Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks

H. T. Kung and Koling Chang
Division of Applied Sciences, Harvard University, 29 Oxford Street, Cambridge, MA 02138, USA

Abstract

*In credit-based flow control for ATM networks, buffer is first allocated to each VC (virtual circuit) and then credit control is applied to the VC for avoiding possible buffer overflow. Receiver-oriented, adaptive buffer allocation allows a receiver to allocate its buffer dynamically, to VCs from multiple upstream nodes based on their bandwidth usage. This paper describes, in detail, such an adaptive algorithm capable of supporting a wide range of link speeds and propagation delays, and also packing multiple allocation and credit records in a single message. Analysis and simulation results show that even under highly bursty traffic, the adaptive scheme guarantees no cell loss due to congestion, and achieves excellent performance in utilization, fairness, ramp-up and packing, while requiring only relatively small node memory and bandwidth overhead. The required memory need only be 4*RTT + 2*N, where RTT is the link round-trip time in cell cycles and N is the number of VCs.*

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 as easily as they have been using conventional LANs. That is, they can use the network at any time without first negotiating a "traffic contract" with the network. They would be able to acquire as much network resources as are available at any given moment, and 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 [6, 7, 9].

This paper gives new results related to adaptive buffer allocation in credit-based flow control, including:

- *A receiver-oriented buffer allocation scheme* (Section 6). Unlike previous sender-oriented buffer allocation schemes [6, 9], the receiver-oriented scheme here is designed to handle the case where VCs from *multiple* input links dynamically share the same buffer pool at the receiver. The description is given in such detail that the algorithm is fully specified without any ambiguity.
- *Analysis of the adaptive scheme* (Section 8). We show that the adaptive allocation does not cause cell loss, and provides fairness and guaranteed ramp-up with only modest amount of node memory.
- *Extensive simulation results* (Section 9). The receiver-oriented scheme has been simulated against stressful scenarios involving large disparity in link bandwidth and propagation delays. The scheme works well in all the tests.

2. Why credit-based flow control?

We briefly review credit-based flow control. A credit-based flow control method generally works over a flow-controlled link for a VC as follows. As depicted by Figure 1, before forwarding any data cell over the link, the sender needs to receive credits for the VC from the receiver. At various times, the receiver sends credits 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.

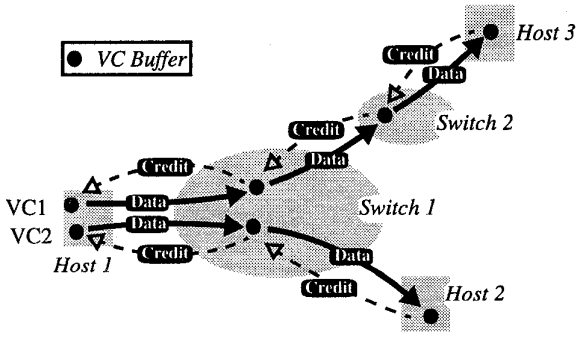

Figure 1: Credit-based flow control applied to each link of a VC

Simulation and analysis shown in this paper and others (see, e.g., [6, 8]) demonstrate that for a wide variety of traffic patterns, credit control is fair, uses links efficiently, minimizes delay and prevents buffer overruns. The credit system is especially well

This research was supported in part by BNR and Intel, 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.

suited to *bursty* data traffic that is unpredictable and has little tolerance for delay. Examples of bursty usages include interactive Mosaic users, high-level protocol packets, and RPCs.

In contrast, rate-based flow control (see, e.g., [11, 12]) may require less expensive hardware and may be effective for steady traffic, but appear to have difficulties in handing bursty traffic. Rate control will work well as long as rates can be set perfectly or near optimally. If the traffic is bursty, then rates can never be set correctly, and as a result they must be set adaptively.

Note that adaptation can not be precise due to inherent facts such as incomplete and out-of-date information used, and variable control delay. On the other hand, adaptation should not be precise either; in fact, the assigned rate of a VC should be higher than its fair share to allow opportunistic grabbing of available bandwidth.

Since adaptive rate setting can not and should not be precise, bounding the liability of overrunning switch buffers is a first order issue. Credit flow control is explicit about how much data a sender may transmit without receiving further credit. Lost or delayed feedback messages will not hurt, as the sender would just use the previous allowance. When necessary, the sender's transmission can be stopped completely (i.e., rate = 0). Effective approximation of these credit properties by rate control has been a major challenge. See [8] for elaboration of these reasons of why credit control is attractive for bursty traffic.

Implementing credit-based flow control, an experimental ATM switch [2], with 622-Mbps ports, has been developed by BNR and Harvard. This switch is operational and initial applications are under development.

3. Credit Update Protocol

The *Credit Update Protocol* (CUP) [6] is an efficient and robust protocol for implementing credit-based flow control. As depicted by Figure 2, for each flow-controlled VC the sender keeps a running total `Tx_Cnt` of all the data cells it has transmitted, and the receiver keeps a running total `Fwd_Cnt` of all the data cells it has forwarded. (If cells are allowed to be dropped within the receiver, `Fwd_Cnt` will also count these dropped cells). The receiver will enclose the up-to-date value of `Fwd_Cnt` in each credit record transmitted upstream. When the sender receives the credit record with value `Fwd_Cnt`, it will update the credit balance `Crd-Bal` for the VC:

```
Crd-Bal = Buf_Alloc - (Tx_Cnt - Fwd_Cnt)   (1)
```

where `Buf_Alloc` is the total number of cells allocated to the VC.

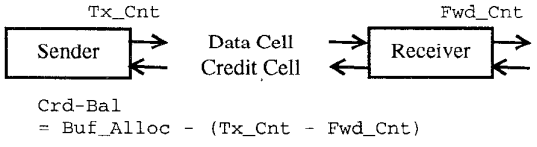

Figure 2: Credit Update Protocol (CUP)

Note that the quantity, `Tx_Cnt - Fwd_Cnt`, represents the "outstanding credits" which are cells of the VC that the sender has transmitted but the receiver has not forwarded. Thus `Crd-Bal` computed by Equation (1) is the proper new credit balance. See [6] for a scheme of using *credit_check cells* periodically to recover from possible loss of data or credit cells.

The frequency that the receiver sends credit records for a VC depends on the VC's progress. More precisely, each time after the receiver has forwarded "`N2`" cells for some positive integer `N2`, the receiver will send a credit record upstream. The value of `N2` can be set statically and dynamically.

A given `Buf_Alloc` value of a VC determines the maximum bandwidth allowed to the VC by credit flow control. For the rest of this section, we make a simplifying assumption that all links have the same peak bandwidth of 1, and represent the rate of a VC as a fraction of 1. Let RTT be the round-trip time, in cell transmission times, of the link between the sender and the receiver (see Figure 2) including both link propagating delays and credit processing time. Assume that the receiver uses fair scheduling policy between VCs when forwarding cells out from the receiver's output link. Then if there are N VCs competing for the same output link, the maximum average bandwidth over RTT that the VC can achieve is:

```
BW = Buf_Alloc / (RTT + N2*N)   (2)
```

The CUP scheme is a lower level and lighter weight protocol than typical sliding window protocols used in, e.g., X.25 and TCP. In particular, CUP is not linked to retransmission of lost packets. In X.25 or TCP, loss of any packets will stop advancing the window until the dropped cells have been retransmitted. To implement this, each packet carries a sequence number. In contrast, CUP does not handle retransmission and reordering problems, and does not require that each cell carry a sequence number.

It can be shown [7] that CUP produces the same buffer management results as the well-known "incremental" credit updating methods (see, e.g., [3, 5]). Instead of sending `Fwd_Cnt` values upstream, these methods send "incremental" credits to be added to `Crd_Bal` at the sender.

4. Static vs. adaptive credit control

We call a credit-based flow control *static* or *adaptive*, if the buffer allocation is static or adaptive, respectively. In a static credit control, a fixed value of `Buf_Alloc` will be used for the lifetime of a VC. Requiring only the implementation of CUP in Section 3 or some equivalent protocol, the method is extremely simple.

There are situations, however, where adaptive credit control is desirable. In order to allow a VC to operate at a high rate, Equation (2) implies that `Buf_Alloc` must be large relative to RTT + N2*N. Allocating a small buffer to a VC can prevent the VC from using otherwise available link bandwidth. On the other hand, committing a large buffer to a VC can be wasteful, because sometimes the VC may not get sufficient data and scheduling slots to transmit at the high rate. The proper rate at which a VC can transmit depends on the behavior of traffic sources, competing traffic, scheduling policy, and other factors, all of which can change dynamically or are not known a priori. In this case, adaptive credit control using adaptive buffer allocation as described below can be attractive.

