joins the new one. Thus a switch that sees multiple configurations participates in the one with the largest tag and eventually ignores all others.

The reconfiguration algorithm assumes that each link is unambiguously "working" or "dead". In reality, however, a faulty link may exhibit intermittent failures. In order to provide a clean abstraction to the reconfiguration algorithm, switch software monitors the links by regularly pinging each neighbor and checking that a correct acknowledgment is received. If this test fails too frequently, a working link is changed to the dead state. Likewise, a dead link's state makes the transition to working if its error rate is acceptably low for a long enough time. Care must be taken that an intermittent fault does not cause a link to make frequent transitions between the two states, for each transition would trigger a reconfiguration, and too-frequent reconfigurations can keep the network from providing service. To prevent this, a *skeptic* module in the software monitor retains a history of a link's failures and recoveries. If failures recur, the skeptic requires an increasingly long period of correct operation before the link is considered to be recovered.

Once the reconfiguration has been completed, routing tables acquire entries as virtual circuits are set up. Circuit setup will be done in accordance with the ATM Forum signaling protocol. When a new virtual circuit is to be created, a cell contain-

ing the ids of the source and destination hosts is sent along a separate signaling circuit. When this cell arrives at a switch, it is passed to the processor on the line card where it arrived. Software there chooses the outgoing port for the circuit (based on the topology information obtained during reconfiguration) and adds the virtual circuit to the line card's routing table. Cells for the new virtual circuit may be sent immediately after the setup cell. If they arrive at a switch before the virtual circuit is established there, they will be buffered until the routing table entry is filled in. The flow-control scheme described in section 5 prevents this from leading to buffer overflow. All cells after the setup cell can be routed in hardware without involving the processor. Virtual circuit setup for guaranteed traffic will be discussed in Section 4.

Further improvements in the reconfiguration and routing techniques are contemplated for later versions of the switch software. The first is aimed at reducing the disruption caused by reconfiguration. In AN1, all switches must collaborate in a reconfiguration, and all packets in transit are dropped when a reconfiguration begins. This is acceptable in small networks, but is unattractive for networks containing thousands of switches. Fortunately, it should often be possible to restrict participation to switches "near" the failing component, and to drop cells only when the path of their virtual circuit goes through a failed link. In this case, the virtual circuit can be rerouted by sending a new circuit setup cell from the point where the path was broken.

A second optimization allows reclamation of resources, such as buffers, that are associated with an idle virtual circuit. Switch software could "page out" a circuit by releasing its buffers, removing it from the routing table, and notifying the downstream switch of this action. The downstream switch could then page it out as well. If further cells for the circuit subsequently arrived, it could be "paged in" by generating a setup cell to recreate the circuit.

So far we have seen that a virtual circuit may be rerouted to bypass a failed link or to reclaim resources. A more speculative option is to reroute circuits to balance the load on the network. The mechanics of rerouting are no more difficult in this case than in the earlier ones. However, algorithms to determine when and where circuits should be moved have yet to be considered.

3 Best-effort Traffic

The previous section described the routing of best-effort traffic between switches. We now turn to its treatment within a switch, discussing buffer organization and *parallel iterative matching*, an algorithm that allows the AN2 switch to deliver both low latency and high bandwidth.

Buffer organization has a major impact on switch throughput. The simplest approach is a FIFO queue of cells at each input; only the first cell in the queue is eligible for transmission across the switch. The difficulty with FIFO buffers arises when the cell at the head of an input queue is blocked because another input has a cell for the same output. All cells behind the first are blocked as well, even if the output they need is available. This is called head-of-line blocking. Karol et al. [Karol et al. 87] have shown that head-of-line blocking limits switch throughput to 58% of each link, when the destinations of incoming cells are uniformly distributed among all outputs.

There are a number of proposals for alternatives to FIFO input buffers [Huang & Knauer 84, Karol et al. 87, Karol et al. 92]. A common approach is to increase the bandwidth of the internal switching fabric (i.e. the crossbar in AN2) by a factor of k, typically by replicating the fabric k times. This allows up to k cells to be transmitted to the same output link in one slot. Since only 1 cell can can depart from an output during each slot, buffers are required at the output line cards, hence this technique is called *output buffering*. The value chosen for k in most implementations is small compared to switch size. It can be shown that such switches have good throughput when arriving cells have independent and uniformly distributed output ports. Traffic in a local area network is not likely to satisfy this assumption, however. This, plus the cost and complexity of the high-bandwidth internal fabric, made output buffering unattractive for AN2.

