Proc. IEEE Hot Interconnects Symposium, ‘93,
Palo Alto, California, 5-6 August 1993

Use of Link-by-Link Flow Control in Maximizing ATM Network Performance: Simulation Results

H. T. Kung, Robert Morris, Thomas Charuhas, Dong Lin
Division of Applied Sciences, Harvard University, 29 Oxford Street, Cambridge, MA 02138

Abstract

Simulations have been performed to verify the effectiveness of using link-by-link flow controlled virtual channels for maximizing ATM network performance. A simulator which accurately reflects the real hardware design of a flow controlled ATM switch is used. The switch is currently under joint development by BNR and Harvard. The simulation results clearly demonstrate that the flow control mechanism is able to provide sufficiently rapid feedback to allow a network to adapt to load changes and maximize its performance. The simulations also show that, when compared to VCs using other traffic management approaches, flow controlled virtual circuits are efficient in terms of buffer usage and in point-to-multipoint multicast implementations.

1 Introduction

A set of *credit-based flow control* schemes for implementing *link-by-link flow controlled virtual channels* has been proposed, in [5], for asynchronous transfer mode (ATM) networks [1, 3]. These three increasingly memory-efficient credit-based flow control schemes with increasing implementation cost are named *N123*, *N123+* and *N23* respectively. A credit cell format, compatible with AAL type 5, is also proposed in [5] along with a set of credit-related transaction types. These credit-based flow control schemes have been implemented in a 622-Mbps ATM switch design under joint development by BNR and Harvard.

Defined in terms of ATM cells, these credit-based flow control schemes can operate on top of various underlying physical media. They can efficiently implement flow controlled VCs of any bandwidth, and can also limit the bandwidth used by the flow control overhead to any given fraction of the total link bandwidth.

Moreover, the schemes are robust in the sense that they can recover automatically from link errors. Adjustments can be easily made to increase the degree of automatic protection against errors at the expense of increased bandwidth overhead or buffer memory size. It is shown in [5] that while enjoying automatic protection against errors, the *N23* Scheme is equivalent to the “additive” credit updating methods [6, 7, 12], as far as the effect of flow control on buffer management is concerned.

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.

A major motivation for using the per VC link-by-link flow control (LLFC) approach is to maximize ATM network performance. We have been performing simulations to validate the approach based on these credit-based flow control schemes. The simulations have been performed on a simulator which accurately reflects the hardware design of the BNR/Harvard switch. The paper is an interim report on the performance simulations we have done so far.

The organization of the paper is as follows: first, motivations for per virtual circuit (VC), link-by-link flow control (LLFC) are given. This is followed by an overview of the credit-based flow control approach and a description of the *N23* Scheme. Then, we present the main contributions of this paper, that is, our simulation approach, and descriptions and results of various simulations we have performed. Finally, comparisons with some other approaches are given. Note that for completeness, some background information in [5] is repeated in this paper.

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

A basic reason to use per VC LLFC is to provide *fast* congestion *feedback* for individual VCs. Measurements have shown that data [9, 14] and video [10] traffic often exhibit large bandwidth variations even over time intervals as small as 10 milliseconds. With the presence of very high-bandwidth traffic sources such as a high-speed host computer with a 800-mbps HIPPI [4] network interface, the network must be prepared for further increase in load fluctuations [13]. A single traffic source of this kind, or just a few of them, will be able to pump data into a network at such a high rate to consume a large fraction of the peak bandwidth of a network link. On the other hand, the traffic source can complete its data transmission in a short time because of the very fact that data is transmitted at high rate. Once the transmission is complete, the network load will suddenly drop sharply. Thus traffic burstiness will increase as the speed of traffic sources increases.

In addition, for high-speed networks there is the problem of increased mismatches in bandwidth [13]. 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 occur quickly.

The highly bursty traffic and increased bandwidth mismatches expected will increase the chance of transient congestion. It therefore becomes absolutely imperative to ensure that transient congestion does not persist and evolve into permanent network collapse. To achieve good network performance under these situations, the network must provide fast congestion feedback on a per VC basis [13]. LLFC implements the required feedback at the fastest possible speed.

Using LLFC, 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 propagate to other traffic.

The "per VC" LLFC 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 LLFC, is especially useful for implementing high-performance, reliable multicast VCs. At any multicasting point involving more than a moderate number of ports, the probability that one or more of them are not available at a given cell cycle is likely to be high, and the delay before a cell is forwarded to 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. 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 unreliable port will not hold up the whole multicast VC for an unbounded amount of time. In addition, a certain degree of "asynchrony" will be allowed so that some multicasting ports may proceed ahead of others by some limited number of cells in order to increase switch utilization and the multicast VC's throughput. (See section 8.2)

Flow control will allow new services for hosts with high-speed network access links operating, for example, at 100 megabits per second. For instance, these hosts can be offered a new kind of service, which may be called a "greedy" service, where the network will take as much traffic as possible at any instant from VCs under this service. Flow control can be used to throttle these VCs on a per VC basis when the network load becomes too high. There will be no requirements for predefined service contract parameters, which are difficult to set dynamically. This "greedy service" is expected to serve many types of best-effort traffic effectively and efficiently.

3 Credit-Based, Per VC Link-by-Link Flow Control

An efficient way of implementing per VC LLFC is to use a *credit-based* approach. A flow controlled VC is composed of one or more flow controlled *VC links* or simply *links*, connecting various network subsystems such as switches and adapters. Figure 1 depicts two flow controlled VCs (*VC1* and *VC2*) for which credit-based flow control is used for each link.

Figure 2 is a magnified view of the two flow controlled VC links between Adapter 1 and Switch 1. During the operation of a VC, two types of ATM cells, called *data* and *credit cells*, will be used. A data cell transports data belonging to the VC. A credit cell transports credit values and various credit-related management information for the VC. All credit cells are transported over some reserved VCs, called the *credit-carrier VCs*. Refer to [5] for a proposed credit cell format and some credit-related transaction types.

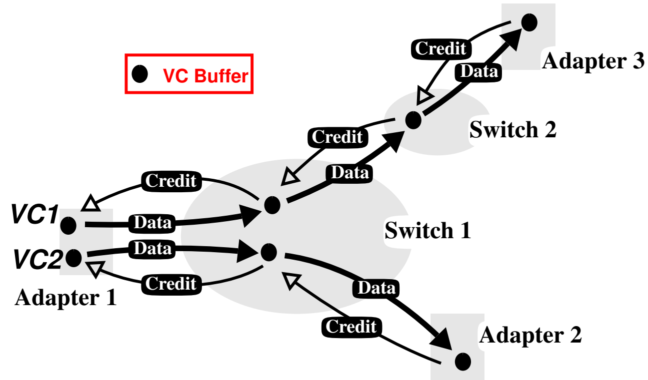

Figure 1 Credit-based flow control applied to each

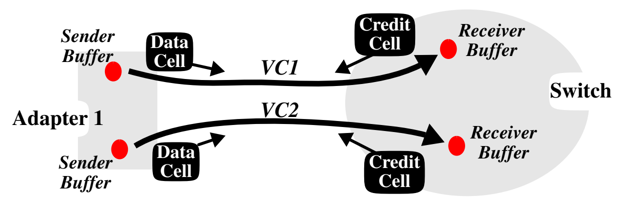

Figure 2 Magnified view of two flow controlled VC links in Figure 1

Each VC link is associated with a pair of *sender* and *receiver buffers*, which are also called *VC buffers*[1]. Transporting data cells from the sender buffer to the receiver buffer over the VC can be *flow controlled* to prevent overrun of the receiver buffer. For two consecutive links of the same VC, the receiver buffer of the upstream link is also used as the sender buffer of the downstream link.

