Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing

H. T. Kung[1], Trevor Blackwell[1, 2] and Alan Chapman[2]

[1]Division of Applied Sciences, Harvard University, 29 Oxford Street, Cambridge, MA 02138, USA

[2]Bell-Northern Research, P.O.Box 3511, Station C, Ottawa, Ontario K1Y 4H7, Canada

Abstract

This paper presents three new results concerning credit-based flow control for ATM networks: (1) a simple and robust credit update protocol (CUP) suited for relatively inexpensive hardware/software implementation; (2) automatic adaptation of credit buffer allocation for virtual circuits (VCs) sharing the same buffer pool; (3) use of credit-based flow control to improve the effectiveness of statistical multiplexing in minimizing switch memory. These results have been substantiated by analysis, simulation and implementation.

1. Introduction

Flow control is essential for asynchronous transfer mode (ATM) networks [1] in providing "best-effort" services, or ABR (Available Bit Rate) services in the ATM Forum terminology. With proper flow control, computer users would be able to use an ATM network in the same way as they have been using conventional LANs, namely, they can use the network at any time without first negotiating a "traffic contract" with the network. Any one user would be able to acquire as much network resources as are available at any given moment, and all users compete equally for the available bandwidth.

An efficient way of implementing flow-controlled ATM networks is through the use of credit-based, per VC, link-by-link flow control [11]. This paper gives several new results related to the credit-based flow control. All the VCs are assumed to be under "best-effort" or ABR services, unless stated otherwise.

The organization of the paper is as follows: First, motivations for per VC, link-by-link flow control are given. This is followed by an overview of the credit-based flow control approach and a summary of its advantages. Then three main results of this results are presented:

- In Section 5, we describe a credit update protocol (CUP), which allows relatively simple hardware/software implementation and is robust against transient errors.
- In Section 6, we describe an adaptive credit allocation scheme where a number of VCs can dynamically share the same buffer pool while still guaranteeing no data loss due to congestion and ensuring high link utilization. The credit buffer allocated to an individual VC will adjust automatically according to the actual bandwidth usage of the VC. There are two advantages of this adaptation capability. First, since the credit buffer size can be derived automatically, there is no need for the user or the system to specify it. This significantly eases the use and implementation of "best-effort" or ABR services. Second, since inactive VCs can automatically yield their unused buffer space to other active ones, the total buffer size required by the flow-controlled VCs at the node can be minimized. In practice, the total buffer for all the VCs need not be larger than a small multiple of the product of the link bandwidth and round-trip link propagation delay. We present simulation results demonstrating the effectiveness of this adaptive credit scheme.
- In Section 7, we note that credit-based flow control can help statistical multiplexing in minimizing switch memory. This result is especially useful for WAN switches which may have to depend on statistical multiplexing to reduce the otherwise large memory required to cover large propagation delays. Credit-based flow control can help because it will automatically limit burst sizes to be no more than the allocated credit size, thereby improving the effectiveness of statistical multiplexing. We present simulation results demonstrating significant memory reduction while achieving zero or low rate of cell loss. The approach is particularly attractive for traffic with large bursts for which statistical multiplexing without flow control would perform poorly.

These three results are complementary. CUP provides a baseline, efficient and robust protocol for implementing credit-based flow control. Adaptive credit allocation allows efficient sharing of a given buffer pool between multiple VCs, and eases the use of credit-based flow control. Improved statistical multiplexing due to credit-based flow control will allow a switch memory of the same size to serve an expanded number of VCs and to handle links of increased propagation delays.

A version of the proposed credit-based flow control scheme has been implemented on an experimental ATM switch with 622-

This research was supported in part by BNR, and in part by the Advanced Research Projects Agency (DOD) monitored by ARPA/CMO under Contract MDA972-90-C-0035 and by AFMC under Contract F19628-92-C-0116.

Mbps ports, currently under joint development by BNR and Harvard. This switch will be operational in fall 1994.

2. Why Per VC Link-by-Link Flow Control?

The Flow-Controlled Virtual Connections (FCVC) approach [11], using per VC, link-by-link flow control, is different from other proposals on congestion control (see, e.g., [2, 8, 15]). Our interest in FCVC is primarily due to its effectiveness in maximizing network utilization, controlling congestion, and implementing "best-effort" or ABR services.

2.1. Maximizing Network Utilization

FCVC provides effective means of using fill-in traffic to maximize network utilization, as depicted in Figure 1. Using FCVC, best-effort traffic can effectively fill in bandwidth slack left by scheduled traffic with guaranteed bandwidth and latency such as video and audio. In the fill-in process, various scheduling policies can be employed. For example, high-priority best-effort traffic can be used in the fill in before the low-priority one.

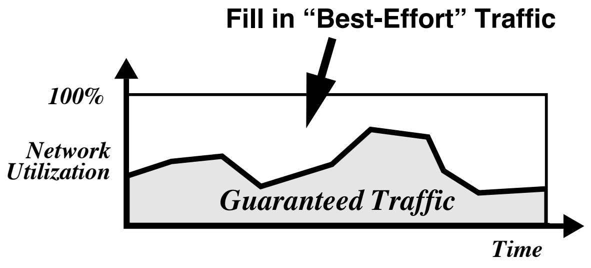

Figure 1: Fill in bandwidth slacks with "best-effort" traffic

For effective traffic fill in, fast congestion feedback for individual VCs is needed. Measurements have shown that data [6, 13] and video [7] traffic often exhibit large bandwidth variations even over time intervals as small as 10 milliseconds. With the emergence of very high-bandwidth traffic sources such as high-speed host computers with 800-Mbps HIPPI network [3] interfaces, networks will experience further increases in load fluctuations [10]. To utilize slack bandwidth in the presence of highly bursty traffic, fast congestion feedback is necessary.

To illustrate the need of fast feedback or flow control for effective fill in, consider a simple case of maximizing the utilization of a link. As depicted in Figure 2, there are multiple VCs from the sender to the receiver sharing the link. The VC scheduler at the sender selects (when possible), for each cell cycle, a VC from which a cell will be transmitted over the link. It is intuitively clear how the scheduler should work; that is, after satisfying VCs of guaranteed performance, the scheduler will select other VCs ("fill-in" VCs), with high priority ones first, to fill in the available bandwidth of the link.

However, two additional conditions (both requiring fast flow control) must be satisfied in order to achieve effective fill in:

- First, data to be used for fill in must be "drawn" to the sender in time. That is, these fill-in VCs should try to hold in their buffers at the sender a number of cells that are ready to be forwarded. There should be sufficiently many of these cells so that they can fill in slack bandwidth at a high rate as soon as the bandwidth becomes available. Note that how long these cells will stay at the sender depends on the load of other VCs.

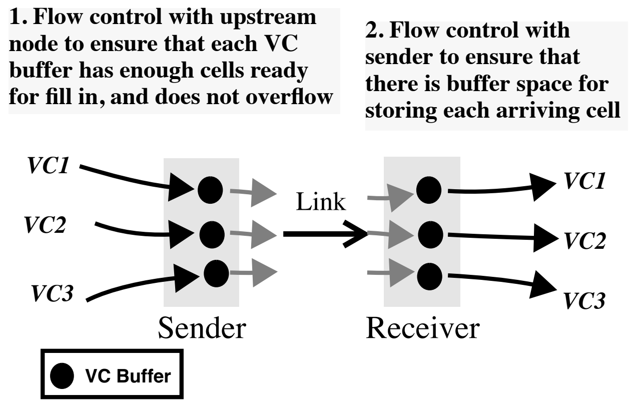
1. Flow control with upstream node to ensure that each VC buffer has enough cells ready for fill in, and does not overflow
2. Flow control with sender to ensure that there is buffer space for storing each arriving cell
VC1
VC2
VC3
Link
VC1
VC2
VC3
Sender
Receiver
● VC Buffer

Figure 2: Two reasons for flow control in achieving effective traffic fill in

When the cells of a VC are not moving out, the upstream node of the VC needs to be flow controlled to avoid buffer overflow. On the other hand, when these cells start moving out, the flow control mechanism should be able to draw in additional cells from the upstream node to fill VC buffers at the sender.

- Second, only "deliverable" traffic should be transmitted over the link in the sense that transmitted data should not be dropped at the receiver due to lack of buffer space. That is, the receiver should have buffer space for storing each arriving cell. Flow control is thus needed for the receiver to inform the sender about buffer space availability. The cost of retransmitting dropped packets increases with both the bandwidth and size of the network, such that on nationwide gigabit networks the penalty is very high.

By using link-by-link flow control, FCVC implements the required feedback at the fastest possible speed. Performance simulation [12] has confirmed the effectiveness of FCVC in filling in traffic, and thus in maximizing network utilization.