In summary, adaptive credit control is static credit control plus adaptive adjustment of `Buf_Alloc` of a VC according to its

current bandwidth usage. For configurations where a large `Buf_Alloc` relative to RTT + N2*N is not prohibitively expensive, it may be simplest to implement just static credit control. Otherwise, some adaptive buffer allocation scheme may be used to adjust `Buf_Alloc` adaptively. The adaptation can be carried out by software.

5. Adaptive buffer allocation

Adaptive buffer allocation allows multiple VCs to share the same buffer pool in the receiver adaptively, according to their needs. That is, `Buf_Alloc` of a VC will decrease, if the VC does not have sufficient data to forward or is back-pressured due to downstream congestion. The freed up buffer space will automatically be assigned to other VCs which have data to forward and are not congested downstream.

Adaptive buffer allocation can be implemented at the sender or receiver. As depicted by Figure 3, in a sender-oriented adaptive scheme [6] the sender dynamically allocates a shared input-buffer at the receiver among a number of VCs from the sender that share the same buffer pool. The sender allocates buffer for the VCs based on their measured, relative bandwidth usage on the output port *p*. Adaptive results in [6, 9] all concern with sender-oriented adaptive buffer allocation.

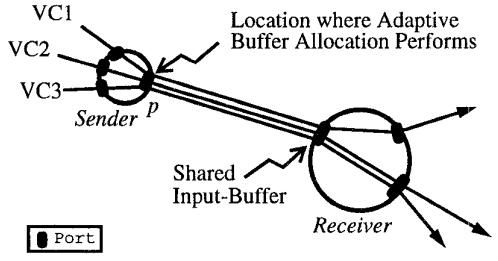

Figure 3: Sender-oriented adaptation

Receiver-oriented adaptation is depicted by Figure 4. The receiver dynamically allocates a shared output-buffer among a number of VCs from one or more senders that share the same buffer pool. The receiver allocates buffer for the VCs based on their measured, relative bandwidth usage on the output port *q*. This paper studies receiver-oriented buffer adaptation.

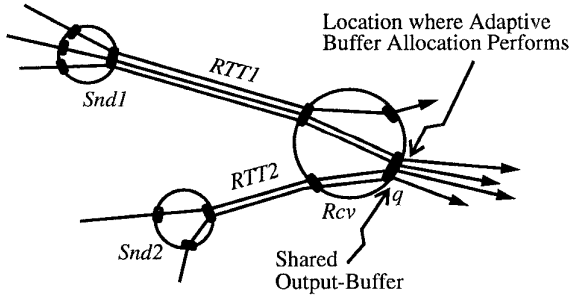

Figure 4: Receiver-oriented adaptation

Receiver-oriented adaptation is suited for the case where a common buffer pool in a receiver is shared by VCs from *multiple* upstream nodes. Figure 4 depicts such a scenario: the buffer pool at output port *q* of the receiver switch *Rcv* is shared by four VCs from two switches *Snd1* and *Snd2*. Note that the receiver (*Rcv*) can observe the bandwidth usage of the VCs from *all* the senders (in this case, *Snd1* and *Snd2*). 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.

Moreover, receiver-oriented adaptation naturally supports the adaptation of `N2` values for individual VCs to minimize credit transmission overhead and increase buffer utilization. As only the receiver needs to use N2 values, it can conveniently change them locally. See Section 7.5.

6. A receiver-oriented adaptive scheme

This section describes the receiver-oriented adaptive scheme of this paper. The scheme intends to achieve a set of goals, including fast ramp up, small memory requirement, low transmission overhead, robustness against transient errors, low implementation cost, and ease of use (i.e, no complex parameters to adjust).

To be precise, the scheme is given in pseudocode. The pseudocode is designed for simulation purposes. (See Section 9 for simulation results of the code.) For actual implementation on real systems, some "re-engineering" may be applied. For example, the scheme allows allocation to be computed at any (slow) speed to suit the implementation.

It should be relatively easy to read the pseudocode. The most important part is the adaptation formula used in the `Compute New Buffer Allocation` routine in Section 6.8. Note that sender and receiver in Figure 2 are also called upstream and downstream nodes, respectively. See Figure 5 for a "visualization" of the pseudocode.

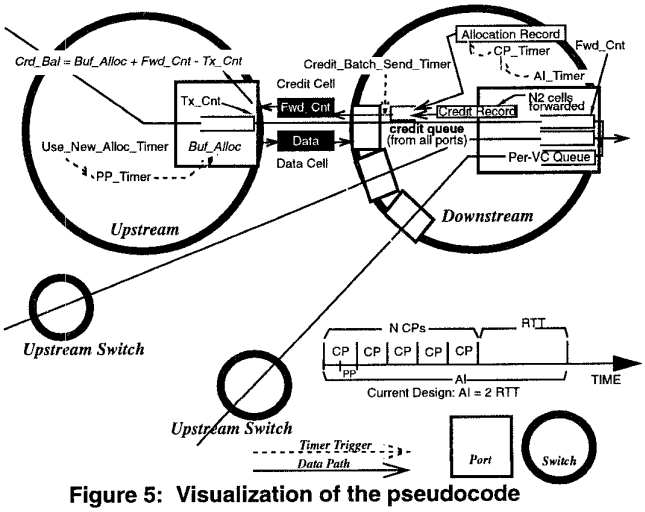

Figure 5: Visualization of the pseudocode

6.1. Shared buffer pool model

A shared buffer pool mode is depicted by Figure 6. The model allows links of various speeds and propagation delays. A cell time

is the time to transmit a cell over a link. Different input links may have different cell times. We use the following notations:

OLCT (Output_Link_Cell_Time)
: Cell time for the output link

ILCT (Input_Link_Cell_Time)
: Cell time for an input link.

MCT (Minimum_Cell_Time)
: Minimum cell time for all input or output links.

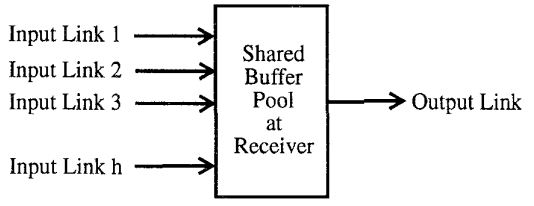

Figure 6: Shared buffer pool model

For simulation simplicity, we assume that the OLCT and the ILCT for any input link are integral multiples of MCT. If, for example, Input Link 2 is three times as slow as a link with the maximum speed, then the ILCT for Input Link 2 is 3* MCT.

The shared buffer pool in the receiver is assumed to have sufficient bandwidth for supporting the output link and all the input links simultaneously. That is, if all the links are of the maximum speed, then for each MCT, the shared buffer pool can input one cell from each of the input links and output one cell to the output link.

The output link is associated with separate per-VC buffers and a round-robin scheduling policy to transmit cells from "eligible VCs". Eligible VCs are defined as those having both positive credit balance and one or more data cells to send. For each OLCT, the scheduling policy selects a VC to send a data cell from among those that are eligible.

6.2. Parameters

Per VC parameters

RTT
: Round-trip time for VC, in number of MCTs. Included in RTT are (1) round-trip link propagation time, (2) credit return and processing time, and (3) credit record queueing delay due to credit packing (see Section 6.3).

VC_Weight
: VC's weight reflecting its RTT relative to RTT_{min} (see the definition of RTT_{min} below), i.e., VC_Weight = RTT/RTT_{min}

Alloc_Bound
: Maximum buffer allocation (in cells) VC ever needs for achieving its fair share of bandwidth. This bound will be used to avoid giving excessive buffer allocation to VC.

PD
: Propagation delay, in number of MCTs, of VC's input link.

Global parameters

N
: Active VCs are VC[1] through VC[N].

RTT_{max}
: Maximum RTT for any of the input links

RTT_{min}
: Minimum RTT for any of the input links

PD_{max}
: Maximum PD for any of the input links

AI
: Buffer allocation interval for VC. A new buffer allocation for each VC is computed once for each AI.

CP
: Buffer allocation computation period, i. e., the time interval between two consecutive buffer allocation computations taking place at the receiver for individual VCs.

Packing_Limit
: Maximum number of credit or alloc records a credit cell can pack.