The credit-based flow control over a VC link generally works as follows. 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 that there is a certain amount of buffer space available for receiving data cells of the VC. After having received credits, the sender is eligible to forward data cells of the VC to the receiver. Each time the sender forwards a data cell of a VC, it decrements its current credit count for the VC by one.

When receiving a credit cell for a VC the sender updates its credit count for the VC using an *absolute* updating method, as opposed to a relative or additive method. This means that the new credit count will be computed entirely from the newly received credit, independently of the old credit count. In

[1] VC buffers are denoted by dots in Figures 1 and 2.

particular, the new credit count will not be relative to the old credit count. This is in contrast with relative or additive updating used in some previously proposed "credit-like" flow control schemes [6, 7, 12], where the new credit count is equal to the old credit count plus the newly received credit value. The absolute credit updating allows a robust flow control scheme in the sense that any effect of a corrupted credit can be recovered automatically by the arrival of the next successfully transmitted credit [5].

Three credit-based flow control schemes, called the *N123*, *N123+* and *N23* Schemes, are described in [5], all of which use absolute credit updating. Generally speaking, the *N23* scheme is the most attractive one of the three methods, as it requires the smallest buffer memory. Moreover, it is shown in [5], while using an absolute (and thus fault-tolerant) credit updating method, the *N23* Scheme actually achieves the same effect as an additive updating method with perfect transmission, as far as buffer management is concerned. The other two credit-based flow control schemes, *N123* and *N123+*, can also be attractive due to their relatively simple implementation. Their extra buffer space is proportional to the link propagation delay and thus can be insignificant for local area networks where propagation delays are small. In this paper, we study only the *N23* method.

4 The *N23* Scheme: A Credit-based Flow Control Scheme

The "N23 Scheme" is a method of implementing the credit-based flow control scheme described above. This method has a number of desirable features [5], including (1) provision for transmitting credit cells at any low bandwidth, (2) robustness against corrupted credit cell, (3) size of the VC buffer for a VC bounded by a quantity depending only on the targeted bandwidth of the VC rather than the peak link bandwidth, and (4) bounded maximum bandwidth achievable by individual flow controlled VCs. Of course, this method has the usual features such as prevention of buffer overflow and underflow typically found in other flow control methods. This section briefly describes the *N23* Scheme. For details, refer to [5].

4.1 Definitions and Terminologies

For describing the *N23* Scheme, it is convenient to consider three consecutive nodes of a VC, referred to as *upstream*, *current* and *downstream* nodes. Figure 3 depicts these nodes along with their VC buffers. With respect to the link between the upstream and current nodes, they are the sender and receiver nodes, respectively. Similarly, with respect to the link between the current and downstream nodes, they are the sender and receiver nodes, respectively.

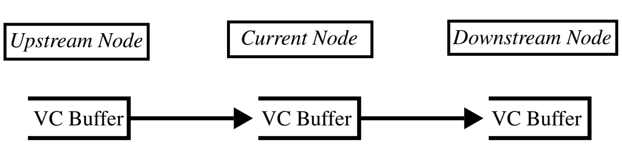

Figure 3 Three consecutive nodes of a VC and their VC buffers

The following definitions and notations will be used throughout:

- R = Round-trip link delay between the current and upstream nodes, including both the link propagation delay and the time for handling data cells and processing credit cells at the two endpoints. R is a system parameter, which can be determined at the system configuration time, and can be measured by executing a built-in looping routine.
- B_{VC} = Targeted bandwidth of a VC over time R.
- B_{link} = Peak bandwidth of the underlying physical link over time R.
- *Cell_Size* = 53 bytes, for ATM cells.

Note that $B_{VC} \leq B_{link}$ is always true, and in general, B_{VC} can be much smaller than B_{link}. This is the reason why the *N23* Scheme is designed so that the VC buffer size for a VC is bounded above by a quantity proportional to B_{VC} rather than B_{link}.

4.2 The *N2* and *N3* Zones of VC Buffer

For the *N23* scheme, the VC buffer is composed of two zones for each VC crossing the current node. As depicted in Figure 4, they are called *N2* and *N3* zones, each possessing *N2* and *N3* cells, respectively.[2]

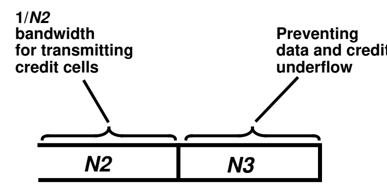

Figure 4 Two zones of each VC buffer

The *N2* zone allows less frequent sending of credit cells (while still preventing data and credit underflow) to minimize the bandwidth for transmitting credit cells.

The value of *N2* can be the same for all flow controlled VCs in a network. The value can be a design or engineering choice. For example, a reasonable value for *N2* in this case could be 10.

- It is also possible that different VCs may use different *N2* values. For example, a VC for which automatic recovery from errors is critical may choose to use a relatively small *N2* in order to increase the chance that another credit cell will immediately follow a corrupted credit cell. See Section 4.5 for further discussions on the error recovery issue.

The *N3* zone prevents data and credit underflow, so that the VC can sustain its targeted bandwidth as long as the upstream node has data to forward and the downstream node has space to receive them. The value of *N3* is given by Equation (3).

- The *N3* zone must be large enough to prevent data underflow. It must hold enough data cells to continue sending downstream at the targeted

[2] The other two credit-based schemes in [5], called *N123* and *N123+*, require three buffer zones.

bandwidth, B_{VC}, while waiting for new data cells from upstream reflecting the most recently sent credit cell on this VC.

- To prevent credit underflow, the upstream node must receive credit cells each with a sufficiently large credit value. This prevents the upstream node (when operating at the targeted VC bandwidth) from depleting its current credit before the next credit cell arrives. Note that each new credit value is *N2* + *N3* less the buffer fill. Thus, for a given value of *N2*, the *N3* value must be large enough to allow the sending of sufficiently large credit values.

4.3 Basic *N23* Algorithm

Figure 5 depicts the basic algorithm for the *N23* Scheme.[3] The current node will send a credit cell (to the upstream node) for a VC each time after it has forwarded *N2* data cells of the VC (to the downstream node) since the previous credit cell 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. A credit cell need not be sent when the combined area is totally occupied.