2.2. Controlling Congestion

Another reason for FCVC is congestion control. For high-speed networks, in addition to highly bursty traffic mentioned above, there is the problem of increased mismatches in bandwidth [10]. When the peak speed of links increases in a network, so may bandwidth mismatches in the network. For example, when a 1-Gbps link is added to a network which includes a 10-Mbps Ethernet, there will be two orders of magnitude difference in their speeds. When data flows from the high-speed link to the low-speed one, congestion will build up quickly. This represents additional congestion scenarios beyond the usual congestion caused by the merging of multiple traffic streams.

The highly bursty traffic and increased bandwidth mismatches expected will increase the frequency of transient congestion. It is therefore important to ensure that transient congestion does not persist and evolve into permanent network collapse.

Using FCVC, a VC can be guaranteed not to lose cells due to congestion. When experiencing congestion, backpressure will build up quickly along congested VCs spanning one or more hops. When encountering backpressure, the traffic source of a congested VC can be throttled. Thus excessive traffic can be blocked at the boundary of the network, instead of being allowed to enter the network and cause congestion problems to other traffic.

By using "per VC" flow control, FCVC allows multiple VCs over the same physical link to operate at different speeds, depending on their individual congestion status. In particular, congested VCs cannot block other VCs which are not congested.

The throttling feature on individual VCs, enabled by FCVC, is especially useful for implementing high-performance, reliable multicast VCs. At any multicasting point involving more than a few ports, the delay before a cell is forwarded out all the ports can fluctuate greatly. It is therefore essential for reliable multicast VCs to throttle in order to accommodate the inherent high variations in their transmission speeds. Thus, the credit value can be based on the slowest port (the one with the largest queue) to ensure that no buffer will be overrun. Of course, in practice a "relatively" reliable multicast which allows some sort of time-out on blocked multicasting ports will be implemented so that an blocked port will not hold up the whole multicast VC for an unbounded amount of time.

2.3. Implementing "Best-Effort" or ABR Services

Flow control will enable services for hosts with high-speed network access links operating, for example, at 155 Mbps. For instance, these hosts can be offered a new kind of data communications service, which may be called a "greedy" service, where the network will accept as much traffic as it has available bandwidth at any instant from VCs under this service. FCVC can throttle these VCs on a per VC basis when the network load becomes too high, and also speed them up when the load clears. This is exactly the traditional "best-effort" service typical for hosts in LAN environments. There will be no requirements for predefined service contract parameters, which are difficult to set.

3. Credit-Based Flow Control

Flow control based on credits is an efficient way of implementing per VC link-by-link flow control. A credit-based flow control method generally works over each flow-controlled VC link as follows (see Figure 3). Before forwarding any data cell over the link, the sender needs to receive credits for the VC via credit cells sent by the receiver. At various times, the receiver sends credit cells to the sender indicating availability of buffer space for receiving data cells of the VC. After having received credits, the sender is eligible to forward some number of data cells of the VC to the receiver according to the received credit information. Each time the sender forwards a data cell of a VC, it decrements its current *credit balance* for the VC by one.

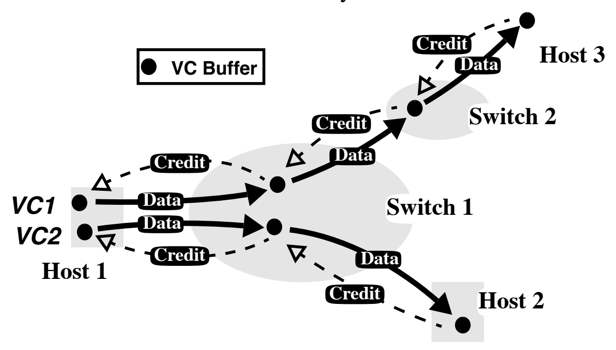

Figure 3: Credit-based flow control applied to each link of a VC

4. The *N23* Scheme: A Credit-Based Flow Control Scheme

The "*N23* Scheme" is a specific scheme for implementing credit-based flow control over a link. (The method is called *N23* for reasons to be explained later.) This section provides an easy-to-understand, *theoretical* definition of the *N23* Scheme. It is important to note that, in practice, other functionally equivalent methods allowing easy and robust implementation, such as the CUP scheme described in Section 5, will likely be used.

As depicted in Figure 4 the receiver is eligible to send a credit cell (with credit value denoted by *C1* or *C2*) to the sender, for a VC, each time after it has forwarded *N2* data cells of the VC (to the receiver's downstream node) since the previous credit cell for the same VC was sent. The credit cell will contain a credit value for the VC equal to the number of unoccupied cell slots in the combined area consisting of the *N2* and *N3* zones.

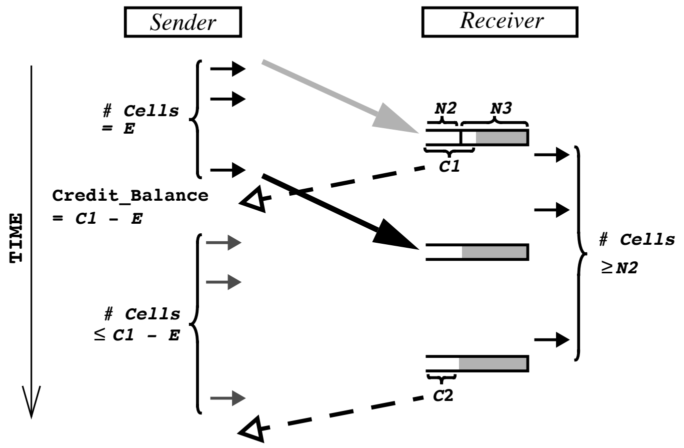

Figure 4: *N23* Scheme for implementing credit-based flow over a link

Upon receiving a credit cell with credit value C for a VC, the sender is permitted to forward up to $C - E$ data cells of the VC before the next successfully transmitted credit cell for the VC is received, where E is defined by Equation (2). Specifically, the sender maintains a count, called Credit_Balance, for the VC. Initially, Credit_Balance is set to be VC's *credit allocation*, *N2* + *N3*. Each time the sender forwards a data cell of the VC (to the receiver), it decrements the Credit_Balance by one. It stops forwarding data cells (only of this VC) when the Credit_Balance reaches zero, and will be eligible to forward data cells (of this VC) again when receiving a new credit cell (for this VC) resulting in a positive value of $C - E$.

More precisely, when receiving a credit cell for a VC, the sender will immediately update its Credit_Balance for the VC using:

$$\text{Credit\_Balance} = \text{Credit value in the newly received credit cell} - E \quad (1)$$

where

$$E = \text{\# of data cells the sender has forwarded over the VC for the past time period of } RTT \quad (2)$$

and

RTT = Round-trip time of the link expressed in number of cell cycles, including the processing delays at sender and receiver (3)

Note that here, only the *RTT* of links connected to a given switch is relevant. Papers proposing end-to-end flow control schemes [17] define *RTT* to mean the round-trip time of the entire network crossed by a VC, which can be not only orders of magnitude larger, but dependent on network congestion. Thus when we report memory requirements as a function of *RTT*, this implies a much smaller memory than a similar-looking formula in an end-to-end paper. One advantage of link-by-link credit-based flow control in providing ABR service is that LAN switches having only short links can have small, inexpensive memories, whereas with end-to-end schemes, every switch must have memory proportional to the network diameter.

The subtraction in Equation (1) takes care of in-flight cells from the sender to the receiver which the receiver had not seen when the credit cell was sent. Thus Equation (1) gives the correct new Credit_Balance.

The *N2* value can be a design or engineering choice. Suppose that x is the number of credit transactions a credit cell can incorporate. (The 48-byte payload of a credit cell can easily hold at least six credit transactions. So we can assume that $x \geq 6$.) Then the bandwidth overhead of transmitting credit cells is no more that 100 / ($N2 \cdot x$ + 1) percent. If $N2 = 1$ or $N2 = 10$ for all VCs, then the bandwidth overhead is at most 14.3% or 1.64%, respectively, assuming $x \geq 6$. The larger *N2* is, the less the bandwidth overhead is but the more buffer each VC will use.

The value of *N2* can also be set on a per VC basis and computed adaptively (see Section 6.3). For example, an adaptive *N2* scheme could give large *N2* values only to VCs of large bandwidth, in order to minimize memory usage.

The *N3* value for a VC is determined by its bandwidth requirement. Let

B_{VC} = Targeted average bandwidth of the VC over time *RTT*, expressed as a percentage of the link bandwidth[1] (4)

Then it can be shown [11] that to prevent data and credit underflow, it suffices to choose *N3* to be:

$$N3 = B_{VC} \cdot RTT \qquad (5)$$

By increasing the *N3* value, the VC can transport data cells at a proportionally higher bandwidth. (Section 6 shows how the *N3* value of a VC can adapt automatically to the actual bandwidth usage of the VC.)