Instead, the AN2 switch avoids the head-of-line blocking problem by using random-access input buffers. Cells that cannot be forwarded in a time

slot are retained at the input in a queue associated with their virtual circuit. The first cell of any queued virtual circuit can be selected for transmission across the switch. Thus a cell is only blocked if its desired output is busy.

With random access buffers, each input may have cells for several different outputs. At each time slot, some pairing of inputs and outputs must be determined such that each input is paired with at most one output, and vice versa, considering only those pairs with a queued cell to transmit between them. This bi-partite matching problem must be solved every time slot, in the half microsecond required to transmit a cell.

The AN2 parallel iterative matching algorithm uses parallelism, randomness, and iteration to accomplish this goal. It is indeed a distributed algorithm, for the processing takes place in parallel at the line cards, with limited communication between them. The algorithm operates by repeating the following three steps (initially, all inputs and outputs are unmatched):

1. Each unmatched input sends a request to *every* output for which it has a buffered cell. This notifies an output of all its potential partners.

2. If an unmatched output receives any requests, it chooses one *randomly* to grant. The output notifies each input whether its request was granted.

3. If an input receives any grants, it chooses one to accept and notifies that output.

The request/grant/accept signals are sent on dedicated wires, one in each direction between each input and output.

Each of these steps occurs independently and in parallel at each input/output port; there is no centralized scheduler. The algorithm ensures that the matching obtained is legal. More than one input can request the same output. The grant phase chooses among them, ensuring that each output is paired with at most one input. More than one output can grant to the same input (if the input made more than one request). The accept phase chooses among them, ensuring that each input is paired with at most one output.

While the matching obtained after one iteration is legal, there may remain unmatched inputs with queued cells for unmatched outputs. An output whose grant is not accepted may be able to be paired with an input, none of whose requests were granted. To address this, the request, grant, accept protocol is repeated, retaining the matches made in previous iterations. Iteration "fills in the gaps." There can be no head-of-line blocking, since all potential connections are considered at each iteration.

If the above steps are repeated until no more matches are obtained, the result is a maximal matching. In the worst case this requires N iterations for an $N \times N$ switch, with one match being formed on each iteration. It can be proved, however, that the average time to find a maximal match is bounded by $\log_2 N + 4/3$, or 5.32 for the AN2 switch. This result is independent of the arrival patterns of cells, due to the random choice in step 2. In fact, simulations show that a maximal match is found within 4 iterations more than 98% of the time. Because of its time limit, AN2 uses just three iterations of parallel iterative matching.

Of course, a maximal match may be substantially smaller than the maximum obtainable. Why not implement a maximum matching algorithm instead? The simplest answer is that we don't know of a fast enough algorithm for maximum matching. Besides, maximum matching can lead to starvation. For example, suppose input 1 consistently has cells for outputs 2 and 3, and input 4 consistently has cells for output 3. The maximum match always pairs input 1 with output 2 and input 4 with output 3, and the virtual circuit between input 1 and output 2 will be starved. In contrast, the randomness in parallel iterative matching protects against starvation.

Simulation studies show that, for a 16×16 switch and a variety of cell arrival patterns, random-access input buffers plus parallel iterative matching yield throughput and latency nearly as good as that of output queueing with $k = 16$ and unbounded buffer capacity. Thus its performance is close to the maximum attainable in the absence of advance knowledge of traffic demands.

4 Guaranteed Traffic

AN2 switches handle guaranteed traffic quite differently from best-effort traffic. With best-effort traffic, parallel iterative matching is used at each time slot to dynamically schedule the cells to be transmitted over the crossbar. With guaranteed traffic, the requirements of each virtual circuit are specified when the circuit is set up. Using this information, the switch creates a schedule for moving guaranteed traffic across the crossbar, giving the required bandwidth to each virtual circuit.

Guaranteed virtual circuits are set up when an application attempts to reserve some bandwidth between a pair of hosts. Bandwidth reservations are based on *frames* of 1024 cell slots. Thus an application expresses its bandwidth request as some number of cells/frame.