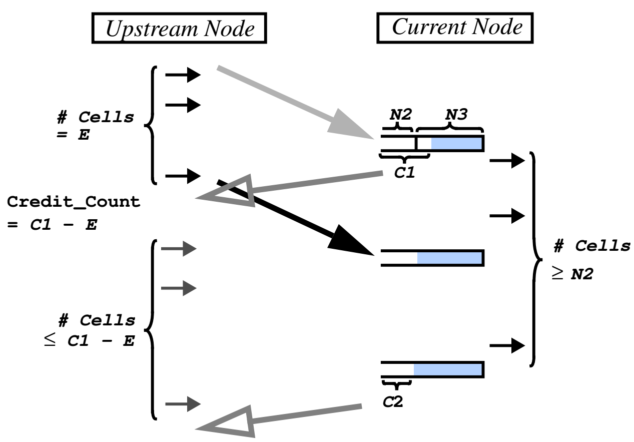

Figure 5 The *N23* Scheme

Upon receiving a credit cell with credit value *C* for a VC, the upstream node 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 upstream node maintains a count, called Credit_Count, for the VC. Credit_Count could be set to be *N2* + *N3* initially. Each time the upstream node forwards a data cell of the VC (to the current node), it decrements the Credit_Count by one. It stops forwarding data cells (only of this VC) when the Credit_Count reaches zero, and will be eligible to forward data cells (of this VC) again when receiving a new credit cell (for this VC) which gives a positive value for $C - E$.

[3] In the figure, time flows from top to bottom, the occupied area of a buffer is shaded, and dashed arrows refer to transmission of credit.

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

$$\text{Credit\_Count} = \text{Credit Value in the Newly Received Credit Cell} - E \qquad (1)$$

where

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

Note that this updating is “absolute”, as defined in the end of Section 3, because the new Credit_Count is computed independently of the old Credit_Count.

One purpose of the *N3* zone, as stated earlier, is to prevent data overflow and underflow. As shown in [5], to achieve this for a VC of targeted average bandwidth of B_{VC} over time *R*, it suffices to choose *N*3 to be:[4]

$$N3 = R \cdot B_{VC} / Cell\_Size \qquad (3)$$

By increasing the *N3* value, the VC can transport data cells at a proportionally higher bandwidth.

4.4 Bounding the Bandwidth for Transmitting Credit Cells and Required *N2* Value

For a given value of *N2*, the current node will send a credit cell for the VC to the upstream node each time after having forwarded *N2* data cells of the VC to the downstream node. Thus over this VC, the link between the current and upstream node will transport credit cells no more than once every *N2* data cells. By using *N2* value of at least *X* for all VCs, the overhead of transmitting credit cells can be limited to an arbitrary fraction (1/*X*) of the link bandwidth. However, the larger the value of *N2* is, the larger the required memory in the *N2* zone is. The selection of the *N2* value is a design or engineering choice. Typically, *N2* is chosen to be about 10 so that credit cells consume no more that about 10% of total network bandwidth.

4.5 Robustness of the *N23* Scheme

Using a strong error check such as a 32-bit CRC (see a proposed credit cell format including CRC-32 in [5]), the probability of undetected incorrect credit cells can be kept at an acceptably low level. A corrupted credit cell detected by the CRC at the sender will be discarded and the arrival of the next successfully transmitted credit cell for the same VC will recover from the error automatically.

After the sender detects and discards a corrupted credit cell, let *A* be the number of future credit cells that will arrive anyhow before the next successfully transmitted credit cell is received. Then $A = \lfloor (B+G)/N2 \rfloor$ where *B* is the number of data cells of the VC in the receiver at the sending time of the credit cell (which becomes corrupted), and *G* is the number of additional data cells of the VC which may still arrive reflecting the last successfully delivered credit cell. Note that *B* + *G* can be as large as the maximum number of cells the VC buffer can hold.

[4] There should actually be a ceiling of the right hand side quantity of equation (2).

Thus after the sender discards a corrupted credit cell, there could be *A* additional future credit cells for the same VC forthcoming, any of which if successfully transmitted, will remove any effect caused by the corrupted cell. The value of *A* can be increased to improve this automatic protection against corrupted credit cells, by decreasing *N2* (at the expense of increased bandwidth overhead for credit cells transmission) or increasing the VC buffer size, or both. Note that networks with large propagation delays have larger VC buffers (because of a large value for *N3*) and therefore the credit-based flow control schemes using absolute updating will be more fault-tolerant than networks with small propagation delays.

However, after the sender detects a corrupted credit cell, sometimes another credit cell will not immediately follow after the corrupted credit cell, because $B + G < N2$. In other words, the corrupted credit cell is the last credit cell generated by a burst of data. In this case, after the sender has waited for credit cells for the VC longer than some time-out period, it can request the receiver to send credit value for a VC (using, for example, *Sender-Request-Credit* in [5]). The receiver can also send redundant credit cells for a VC at any frequency to increase the protection against error. The absolute nature of the credit count updating by Equation (1) is such that as discussed in Section 3, effect of lost credit cells can be automatically recovered by any future successfully transmitted credit cell for the same VC. The only impact of a lost credit will be a potential, temporary delay in forwarding more traffic over the VC.

4.6 Orthogonal Relation to Switching and Scheduling

The credit-based per VC flow control mechanisms discussed in this paper are orthogonal to issues related to switching and scheduling functions associated with a switching node. For a flow controlled VC link, one side of the link does not need to know whether the other side is a switch or not. The flow control itself has no concerns on implementation matters at either side of the link related to how scheduling and/or switching of data cells of various VCs are performed. That is, flow control functions prevent data overflow and underflow, whereas switching and scheduling functions are responsible for implementing various services, such as guaranteed bandwidth and latency for certain VCs, on top of the flow control mechanism. However, as discussed in [5], underlying flow control schemes can facilitate efficient scheduling.

4.7 VC Buffer Sizes for the *N23* Scheme

Figure 6 is a summary of the required per VC buffer sizes for the *N23* Scheme at each node, for various link lengths and targeted VC bandwidths B_{VC}. The calculation assumes that $N2 = 10$ for all VCs, and the propagation delay per km is 5 microseconds. In addition, it assumes that the time for handling data cells and processing credit cells at the two endpoints consumes additional 5 microseconds, as in the BNR/Harvard ATM switch.

Link Length	10 Mbps	100 Mbps	622 Mbps
1 km	10+1 cells 1 KBytes	10+4 cells 1 KBytes	10+23 cells 2 KBytes
10 km	10+3 cells 1 KBytes	10+25 cells 2 KBytes	10+155 cells 9 KBytes
100 km	10+24 cells 2 KBytes	10+238 cells 14 KBytes	10+1475 cells 79 KBytes
1000 km	10+235 cells 13 KBytes	10+2360 cells 126 KBytes	10+14679 cells 779 KBytes

Figure 6 Per VC buffer size in #cells (i.e., $N2 + N3$), and #KBytes at each node for the *N23* Scheme

5 Simulation Approach

The simulator that produced the results in this paper was designed primarily to verify the architecture of the BNR/Harvard switch. It models much of the switch down to the level of registers and clock cycles, and thus provides very accurate timings at the level of individual ATM cells.