Because the new Credit_Balance is computed by subtracting the number of in-flight cells, E, from the received credit, there is no need for the receiver to reserve a buffer space (called the *N1* zone in [11]) to hold these cells. The method is thus named *N23* as it only needs buffer space for the *N2* and *N3* zones.

Below are some important properties of the *N23* Scheme [11]:

P1 There is **no data overflow**, as long as corrupted credit cells can be detected by the CRC in each credit cell [11].

[1] In this paper, the bandwidth of a VC is always expressed as a percentage of the bandwidth of the link in question, and delays or times are given in number of cell cycles.

P2 There is **no data underflow** and **no credit underflow** in sustaining a VC's targeted bandwidth as long as there are no corrupted credit cells. This means that when there are no hardware errors which corrupt credit cells, the VC never has to wait for data or credits due to the round-trip link delay associated with the flow control feedback loop. That is, the flow control mechanism itself will never prevent a VC from sustaining its targeted bandwidth.

P3 Corrupted credit cells, which are detected by the CRC and discarded, could cause some delay for the affected VC due to data or credit underflow, but no further harm. The delay is no more than, and can be much less than, the usual time-out period for recognizing errors plus the round-trip link delay required to recover from them. In fact, any possible effect of a corrupted credit cell will disappear after the successful delivery of the next credit cell for the same VC. The receiver sends the next credit cell either automatically (i.e., after additional *N2* cells have been forwarded) or as part of a background audit process (see Section 5). In this sense the flow control scheme is **robust** and **self-healing**. Note that credit cells are "idempotent" with respect to the sender in that multiple receipts of credit cells, possibly including redundant ones, from the receiver will never cause harm.

P4 **Transmitting credit cells at any low bandwidth** is possible. By increasing the size of the VC buffer (i.e., the *N2* value), the required bandwidth for transmitting credit cells decreases proportionally.

P5 **The average bandwidth achievable** by a flow-controlled VC over time *RTT* is bounded above by $(N2 + N3) / (RTT + N2)$.

5. Credit Update Protocol (CUP)

This section describes a protocol, called Credit Update Protocol (CUP), for implementing the *N23* Scheme. The method is easy to implement because it only requires the hardware to count cell arrivals, departures, and drops. In particular, credit update does not require estimating the buffer fill of the VC at the receiver, nor the quantity *RTT* or E at the sender, which a straightforward implementation of Equation (1) would require.

Consider per VC flow control over a link. For each flow-controlled VC the sender keeps a running total V_s of all the data cells it has forwarded, and the receiver keeps a running total V_r of all the data cells it has forwarded or dropped. The receiver will enclose the up-to-date value of V_r in each transmitted credit cell for the VC[2]. When the sender receives the credit cell with value V_r, it will update the Credit_Balance for the VC by:

$$\text{Credit\_Balance} = N2 + N3 - (V_s - V_r) \qquad (6)$$

Note that

$$V_s - V_r = BF + E \qquad (7)$$

where BF (i.e., "buffer fill") is the number of cells in the VC buffer when the credit cell departs from the receiver, and E is the quantity defined by Equation (2) when the credit cell arrives at the sender.

[2] A wraparound count can be used to store the V value. The count need only be large enough to represent the value of several times $(N2 + N3)$. The same holds for the U value defined below.

Since the credit value in the newly received credit cell, in Equation (1), is $N2 + N3 - BF$, we see that Credit_Balance computed by either Equation (1) or (6) is the same. Thus we can use Equation (6) for the implementation of the *N23* scheme.

Equation (6) can also be explained directly. Note that $N2 + N3$ is the allocated credit for the VC and $V_s - V_r$ is number of cells that can be in flight or in the VC buffer at the receiver. Thus the Credit_ Balance is $N2 + N3 - (V_s - V_r)$, which is exactly what Equation (6) computes. It is easy to see that the scheme is robust against a lost credit cell, in the sense that repair takes place automatically at the arrival of the next successfully transmitted credit cell of the same VC.

Additional steps are required to provide protection against possible loss of data cells. Without these steps, the sender's Credit_Balance for a VC would be forever lower by one additional count each time a data cell of the VC is lost in the link when transmitting from the sender to the receiver.

To provide protection against possible loss of data cells, each node will keep another running total U of all the data cells it has received for each flow-controlled VC. For each of these VCs, the sender will send a Credit-Check cell or "*CC* cell" periodically at some interval, which is an engineering choice. The sender encloses in the *CC* cell the current V_s value for the VC. The receiver, upon receiving the *CC* cell, immediately computes:

$$\#Lost\_Data\_Cells = V_s - U_r$$

where U_r is the current U value for the VC at the receiver. If #Lost_Data_Cells is greater than zero, the receiver will perform the following recovery for the VC:

$$U_r = U_r + \#Lost\_Data\_Cells$$

$$V_r = V_r + \#Lost\_Data\_Cells$$

and will also send a credit cell with the new V_r value to the sender. (Note that to prevent false indication of cells lost on the next link when a *CC* cell is next generated, the receiver may use an additional count.) The receiver need not perform these recovery operations right away - the receiver can continue receiving additional data cells for the VC before the recovery is complete. Note that for a given VC, #Lost_Data_Cells can never be more than $N2 + N3$.

6. Adaptive Credit Allocation

We describe adaptive credit allocation which allows a number of VCs to share the same buffer pool dynamically. The credit allocation for each VC, i.e., the value of $N2 + N3$, will adapt to its actual bandwidth usage. Using this scheme a VC will automatically decrease its $N2 + N3$ value, if the VC does not have sufficient data to forward or is back-pressured because of downstream congestion. The freed up buffer space will automatically be assigned to other VCs which have data to forward and are not congested.

6.1. Basic Adaptation Concepts

6.1.1. "Dividing a Pie"

The problem of allocating credits between the VCs sharing the same buffer pool is like that of dividing a pie. Figure 5 depicts this analogy.

- The size of the pie corresponds to that of the shared buffer pool. To allow fast ramp up of bandwidth for individual VCs, we assume that the size of the shared buffer pool is $\rho \cdot RTT$ for some constant $\rho > 1$.
- Each partition of the pie corresponds to the allocated credit for a VC.
- The shaded area in each partition (Figure 5 (a)) represents the *operating credit* of the corresponding VC, which is the size of the credit buffer required to sustain the current *operating bandwidth* realized by the VC. That is,

$$\text{Operating Credit} = \text{Operating Bandwidth} \cdot RTT$$

The relative ratios between the operating credits indicate the relative bandwidth usages between the VCs over some *measurement time interval* (*MTI*). To simplify discussion, we assume for the rest of Section 6 that *MTI* (given in cell cycles) is *RTT*. However, larger values (2 or 3 times *RTT*) are probably more appropriate.

6.1.2. Credit Allocation Based on Relative Bandwidth Usage of VCs

Figure 5 depicts how, in our adaptive scheme, credit allocation adapts to actual bandwidth usages of individual VCs. Figure 5 (a) depicts the original credit allocation between three VCs. The operating credits (denoted by shaded regions) of the VCs and their relative ratios are shown. Note from Figure 5 (a) that the ratios between the operating credits are not consistent with those between the allocated credits. Figure 5 (b) shows a new credit allocation which is consistent to the relative operating credits or bandwidths of the VCs, where ρ' is the ratio of the pie over the sum of all shaded areas. Note that since the total operating bandwidth over all the VCs must be no more than 100%, the total size of all shaded areas is no more than *RTT*. This implies that $\rho' \geq \rho$

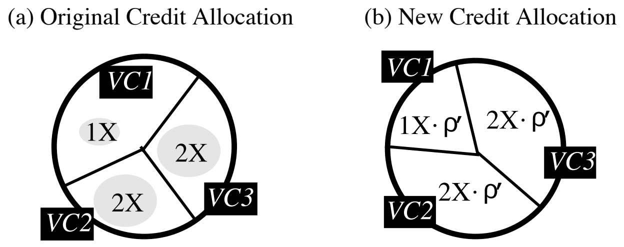

Figure 5: Adaptation of credit allocation: (a) original credit allocation for three VCs, which is inconsistent with the relative ratios (1:2:2) of the VCs' operating credits denoted by shaded areas; and (b) new credit allocation, which is consistent with the ratios of operating credits or bandwidths of the VCs

6.1.3. How Adaptive Credit Allocation Works

A key idea of the adaptive scheme described above is its use of *relative* bandwidth usages in determining new credit allocation (Figure 5). The credit allocation of each VC is always strictly larger than the VC's operating credit by a factor of $\rho' \geq \rho > 1$. As explained below, this will give sufficient headroom for each VC to ramp up its credit allocation rapidly. Note that the relative ratios between operating bandwidths of VCs are exactly the same as the relative ratios between their operating credits. Bandwidth usages can be easily obtained by counting cell departures for individual VCs over some *MTI*. (This counting facility is already present in some switches for other purposes.)