The request to reserve bandwidth is processed by a network service called "bandwidth central". The name is misleading – network central might well be implemented in a distributed fashion. For the first realization of AN2, however, network central resides at a single switch, chosen during reconfiguration. Because it resolves all bandwidth requests, it knows the unreserved capacity of each link in the network. A new request is granted if there is a path between source and destination on which each link has enough unreserved bandwidth. Otherwise, the request must be denied. Bandwidth central chooses the route for the new virtual circuit if more than one possibility exists. A discussion of some heuristics for route selection, in the context of the Paris network, can be found in Awerbuch et al. [1990].

Once bandwidth central has selected a route, it informs the switches involved of the new reservation, so that they can revise their frame schedules. Figure 2 shows an example of such a schedule. Note that it indicates, for each slot and each input, what output (if any) receives a cell from that input in that slot. Suppose that the switch in Figure 2 is notified that it must add a one cell/frame reservation between input 4 and output 3. Such a reservation does not exceed the capacity of any link, but there is no way to fit it into the existing schedule. Fortunately, the Slepian-Duguid theorem [Hui & Arthurs 87] implies that a schedule can be found for any set of reservations that does not over-commit the

Reservations (cells per frame)

Input	Output 1	2	3	4
1		1	1	1
2	2			
3		2		1
4	1		1	

Schedule

Slot 1	$1 \rightarrow 3$	$2 \rightarrow 1$	$3 \rightarrow 2$	
Slot 2	$1 \rightarrow 4$	$2 \rightarrow 1$	$3 \rightarrow 2$	$4 \rightarrow 3$
Slot 3	$1 \rightarrow 2$		$3 \rightarrow 4$	$4 \rightarrow 1$

Figure 2: Guaranteed Traffic: Reservations and Schedule

bandwidth of any link. Moreover, the proof of the theorem provides an algorithm for adding a cell to an existing schedule; the time required is linear in the size of the switch and independent of frame size.

The algorithm works as follows. Suppose a reservation is to be added for a cell from input P to output Q. If there is a slot in the schedule where both input P and output Q are free, the reservation can be added to that slot. Otherwise, there is at least one slot where P is unreserved, and a different slot where Q is unreserved. Call them p and q respectively. (We know such slots exist, because bandwidth central has determined that the new reservation does not over-commit any link.) Add the reservation $P \rightarrow Q$ to one of these slots, say p. This will cause a conflict with an existing connection in that slot, say from R to Q. Move the conflicting connection into slot q. There it may conflict with another item; if so, move the conflicting item to slot p. Repeat the process until no conflicts remain; this will require at most N steps for an $N \times N$ switch. Thus adding a reservation for k cells takes at most $N \times k$ steps.

Figure 3 illustrates the steps involved in adding a reservation for a cell from input 4 to output 3 in the schedule in Figure 2. In the figure, the changes at each step are italicized, and the conflicting reservation, if any, is in boldface. In this case the algorithm terminates after three steps.

Once the schedule has been revised, it controls

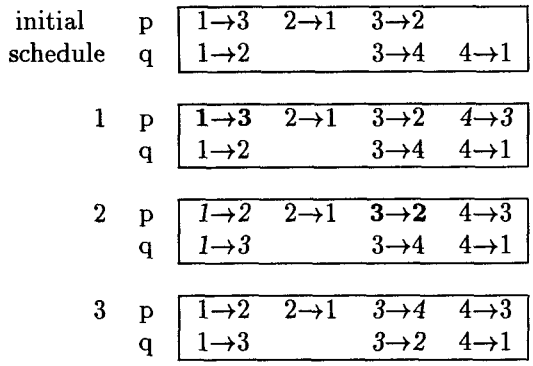

Figure 3: Adding the Reservation 4→3 to an Existing Schedule

the transmission of guaranteed cells. Best-effort cells can be scheduled (by parallel iterative matching) during slots not used by guaranteed traffic. For example, in Figure 2, a best-effort cell can be transmitted from input 2 to output 3 during the third slot. In addition, best-effort cells can use an allocated slot if no cell from the scheduled virtual circuit is present at the switch.

Separate buffer pools are maintained for guaranteed and best-effort traffic. The number of buffers needed to avoid dropping guaranteed cells depends on the characteristics of the network. In a synchronized network like the telephone network, a global clock keeps all switches operating at the same rate. In this case, the number of cell buffers needed at each line card is twice the frame size. Buffers for a single frame are not enough, because neither the frame boundaries nor the transmission order is the same at both switches, and because the switches can rearrange their schedules from one frame to the next.