Reflecting the actual switch architecture, the simulator uses a common memory and output scheduling. Each VC has its own queue of cells, and VCs are divided into a small number of groups for scheduling; within each group, VCs that have both cells and credit are serviced in round-robin. The simulations in this paper either involve only one group, or treat them as priority levels. In general, the credit mechanism does not constrain scheduling except for VCs that have no credit; schedulers for various qualities of service may be implemented without concern for credit or buffer management.

The Awesime C++ threads package [11] provides the simulator's framework. Each hardware functional block is simulated with a thread, and threads communicate only through simulated registers or busses. A thread can allow time to pass between events by suspending itself for that amount of time; this is the primary synchronization method within the simulator. A simulated switch uses roughly 1000 threads of 50 different types.

Each simulation can be configured in two areas. Switches and hosts can be connected by links with specified propagation delays. Virtual circuits can be created with specified traffic patterns. The simplest pattern is called greedy, and sends as much data as it can, limited only by the credit mechanism. Most other patterns alternate idle periods with bursts of back-to-back cells. Each VC has an *N3* value for each link it traverses, usually chosen by Equation (3) to accommodate the VC's desired peak bandwidth. Each VC also has a priority.

The output of the simulator is a time-stamped trace of the cells that enter and leave each switch and host. A specialized visualization program extracts relevant statistics and produces graphs, including those in this paper. Bandwidth graphs show the fraction of link bandwidth used, averaged over an interval surrounding each data point; the interval is usually 25 cell times. The bandwidth used includes the ATM header as well as payload. Bandwidth can be measured for a single VC's data cells over a link, for all traffic on a link, or for credit cells alone. Other graphs include end-to-end delay for a VC's cells, the total number of cells queued in a switch, and the number of cells credited to a VC at a particular host or switch. The horizontal axis of each graph measures time.

6 Full Link Utilization: “Overbooking Inequality”

One can see from Figure 6 that local area networks can support a large number of VCs with large B_{VC}, using only a moderate amount of memory. Each of these VCs may assume a high bandwidth at various times whenever network load permits. For example, on a 1 km link, one megabyte of memory can maintain 1,000 VCs, each of which can operate at a speed as high as 100 Mbps, i.e., at B_{VC} = 100 Mbps.

These VCs obviously cannot all operate at their peak bandwidth simultaneously, over the same physical link of bandwidth B_{link}, equal to say, hundreds of megabits per second. That is, the “overbooking” inequality holds:

$$\sum B_{VC} > B_{link} \qquad (4)$$

where the summation is over all the VCs on the link. In fact, the idea of LLFC is to make the left-hand side much larger than the right-hand side, in order to maximize link utilization. The credit-based flow control schemes of this paper will flow control these VCs dynamically so that they can slow down when the link is congested. However, as soon as the link congestion situation lightens, each of these VCs can immediately operate at speeds as high as possible, up to its peak bandwidth B_{VC}, to make the maximum-possible use of the available bandwidth.

In some sense, the fundamental reason to use link-by-link flow control or fast feedback is to allow the Overbooking Inequality (4), in order to let VCs peak at high speeds whenever possible. This is in sharp contrast with contract-based, rate-control approaches through which the VC admission process will disallow such inequality for traffic entering the network.

Figure 7 shows an example of this overbooking. There are two VCs sharing the same output link of a switch. *VC1* is a non flow-controlled, high-priority VC carrying compressed video traffic with bandwidth alternating between 10% and 66% of the link bandwidth. *VC2* is an *N23* flow-controlled, low-priority VC carrying best-effort (greedy) traffic such as a file transfer.

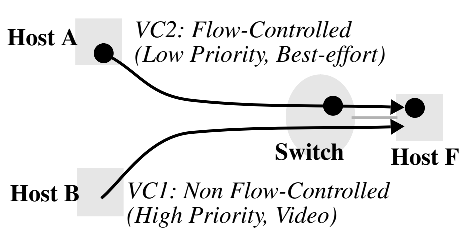

Figure 7 Two flow-controlled VCs competing for the same output link of a switch

Suppose that for *VC2* the round-trip link delay *R* between Host A and the switch is 155 cell times. Assume that the VC buffer at the switch for *VC2* has an *N3* value large enough to sustain the full link rate, that is, $B_{VC} = B_{link}$. Then, by Equation (3), *N3* = 155. Of course, *VC2* cannot sustain at that rate because *VC1* is also sending cells over the same link. Whenever there is competition, the switch scheduler at the output link will do the obvious thing, namely, ensuring that *VC1* will win. Ideally, *VC2* would vary the rate at which it sends cells so as to use all bandwidth not used by *VC1*, keeping the link fully utilized.

The results of a simulation of this scenario are shown in Figure 8. The throughputs of the two VCs always sum to 100%, so *VC2* is in fact filling the gaps in *VC1*’s traffic. Furthermore, *VC2* is doing this without buffering large numbers of cells in the switch: *VC2* never uses more than *N2* + *N3* = 10 + 155 = 165 cell buffers. Notice that the graph of *VC2* lags slightly behind *VC1*, as is expected. Both of these result from the flow control mechanism’s ability to quickly back-pressure and draw-in data for *VC2* as *VC1* changes its load.This allows the scheduler to achieve what is expected, without having to be concerned with buffer management. The rest of this paper explores the detailed behavior of this flow control mechanism..

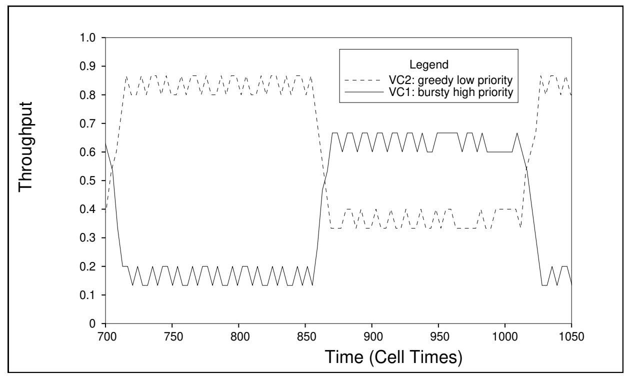

Figure 8 The greedy VC (VC2) takes up the left over bandwidth. The link is fully utilized.

7 Understanding Credit-Based Flow Control

In the following sections, we will present simulation studies and results which will demonstrate the effectiveness of the *N23* credit scheme. Please note that an *N2* of 10 is used for all simulations.

7.1 Flow Controlled VC in Noncongested Situations

Consider the simulated situation in Figure 9. A single VC starts at host A, travels through the switch, and terminates at host B. Each link has a propagation delay of 50 microseconds, which is the time it takes to send 77.5 cells; thus *R* is 155 cell times. Host A generates a cell for the VC every third cell time and Host B consumes cells as fast as they arrive. The simulation uses an *N3* of 51, which is the *R* / 3 that Equation (3) predicts will guarantee a bandwidth of at least one third of the link capacity.