Figure 6 depicts a 3-VC example with $\rho = 2$ and $RTT = 100$. Initially VC1, VC2 and VC3 operate at 10%, 10% and 80%, respectively, of the link bandwidth. Since $\rho = 2$, the initial credit allocation for VC1, VC2 and VC3 is 20, 20 and 160, respectively. Suppose now that VC1 has an increased amount of data to forward and is not congested downstream, while the offered load for VC2 and VC3 will remain the same. Assume that the three VCs are scheduled fairly. Then as Figure 6 shows, after three rounds of credit allocation, VC1 can reach its target bandwidth target, i.e., 45% of the link bandwidth. Notice the allocated bandwidth or credit for VC1 doubles after each round.

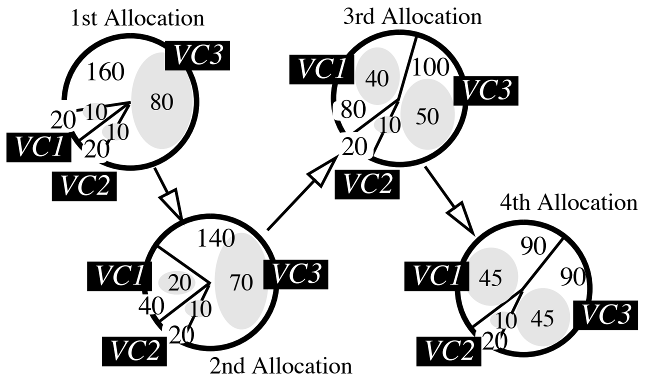

Figure 6: Suppose that $\rho = 2$. The allocated credit or bandwidth for VC1 doubles after each round of credit allocation until the target bandwidth is reached

6.1.4. Proof of Exponential Ramp Up

We give an analysis for the fast ramp-up result illustrated by the example in Section 6.1.3. The following notations are used:

- X = Current operating bandwidth of the VC which is ramping up. (This VC is VC1 in Figure 6.) X is expressed as a percentage of the link bandwidth. Thus the operating credit for the VC is $X \cdot RTT$.
- C = Total operating bandwidth of all the other VCs. (These are VC2 and VC3 in Figure 6.) C is expressed as a percentage of the link bandwidth. Thus $C + X \leq 1$.

Suppose that the total allocated credit among all VCs is $\rho \cdot RTT$. Then the allocated credit of the ramping up VC will be $\rho \cdot RTT \cdot X / (C + X)$ at the end of the current *MTI*. According to the adaptive algorithm the operating bandwidth of the VC will be $\rho \cdot X / (C + X)$ for the next *MTI*. This implies after the current *MTI*, the bandwidth of the VC will be ramped up by a factor of:

$$[\rho \cdot X / (C + X)] / X = \rho / (C + X) \geq \rho \qquad (8)$$

Thus the adaptive scheme can ramp up the bandwidth of a VC at an *exponential* rate, i.e., after the *i*-th round of credit allocation, the VC's credit can be as high as ρ^i times its initial credit.

6.1.5. Discussion

The above analysis assumes that when credit reallocation takes place, the entire shared pool is not occupied by cells and will not be occupied by in-flight cells. To remove this assumption, the "pie" to be divided during reallocation should just consist of the shared buffer pool minus these cells.

It could be the case that the new credit allocation for a VC is smaller than its current used credit. In this situation the new credit allocation will not take full effect until enough data cells have departed from the receiver and the used credit is no longer larger than the new credit allocation.

When a sender or receiver decides give some *N3* value to a particular VC, it does not know the conditions the VC will encounter through the rest of the network. Thus, sometimes it will give a large *N3* to a VC that later becomes blocked. Cells from this VC will continue to occupy memory in the receiver for some period of time. To ensure that the memory occupied by a given VC never exceeds its *N2+N3* value, the adaptive algorithm cannot redistribute its *N3* to other VCs that might need it.

This problem can be mitigated in a few ways. First, we guarantee every VC a minimum *N*3 value of one or more, so that no VC can ever be totally blocked by the credit allocation policy. This ensures freedom from deadlock, and that all VC queues occupying memory will eventually drain. Second, we can vary the agressiveness of the adaptive algorithm depending on the available memory. The ρ and α values can be reduced as *N3Sum* approaches *N3T*. Thus under light network load, it will ramp up *N3* values quickly to achieve low delay, and when congestion develops it will only give large *N3* values to VCs which have demonstrated high rates for a longer period. Third, we can make the receiver memory size large. If the memory is 50**RTT*, then 50 full-speed VCs would have to become blocked before the memory fills up.

6.2. Sender-Oriented Adaptation

The adaptive credit allocation can be implemented at the sender or receiver. This section describes a sender-oriented adaptive scheme.

As depicted by Figure 7, a number of VCs from the sender share the same buffer pool at the receiver. The sender will dynamically adjust the *N3* value for each VC to reflect its actual bandwidth usage at a given time.

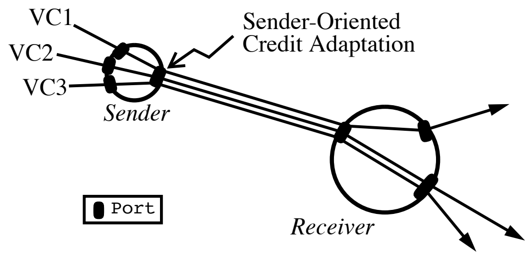

Figure 7: Sender-oriented adaptation

The *N3* adjustment algorithm is run periodically. Assuming a fixed *N2* value for all the VCs, the algorithm cycles through them; for every iteration of the algorithm, the *N3* of a single VC is updated. A good implementation would service the more active VCs more frequently. The number *N3T*, which stands for "*N3* Total", represents the size of the memory that these VCs share in the receiver less $N2 \cdot A$ with *A* being the total number of these VCs. The value of *N3T* may be statically configured, or dynamically varied by some other protocol using relatively large time constants.

The adaptive algorithm computes a target *N3* for a VC as a fraction of $\rho \cdot RTT$, proportional to the VC's fraction of egress traffic on the link. The value is limited by a minimum and maximum value to prevent *N3* going to zero for inactive VCs, and to prevent any VC from exceeding a limit. The algorithm ensures that two properties always hold:

- The total *N3* for all VCs is no greater than *N3T*.
- The number of cells of a given VC actually present in the receiver's memory is no more than the *N2* + *N3* for that VC.

The second property is ensured by never decreasing a VC's *N3* such that its current credit amount becomes negative. The combination of these properties ensures that the memory use in the downstream node is bounded by $N3T + N2 \cdot A$.

Both the aggregate link bandwidth, and operating bandwidth of individual VCs are measured by counting the number of cells over a period *MTI*, typically a few round-trip link times.

The parameter α defines a single pole low-pass IIR filter, which insulates target *N3* values from noise in the egress traffic. In our simulations we set α close to 1, but lower values may improve performance for traffic sources expected to be fairly smooth.

The implemented code handles multicast VCs; however to simplify the discussion a single input and output port is assumed. To make its operation clearer, computation with real numbers is assumed. However, it can be implemented efficiently (as in our simulator) using only integer representations.

Sender-Oriented Adaptive Algorithm

```
vcID := a VC to adjust
bandwidthFraction := # cells sent for the given VC, divided by the
                       total number of flow-controlled cells sent
                       over the last K cell times

targetN3x := bandwidthFraction * RTT * ρ
targetN3 := α*targetN3x + (1-α)*targetN3

limit targetN3 to be:
            no less than minN3 (minN3 >= 1)
            no more than maxN3 (maxN3 <= RTT)

targetDelta := targetN3 - N3[vcID]
limit targetDelta to be:
          no less than (0-creditAmount[vcID])
          no more than (N3T - N3Sum)

increase N3[vcID] by targetDelta
increase N3Sum by targetDelta
increase creditAmount[vcID] by targetDelta
```

We note some properties of the adaptive scheme that make it easy to implement. It requires no communication beyond the credit messages specified for the CUP scheme in Section 5. The sending node only needs to know how much memory it is allowed to use in the receiving node. We expect that much can be done to tune this algorithm for optimum performance in various networking configurations.

6.3. Receiver-Oriented Adaptation

Adaptive credit allocation can also be done at the receiver. A receiver-oriented adaptation, described in this section, is natural for the case where a common buffer pool in a switch is shared by VCs from multiple input links. Figure 8 depicts such a scenario: the buffer pool at output port *p* of switch *R* is shared by four VCs from two switch *S1* and *S2*. Note that the receiver (*R*) can observe the bandwidth usage of the VCs from *all* the senders (*S1* and *S2*). In contrast, each sender can only observe the bandwidth usage of those VCs going out from the same sender. Therefore, it is natural to use receiver-oriented adaptation in this case.