In a network like AN2 with no global synchronization, buffer requirements depend on network parameters like network diameter, link and switch latency, and the variation in switch clock rates. For a typical local area installation, four frames worth of buffers are sufficient.

Next, let us consider the latency bounds for guaranteed traffic. It turns out that for both synchronous and asynchronous networks, the time for a guaranteed cell to reach its destination is at most $p \times (2f + l)$, where p is the path length, f is the frame time, and l is the maximum link latency. This bound is derived from the buffer requirements in a straightforward way in the synchronous case. In the asynchronous case, the derivation is subtle; its basis is the fact that a cell delayed for a long time in one switch cannot be very much delayed in later switches.

With 1 gigabit-per-second links, it takes less than half a millisecond to transmit a frame. Thus the latency and jitter of a guaranteed cells is less than 1 millisecond per switch. This should be quite satisfactory for multi-media applications. In contrast, a best-effort cell on a lightly loaded network should experience only a 2 microsecond delay at each switch. In a heavily loaded network, however, queueing delays could make best-effort cell latency arbitrarily large.

Later versions of the AN2 switch may provide improvements to the guaranteed-bandwidth service. One area to be explored is greater flexibility in frame size. Large frames are attractive because they provide a fine-grained allocation unit, but small frames yield better latency and jitter bounds. Nested frames could provide the benefits of both. For example, allocation could be based on 1024-slot frames, with cell re-ordering restricted to 128-slot units. Such a change would require a more sophisticated algorithm for building frame schedules.

Another potential improvement is arranging the frame schedule to give better service to best-effort traffic. Best-effort cells can only be transmitted in slots where neither their input nor their output is busy with reserved traffic. Such slots will be more frequent if reserved traffic is packed into a small number of slots, leaving other slots completely free for best-effort traffic. Best-effort cells will also fare better if the unreserved slots are distributed throughout the frame rather than grouped at one point. Finding the best way to arrange the frame schedule is a matter for further study.

5 Flow-control and Deadlock

We have seen that cells from both traffic classes are buffered in the line card at which they arrive. If cells arrive more quickly than they can depart, e.g. because there is contention for the required output link, then the buffers may overflow. There are three

common ways of dealing with this problem:

- match transmission rate with reserved bandwidth so that buffer capacity is never exceeded,
- use flow-control, that is, some form of feedback that inhibits message transmission when the buffer is in danger of overflowing, or
- drop messages when buffer capacity is exceeded. If messages are dropped, they are typically retransmitted by higher levels of the system.

In AN2, guaranteed traffic uses the first of these approaches. The network controller prevents a host from sending more than its reserved bandwidth on a guaranteed virtual circuit. With this restriction, the buffer sizes given in Section 4 will always be sufficient.

For best-effort traffic, AN2 uses flow control based on *credits*. Figure 4 illustrates the protocol. Buffers for each best-effort virtual circuit traversing the link are allocated at the downstream switch. The upstream switch maintains a credit balance for buffers in the downstream switch; this is the number of buffers known to be empty. Whenever the upstream switch sends a cell, it decrements the balance for the corresponding virtual circuit. Whenever a cell buffer is freed in the downstream switch, due to the successful forwarding of a cell through the crossbar, a credit is transmitted back to the upstream switch, and the credit balance for the circuit is incremented. Cells are only transmitted for circuits with non-zero credit balances.

The credit-based scheme is robust in the face of lost flow-control messages. With credits, a lost message can only cause reduced performance. Performance can be regained by having the upstream switch periodically trigger a resynchronization of credits. Devising the re-synchronization protocol is in itself an interesting problem in distributed computing, but we will not cover it here.

Using flow-control to prevent buffer overflow introduces the possibility of deadlock. A cell effectively holds a buffer at the upstream switch while attempting to acquire one at the downstream switch. With AN1's FIFO buffers, if the first packet in the queue is blocked, the entire link is blocked as well. If a cycle of blocked links could arise, where each link has a packet waiting for a buffer in the next link, then deadlock could occur. In AN1 this possibility is prevented by restricting the possible routes for packets.