Credit_Batch_Send_Period
: Time delay between the time when the receiver finishes the sending of a batch of packed credit cells, and the time when it starts sending the next batch. Credit_Batch_Send_Period is given in number of ILCT of the input link over which the credit cells will be sent.

MI
: Measurement interval over which each VC's bandwidth usage is measured

M
: Size of the shared memory or buffer pool in the receiver, in number of cells

Parameters only for upstream node

(Consider the upstream node corresponding to an input link.)

N'
: Active VCs over the input link are also called VC [1] and VC [N']

PP
: New allocation phase in period, i.e., time delay before phasing in the new buffer allocation for the next VC

6.3. Guidelines for setting parameters

Packing_Limit
: This code assumes that Packing_Limit = 6, i.e., up to six credit or alloc records can be packed in the 48-byte payload of a credit cell.

Credit_Batch_Send_Period = A * Packing_Limit, with $A \geq 1$
: This code assumes A = 10. This choice of the A value should, in general, result in a high packing degree of credit or alloc records for congested links. Smaller values for A can also work well for a downstream node with a large number of ports.

RTT
: To account for the queueing delay of credit record due to packing, RTT should include Credit_Batch_Send_Period * B, where B is ILCT (of the input link) divided by MCT. (See the definition of RTT in Section 6.2.)

AI
: AI must be at least RTT_{max}. This is the minimum-possible adaptation interval from a control theoretical viewpoint. In practice, networks should be engineered to avoid buffer sharing between VCs with extreme RTT disparities, in order to allow fast adaptation for VCs with relatively small RTTs. This code assumes AI = RTT_{max} + N * CP.

PP
: PP should be smaller than CP. This code assumes PP = CP/2.

MI
: To simplify implementation, MI is typically an integral multiple (>1) of AI. Assuming MI = 2 * AI, this code maintains both Prev_Fwd_Cnt and PPrev_Fwd_Cnt counters.

Alloc_Bound = $\lceil (4/3)*RTT*MCT / MAX(OLCT, ILCT) + 4/3 \rceil$
See Section 8.1 for a proof that Alloc_Bound given here would allow the VC to achieve its fair share of bandwidth.

M = $C*RTT_{max}*(MCT/OLCT) + D*N$, with $C \geq 4$ and $D \geq 2$
This pseudocode assumes C = 4 and D =2. See Section 8.2 for a proof that this value of M guarantees ramp up. Note that with any choice of the M value, this pseudocode guarantees that there will never be any cell loss due to congestion. However, larger M would allow faster ramp up.

6.4. Upstream node variables

Per VC variables

Tx_Cnt Number of cells transmitted by the upstream node

Crd-Bal Available credit at the upstream node

VC_Activated
Binary flag, when TRUE indicates that VC has been activated

Cur_Alloc Current buffer allocation for VC

New_Alloc[2]
VC's new buffer allocation for the next AI.Two new allocation banks (NABs) are used to allow incremental phase in of new buffer allocation for individual VCs.

Valid_NAB
Flag indicating from which NAB the value of New_Alloc will be read. (Initially, Valid_NAB = 0)

Global variables

Use_New_Alloc_Timer
This is a timer associated with an output of an upstream node. When this timer reaches AI, all outgoing VCs will start using their New_Alloc. See Section 7.3 on initializing this timer for allocation synchronization.

Use_New_Alloc_Timer_Expired
Binary flag, when TRUE indicates that Use_New_Alloc_Timer has reached AI

Global_NAB
Indicate the NAB from which values of New_Alloc should be read for all the VCs going out from the upstream node over the link in question. Global_NAB toggles for each AI. (Initially, Global_NAB = 0)

PP_Timer A new buffer allocation can phase in for some VC when this timer reaches PP. (Initially, PP_Timer = 0)

q Pointer to the next VC for which the new buffer allocation will be phased in when PP_Timer reaches PP. (Initially, q = 1)

6.5. Downstream node variables

Per VC variables

Rx_Cnt Number of cells received by the shared buffer system

Fwd_Cnt[2] Number of cells forwarded from the buffer system. Two memory banks are used to provide a "frozen" copy of Fwd_Cnt for incremental calculation of bandwidth usage.

Valid_Bank Indicate which memory bank in which the most recent Fwd_Cnt is stored

Prev_Fwd_Cnt
Fwd_Cnt for the previous allocation interval

PPrev_Fwd_Cnt
Fwd_Cnt for the allocation interval just before the previous one

N2 Each time after N2 data cells have been forwarded, VC is eligible for queueing a new credit record to be transmitted upstream.

N2_Cnt Number of cells forwarded since last queueing of credit record

VU Weighted VC bandwidth usage over the past MI cell cycles. (Initially, VU[x]= 0 for all x).

X Newly computed buffer allocation

Global variables

TU Total weighted bandwidth usage among all VCs over the past MI cell cycles. (Initially, TU = 0).

TQ Total queue length, or memory usage, of all VCs. (Initially, TQ = 0)

TQ_Snapshot
A copy of TQ to be used in buffer allocation computations for all the VCs with respect to the same allocation interval

AI_Timer A new buffer allocation interval starts when this timer reaches AI.

CP_Timer A new buffer allocation computation can start for some VC when this timer reaches CP. (Initially, CP_Timer = 0)

Credit_Batch_Send_Timer
When this timer reaches Credit_Batch_Send_Period, the receiver will start sending a batch of packed credit cells until the Credit_Send_Queue is emptied. Credit_Batch_Send_Timer increments by one for each ILCT of the input link over which credit cells will be sent. (Initially Credit_Batch_Send_Timer = 0)

Packing_Cnt
During packing, Packing_Cnt counts credit or alloc records in the Draft_Cell_Payload to be used in the next credit cell. (Initially, Packing_Cnt = 0)

p Pointer to the VC to which the next buffer allocation will be performed. (Initially, p = 1)

Tx_Credit_Cells
Binary flag, when TRUE indicates that it is time to send credit cells

AI_Expired Binary flag, when TRUE indicates that AI_Timer has reached AI

Use_Bank Indicate memory bank to which updated Fwd_Cnt should be written. Use_Bank toggles for each new AI. (Initially, Use_Bank = 0)

6.6. Switch interface variables

Per VC variables

Data_to_Send
Indicates that VC has data ready to send

OK_to_Send
Indicates VC has a positive credit balance

Send_Grant Indicates that VC has permission to send a cell at the next cell interval

Cell_Sent Indicates that a cell for VC has been transmitted

6.7. Upstream node pseudocode

Activate VC[i]

```
Cur_Alloc[i] = 1;
New_Alloc[i] = 1;
Crd-Bal[i] = 1;
VC_Activated[i] = TRUE;
```

Transmit a Data Cell for VC[i]

```
(For Each ILCT)
begin
If (Send_Grant[i] == TRUE)
 If (Crd-Bal[i] > 0)
   Send: Data_Cell for VC[i];
   Increment: Tx_Cnt[i] = Tx_Cnt[i] + 1;
   If (Valid_NAB[i] != Global_NAB)
     Set: Valid_NAB[i] = Global_NAB
     Update: Crd-Bal[i] = Crd-Bal[i]
       + New_Alloc[Valid_NAB[i]][i] - Cur_Alloc[i];
     Copy: Cur_Alloc[i] =
          New_Alloc[Valid_NAB[i]][i];
     Set: New_Alloc[(Global_NAB+1) mod 2][i] = 1;
   Decrement: Crd-Bal[i] = Crd-Bal[i] - 1;
   If (Crd-Bal[i] == 0)
     Set: OK_to_Send[i] = FALSE;
   Else Set: OK_to_send = TRUE;
end
```

Receive a Credit Cell

```
(For Each ILCT)
begin
For each record in the credit cell do:
If the record is for VC[i]
If (VC_Activated[i] == FALSE)
 Discard record;
Else
 If (Credit_Record)
   Receive: Credit_Record{i; Fwd_Cnt[i]};
   If (Valid_NAB[i] != Global_NAB)
     Set: Valid_NAB[i] = Global_NAB
     Copy: Cur_Alloc[i] =
          New_Alloc[Valid_NAB[i]][i];
     Set: New_Alloc[(Global_NAB+1) mod 2][i] = 1;
   Set: Crd-Bal[i] = Cur_Alloc[i]
        + Fwd_Cnt[i] - Tx_Cnt[i]
   If Crd-Bal[i] > 0
     Set: OK_to_Send[i] = TRUE;
   Else Set: OK_to_Send[i] = FALSE;
 If (Alloc_Record)
   Receive: Alloc_Record{i, X[i]};
   If (Valid_NAB[i] != Global_NAB)
     Set: Valid_NAB[i] = Global_NAB
     Update: Crd-Bal[i] = Crd-Bal[i]
       + New_Alloc[Valid_NAB[i]][i] - Cur_Alloc[i];
     If (Crd-Bal[i] > 0)
       Set: Ok_to_Send[i] = TRUE;
     Else Set: Ok_to_Send[i] = FALSE;
     Copy: Cur_Alloc[i] =
          New_Alloc[Valid_NAB[i]][i];
   Set: New_Alloc[i(Global_NAB+1) mod 2][i] =
X[i];
end
```