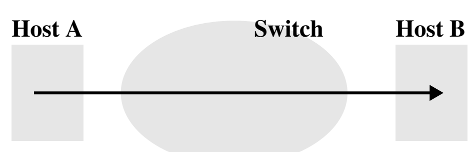

Figure 9 Single Noncongested VC

Figure 10 displays the simulated bandwidth used by the VC as

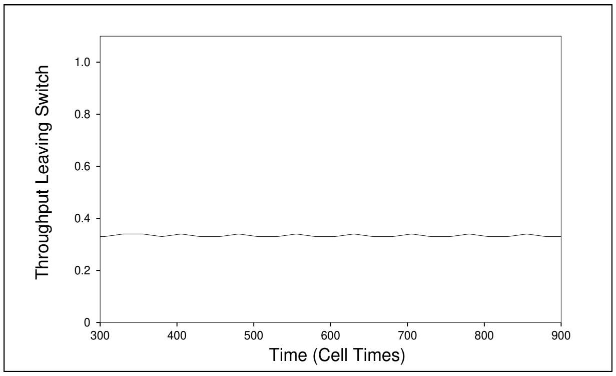

Figure 10 Offered load of 33%. Desired bandwidth of 33% is achieved.

it leaves the switch; the VC achieves the desired bandwidth of one third. As indicated in Figure 11, credit is always available at the switch for the VC, so the switch can pass on the VC's cells without delay. Since host B consumes cells as they arrive, it sends one credit cell back to the switch for every *N2* data cells. The vertical rises in the graph correspond to the arrival of these cells at the switch. Host B sends *N2* + *N3* credits in each cell, but the switch subtracts *E* from this value. Since the switch maintains the bandwidth of 1/3, *E* is always roughly *R* / 3, which is equal to *N3*, so the effective credit is roughly *N2*. Data cells arrive at the switch on every third cell cycle, and leave immediately, so the credit count in Figure 11 decreases by one every three cell times. Since Host B sends a credit cell with an effective value of *N2* for every *N2* cells it receives, a new credit arrives at the switch just as the switch is running out of the credit contained in the previous credit cell.

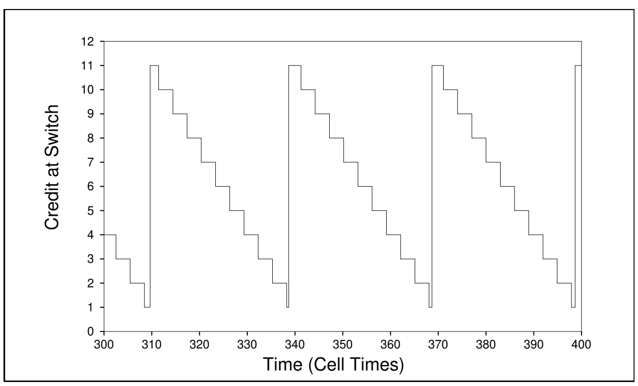

Figure 11 Offered load of 33%. Credit arrives just before it is depleted.

Figure 12 and Figure 13 show a situation with one change: the VC is greedy, and thus willing to send as much data as the flow control system will allow. In each period of *R*, the switch sends *N2* + *N3* cells as a full-speed burst; from then until the end of the period, its *E* is as large as the credits returned by host B, so the switch cannot send more data. Figure 13 does not show the credits received during this time, since they all have effective values of zero. The first credit cell that arrives after the end of the period triggers another burst. The burst sometimes falters at the beginning, if the first credit arrives when *E* is greater than *N3*, and the effective credit is thus less than *N2*. Once *E* has decreased enough, the switch can send another continuous burst of *N2* + *N3* cells.

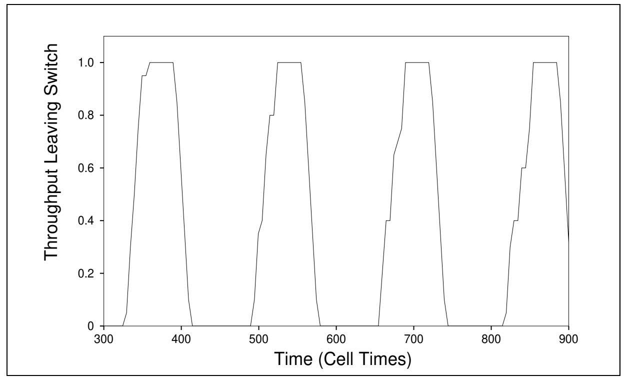

Figure 12 Offered load of 100%. Average bandwidth of 1/3 is achieved, but in bursts.

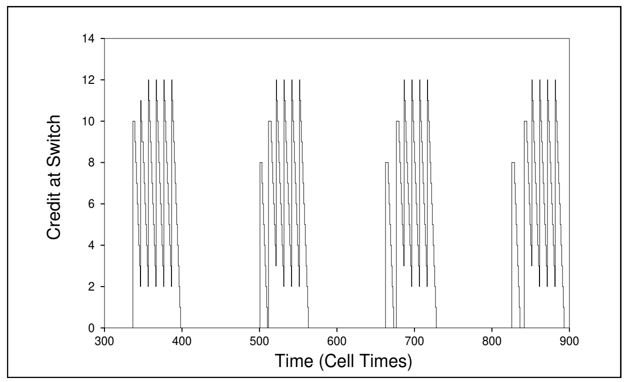

Figure 13 Offered load of 100%. Credit is depleted early during each round trip time period.

The average bandwidth achieved by the VC in Figure 12 is 39% of the link rate, which is the upper bound (*N2* + *N3*) / *R* proven in [5]. Increasing *N3* would allow the VC to send faster, by sending more cells in each *R* period; any *N3* value less than *R* has the effect of limiting a VC's peak bandwidth. If *N3* were increased to *R*, the VC would experience the same smooth flow seen in Figure 11, but at the full link rate. An increase in *N3* would not affect the throughput in Figure 10, because host A in that scenario only offers a load of 33%; such an increase would only increase the credit values.

If a VC is not congested and has a sufficiently large *N3*, credit-based flow control creates no additional delay at each node. Simulations such as that of Figure 11 show that for the *N23* Scheme, in noncongested situations the VC buffer need never store more than one cell. When congestion occurs, data cells may be buffered in the VC buffer. After congestion clears, the extra delay at the node is no more than that determined by the size of the VC buffer.

7.2 Resource Sharing under Congestion

If the switch is congested, the *N23* credit scheme prevents overflow of data cells. Together with a round robin scheduler, it is guaranteed that each competing VC gets an equal share of the output link.