The rules for route restriction are based on the spanning tree formed during reconfiguration. Each link in the network is assigned an orientation, with *up* being toward the root of the tree. (If the two ends of the link are at the same level in the tree, then up is toward the higher-numbered switch.) Messages are only routed on *up*/down** paths, i.e. paths in which no traversal down a link is followed by an upward traversal. This restriction is sufficient to prevent cycle formation and thus to prevent deadlock. Up*/down* routing may eliminate some potential routes and thus have a negative effect on performance. The impact depends on both the topology and the workload. However, it has the advantage of requiring minimal buffer space, which was important at the time when AN1 was developed.

By the time AN2 was designed, however, memory prices had fallen, and the cost of large buffers was acceptable. In AN2, each best-effort virtual circuit that passes through a link has its own set of buffers in the downstream line card. The buffers for different virtual circuits are independent, so if one virtual circuit is blocked, other virtual circuits passing over the same link are not affected. Since the links of a single virtual circuit can not form a cycle, deadlock cannot occur.

This approach prevents deadlock even if each virtual circuit is allocated only one buffer. A larger allocation is needed for good performance, however. Suppose that a virtual circuit encounters no contention for the links on its route. The circuit should be able to transmit at the full link rate, which would be impossible if the upstream switch on a link ever ran out of credits. To guarantee that it never does, it must start with enough credits to cover a round-trip on the link; this allows time for the cell to reach the downstream switch and a credit to be returned. Thus enough buffers are needed for each virtual circuit to hold as many cells as can be transmitted in one round-trip time on the link. With 1000 virtual circuits per link and a maximum link length of 10 km, the required memory costs much less than the opto-electronics in the line card.

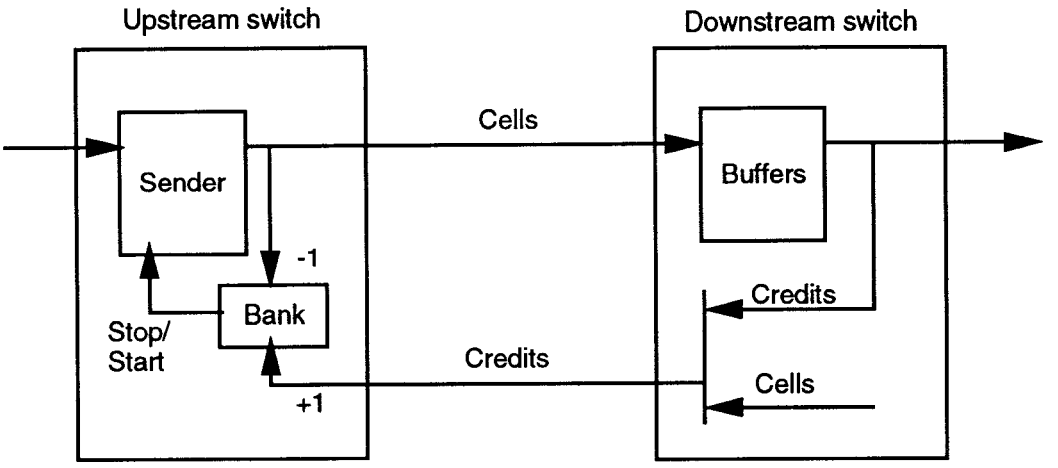

Figure 4: Flow Control for Best-effort Traffic

The initial AN2 implementation statically allocates this number of buffers to each best-effort virtual circuit. For a lightly-used circuit, this may be more buffers than necessary. More sophisticated schemes, such as dynamically altering buffer allocation based on use, may be considered later. This could allow the link to support more virtual circuits without adversely affecting performance.

6 Conclusion

The ideas presented here are explored more thoroughly in other AN1/AN2 papers. Schroeder et al. [1991] cover most of the features of AN1, including the reconfiguration algorithm and up*/down* deadlock avoidance. Rodeheffer and Schroeder [1991] describe AN1 error monitoring and recovery; similar techniques are used in AN2. Owicki and Karlin [1992] assess the performance impact of some design decisions in AN1. Anderson et al. [1992] motivate a number of the choices made in the AN2 switch and present the switching techniques for guaranteed and best-effort traffic. Finally, Goguen [1992] gives an overview of AN2, including flow control, fast circuit setup, and reconfiguration.

