FLOWmaster Flow-Control

Bob Simcoe

Jim Scott

Doug Washabaugh

Version 0.22

05 January, 1995

1. Overview

For data applications, the ATM Forum has defined a class of service named Available Bit Rate (ABR). ABR service is envisioned to provide LAN-like performance, which has the following properties:

- All of the unused link bandwidth is available to all of the active ABR VCs,
- The link bandwidth is shared fairly between those VCs,
- Has negligible cell loss, which can help avoid thruput collapse.

Digital's solution for flow-control of ABR traffic is FLOWmaster. FLOWmaster is per-hop, per VC, credit based flow control. FLOWmaster delivers high performance and insures that no cells are lost because of congestion. It also enables flow-controlled VCs to utilize full link bandwidth when the link becomes idle.

2. Theory of Operation

The fundamental concept of credit-based flow-control is that the transmitter does not send a cell unless it knows that the receiver has a buffer to hold the cell. The transmitter maintains a "credit balance" on a per-VC basis. This credit balance is the number of cells that the transmitter expects the receiver can buffer. The transmitter can safely send cells on the VC as long as the VC's credit balance is greater than zero.

When the transmitter sends a cell on a VC, it decrements the VC's credit balance. When the receiver frees a buffer for a VC (by forwarding a cell from the VC), it sends a credit to the transmitter. The transmitter then increments the VC's credit balance.

In Figure 1, a virtual circuit is created from A to C, through B. Each direction of each channel (A →B, B→C, C→B and B→A) is flow-controlled independently. For example, assume that each VC has 3 cells of buffering (credits). Entity A can then send 3 cells to B, but then pause transmissions on the VC until it receives credits from B. As B forwards cells to C and frees the associated buffers, it sends credits to A, thus allowing A to send more cells to B.

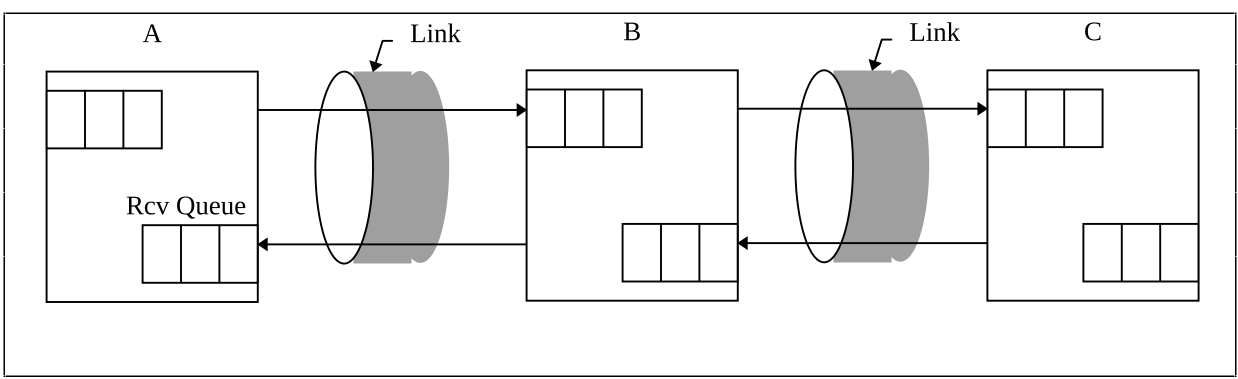

Figure 1 Multiple Hop FLOWmaster

Flow control operates independently on each direction of each link. In Figure 1, the "A" transmitter on the A→B channel sends cells to B only if it has a non-zero credit balance. Whether or not the A's transmitter on A→B channel is allowed to transmit is not affected by the state of B's transmitter on the

B→A channel. Note that the credit balance counters are on a per-VC basis so that other VCs on the link with a non-zero balance can transmit if a particular VC is blocked because its credit balance is zero.

2.1 Device Model

Figure 2 shows a model of a device that implements FLOWmaster flow-control. This model helps in understanding credit resynchronization.