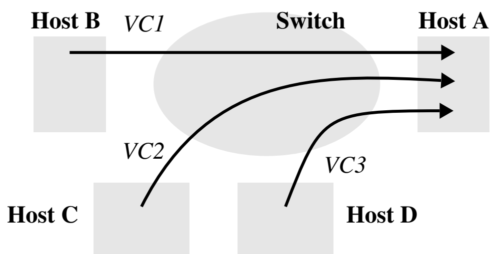

Figure 14 Resource sharing under congestion

Consider the simulated situation in Figure 14, three greedy VCs competing for the same output link. *VC1*, *VC2* and *VC3* are greedy VCs whose *N3*s are set so as to allow them to send at a maximum rate of 100%, 72% and 40% of B_{link} respectively. *R* is 155 cell times and *N3* is set to 155, 113, and 63 for *VC1*, *VC2* and *VC3* according to Equation (2). The initial credit value at the switch is *N2* + *N3*. Simulation results show that the switch is congested because the traffic competing for the output link sums up to 212% B_{link}. Eventually the sender generates data cells at the rate of 1/3 of B_{link} because of the equal sharing of the output link

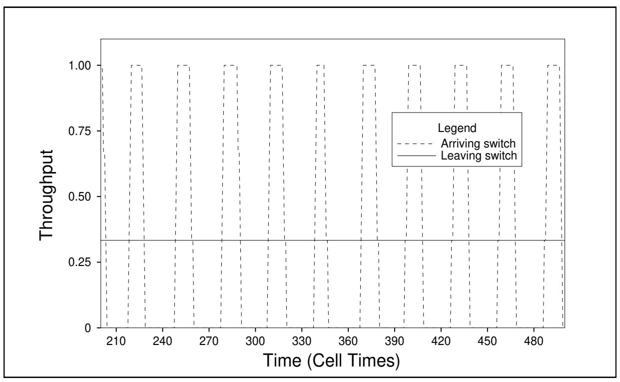

Figure 15 Input is bursty, output is smooth. Both are 1/3 of the link bandwidth.

and the *N23* back pressure. Figure 15 shows the throughput arriving and leaving the switch for *VC1*. The sawtooth pattern for the arriving curve arises because of the bursts of 10 cells sent by the upstream node upon receiving a credit value of 10. The straight line for the departing curve shows there is exactly one cell leaving the switch during any interval of three cell times for *VC1*. Note that *N3* is the predominant defining factor of queue length and delay for greedy traffic. Figure 16 shows the queue lengths of the three VCs inside the switch. Since *VC1* initially sends at the full link speed (*N2*+*N3* = 165 cells back to back) into the switch, but can only send 55 cells over 165 cell times out of the switch, about 110 cells must be buffered in the switch. Thereafter, the number of cells buffered remains at about 110, because after the sender is back pressured, both input and output links operate at the same speed (1/3 of the link bandwidth). The fluctuation of 7 cells within a 30 cell time period arises from the fact that the sender bursts 10 cells back to back, whereas the switch sends 3 cells for the first 10 cell times and another 7 during the next 20 cell times while the sender runs out of credit, and thus is silenced. Because each VC is running at 1/3 of the link speed, we expect the delay to be 3 times the queue length.

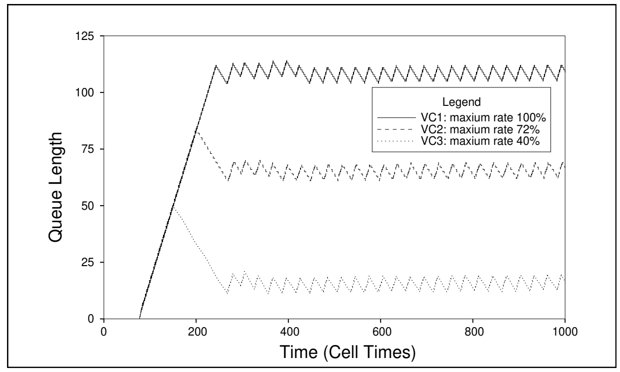

Figure 16 Queue length increases with increasing *N3*

This demonstrates that although the VCs are treated equally, they may have different queue lengths and thus experience different delays because of their different *N3* values.

7.3 Resource Sharing for Multiple Switches

Consider the two-switch case, shown in Figure 17, with four "greedy" VCs. *VC1* and *VC2* compete for the output link of the first switch and travel through the second switch, where *VC1* competes with VC3 and VC4 for another output link. The propagation delay for all links is 50µs, so *R*=155 cell times. All VCs are greedy with an *N3* of 155 so as to facilitate the sending at the full link bandwidth.

Simulation results show that bandwidth usage of the three VCs competing for the output link of Switch 2 quickly converges to exactly 1/3 each. Since all three VCs have the same priority, the scheduler sends cells from these VCs in a round robin fashion. As we shall see, there are always cells to send for each VC, since the buffer fill remains at over 100 cells per VC.

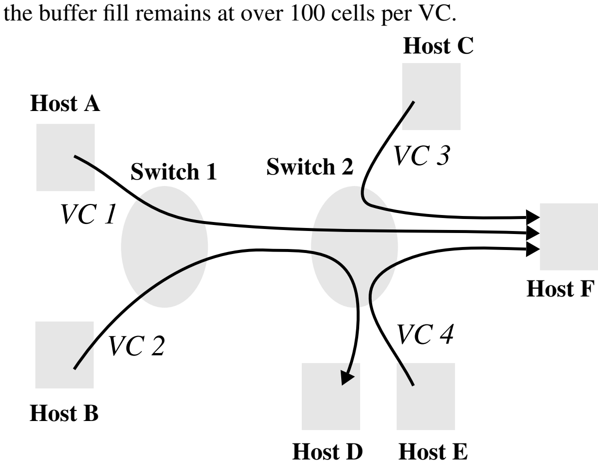

Figure 17 Resource sharing for multiple switches

Figure 18 shows the throughput of *VC1* and VC2 leaving Switch 1. Notice that initially, both VCs send at 1/2 the bandwidth. However, since *VC1* is only given 1/3 of the bandwidth leaving Switch 2, the credit being sent back to Switch 1 for *VC 1* effectively exerts a back pressure, causing the throughput of *VC1* to throttle back to 1/3 of the link bandwidth. *VC2*, being greedy as well, quickly uses the remaining 2/3 of the link bandwidth. This demonstrates how the flow controlled

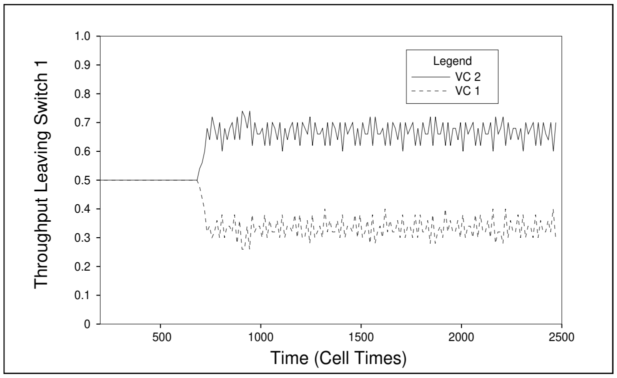

Figure 18 Throughput of VC1 drops to 1/3 & VC2 takes the remaining 2/3 of the link bandwidth.

network as a whole ensures that every VC gets a fair share of its most congested link.

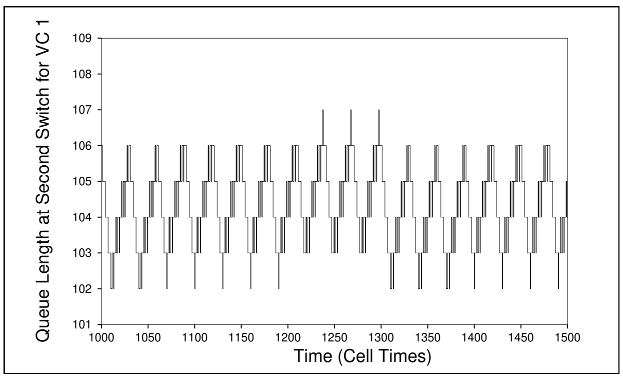

Figure 19 Buffer fill has a lower end of 102-103 and fluctuates by 3 or 4 cells.