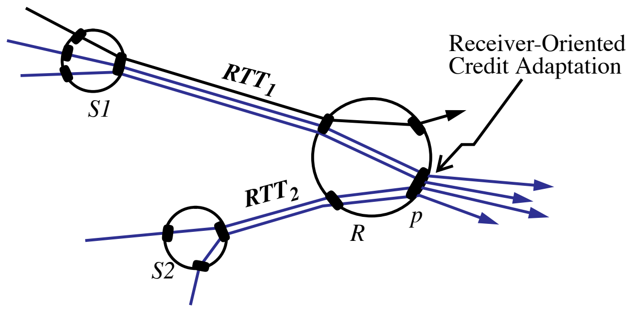

Figure 8: Receiver-oriented adaptation

The receiver dynamically changes the credit allocation to a VC based on its relative bandwidth usage of the VC on the outgoing link. The increase or decrease of the credit allocation will be reflected by the V_r value (see Section 5) in the next credit cell sent by the receiver for the VC. That is, V_r will increase or decrease in the same way as the credit allocation.

Suppose that the round-trip times (RTT_1 and RTT_2) of the two links connecting *R* to *S1* and *S2* have different values. Then an adaptive scheme should allow for adjusting credit allocations based on the relative bandwidth usage of a VC *weighted* by the VC's *RTT*. The size of the buffer pool at the receiver should be related to $\rho \cdot RTT_{max}$, where RTT_{max} is the maximum *RTT* value for all the links, and as defined in Section 6, $\rho > 1$ is a small constant.

There two important features of the receiver-oriented adaptation:

- As noted above, the size of the buffer pool at the receiver is related to R_{max}, independent of the number of input links sharing the same buffer.
- In addition to the *N3* adaptation, the *N2* value of each VC can also be adaptive. This works naturally for the receiver-oriented adaptation as only the receiver needs to use *N2* values and thus can conveniently change them locally as it wishes. For a given credit allocation of a VC, the allocation can simply be split between *N2* and *N3* according to some policy. For example, *N2* can be given to be one half, or some other proportion, of the allocated credit. This allows only those VCs of large bandwidth usages to use large *N2* values, so that the bandwidth overhead of transmitting credit cells can be minimized while still keeping the total *N2* zones of all the VCs relatively small. In general, an inactive VC can be given an *N2* value of 1. The *N2* value will increases as VC's bandwidth ramps up. Thus the total allocated memory for *all* VCs can be as small as $\rho \cdot RTT_{max}$ + # VCs.

The ramp up for the receiver-oriented adaptation over a link will be delayed by about *RTT*, compared to the sender-oriented adaptation. However, for a multiple-hop connection, while a VC is ramping up on one link, it can also start ramping up the next link as soon as data that have caused the ramp up on the first link begin to reach the second link. Thus, the total extra delay in the receiver-oriented adaptation is expected to be only about one *RTT* corresponding to that hop which has the largest *RTT* value. This has been validated by simulation results, which will be reported in another paper.

7. Flow-Controlled Statistical Multiplexing for Minimizing Switch Memory

For the *N23* Scheme, or any other similar credit-based flow control method, the total amount of memory required to allow all the VCs to reach their desired peak bandwidth can be large, especially for WANs where propagation delays are large. However if we allow some (very) small probability of cell loss, then the memory size can be significantly reduced by statistical multiplexing. It will be shown that credit-based flow control will improve the effectiveness of statistical multiplexing.

We use the notion of "virtual memory" to describe the concept of using statistical multiplexing in reducing the size of the "real memory" of a switch. This is depicted in Figure 9.

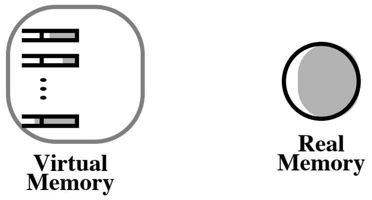

Figure 9: "Virtual memory" for credit allocation and "real memory" for storing data cells

- VC's buffer (the *N2* and *N3* areas) for supporting credit-based flow control is allocated from the virtual memory of the switch.
- Buffer space actually occupied by data cells at any given time (shaded area) is allocated from the real memory of the switch. When the real memory is overflowed, data cells will be dropped. The real memory is sized for low cell loss.

There are advantages of using a virtual memory substantially larger than the real memory. These include fast bandwidth ramp up in adaptive credit allocation, and increased number of admitted flow-controlled VCs.

There are reasons to expect that this virtual memory approach based on statistical multiplexing can be effective. Obviously, an inactive VC with no data cells to forward will not consume any real memory. Even an active VC, under a non-congestion situation, will occupy at most one cell in the real memory at any time, independently of the VC's bandwidth. As long as data is flowing on the links, then one *RTT* worth of data is included in the *N2* + *N3* values, but never occupies switch memory.

It is well known that statistical multiplexing is expected to be effective when a large number of VCs of relatively small average bandwidths and small bursts share a real memory. Using the *N23* credit-based flow control, bursts will be bounded by *N2* + *N3* cells. Using *N2* = 10 and a relatively small value for *N3*, we can ensure that the carried traffic will have small bursts and therefore the use of statistical multiplexing in minimizing memory will work well. The is validated by the simulation results in the next section.

In summary, suppose that there are *A* flow controlled VCs, and they use the same *N2* and *N3* values. If the flow control mechanism needs to guarantee that there is never any cell loss due to congestion, then *M* must be at least *A**(*N2*+*N3*). However, if some non-zero probability of cell loss is acceptable, then *M* can be much smaller than *A**(*N2*+*N3*).

Another way of looking at this statistical multiplexing approach is that when a moderate number of VCs are congested, the flow control mechanism achieves zero cell loss, and provides backpressure to control the sources. Under heavy congestion, when many VCs are congested, cells will be lost. This is illustrated by the conceptual load-loss curves [18] of Figure 11. While flow control

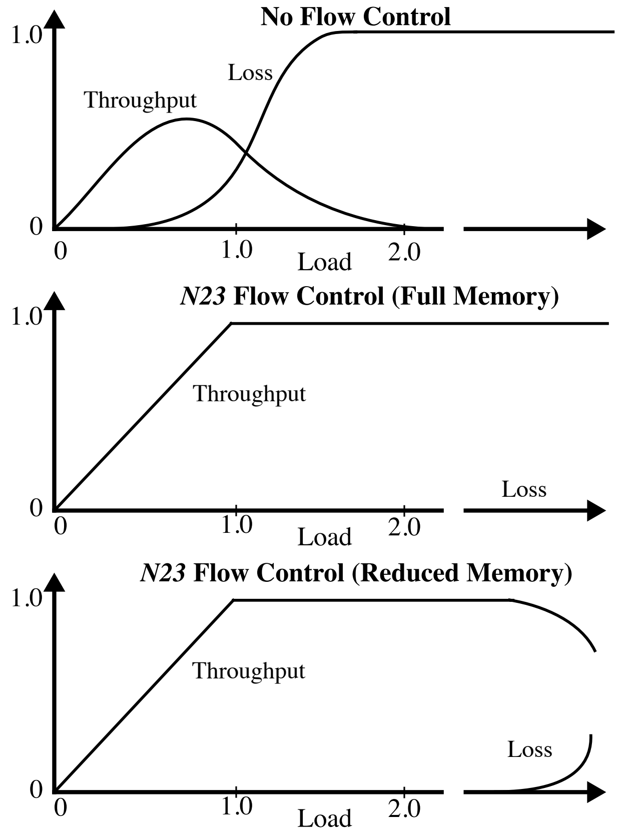

Figure 10: Load-loss curves for various schemes

with the full amount of memory to guarantee zero cell loss has an ideal load-loss curve, with reduced memory using statistical multiplexing there is still a large region of the load-loss curve which gives efficient operation. The load level at which loss occurs depends on the size of the real memory, and on the characteristics of the traffic and the rest of the network. Reasonable end system protocols, such as TCP/IP, should be able to make effective use of this broad region of efficient operation.

8. Simulation Configuration

All the simulations to be presented in the rest of the paper assume a simple configuration shown in Figure 11. This configuration was chosen to allow easy interpretation of simulation results and also it is sufficiently general to cover most of the key issues we want to address.

There are *N* VCs originating from some number of source hosts (indicated by shaded rectangles) and passing through two switches (indicated by shaded circles). There are several VCs on each input port of Switch-1, and they depart from the same output port of so there will be congestion at the port which we study. (In Figure 11, each solid bar indicates a switch input or output port.) All the VCs will share the same *M*-cell memory in Switch-1. (Switch-1 is referred to as "the switch" in the rest of the paper, unless otherwise stated explicitly.) We assume a simple output buffered switch where VCs can be individually scheduled and accessed at each output port.