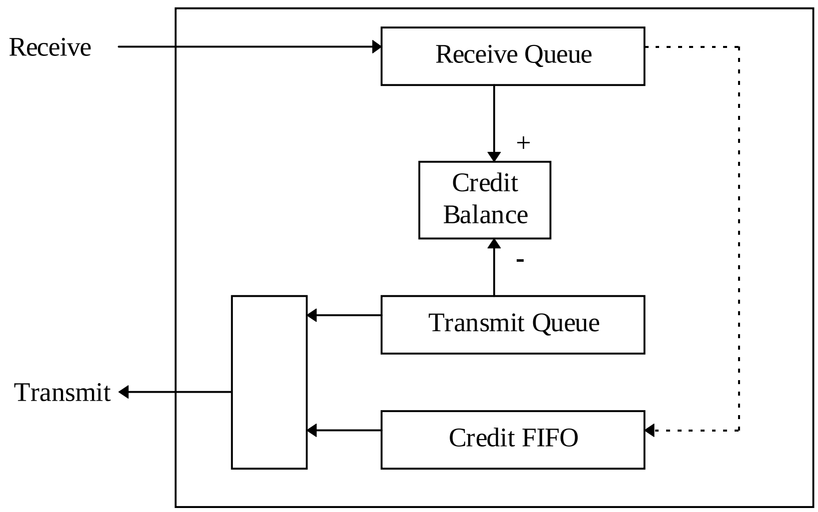

Figure 2 FLOWmaster Device Model

The *Credit Balance* is a data structure that contains the number of credits for each VC. The credit balance for a VC is initialized to a non-zero value. The receive logic increments the VC's credit balance when it receives a credit for the VC, and the transmit logic decrements the VC's credit balance when it sends a cell on the VC.

The *Credit FIFO* contains credits to be returned to the "upstream node". Each of these entries contains a single credit for some VC.

The *Receive Queue* contains cells that have been received on the open VCs. When these cells are freed (forwarded), then a credit for the VC is placed on the Credit FIFO queue.

The *Transmit Queue* contains cells that are waiting to be transmitted. The device transmits a cell by removing an entry from the head of the Transmit Queue and Credit FIFO to form a cell header and payload.

Each device that supports FLOWmaster must have the following capabilities:

- The device's software must ensure that it does not transmit cells on a VC that is being resynchronized. It is, however, permitted to queue the VC's cells.
- The device's software must be able to guarantee that all of a VC's credits on the Credit FIFO are transmitted before the credit resynchronization completes.
- The device's software must be able to detect when all cells for a VC have been removed from the Receive Queue.

2.2 Number of Credits Required for a VC

For a transmitter to be able to send continuously at full link speed, the receiver must have a sufficient amount of guaranteed buffering (credits). This amount of buffering (in cells) is given by the following equation:

```
(VC_Max_Speed * maximum  time to return a credit)
```

The maximum time to return a credit is the sum of the following (all expressed in cell times):

- The time to transmit a cell,
- The length of the link connecting the transmitter and receiver,
- The time required to transfer the cell into the receiver's memory,
- The time required to free the buffer,
- The time required to generate a credit and place it on the link,
- The length of the link connecting the transmitter and receiver (return trip).
- The time for the transmitter to process the received credit.

For LAN connections running 100 meters at full OC-3 link rates, 4 cells of buffering are required to keep the link performing at full speed. Additional buffering is required if framing (or other chips containing FIFOs) add latency. Current designs with SUNI SONET framers require 10 cells of buffering per VC.

3. Credit Resynchronization

Generally, cell header errors that link failures cause are detectable. However, some link errors may cause undetected cell header errors. These undetected errors can lead to VCs either gaining or losing credits. If a VC loses credits, its performance may suffer because it might not have enough credits to sustain full link bandwidth. If a VC gains credits then it might overrun the receiver and cause cell loss which can lead to excessive retransmission of data. Therefore, FLOWmaster devices must periodically run a credit resynchronization algorithm on all active VCs. Refer to Section 8 for specifics as to when to run the credit resynchronization algorithm.

Like credit based flow control, credit resynchronization is similar to in that it is done on a per VC, per hop basis.

To perform credit resynchronization, the transmitter must:

1. Stop sending cells on the VC to be resynchronized.
2. Wait for all of the buffered cells for that VC to be transmitted, OR insert the credit resynchronization request before any buffered cells for that VC.
3. Send a credit resynchronization request to the downstream device for that VC.

When the downstream device receives a credit resynchronization request, it must wait until:

- The receive buffers do not contain any cells for that VC, and
- All credits for that VC have been drained from the credit FIFO.

The downstream device must then send a message to the upstream device to indicate that credit resynchronization is complete. The upstream node resets the credit count for the VC and re-enables transmission on that VC.

Refer to Appendix A for a description of the format of the cells that carry FLOWmaster credits.

4. Protocol Parameters

The operation of the protocol is affected by the following parameters:

Parameter	Description
T	The amount of time that a device can process a synchronization request before it must send a response. The default value is 500 milliseconds.
E	The worst case maximum round trip delay. The default is 50 milliseconds.
R	The minimum number of outstanding FLOWmaster requests supported by a downstream FLOWmaster device. The default is 16.

5. FLOWmaster Control Messages

FLOWmaster control messages are sent in a single cell AAL5 packet, on the VCI specified by the MIB object *adpResyncVCI*. This VCI is **not** FLOWmaster flow-controlled. Note: All values are big-endian on the wire. This example shows the control message in a UNI cell, but control messages can also be carried in an NNI cell.