Increment Adaptive Timer

```
(For Each MCT)
begin
Increment: Use_New_Alloc_Timer = Use_New_Alloc_
Timer + 1;
If (Use_New_Alloc_Timer == AI)
 Global_NAB = (Global_NAB + 1) mod 2;
 Set: Use_New_Alloc_Timer = 0;
 Set: Use_New_Alloc_Timer_Expired = TRUE;
 Set: PP_Timer = 0;
end
```

Refresh Buffer Allocation

```
(For Each MCT)
begin
If (Use_New_Alloc_Timer_Expired == TRUE)
 If (PP_Timer == 0)
   If (Valid_NAB[q] != Global_NAB)
     Set: Valid_NAB[q] = Global_NAB
     Update: Crd-Bal[q] = Crd-Bal[q]
        + New_Alloc[Valid_NAB[q]][q]
        - Cur_Alloc[q];
   If Crd-Bal[q] ≤ 0
        Set: OK_to_Send = FALSE;
     Else Set: OK_to_send = TRUE;
     Copy: Cur_Alloc[q] =
        New_Alloc[Valid_NAB[q]][q];
     Set: New_Alloc[(Global_NAB+1) mod 2][q] = 1;
// For the next allocation if the Alloc_Record associated with X[q] is lost
// in transmission, this default allocation of 4 cells will be used.
     Increment: q = (q mod N') + 1;
 PP_Timer = (PP_Timer + 1) mod PP;
 If (q = 1)
   Set: Use_New_Alloc_Timer_Expired = FALSE;
   Set: PP_Timer = 0
end
```

6.8. Downstream node pseudocode

Activate VC[i]

```
Rx_Cnt[i] = 0;
Fwd_Cnt[0:1][i] = 0;
Prev_Fwd_Cnt = 0;
PPrev_Fwd_Cnt = 0;
N2[i] = 1;
N2_Cnt[i] = 0;
VC_Activated[i] = TRUE;
Valid_Bank[i] = 0;
```

Receive a Data Cell for VC[i]

```
(For Each ILCT)
begin
If (VC_ACTIVATED[i] == FALSE)
 Discard cell;
Else
 Receive data cell and put it in buffer
 Increment: TQ = TQ + 1;
 Increment: Rx_Cnt[i] = Rx_Cnt[i] + 1;
 Set: Data_to_Send[i] = TRUE;
end
```

Forward a Data Cell for VC[i]

```
(For Each OLCT)
begin
If (Send_Grant[i] == TRUE)
  Decrement: TQ = TQ - 1;
  Increment: Fwd_Cnt[Use_Bank][i] =
     Fwd_Cnt[VAlid_Bank[i]][i] + 1;
  Valid_Bank[i] = Use_Bank;
  Increment: N2_Cnt[i] = N2_Cnt[i] + 1;
  If (N2_Cnt[i] == N2)
    Enqueue:
      Credit_Record {i, Fwd_Cnt[Valid_Bank[i]][i]};
    Set: N2_Cnt[i] = 0;
  If (Rx_Cnt[i] - Fwd_Cnt[Valid_Bank[i]][i] == 0)
    Set Data_to_Send[i] = FALSE;
end
```

Increment Packing Timer

```
(For Each ILCT)
begin
If (Tx_Credit_Cells = FALSE)
  Increment: Credit_Batch_Send_Timer =
      Credit_Batch_Send_Timer + 1;
  If (Credit_Batch_Send_Timer ==
         Credit_Batch_Send_Period)
    Set: Tx_Credit_Cells = TRUE;
end
```

Send a Credit Cell

```
(For Each ILCT)
begin
If (Tx_Credit_Cells == TRUE)
  While (Packing_Cnt < Packing_limit)
    If (Credit_Send_Queue is not empty)
      Dequeue: Credit_Send_Queue;
      Insert: credit or alloc record
           in Draft_Cell_Payload;
    Increment: Packing_Cnt = Packing_Cnt + 1;
  Send: credit cell with Draft_Cell_Payload;
  Set: Draft_Cell_Payload is reset to empty;
  Set: Packing_Cnt = 0;
  If (Credit_Send_Queue is empty)
    Set: Tx_Credit_Cells = FALSE;
    Set: Credit_Batch_Send_Timer = 0;
end
```

Increment Adaptive Timer

```
(For Each MCT)
begin
Increment: AI_Timer = AI_Timer + 1;
  If (AI_Timer == AI)
    Use_Bank = (Use_Bank + 1) mod 2;
          // Switch to the other memory bank
    For (all VC[i]s)
      VU[i] = (Fwd_Cnt[(Use_Bank+1) mod 2][i] -
         PPrev_Fwd_Cnt[i])*VC_Weight[i];
      // Since Fwd_Cnt[(Use_Bank+1) mod 2] will not be rewritten
      // in this AI, this loop can be done incrementally over multiple
      // cell cycles, provide the loop is complete before the next AI.
      Increment: TU = TU + VU[i];
      Set: PPrev_Fwd_Cnt[i]=Prev_Fwd_Cnt[i];
      Set: Prev_Fwd_Cnt[i] =
           Fwd_Cnt[(Use_Bank+1) mod 2][i];
      If (Valid_Bank[i] != Use_Bank)
        Fwd_Cnt[Use_Bank][i] =
           Fwd_Cnt[Valid_Bank[i]][i];
        Valid_Bank[i] = Use_Bank;
    Set: TQ_Snapshot = TQ;
    Set: AI_Timer = 0;
    Set: AI_Expired = TRUE;
    Set: CP_Timer = 0;
end
```

Compute New Buffer Allocation

```
(For Each MCT)
begin
If (AI_Expired == TRUE)
  If (CP_Timer == 0)
    If (TU == 0)
      X[p] = MIN(MAX(⌊(M/2 - TQ_Snapshot - N)
           *(1/N)⌋,0) + 1, Alloc_Bound[p]);
    Else
      X[p] = MIN(MAX(⌊(M/2 - TQ_Snapshot - N)
           *(VU[p]/TU)⌋,0) + 1, Alloc_Bound[p]);
    N2[p] = ⌈X[p]/4⌉;
    If N2_Cnt[p] ≥ N2[p]
      Set: N2_Cnt[p] = 0;
      Enqueue:
        Credit_Record{p, Fwd_Cnt[Valid_Bank[p]][p]};
    If X[p] > 1
      Enqueue: Alloc_Record{p, X[p]};
    Increment: p = (p mod N) + 1;
  CP_Timer = (CP_Timer + 1) mod CP;
  If (p = 1)
    Set: TU = 0;
    Set: AI_Expired = FALSE;
    Set: CP_Timer = 0
end
```

7. Design issues for receiver-oriented adaptive schemes

There are some significant issues inherent in any receiver-oriented adaptive schemes. This section examines these issues and indicates how they are taken care of by the adaptive scheme of Section 6 above.

7.1. Links of different round-trip times

Suppose that the links (e.g., those connecting *Rcv* to *Snd1* and *Snd2* in Figure 4) have different round-trip times (RTT_1 and RTT_2). Then an adaptive scheme should adjust buffer allocations based on the relative bandwidth usage of a VC *weighted* by the VC's *RTT*. In the pseudocode above, the per VC parameter, `VC_Weight`, captures this weighting information.

Moreover, the size of the buffer pool at the receiver should be related to RTT_{max}, where RTT_{max} is the maximum RTT value for all the input links. As noted in the pseudocode above, the required

memory size is C*RTT_{max}*MCT/OLCT + D*N, with C ≥ 4 and D ≥ 2.

7.2. Links of different speeds

Suppose that the links (e.g., those connecting *Rcv* to *Snd1* and *Snd2* in Figure 4) have different speeds. Then the buffer allocation upper bound should reflect the differences. That is, VCs through lower bandwidth links will have smaller upper bounds. Note that in the pseudocode the value of `Alloc_Bound` indeed reflects link bandwidths.