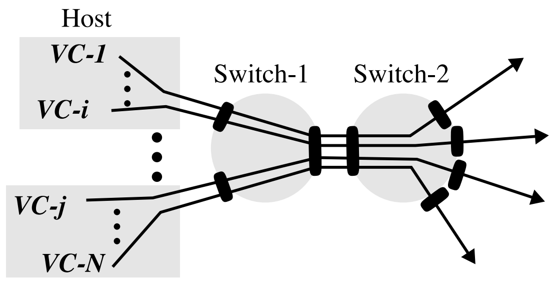

Figure 11: Simulation configuration

8.1. Notations

B:	VC burst size (# cells)
D:	Round-trip propagation delay between each host and the switch (# cells)
F:	Total average offered load (% of the link bandwidth of the switch output port).
L:	Cell loss rate (%)
M:	Size of switch memory (# cells)
MMU:	Maximum observed memory usage (# cells) for a given traffic load over the length of the simulation
N:	Number of VCs
T:	Simulated time (# cell cycles)
α	Memory Reduction Factor

9. Simulation Suite A: Performance as a Function of Allocated Credit Size

The A series of simulations show how throughput, delay, and loss rates are affected by the *N3* parameter. Clearly, a VC with larger *N3* will have higher throughput and lower delay than a VC with lower *N3*, because it will be blocked less often due to lack of credit.

In the following two series of simulations, with 100 and 200 VCs respectively, we show the loss rate, memory usage, average delay, and link utilization as a function of the VC's *N3* value.

We also take a different approach to showing memory requirements in the two series. Simulation A-S1 shows the number of dropped cells with a fixed size memory. Simulation A-S2 shows the maximum memory required so that no cells were dropped over the simulation interval. Note that in both simulations, 4800 cell times of the delay is due to link propagation time.

9.1. Simulation A-S1 (Figure 13)

B (VC burst size) = 172
D (Round-trip propagation delay) = 3200
F (Total offered load) = 95%
M (Switch memory size) = 4096
N (Number of VCs) = 100
T (Simulated time) = 1,000,000 cell periods
N2 = 10

9.2. Simulation A-S2 (Figure 12)

B (VC burst size) = 172
D (Round-trip propagation delay) = 3200
F (Total offered load) = 95%
M (Switch memory size) = Unlimited
N (Number of VCs) = 200
T (Simulated Time) = 1,000,000 cell periods
N2 = 10

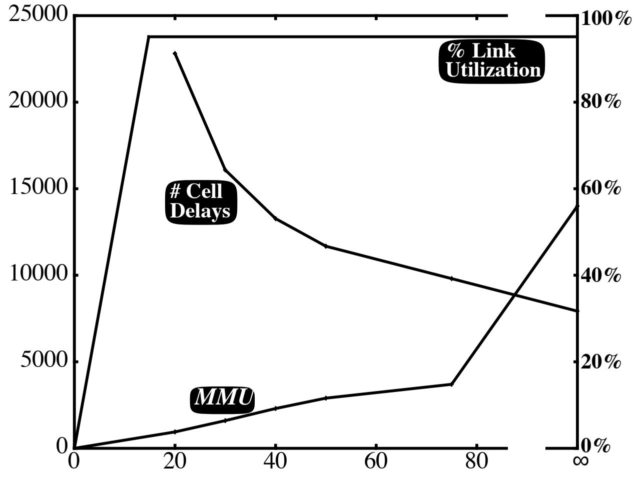

Figure 12: Link utilization, # cell delays, and maximum memory usage (*MMU*) as function of *N3* values (Simulation A-S2)

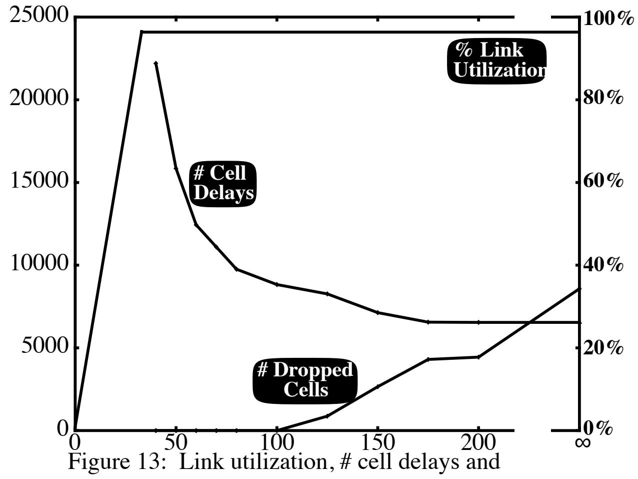

Figure 13: Link utilization, # cell delays and # dropped cells as function of *N3* values (Simulation A-S1)

10. Simulation Suite B: Comparing FC And Non FC Statistical Multiplexing

The B series of simulations show the memory use characteristics of flow-controlled (FC) vs. non flow-controlled (non FC) traffic. Simulating switches with unlimited memory, we report two values: minimum memory such that the loss rate with FC was very low (no cells lost in a 1 million cell simulation), and minimum memory to achieve low cell loss without flow control.

10.1. Simulation Assumptions

A1. The *N* VCs have identical load involving *B*-cell bursts. The inter-burst times are exponentially distributed such that the average offered load of each VC is (Offered Load)/*N* of the link bandwidth.

A2. The switch memory, shared by all the *N* VCs, has $M = \alpha * N * B$ cells where $\alpha < 1$ is the "Memory Reduction Factor" due to statistical multiplexing. (A goal of a statistical multiplexing method in minimizing *M* would be to achieve a small value of α for a wide range of *B* and *N* values.)

10.2. Theory to Be Validated

T1. For the same *N* and *B*, flow control can significantly reduce α while achieving the same throughput and the same rate of cell loss.

10.3. Simulation B-S1

B (VC burst size) = 172 (corresponding to an 8K-Byte block in a datagram protocol such as NFS)
D (Round-trip propagation delay) = 3,200 (approximately one-way 225km for the OC-12 rate, or 900km for the OC-3 rate)
F (Total offered load) = 95%
N (Number of VCs) = 100
T (Simulated time) = 1,000,000 cell periods
N2 = 10
N3 = 48 (hence peak rate is 1.5% of link rate)

In the both FC and non-FC cases the average utilization of the output port was about 95%.

	FC	Non FC
Memory Size (*M*)	Cell Loss Rate (*L*)	
370 Cells	0	Large (e.g., $L >> 1\%$)
3250 Cells	0	Small (i.e., $L < .1\%$)
	α Value	
	2%	> 19%

Table 1: Comparing cell loss rates for FC and non FC statistical multiplexing at two memory sizes (Simulation B-S1)

10.4. Comments on the Simulation Results

Note that the above results of simulation B-S1 validate theory T1 in Section 10.2.

By analysis one can expect that for multi-switch configurations flow control is just as effective as for the single-switch case, as far as minimizing switch memory by statistical multiplexing is concerned. This has been confirmed by our multi-switch simulations, which are not reported here.

11. Simulation Suite C: Adaptive Credit Allocation by Tracking Traffic Load

The C series of simulations shows the effect of the adaptive *N3* algorithm, described in Section 6, on delay and memory usage.

11.1. Simulation C-S1

B (VC burst size) = 172
D (Round-trip Propagation delay) = 3200
F (Total offered load) = 95%
M (Switch memory size) = Unlimited
N (Number of VCs) = 100
T (Simulated time) = 1,000,000 cell periods

Figure 14 plots the *N3* values of a representative VC against iterations of the adaptive credit algorithm - one iteration is 500 cell cycles. The N3 values change rapidly to adapt to offered load changes. Note that changes of the *N3* values at Switch-1 closely track those at the source host, and those at Switch-2 closely track those at Switch-1. (The three traces are so close that they almost coincide in Figure 14.) A blown up version of part of this tracking is shown in Figure 15.

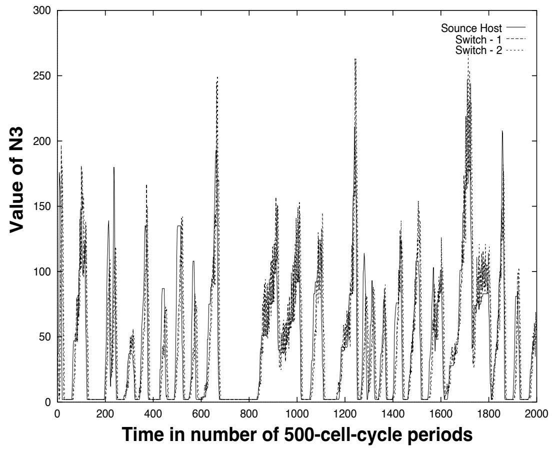

Figure 14: Changes of the *N3* at the source host, Switch-1 and Switch-2 track each other closely (Simulation C-S1)

Notice that the *N3* values of consecutive switches track each other with only a short delay. As can be seen in Figure 15, a switch lags behind the switch upstream by no more than 5,000 cell cycles. This reaction time is only a little more than a roundtrip time (3200 here).