<table>
<tr><td>GFC</td><td colspan="2">Credit[11:8]</td></tr>
<tr><td colspan="3">Credit[7:0]</td></tr>
<tr><td colspan="3">VCI</td></tr>
<tr><td>VCI</td><td>PTI</td><td>CLP</td></tr>
<tr><td colspan="3">HEC</td></tr>
<tr><td colspan="3">Command</td></tr>
<tr><td colspan="3">VCI_Range_Start</td></tr>
<tr><td colspan="3">VCI_Range_End</td></tr>
<tr><td colspan="3">Sequence Number</td></tr>
<tr><td colspan="3">Pad</td></tr>
<tr><td colspan="3">Control</td></tr>
<tr><td colspan="3">Length</td></tr>
<tr><td colspan="3">CRC 32</td></tr>
</table>

Figure 3 Format of a FLOWmaster Control Message

Table 1 Description of FLOWmaster Control Message Fields

Field	Size	Description
GFC	4 Bits	Refer to ATM UNI Specification.
Credit<11:8>	4 Bits	Bits 11:8 of the VCI to which a credit is given.
Credit<7:0>	1 Byte	Bits 7:0 of the VCI to which a credit is given.
VCI<11:4>	1 Byte	Refer to the ITU draft specification.
VCI<3:0>	4 Bits	Refer to ATM UNI Specification.
PTI	3 Bits	Refer to ATM UNI Specification.
CLP	1 Bit	Refer to ATM UNI Specification.
HEC	8 Bits	Refer to ATM UNI Specification.
Command	4 Bytes	The commands are described in subsequent sections: SYNC_REQ = 0x0F0000FF SYNC_BOTH_REQ = 0x0F0022FF SYNC_ACK = 0x0F0011FF SYNC_NACK = 0x0F0033FF
VCI_Range_Start	4 Bytes	Refer to the description of the command for which this field is used.
VCI_Range_End	4 Bytes	Refer to the description of the command for which this field is used.

Field	Size	Description
Sequence Number	4 Bytes	Sequence number of the message. This number must not be repeated in the time (T + E).
Pad	24 Bytes	Refer to ATM UNI Specification.
Control	2 Bytes	Refer to ATM UNI Specification.
Length	2 Bytes	Refer to ATM UNI Specification.
CRC 32	4 Bytes	Refer to ATM UNI Specification.

5.1 SYNC_REQ Command

The upstream node uses this command to resynchronize credits for circuits on a VC.   The *VCI_Range_Start* and *VCI_Range_End* fields specify the inclusive range of VCIs to resynchronize. A device can send a SYNC_REQ message even if a VC to resynchronize may be in the middle of a packet [this is convenient for switches which have knowledge of cells, not packets].   The downstream device must be able to handle this without dropping cells.

Once the upstream node sends a “SYNC_REQ” message for a single VC, it cannot send any more cells on the VCs that it is synchronizing until the synchronization process is complete. However, if the upstream nodes sends a “SYNC_REQ” message for a range of VCs, it can optionally continue to send cells on all VCs as long as it marks the VCs on which cells are sent. When the range resynchronization completes, it must perform an individual credit resynchronization on any marked VC.

For the upstream node, the resynchronization process is complete when:

1. (T + E) seconds have passed, where T is the timeout period and E is the maximum round trip delay on the line, OR
2. A “SYNC_ACK” message is received, and its sequence number matches the sequence number of “SYNC_REQ” message, OR
3. A “SYNC_NACK” message is received, and its sequence number matches the sequence number of “SYNC_REQ” message.

When a downstream node receives a “SYNC_REQ” message, it attempts to resynchronize each VCI in the range.

A VCI is resynchronized if any of the following is true:

- The VCI does not have any credits queued it , OR
- The VCI carries CBR traffic (note that CBR traffic is not FLOWmaster flow-controlled, so CBR VCs are considered “resynchronized” by default), OR
- A cell is observed at some point to arrive on the VC. This rule enables downstream node implementation to avoid special treatment for CBR VCs.

For a range of VCIs, if the downstream node can synchronize all of them immediately (one pass), then it sends a SYNC_ACK message to the upstream node. Otherwise, it sends a SYNC_NACK message to the upstream node, with the *VCI_Range_Start* field set to the first VCI that could not be resynchronized. If the downstream node does not support resynchronization of a range of VCIs, then the *VCI_Range_Start* field of SYNC_NACK message must contain the value from the *VCI_Range_Start* field of SYNC_REQ message.

For a single VCI, if the downstream node can resynchronize the VCI in the time-out period, then it sends a SYNC_ACK message to the upstream node. Otherwise, it must send a SYNC_NACK message to the upstream node.