Figure 19 shows the number of cells queued in the buffer for *VC1* at Switch 2. The buffer fill graphs of *VC3* and *VC4* similar to that of *VC1* in the steady state case. Thus the delay across Switch 2 for these three VCs is similar as well. Note that the buffer fill remains between 102 and 107 cells. This can be explained as follows: Let us focus on *VC1*. Cells are exiting Switch 2 at 1/3 the bandwidth. Initially, they are arriving at 1/2 the bandwidth. This mismatch causes the buffer in Switch 2 to begin filling up. It takes 30 cell cycles for 10 cells to be sent out of Switch 2, so a credit cell is sent back to Switch 1 every 30 cell cycles. That credit cell contains the value *N2+N3-Bufferfill*, where *Bufferfill* is the number of cells buffered in Switch 2. Upon receipt of the credit cell, Switch 1 subtracts *E* from that credit value, which is the number of cells it has sent in the last round trip time. This effective credit decreases as the buffer fills up until the following equilibrium is achieved:

$$N2 + N3 - Bufferfill - E = 10$$

where *E* = 155/3 since the VC ends up sending at 1/3 the link bandwidth. Thus the left hand side represents the credit calculated by Switch 1. The right hand side represents the actual value the credit cell must contain in order for the VC to use exactly one third of the bandwidth. Since a credit cell is sent back from Switch 2 every 30 cell cycles, it must have a value of 10 in order for the VC to send at 1/3 the bandwidth. Thus, according to the equality, *Bufferfill* = 103, which corresponds to the lower end value in Figure 19. This is case for *VC3* and *VC4* as well.

To explain the fluctuation of 3 or 4 cells in the buffer fill, we must examine the rate of flow into and out of Switch 2 during one credit cell sending period, which, as mentioned above, is 30 cell cycles. During the first 20 of these cell times, Switch 1 sends at a rate of 1/2, thus sending approximately 10 cells during this period, during which time Switch 2 sends out 6 or 7 cells, since it is sending at a rate of 1/3. During the next 10 cell cycles, no more cells are sent from Switch 1, since the credit has been depleted, and Switch 2 sends out those 3 or 4 cells. Thus, the buffer fluctuates by 3 or 4 cells as is verified by Figure 19.

7.4 Competition between Credit and Data cells

Whenever two VCs share a link, but flow in opposite directions, credit and data must compete for bandwidth. The average credit bandwidth in one direction is the data bandwidth in the other divided by *N2*. In the switch being simulated, credit cells have priority over data, so any increase in credit bandwidth decreases data bandwidth in the same direction. These relationships form a cycle.

Figure 20 is an example of this. Suppose that both VCs are greedy and have a large enough *N3* to sustain the full link bandwidth, so that only competition with credit will prevent them from sending at full speed. Let *B1* and *B2* be the data bandwidths of the two VCs; then $B1 = 1 - (B2 / N2)$ and $B2 = 1 - (B1 / N2)$. Substituting $1 - (B2 / N2)$ for *B1* in the second equation and solving for B2 indicates that $B2 = N2 / (N2 + 1)$. If *N2* is 10, then 91% of the bandwidth in each direction should be used by data, and 9% by credit. Simulation confirms this prediction.

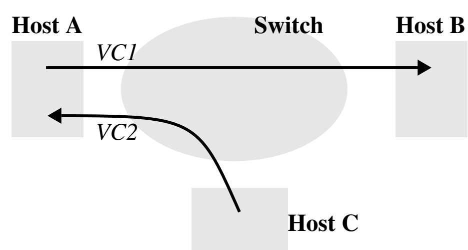

Figure 20 Credit and data competition

8 Comparing with Other Approaches

8.1 Request-Response with and without Flow Control

Applications which send requests and wait for responses seem particularly suited to link-level flow control. Consider the NFS [17] file system protocol, which is based on a remote procedure call (RPC) mechanism. NFS implementations usually adjust time-out periods based on round-trip times, but do not explicitly adjust the amount of load offered in response to congestion (though see [15]). The underlying RPC mechanism limit the load to some extent, since clients must often wait for the reply to one RPC before sending the next. However, NFS requests and responses range in size from a few hundred bytes to 8 kilobyte disk blocks, so this limit is not very stringent or precise. Since

NFS clients must wait during any delay imposed by the network, artificially smoothing bursts in NFS traffic is not desirable.

An underlying link-level VC flow control mechanism would allow NFS to send at full speed when possible, but would throttle it when the network was congested. Without such flow control, NFS would force the network to buffer large amounts of data, pausing only after each RPC request. Simulations based on traces of real NFS traffic indicate that flow control substantially decreases the amount of buffer space required to handle a given number of NFS connections, without increasing delay. For instance, Figure 21 contains the number of cell buffers that

# of clients	with flow ctl.	w/o flow ctl.
4	6	6
8	35	120
12	70	340
15	100	550

Figure 21 Peak cell buffers required for NFS

would be required in a switch to handle different numbers of clients running a particular set of traces. These numbers are the maximum queue lengths observed during one second of simulated traces. The traces are of clients reading each file in a directory hierarchy; each client communicates with a different server, but all connections share a single bottleneck link with a round-trip time of five microseconds. The worst case buffer use occurs when many clients send a large request at the same time; these simulations give a feel for how likely that is to happen.

8.2 Multicast with and without Flow Control

Our BNR/Harvard switch features common memory and output port scheduling, and thus is well suited to multicasting. For a multicasting VC, the input branch enters a switch through one input port, and the branches exit through a number of output ports. Each of these output ports schedules its own branch asynchronously. In other words, if a port has sent cell1, it can send cell2 as long as it has credit, even if one of the other branches has not sent cell1 yet. At any time, there is only one copy of a data cell, stored in the common memory. When a data cell has been scheduled for transmission through every branch's output port, this cell's memory location is recycled. After every *N2* cell buffers have been recycled for a multicasting VC, a credit cell is sent back through the input port to the upstream node. The credit value is *N2*+*N3* minus the number of cells of the VC still residing in the common memory.

Non flow controlled multicast can consume large amount of buffer space, because a cell's buffer space can not be freed until all branches have sent out the cell. Consider the situation in Figure 22: one 5-port low priority bursty multicast VC shares output links with 5 high priority bursty unicast VCs. The upstream node of the multicast VC sends cells at the rate of 1/3 of the link speed on average, whereas the unicast VCs send at 1/2 of the full link speed on average. Because the VCs are all bursty, it is possible that the output links can become congested in the absence of any flow control, thus requiring a large buffer

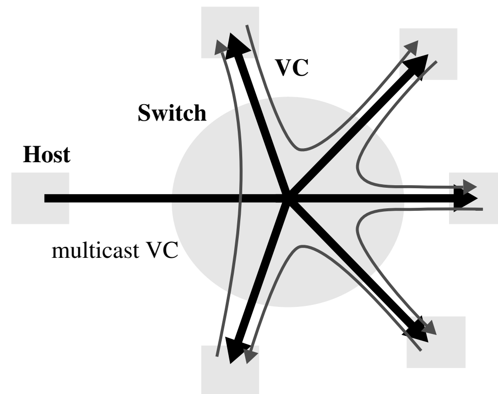

Figure 22 Multicast VC competes with a unicast VC at each branch port.