The peak *N3* reached in response to a burst is about equal to the burst size (172) in most cases. Figure 14 shows that it never exceeds twice the burst size for this particular set of parameters.

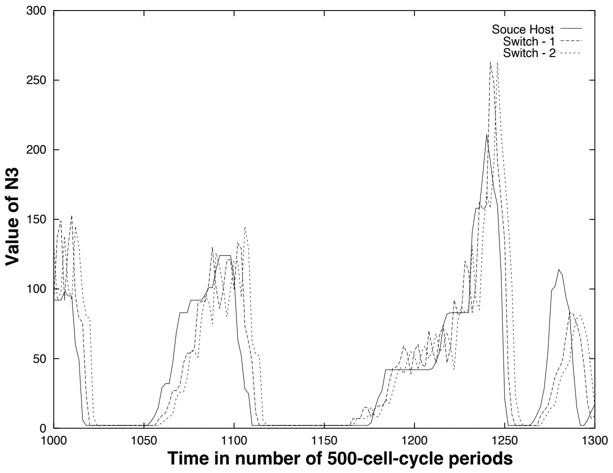

Figure 15: A blown up view of part of the tracking shown in Figure 14 (Simulation C-S1)

The adaptive *N3* can reach values much higher than would be practical for a statically defined *N3*. This allows the switch to clear the burst through as quickly as possible, reducing delay.

The increase of the *N3* value in response to an arriving burst of cells is exponential under light network loads, but linear under heavy loads. Consider a lightly loaded network. The adaptive *N3* algorithm in Section 6 will increase *N3* rapidly. A few cells sent on the VC will increase its *N3* by more than the number of cells sent, enabling more cells to be sent -- *N3* increases exponentially.

However if link usage is high, *N3* will increase by less than the number of cells sent out, so that *N3* will increase by only a small amount when the initial cells are sent out. *N3* will not increase further until the switch receives a credit cell enabling it to send more cells.

This exponential/linear increase scheme provides low latency under low network loads, and stability under high loads.

Figure 16 shows the memory usage in Switch-1 as a function of time in cell cycles. Memory usage is lower than that of usage of non flow-controlled traffic as shown in Table 1. The low memory usage is achieved without any loss of cells.

11.2. Simulation C-S2

B (VC burst size) = 172
D (Round-trip Propagation delay) = 3200
F (Total offered load) = 95%
M (Switch memory size) = Unlimited
N (Number of VCs) = 200
T (Simulated time) = 1,000,000 cell periods
Table 2 summarizes the performance of the adaptive credit allocation method, in terms of its *MMU*, delay and link utilization. Again, note that relatively small memory usage is achieved

12. Simulation Suite D: NFS

12.1. Simulation Configuration

The simulation configuration for NFS [16] traffic is depicted in Figure 17. Clients on the right-hand side issues read requests to the servers on the left-hand side. The figure shows 80 VCs each carrying data responding to requests carried on paired VCs from

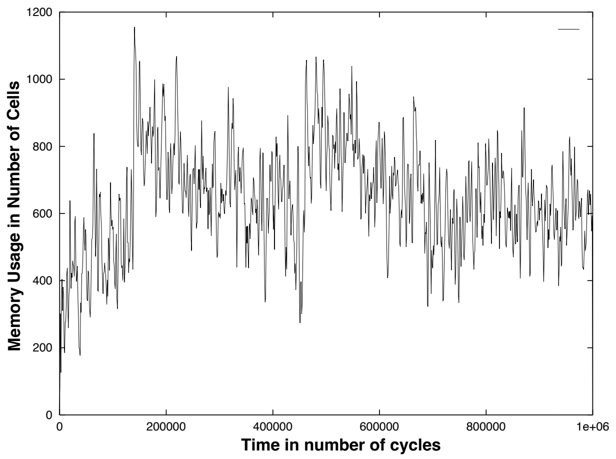

Figure 16: Memory usage profile (Simulation C-S1)

	MMU	# Cell Delays	Link Utilization
Adaptive Credit Allocation	1,150	14,088	95%
Static Credit Allocation (Simulation B-S2, Figure 12)	1,600 (N3=30) 2,300 (N3=40)	16,083 (N3=30) 13,272 (N3=40)	95%

Table 2: Performance comparisons between adaptive and static credit allocation (Simulations B-S2 and C-S2)

right to left; the traffic on client-to-server VCs is small. NFS traffic is essentially a request/response transaction stream, which is typical of many database access and transaction processing protocols.

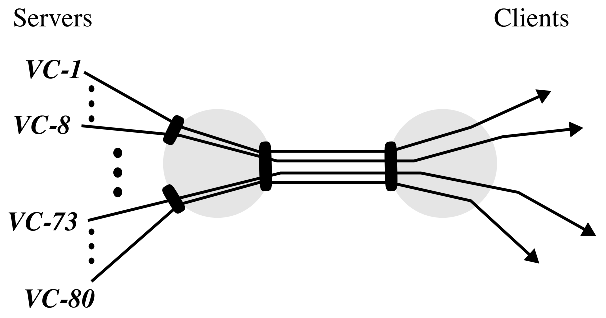

Figure 17: NFS simulation configuration

In this simulation, we measured the total number of transactions completed between clients and servers, as well as the number of datagrams lost.

The simulation was driven from a trace of real activity on our network. The only activity was reading through all the files in a large home directory. We multiplied the size of all packets by 4, in order to simulate a more aggressive NFS implementation. (NFS typically handles blocks of at most 8KB - the maximum size in this simulation is 32KB). We then simulated having 8 clients, each running from a different part of the trace with each of 10 servers. Thus the simulated traffic was 320 (4*80) times the measured traffic. The simulated clients and servers also took half as long to respond as the measured response time - this did not change the network load very much.

12.2. Simulation D-S3

Max Data Packet Size = 693 cells
(32 KB data + 1/2 KB header)
D (Round-trip propagation delay) = 3,200
N (Number of VCs) = 80 (8 clients * 10 servers)
T (Simulated time) = 1,000,000 cell periods = 2.7 sec
N2 = 10
N3 = Adaptive
Data rate = 155 Mb/sec
NFS retransmit time-out = 1.0 second

	FC, Adaptive N3	Non FC
Memory Size (*M*)	Cell Loss Rate & Datagram Loss Rate (*L* & *DL*)	
3,700 Cells	0% & 0%	6% & 4.5%
10,000 Cells	0% & 0%	4.1% & 2.6%
	# Complete Transactions	
5,000 Cells	2,681	1,907
	Datagram Transit Time (cell times)	
5,000 Cells	16,700	7,300

Table 3: Performance comparison of NFS over FC and non-FC networks (Simulation D-S3)

Without flow control, a very large amount of memory is required to low datagram loss levels - approximately 40000 cells. Flow control, with the adaptive scheme, reduces the loss level to zero while using substantially less memory (3700 cells). In the simulation, the population of users is large enough that the lost traffic of a few clients does not impact the total transaction rate substantially. In the real world, every lost datagram causes a pause of 1 or more seconds for some client, while it waits to retransmit. Clients that lose multiple datagrams in a row double their timeout every time, within some bounds. So a loss of 3 consecutive datagrams implies a 7 second pause. These pauses have a large effect on the average speed for a given set of tasks.

More detailed analysis of the simulation traces showed behavior far worse than evidenced from the table. Essentially, about half of the NFS clients experienced a lost packet in the initial 50 mS, and were therefore idle for 1 second. When they retransmitted, there was again high packet loss, and within a few mS a (different) half of the NFS clients lost packets and became idle. Thus, only half of the clients were ever active at one time, while the rest were sitting idle.

Assuming 5000 cells of available switch memory, the network latency is 22 mS for FC, and 10 mS for non FC. With FC, if a client needs to make 1,000 serial reads, total time will be 22 seconds in addition to a few seconds of server time. Without FC, assuming losses cause a 1 second pause, total time will be 63 seconds plus server time. The improvement is even more dramatic on shorter links (where NFS is more likely to be used.)

13. What Do All the Above Simulation Results Mean?

We draw some general conclusions from the simulations reported in the preceding section and from other simulations (such as those with offered traffic load *F* at 50% instead of 95%) which we have done but are not reported in this paper. Table 4 summarizes these conclusions.