The waiting time for the packing of credit or allocation records should also be adjusted accordingly. That is, in order to achieve the same degree of packing, the receiver should extend this period for low bandwidth links. In the pseudo code `Credit_Batch_Send_Period` is proportionally increased for low-speed VCs.

7.3. Allocation synchronization

All upstream nodes of a receiver should be synchronized, so that data from them will arrive at the receiver reflecting the allocation for the same allocation interval (`AI`). Otherwise, a VC with decreased new allocation may still forward an excessive amount of data according to previous allocation. As a result, cells could get lost at the receiver because of memory overflow.

In our pseudocode, we use a `Use_New_Alloc_Timer` for each of the upstream nodes to enforce synchronization. The timers are initialized to offset differences in link propagation delays. A positive side benefit of the use of these timers is that the receiver can send allocation records (`Alloc_Record`) for individual VCs incrementally without worrying about when senders will get them.

More precisely, if the upstream nodes have different propagation delays, then the initial values of `Use_New_Alloc_Timer` at these nodes are staggered to account for the differences. That is, the initial value of `Use_New_Alloc_Timer` associated with an upstream node is -[AI + (PD_{max} - PD)], where PD is the propagation delay, in number of MCT, for the input link connecting to the upstream node. This ensures that data arriving at the receiver from various upstream nodes will all reflect the buffer allocation results computed by the receiver for the *same* AI awhile ago.

This synchronization method is a little heavy-handed, mainly for simulation purposes. In practice, any method which guarantees synchronization within AI will be sufficient.

7.4. Buffer allocation bound

Since we require the buffer space in the receiver to be more than two RTT_{max} it is possible to allocate more buffer to a VC than it can possibly use. Preventing excessive allocation to a VC would improve the chance for other VCs to use the memory when their needs arise. As explained in Section 8.1, our pseudocode uses a tight upper bound (`Alloc_Bound`) on the maximum-possible allocation a VC would need in terms of link speeds, propagation delay and the current `N2` value.

7.5. Adaptive Credit Transmission Frequency

It is desirable to adapt the frequency of transmitting credit cells of a VC, i.e., the `N2` value, according to the VC's currently bandwidth usage. Those VCs with large bandwidth usage could use large `N2` values, and thus would reduce their bandwidth overhead of transmitting credit records upstream. On the other hand, an inactive VC could be given an `N2` value as small as one, to increase memory utilization. The `N2` value would increases only when VC's bandwidth ramps up. Thus the required memory for each VC could be as small as one cell.

The pseudocode implements the `N2`-adaptation idea described above. For a given buffer allocation of a VC, the `N2` value is simply chosen as a fraction (such as a quarter) of the allocation.

Allowing `N2` to be set automatically is also attractive from the ease-of-use viewpoint. That is, there is no need for the higher-level software to choose `N2` values for individual VCs.

7.6. Packing of credit or allocation records

It is possible to carry up to 6 credit or allocation records in the 48-byte payload of a credit cell. In order to achieve a high degree of packing while limiting credit record's waiting time, our pseudocode sends a batch of packed credit cells each time after `Credit_Batch_Send_Period` expires. That is, the scheduler will serve the queued credit or alloc records only when the period has expired. Once the period expires, the scheduler will dequeue all the records in the queue. As noted in Section 7.2, `Credit_Batch_Send_Period` has a large value for links of small bandwidth.

8. Analysis of the receiver-oriented scheme

The section analyzes some important properties of the receiver-oriented adaptive scheme presented in Section 6.

8.1. Upper bound on buffer required by a VC

We prove in this section that Alloc_Bound given in Section 6.3 is an upper bound on the buffer required by a VC.

Let *Credit Round Trip* (CRT) be the length of the time interval starting when an upstream node transmits a data cell and finishing when it receives the credit cell triggered by the forwarding of this data cell (and possibly some subsequent data cells) at the receiver, assuming that there is no congestion. Let X be the new buffer allocation for the VC. Then

CRT = RTT*MCT + MAX(N_{up}*N2*ILCT, N_{dw}*N2*OLCT),

where N_{up} is the current number of VCs competing over the input link, N_{dw} is that over the output link, and `N2` is $\lceil X/4 \rceil$ in this pseudocode. Note that N2 ≤ X/4 + 1.

At any given time, one of the following two cases must hold. The first case is when

N_Dw*N2*OLCT ≥ N_{up}*N2*ILCT.

Then for the VC to achieve $1/N_{dw}$ of the output link bandwidth, X need only be as large as that required for the following equation:

X*OLCT/CRT = $1/N_{dw}$, or

$X*OLCT/[RTT*MCT + N_{dw}*(X/4 +1) *OLCT] = 1/N_{dw}$.
By solving X from the above equation, we see that

$$X = (4/3)*RTT*MCT/(N_{dw}*OLCT) + 4/3 \quad (3)$$

This X value can already allow the VC to achieve its fair share, i.e., $1/N_{dw}$ of the output link bandwidth, and thus any larger value for X would not be needed. Therefore Equation (3) gives an upper bound on X for the first case.

Similarly, we consider the second case when
$N_{dw}*N2*OLCT < N_{up}*N2*ILCT$. Then for the VC to achieve $1/N_{up}$ of the input link bandwidth, X need only be as larger as that required for the following equation:
$X*ILCT / CRT = 1/N_{up}$,
or
$X*ILCT / [RTT*MCT + N_{up}*(X/4 + 1)*ILCT] = 1/N_{up}$.
By solving X from the above equation, we see that

$$X = (4/3)*RTT*MCT/[N_{up}*ILCT] + 4/3 \quad (4)$$

This X value can already allow the VC to achieve its fair share, i.e., $1/N_{up}$ of the input link bandwidth, and thus any larger value for X would not be needed. Therefore Equation (4) gives an upper bound on X for the second case.

Now note that the following single formula captures the two upper bound results (3) and (4):

$$X=(4/3)*RTT*MCT/MAX[N_{dw}*OLCT, N_{up}*ILCT]+4/3 \quad (5)$$

That is, if $N_{dw}*N2*OLCT \geq N_{up}*N2*ILCT$,
(or $N_{dw}*OLCT \geq N_{up}*ILCT$)
then the above formula gives (3); otherwise it gives (4).

The upper bound (5) implies that for any $N_{dw} \geq 1$ and $N_{up} \geq 1$, X need not be larger than
(4/3)*RTT*MCT / MAX[OLCT, ILCT] + 4/3
Thus we let Alloc_Bound
$= \lceil (4/3)*RTT*MCT / MAX[OLCT, ILCT] +4/3 \rceil$.

8.2. Guaranteed Ramp Up

This section gives a proof that with M given in Section 6.3,

$$M \geq (4*RTTmax + N)*MCT/OLCT \quad (6)$$

a VC can always ramp up its buffer allocation to achieve its fair share of bandwidth. Analysis here is similar to that for Alloc_Bound in Section 8.1.

Suppose that the current buffer allocation X for a VC is insufficient for the VC to achieve its fair share of bandwidth. We show that buffer allocation for the VC will ramp up, that is, the VC's new buffer allocation X' will be larger than X, provided that

$$TQ < (2/3)*RTTmax*MCT/OLCT \quad (7)$$

where TQ is the current total queue size at the receiver.

Consider the first case when
$N_{dw}*N2*OLCT \geq N_{up}*N2*ILCT$.
Because X is not large enough for the VC to achieve $1/N_{dw}$ of the output link bandwidth, $X*OLCT/CRT_{abs} < 1/N_{dw}$.
This implies that
$X*OLCT / [RTT*MCT + N_{dw}*X / 4 *OLCT] < 1/N_{dw}$, or

$$N_{dw}*X/4 < (1/3)*RTT*MCT/OLCT \quad (8)$$

In this analysis, we make the simplifying assumption that $\lceil X/4 \rceil$ is X/4.

Note that in this code the new allocation X' is obtained by dividing the pie M/2 - N - TQ according to VC's bandwidth usage, weighted by its RTT, over the output link. Thus,
$X' \geq (M/2 - N - TQ)*X*OLCT*(RTT/RTTmax)$
$/ [RTT*MCT + N_{dw}*(X/4)*OLCT]$.
By (6), (7) and (8), we see $X' > X$.