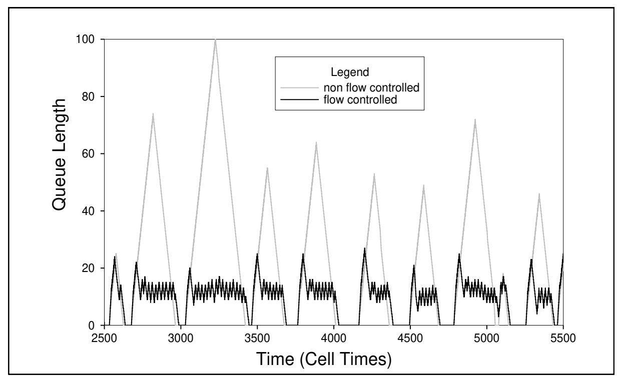

Figure 23 Flow controlled multicast VC consumes much less buffer space than the non flow controlled VC.

utilization. Figure 23 shows that flow controlling the low priority multicast VC creates a savings of the peak buffer space required. Figure 24 depicts the total number of cells the multicast VC has forwarded up to each of the cell times shown. The figure shows that the credit restriction does not affect the throughput: both curves average about 1/3 of the link speed. The only difference is that the bursty traffic is smoothed at the output side for the flow controlled VC, which is not shown in the figure.

We have simulated another situation in which the multicast VC has 15 branches. The results show that the throughput is slightly lower than the non flow controlled target rate. In order to send back each credit cell, all branches in the switch must send out at least *N2* cells. Thus any link with a competing burst will be a bottleneck, by preventing credit from being sent back. When the number of branches becomes larger, the probability that at least one branch is slowed down by a burst becomes higher and it is more likely that the throughput of all the output links are slightly lower than the target. For this reason we use a larger *N3* for multicast VCs; the, simulation results shown in Figure 23 and Figure 24 correspond to this larger *N3*.

8.3 Link-by-link Flow Control Compared with End-to-End Flow Control

End-to-end windowed flow control [18] is commonly used in applications such as file transfer. Such protocols often include congestion control algorithms to adjust a host's window size in

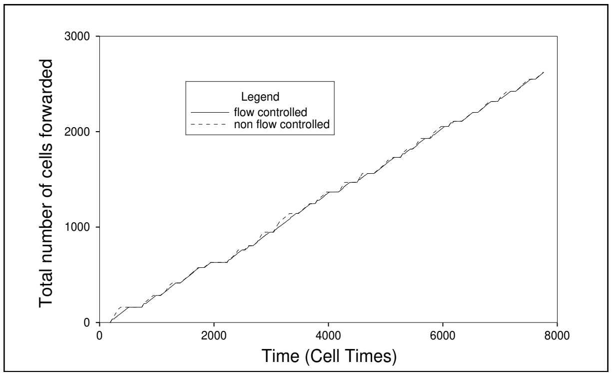

Figure 24 The throughput of the multicast VC is identical, whether flow controlled or not.

response to changes in the rate the network is able to sustain. Such protocols cannot react in less than one end-to-end round trip time. If congestion occurs suddenly on some link, the switch feeding that link must be prepared to buffer an entire end-to-end round-trip time's worth of data. Thus, if a network wishes to avoid discarding data, it must allocate an end-to-end round-trip time's worth of data in every switch for every connection.

In contrast, link-by-link flow control reacts to congestion within one link round-trip time. Thus, each switch must be prepared to buffer one link round-trip time's worth of data in every switch for every connection; the total amount of buffering devoted to a connection is one end-to-end round-trip time's worth. The ratio of switch memory requirements for the two methods is equal to the number of links traversed by a connection. In any network with more than a few switches, link-by-link flow control will require far less memory to provide lossless transmission than end-to-end flow control.

9 Conclusion

The simulation results presented in this paper confirm several beneficial properties of using link-by-link flow controlled virtual circuits. As described in Section 6, low-priority best effort traffic can dynamically fill the bandwidth gap possibly left by other high-priority traffic, thus achieving full link utilization. "Greedy" services, as defined in Section 2, can therefore be efficiently implemented.

When network congestion occurs, the size of the VC buffer of a flow controlled VC need never grow beyond a predetermined limit related to the VC's desired bandwidth and the round-trip link delay. Simulation results of Section 8.2 demonstrate that a flow controlled multicast VC uses a much smaller peak buffer size than a non flow controlled one while delivering the same throughput. Similar savings in buffer space is also observed in Section 8.1 for NFS traffic.

Simulations of Section 7 show that the flow control mechanism itself does not create extra delays under uncongested situations. If congestion does occur because the size of the buffer of a flow controlled VC is limited, the delay will be bounded when the congestion clears up. Other simulations in the section validate some properties of flow control that one would expect, such as fair resource sharing when using fair scheduling.

References

[1] ATM Forum, "ATM User-Network Interface Specification," Version 2.1, May 1993.

[2] ANSI T1S1.5, "AAL5 - A New High Speed Data Transfer AAL," Stds. Proj. T1S1.5 AAL-ATM, Nov. 4-8, 1991.

[3] "Preliminary Report on Broadband ISDN Transfer Protocols," Bellcore Special Report, SR-NWT-001763, Issue 1, Dec. 1990.

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

[5] H. T. Kung and A. Chapman, "The FCVC (Flow Controlled Virtual Channels) Proposal for ATM Networks," Version 1.3, October 1993. A summary of the paper is to appear in *Proc.1993 International Conf. on Network Protocols*, San Francisco, California, October 19-22, 1993, pp. 116-127.

[6] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. A. Webb, "iWarp: An Integrated Solution to High-Speed Parallel Computing," *Proceedings of Supercomputing '88 Conference*, Orlando, Florida, November 1988, pp. 330-339.

[7] 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.

[8] 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.

[9] 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.

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

[11] D. Grunwald, "A Users Guide to Awesime: An Object Oriented Parallel Programming and Simulation System," University of Colorado at Boulder Technical Report CU-CS-552-91, 1991.

[12] 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.

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

[14] 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.

[15] R. Macklem, “Lessons Learned Tuning the 4.3BSD Reno Implementation of the NFS Protocol,” *USENIX Winter Conference Proceedings*, January 1991.

[16] 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, March 1993.

[17] R. Sandberg, et. al., “Design and Implementation of the Sun Network Filesystem,” *USENIX Summer Conference Proceedings*, pp. 119-130, June 1985.

[18] Postel, J. “RFC793: Transmission Control Protocol.” 1981.