	Non FC	Statically FC	Adaptively FC
Delay	*x*	*2x*	*2x*
Memory Usage	*>> RTT*	*N*RTT*BWpeak* *(or lower as in Section 7)*	*3*RTT* *(or lower as in Section 7)*
Loss	*High*	*Zero* *(or low if memory is reduced)*	*Zero* *(or low if memory is reduced)*
Ease of Use	High	Low	High

Table 4: Performance comparison between non flow-controlled (Non FC), adaptively flow-controlled (Adaptively FC), and statically flow-controlled (Statically FC) networks. *N* is # of connections

One should read Table 4 as follows. Consider an offered traffic giving say, *F*= 95% load, such as that used by a S1 or S2 simulation in the preceding section. Suppose that a non flow-controlled network (with unlimited memory and per-VC queueing) has an average delay of *x*. Then if the network is adaptively flow-controlled (by the *N3* adaptive algorithm of Section 6), the average delay is expected to be no more than *2x*. Moreover, this adaptively flow-controlled network is expected to need only about a half of the memory (having say, *y* cells) required by the corresponding statically flow-controlled network achieving the same average delay. The required memory for the non flow-controlled network to achieve a reasonably low cell loss rate will need to have much, much more than *y* cells. Both the non flow-controlled and adaptively flow-controlled networks are easier to use than a statically flow-controlled network, because they don't require users' assistance in allocating credit buffer (i.e., setting the *N3* value). This analysis implies that the adaptive FC is the winner among the three approaches.

14. Concluding Remarks

Existing ATM protocol standards are expected to perform well with steady, predictable traffic; however data traffic such as on demand data transfer and interactive sessions are highly bursty and unpredictable. ATM networks without flow control do not handle this traffic well. Flow control allows *best-effort* traffic to attain high throughput, and experience low latency and loss with minimal buffer reservation through the network.

Some early proposals for flow control in ATM networks required large amounts of buffer memory, proportional to the link length times the total peak capacity of all VCs. Memory for gigabit ATM switches can be expensive due to the high bandwidth involved. Although these flow control proposals, without employing techniques of this paper, could be practical for LANs with small link propagation delays, the large memory requirements created many difficulties for ATM WANs:

- Hosts had to make accurate estimates of how much bandwidth they would require, in order to request an appropriate credit buffer size (i.e., some *N3*-equivalent value).
- Idle VCs consumed significant switch resources. Attempting to deactivate idle VCs imposed significant protocol overheads on the hosts and switches.
- Traffic requiring large peak bandwidths but with low average bandwidth (X Window System connections, for example) used network resources very inefficiently

The results reported in this paper improve this situation in two ways. First, we have shown that much smaller memories can provide zero or low loss rates through statistical multiplexing - in fact the use of flow control can reduce total switch memory requirements for bursty traffic.

Second, the adaptive *N3* protocol eliminates the need for hosts to estimate their traffic requirements, and allows highly bursty, variable traffic sources to use network resources efficiently. Finally, we can expect that ATM can be as simple and efficient for computer communications as TCP over IP networks.

Our initial flow-controlled switches designs required substantial *hardware* support for credit management. Many aspects of credit cell management required sub-microsecond processing. The CUP method described in this paper, however, is designed with a *software* implementation in mind.

A "Credit Card" added as an overlay on a conventional ATM switch can run the credit protocol in software. Because the *N2* is used to reduce the credit processing, the software only needs to process about 1 event for every 10-20 cells of flow-controlled traffic, requiring perhaps 10 memory references

Our reference architecture, used for all simulations reported here, is designed so that delays in processing by the software will only result in a slight degradation in total throughput of flow-controlled cells, but will never produce incorrect behavior such as dropping cells.

In our reference implementation, all the machinery required for flow control runs only at the link rate (not a multiple thereof, as do many parts of ATM switches). It requires only the following switch features above the features common to any ATM switch:

- A Credit Card, containing a fast microcontroller, capable of sending and receiving cells at the nominal link rate. It might replace a port card in a backplane-style switch, and it can be the same engine that runs connection setup processing.
- Egress, ingress, and drop counters for each VC, readable by the Credit Card. (Some ATM switches already have these counters).
- Per-VC credit counters on each port card, which are decremented when a cell is sent. The scheduler must not send cells for VCs with no credit.
- Per-VC *N2* counters on each port card which are decremented for every cell sent. When a counter reaches zero, the ID number of that VC is enqueued in a FIFO, readable by the Credit Card. This ID signals the Credit Card to send a credit cell.

The adaptive *N3* protocol used in the simulations is still at an early stage of development. We suspect that many interesting algorithms will be developed that can improve performance substantially over the results we have shown. This effort will be aided by the nature of CUP:

- The implementation of the algorithm as software allows easy experimentation and sophisticated algorithms.
- The basic credit cell primitive, and the lost data cell recovery message, are the only protocol elements required between switches. Simple and easy to standardize, the CUP protocol will allow switch manufacturers to develop new and better adaptive *N3* algorithms to optimize throughput of their switches.

CUP makes a uniform flow control possible on both LAN and WAN ATM networks, and is simple enough that it can be easily standardized for heterogenous networks. We have shown that using flow control can reduce memory requirements, and adaptive N3 control can reduce them still further, while also simplifying connection setup. We think CUP is the right foundation on which to provide best-effort capabilities in ATM networks.

References

[1] ATM Forum, "ATM User-Network Interface Specification," Version 3.0, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] "ISDN - Core Aspects of Frame Protocol for Use with Frame Relay Bearer Service," ANSI T1.618-1991.

[3] "High-Performance Parallel Interface - Mechanical, Electrical and Signalling Protocol Specification (HIPPI-PH)", ANSI X3.183-1991.

[4] S. Borkar, R. Cohn, G. Cox, T. Gross, H. T. Kung, M. Lam, M. Levine, M. Wire, C. Peterson, J. Susman, J. Sutton, J. Urbanski and J. Webb, "Integrating Systolic and Memory Communication in iWarp," *Conference Proceedings of the 17th Annual International Symposium on Computer Architecture*, Seattle, Washington, June 1990, pp. 70-81.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. SIGCOMM '89 Symposium on Communications Architectures and Protocols*, pp.1-12.

[6] H. J. Fowler and W. E. Leland, "Local Area Network Traffic Characteristics, with Implications for Broadband Network Congestion Management," *IEEE J. on Selected Areas in Commun.*, vol. 9, no. 7, pp. 1139-1149, Sep. 1991.

[7] M. W. Garrett, "Statistical Analysis of a Long Trace of Variable Bit Rate Video Traffic," Chapter IV of Ph.D. Thesis, Columbia University, 1993.

[8] V. Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM '88 Symposium on Communications Architectures and Protocols,* Aug. 1988.

[9] M. G. H. Katevenis, "Fast Switching and Fair Control of Congested Flow in Broadband Networks," *IEEE J. on Selected Areas in Commun.*, vol. SAC-5, no. 8, pp. 1315-1326, Oct. 1987.

[10] H. T. Kung, "Gigabit Local Area Networks: A Systems Perspective," IEEE Communications Magazine, 30 (1992), pp. 79-89.

[11] H. T. Kung and A. Chapman, "The FCVC (Flow-Controlled Virtual Channels) Proposal for ATM Networks," Version 2.0, 1993. A summary appears in *Proc. 1993 International Conf. on Network Protocols*, San Francisco, California, October 19-22, 1993, pp. 116-127.

[12] H.T. Kung, R. Morris, T. Charuhas, and D. Lin, "Use of Link-by-Link Flow Control in Maximizing ATM Networks Performance: Simulation Results," *Proc. IEEE Hot Interconnects Symposium, '93* Palo Alto, California, Aug. 1993.

[13] W. E. Leland, M. S. Taqqu, W. Wilinger and D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *Proc. SIGCOMM '93 Symposium on Communications Architectures and Protocols*, 1993.

[14] A. Parekh and R. G. Gallager, "Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks - The Multiple Node Case," *IEEE INFOCOM '*93, San Francisco, Mar. 1993.

[15] K.K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," ACM Transactions on Computer Systems, Vol. 8, No. 2, pp. 158-181, May 1990.

[16] Sun Microsystems. "NFS: Network File System Protocol Specification," RFC 1094, Mar 1988.

[17] N. Yin and M. G. Hluchyj, "On Closed-Loop Rate Control for ATM Cell Relay Networks," submitted to IEEE Infocom 1994.

[18] C.L. Williamson, D.L. Cheriton, "Load-Loss Curves: Support for Rate-Based Congestion Control in High-Speed Datagram Networks,", *Proc. SIGCOMM '91 Symposium on Communications Architectures and Protocols*, pp.17-28.