Similarly, consider the second case when
$N_{dw}*N2*OLCT < N_{up}*N2*ILCT$.
Because X is not large enough for the VC to achieve 1/N_Up of the input link bandwidth, $X*ILCT/CRT_{abs} < 1/N_{up}$.
This implies that
$X*ILCT / [RTT*MCT + N_{up}*X/4 *ILCT] < 1/N_{up}$, or

$$N_{up}*X/4 < (1/3)*RTT*MCT/ILCT \quad (9)$$

Note that in this code the new allocation X' is obtained by dividing the pie M/2 - N - TQ according to VC's bandwidth usage, weighted by its RTT, over the output link. Thus,
$X' \geq (M/2 - N - TQ)*X*OLCT* (RTT/RTTmax)$
$/ [RTT*MCT + N_{up}*(X/4)*ILCT]$.
By (6), (7) and (9), we see $X' > X$.

Even when (7) does not hold, for either of the above two cases $X' > X$ can still be true as the inequality (8) or (9) is usually not tight.

8.3. Weighted allocation to account for different link delays

In computing new allocation, we apply linear weighting to measured bandwidth usage to account for different link RTT values. The weighting is necessary since the achievable bandwidth, as described in Section 8.1, is inversely proportional to CRT which depends on RTT and N2. Since in our algorithm the N2 value is also proportional to the total buffer allocation (i.e., N2 = $\lceil X/4 \rceil$), the linear weighting used in our code is exactly what we need.

9. Simulation results

We have developed a cell-level simulator based on the pseudocode of Section 6. Without loss of generality, our simulation uses MCT as time unit and cell as data unit. The bandwidth unit is the bandwidth of the fastest link. Thus the maximum bandwidth is 1.

The purpose of the simulation is to evaluate the performance of our receiver-oriented adaptive buffer allocation scheme as described in Section 6. We want to see if the algorithm can demonstrate fair buffer allocation, fair bandwidth sharing, limited queue length, high link utilization, and tolerance of significant bandwidth and length disparity among input links.

We use three network configurations for the simulation. Our tests let multiple VCs start at the same time or at different times.

We also consider stressful situations such as sudden bandwidth decrease and increase for a link. We observe how fair the VCs share the bandwidth, how fast they ramp up, buffer usage, etc.

9.1. Three network configurations and basic simulation results

The simulation results reported here use three network configurations. The first two are known at the ATM Forum as "Generic Fairness Configurations" [10] (GFC1 and GFC2) and the third is a single switch configuration we call GFC3.

The GFC1 topology and bandwidth setup are shown in Figure 7. The link propagation delay between any two connecting switches is 1,800 cells (corresponding to a 1,000km, 155.5 Mbps link) and that between a switch and a host is 1 cell. Table 1 shows the expected bandwidth for each group of VCs in the steady state, that can be derived by simple analysis.

PD: propagation delay in # MCTs
PD = 1 for link between host and switch
[ ]: Link bandwidth
Link bandwidth = 1 if not indicated
(N): Number of VCs in the VC group

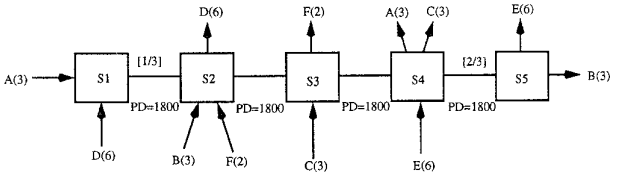

Figure 7: Generic Fairness Configuration 1 (GFC1)

Figures 8, 9 and 10 are simulation results of our adaptive scheme for GFC1. The bandwidth of several VCs and the total buffer usage of several switches are shown in Figure 8 and Figure 9, respectively. The number of received cells on a VC at the destination is given in Figure 10. From this figure we have a clear view of the long-term bandwidth achieved by the individual VCs. We see that bandwidth results from Figure 8 and Figure 10 match perfectly the expected bandwidth depicted in Table 1.

Group	Bandwidth	Bottleneck Link
A	1/27 = 0.037	S1-S2
B	2/27 = 0.074	S4-S5
C	2/9 = 0.222	S3-S4
D	1/27 = 0.037	S1-S2
E	2/27 = 0.074	S4-S5
F	1/3 = 0.333	S2-S3

Table 1: Expected bandwidth for GFC1

GFC2, depicted in Figure 11, has a network topology similar to GFC1, but involving more switches and more varieties in link propagation delay. The expected bandwidth of VCs in the steady state for GFC2 is shown in Table 2.

Figure 12 shows the simulation results of our adaptive scheme. We note that the VCs indeed achieve the expected bandwidth of Table 2.

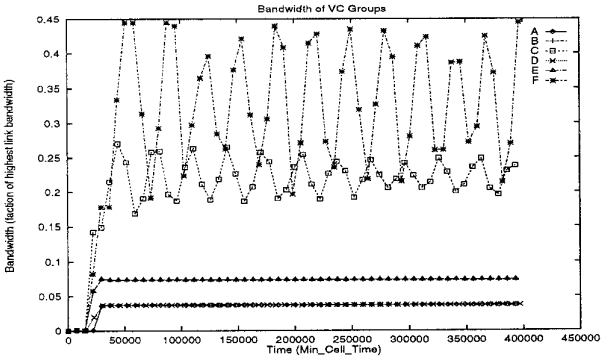

Figure 8: Bandwidth of VCs for GFC1

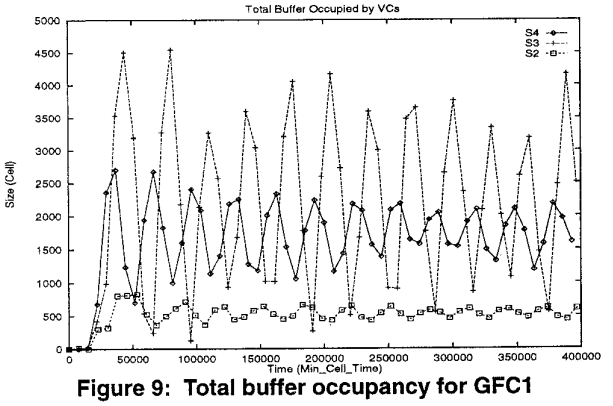

Figure 9: Total buffer occupancy for GFC1

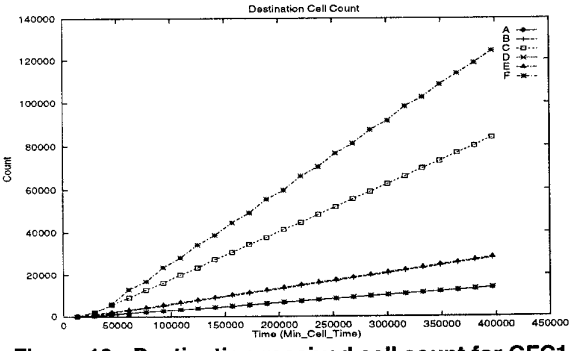

Figure 10: Destination received cell count for GFC1

PD: propagation delay D = 500 MCTs
PD = 1 MCT for link between host and switch
[ ]: Link bandwidth
Link bandwidth = 1 if not indicated
(N): Number of VCs in the VC group

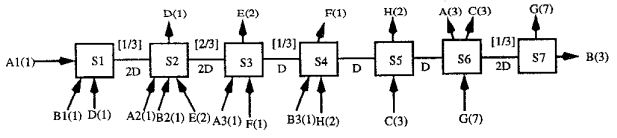

Figure 11: Generic Fairness Configuration 2 (GFC2)

Group	Bandwidth	Bottleneck Link
A	2/30=0.066	S3-S4
B	1/30=0.033	S6-S7
C	7/30=0.233	S5-S6
D	7/30=0.233	S1-S2
E	7/30=0.233	S2-S3
F	2/30=0.066	S3-S4
G	1/30=0.033	S6-S7
H	10.5/30=0.35	S4-S5