SRC is a laboratory with considerable interest and experience in building distributed systems, and that has undoubtedly shaped the design of AN1 and AN2. The inter-switch interactions are much like those in any distributed system, with problems of synchronization, fault-tolerance, and data locality to be resolved. The reconfiguration algorithm presented here is a prime example. Even within the switches, there are interesting distributed problems, such as scheduling the crossbar. Here fault-tolerance and synchronization are not at issue, but exploiting parallelism and dealing with data locality remain important.

It is striking how much theoretical work has gone into the development of AN1 and AN2. This is most obvious in the algorithms, including reconfiguration, parallel iterative matching, Slepian-Duguid, virtual circuit establishment. Also important have been various sorts of theoretical analyses. The reconfiguration algorithm, in particular, benefited from program verification; flaws in several early versions were discovered during that process. Determining the buffer requirements for guaranteed traffic required some subtle reasoning and careful abstraction from the concrete network behavior. In contrast, the decision to use very large buffers for best-effort traffic was a triumph of practical engineering. The blend of both kinds of insight has been important to the success of AN1 and AN2

This paper has described a number of the solutions devised for AN2 as well as a number of areas where more work is needed. I hope the PODC community will be moved to investigate some of these

areas, both because they are inherently interesting problems and because the results could enhance the quality of networks like AN2.

Acknowledgments

Many people have contributed to AN1 and AN2. In addition to the authors of the AN1/AN2 papers cited here, the following people have played a role in its development: Chuck Jerian, Jim Collias, John Dillon, Hans Eberle, Manolis Katevenis, Leslie Lamport, Butler Lampson, Bill Ramirez, and Herb Yeary.

References

[Ahmadi & Denzel 89] Ahmadi, H. and Denzel, W. A Survey of Modern High-Performance Switching Techniques. *IEEE Journal on Selected Areas in Communications*, 7(7):1091–1103, September 1989.

[Anderson et al. 92] Anderson, T., Owicki, S., Saxe, J., and Thacker, C. High Speed Switch Scheduling for Local Area Networks. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 98–110, October 1992.

[Awerbuch et al. 90] Awerbuch, B., Cidon, I., Gopal, I., Kaplan, M., and Kutten, S. Distributed Control for Paris. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 145–160, August 1990.

[Cidon & Y.Ofek 90] Cidon, I. and Y.Ofek. MetaRing – A full-Duplex Ring with Fairness and Spatial Reuse. In *Proceedings of INFOCOM '90*, 1990.

[Goguen 92] Goguen, M. AN2: a Self-Configuring Local ATM Network. In *Proceedings of the National Communications Forum (NCF) 92*, 1992.

[Huang & Knauer 84] Huang, A. and Knauer, S. Starlite: A Wideband Digital Switch. In *Proceedings of GLOBECOM '84*, pages 121–125, December 1984.

[Hui & Arthurs 87] Hui, J. and Arthurs, E. A Broadband Packet Switch for Integrated Transport. *IEEE Journal on Selected Areas in Communications*, 5(10):1264–1273, October 1987.

[Karol et al. 87] Karol, M., Hluchyj, M., and Morgan, S. Input Versus Output Queueing on a Space-Division Packet Switch. *IEEE Transactions on Communication*, 35(12):1347–1356, December 1987.

[Karol et al. 92] Karol, M., Eng, K., and Obara, H. Improving the Performance of Input-Queued ATM Packet Switches, . In *Proc. INFOCOM '92*, pages 110–115, May 1992.

[Ofek & Yung 90] Ofek, Y. and Yung, M. Principles for High Speed Network Control: Losslessness and deadlock-freeness, self-routing and a single buffer per link. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 161–175, August 1990.

[Owicki & Karlin 92] Owicki, S. and Karlin, A. Factors in the Performance of the AN1 Computer Network. In *Proceedings of the 1992 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 167–180, June 1992.

[Rodeheffer & Schroeder 91] Rodeheffer, T. and Schroeder, M. Automatic Reconfiguration in Autonet. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 183–197, October 1991.

[Schroeder et al. 91] Schroeder, M., Birrell, A., Burrows, M., Murray, H., Needham, R., Rodeheffer, T., Satterthwaite, E., and Thacker, C. Autonet: a High-speed Self-configuring Local Area Network Using Point-to-point Links. *IEEE Journal on Selected Areas in Communications*, 9:1318–1335, October 1991.