Table 2: Expected bandwidth for GFC2

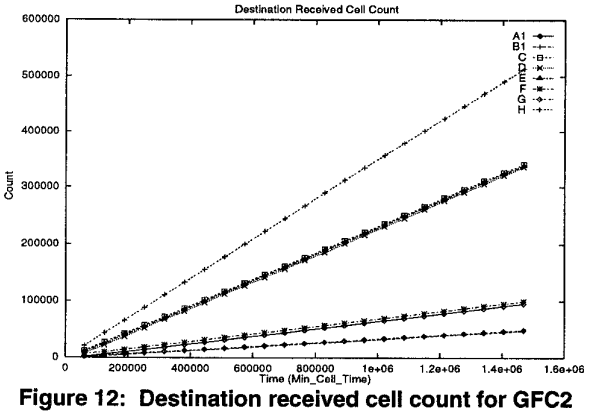

Figure 12: Destination received cell count for GFC2

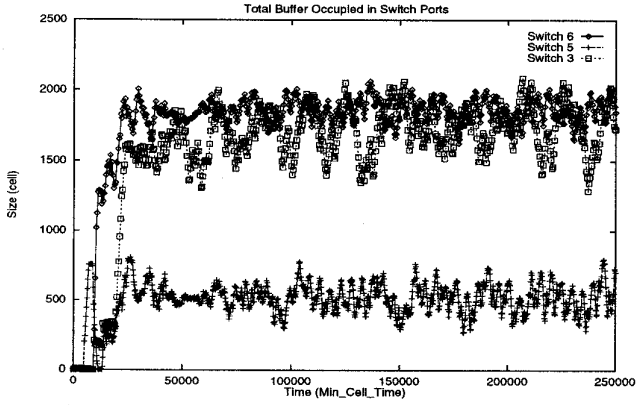

Figure 13: Total switch buffer occupancy for GFC2

GFC3 is composed of one switch and 11 hosts. The bandwidth of the input links varies from 1 to 1/100 and link propagation delay varies from 1 to 2,000 MCTs. As shown in Figure 14, there are 50 VCs coming in from each input port for a total of 500 VCs sharing the output port. In the steady state, VCs in group D and G can only reach 1/5,000=0.0002 of the output link bandwidth while rest of the VCs can reach 98/40,000 = 0.00245 of that.

The bandwidth sharing result, the destination received cell count, and the total queue length for our adaptive scheme are shown in Figure 15, 16 and 17, respectively. We see that the achieved bandwidth agrees with the expected bandwidth described in the preceding paragraph.

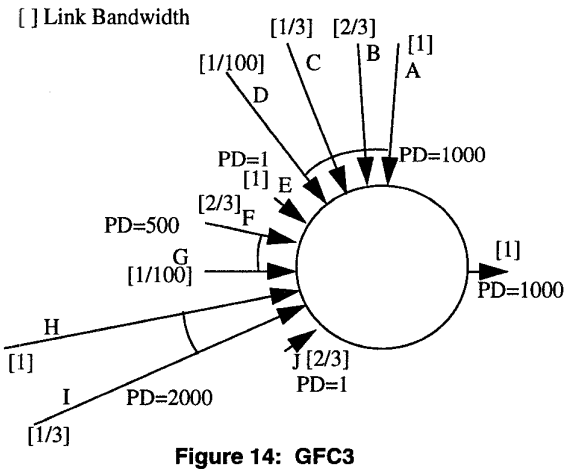

Figure 14: GFC3

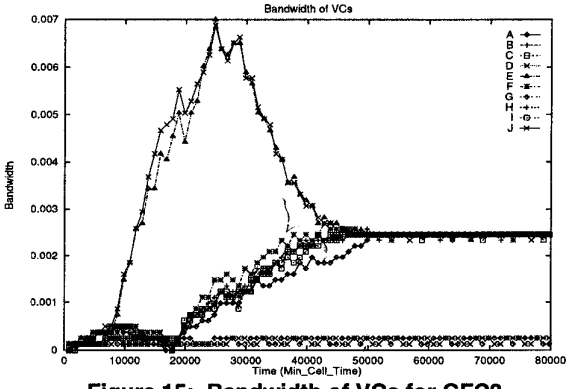

Figure 15: Bandwidth of VCs for GFC3

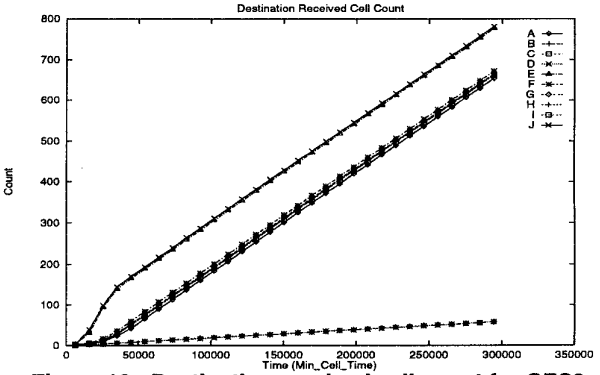

Figure 16: Destination received cell count for GFC3

9.2. Ramping up test

This section shows simulation results with VCs starting at different times. We want to see how fast a newly started VC can ramp up. In the GFC2 case, we let A3, B3, and C1 starting at the

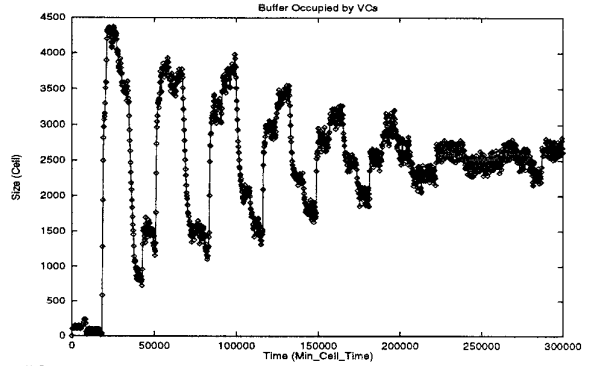

Figure 17: Total switch buffer occupancy for GFC3

300,000th, 500,000th and 700,00th MCT, respectively. The bandwidth of these three VCs observed on the link from S5 to S6 is shown in Figure 18. The buffer allocation of these three VCs is shown in Figure 19. Figure 20 shows how the bandwidth (i.e., slope) changes for some VCs over a long period of time. The simulation results in all these figures match with those predicted in Table 2.

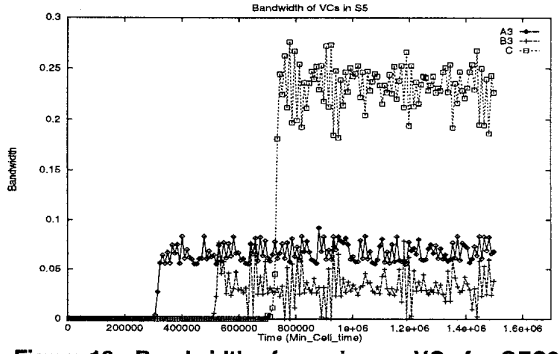

Figure 18: Bandwidth of ramping up VCs for GFC2

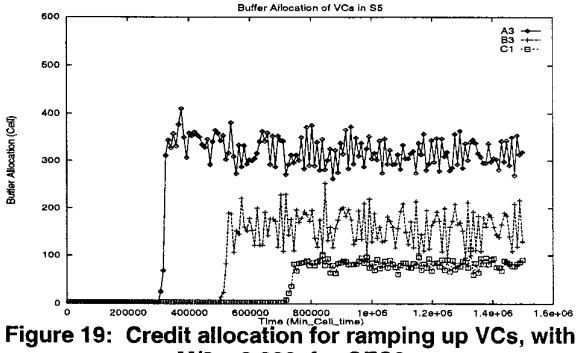

Figure 19: Credit allocation for ramping up VCs, with M/2 = 2,000, for GFC2

In Figure 18 we find that it takes less than 50,000 MCTs for VC C to ramp up. The same configuration has also been tested with several different sizes of M. The simulation result on the bandwidth of VC A3 seen in switch S5 is shown in Figure 21. We can see how the ramp up speed improves as the switch memory increases its size.

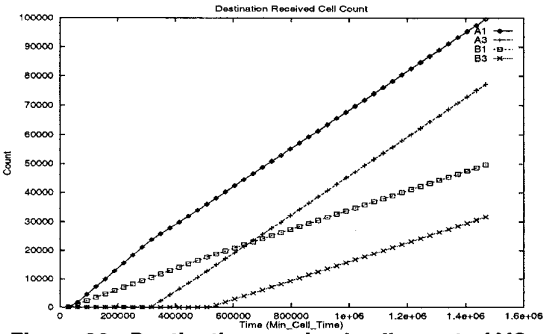

Figure 20: Destination received cell count of VCs with various starting times for GFC2

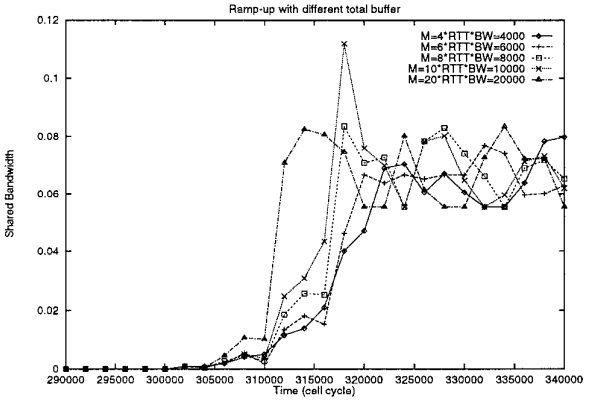

Figure 21: The ramp-up of VC A3 for GFC2 under various switch buffer sizes

9.3. Sudden bandwidth decrease in GFC3

In order to observe the behavior of the adaptive algorithm during severe congestion, we create a congestion situation by reducing a bottleneck link's bandwidth from 1 to 1/100 suddenly. We assume the GFC3 configuration of Figure 14, with 10 VCs (one from each port) going through the switch.

For the output link we start its bandwidth with 1, reduce it at the 500,000th MCT to 1/100, and then increase it at the 1,100,000th MCT to 1 again.

After the bandwidth reduction, those small bandwidth VCs (coming from bandwidth 1/100 link) should get fair share bandwidth, i.e., 1/1000. Figure 22 shows this fair share of bandwidth between the VCs during the bandwidth reduction period.

Moreover, Figure 22 shows that even during the bandwidth reduction period, newly started VCs can still ramp up. In particular, we see that the three VCs (C,E and H) which start at the 700,000th, 800,000th and 900,000th MCT can all ramp up. Figure 25 shows that the three VC achieve the same bandwidth (as their slopes are the same). Thus, during the bandwidth reduction, not only new VCs can ramp up but they will get fair share in bandwidth.

After the bandwidth reduction, the total queue length will go up. In the steady state the queue length will stay higher than before since the high input/output bandwidth ratio will cause high queue level. This is shown by Figure 23.

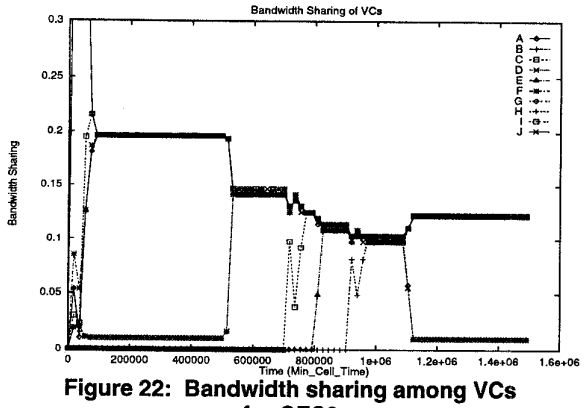

Figure 22: Bandwidth sharing among VCs for GFC3

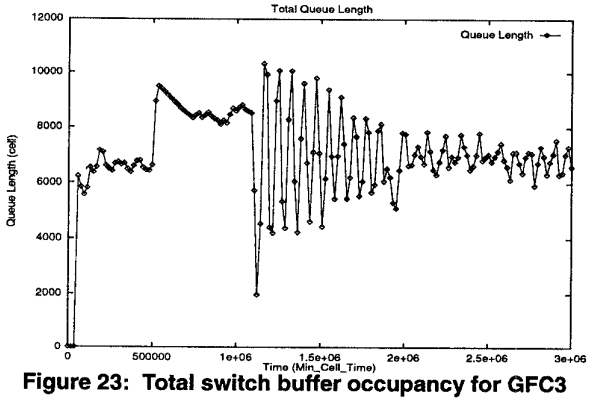

Figure 23: Total switch buffer occupancy for GFC3

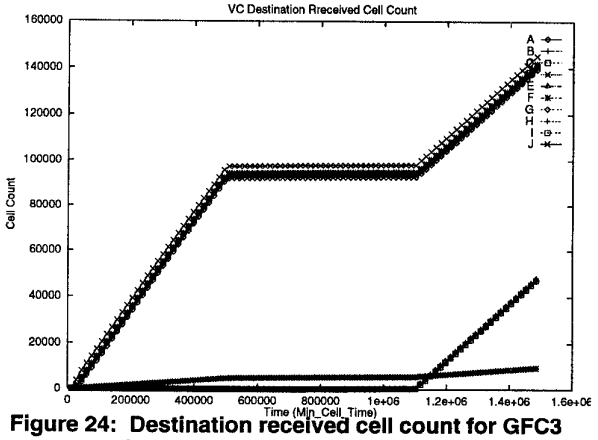

Figure 24: Destination received cell count for GFC3 with sudden bandwidth reduction

9.4. Credit cell traffic overhead

From these simulation results, we note that credit cells traffic is tremendously reduced due to adaptive N2 and credit record packing. Consider, for example, the link between switches S4 and S5 in GFC2. Figure 26 gives the total number of credit cells sent. We see that there are about 1.5 million data cells, about 18,600

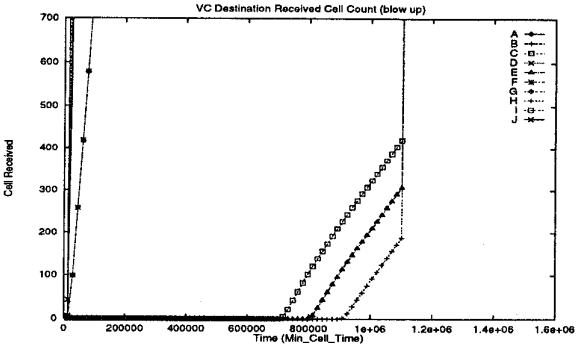

Figure 25: Destination received cell count for GFC3 with three VCs ramping up during bandwidth reduction (blow up of Figure 24)

credit and allocation records, and about 14,000 credit cells. The number of credit cells is less than 1% of that of data cells.

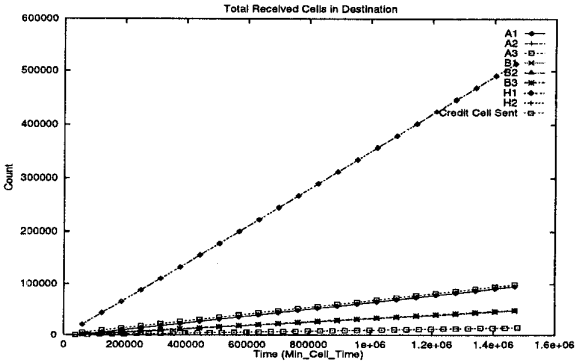

Figure 26: # data and credit cells over the link between S4 and S5 for GFC2. (Data cell number is per-VC and credit cell number is for all VCs.)

10. Conclusions

When a buffer pool in a node is shared by VCs from multiple input links, receiver-oriented rather than sender-oriented adaptive buffer allocation must be used. A receiver-oriented scheme is more complex than the sender-oriented counterpart because the multiple input links may have different RTTs and link capacities, and because allocation at the receiver must be communicated to each upstream node. In this paper we have described a receiver-oriented adaptive scheme which can effectively deal with these issues.

We have shown, by analysis, that our receiver-oriented adaptive scheme will not lose cells during allocations, and will allow newly started VCs to ramp up to get a fair share of bandwidth. We have also demonstrated, by simulation, that the scheme achieves excellent performance in utilization and fairness, while requiring only modest memory and bandwidth overhead.

The small size of the required memory is especially encouraging. All of our simulation results reported in this paper are based on the code of Section 6, which assumes only a memory of 4*RTTmax*MCT/OLCT + 2*N cells, where N is the number of the active VCs. Thus the receiver-oriented scheme is attractive for

links that have large propagation delays and need to support a large number of VCs.

Acknowledgments

We thank the INFOCOM '95 reviewers for their helpful comments on an early version of this paper.

References

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

[2] Blackwell, et al. "An Experimental Flow Controlled Multicast ATM Switch," *Proceedings of First Annual Conference on Telecommunications R&D in Massachusetts*, Vol. 6, pp. 33-38, October 25, 1994.

[3] 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.

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

[5] 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.

[6] H. T. Kung, T. Blackwell and A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing," *Proc. ACM SIGCOMM '94 Symposium on Communications Architectures, Protocols and Applications*, pp. 101-114.

[7] 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*, pp. 116-127. (Postscript files of this and other related papers by the authors and their colleagues are available via anonymous FTP from virtual.harvard.edu:/pub/htk/atm.)

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

[9] C. Ozveren, R. Simcoe, and G. Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," *Proc. ACM SIGCOMM '94 Symposium on Communications Architectures, Protocols and Applications*, pp. 89-100.

[10] R. J. Simcoe, "Configurations for Fairness and Other Test," ATM_Forum/ 94-0557, 1994.

[11] 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.

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