It is possible that the downstream node may not be able to synchronize all of the VCs within the time-out period. If this occurs, the downstream node must send a SYNC_NACK message to the upstream node.

Both the *Seq_Num* field and the *VCI_Range_Start* field are the same as in the request message. The *VCI_Range_End* field contains the first VCI that could not be resynchonized.

The downstream node is expected to be able to handle ***R*** outstanding SYNC_REQ messages at one time. However, concurrent requests may not refer to overlapping ranges of VCI values. Any received request with a VCI range that overlaps with an outstanding request may be discarded.

5.2 SYNC_BOTH Command

The semantics of this message is exactly the same as the SYNC_REQ command, except that it also carries a hint that credit resynchronization in the reverse direction is suggested.

When the downstream node receives a SYNC_BOTH message, it should behave as though it received a SYNC_REQ message. Additionally, if it has any outgoing flow-controlled VCIs in the VCI range, it *may* send a SYNC_REQ message to the upstream node. The range of VCIs should at least cover those in the SYNC_BOTH message.

5.3 SYNC_ACK Command

The downstream node uses this message to indicate that it has successfully drained the data and credits for the range of VCIs in the associated SYNC_REQ or SYNC_BOTH command.

When the upstream node receives a SYNC_ACK message that matches the *Seq_Num* of an outstanding synchronization request, it performs two actions for each VCI in the range indicated by the request:

1. Resets the number of credits.
2. Reenables transmission of data cells.

The *VCI_Range_Start* and *VCI_Range_End* fields are ignored by the upstream node.

5.4 Sync_NACK Command

The downstream node uses this message to indicate that it could not perform the requested resynchronization. The field *VCI_Start_Range* carries the VCI of the first VC that could not be resynchronized. There are several reasons why one or more VCs could not be resynchronized:

1. The number of VCIs in the request is larger than 1, and the downstream node only supports the resynchronization of 1 VCI at a time.
2. The limit of outstanding requests at the downstream node was exceeded.
3. One or more of the VCIs in the range could not be resynchronized.

When the upstream node receives a SYNC_NACK message, the actions that it takes depends on whether it was reynchronizing a single VC or a range of VCs. If it was resynchronizing a single VC, it should continuously retry the resynchronization at convenient intervals. However, if it was resynchronizing a range of VCs, it is recommended that it retry a new range, starting with the VCI after the VCI that failed to resynchronize. Eventually, it must retry an individual resynchronization on all failed VCIs.

6. Escape Mechanism

A special “escape” mechanism provides a method by which a FLOWmaster device can quickly determine the FLOWmaster characteristics of the device at the other end of the link. The mechanism enables all FLOWmaster characteristics to be read with a single request and response packet. [If SNMP were used, many requests and responses would be required].

6.1 Determining if the Neighbor Supports the Escape Mechansim

A device can determine if the link neighbor supports the escape mechanism by using SNMP to read the DEC MIB object. Alternatively if the device does not implement SNMP, it can send the binary requests described in section 10.

6.2 Escape Mechanism Messages

The escape mechanism uses two message types: request and response. Both message types use the same format, as shown in Table 2. All escape messages are sent on VCI 16. Escape messages cannot be sent to a device until the device is known to support the escape mechanism.

An endpoint sends a response message only when it receives a request message. The response message must be sent promptly.

When an endpoint detects that its link is up, it may begin sending escape request messages. Once a response is received, the endpoint should stop sending the requests. If the endpoint does not receive a response to a request message, it may retry the request at 0.2 second intervals If a response is not received after 2 seconds, then it can continue to retry the request at 10 second intervals. If a response is not received after 1 minute, the endpoint should stop sending escape message requests.

Table 2 Format of Escape Message

Name	**Type**	**Key**	**Description**
escape	UINT8	M	Contains the value 0x55 to indicate that “this is not an SNMP packet.”
version	UINT8	M	Version number of the protocol. Currently only the value ‘1’ is supported.
pkt_type	UINT16	M	The type of packet, ‘1’ = request, ‘2’ = response.
Miscellaneous Information			
sender_type	UINT32	M	Refer to description of *adObjectType* in the DEC ATM MIB.
sender_sub_type	UINT32	M	Refer to the description of *adObjectSubID* in the DEC ATM MIB.
max_vci_bits	UINT32	M	Refer to the description of *atmfAtmLayerMaxVciBits* in the ATM MIB.
uid[6]	UINT8	M	The 48 bit UID of the sender, in normal Ethernet style. The “multicast” bit is bit 0 of byte 0.
pad[2]	UINT8	M	Sender sets this field to 0, the receiver ignores this field.
flow_master support	UINT32	M	Refer to the descriptoin of *adpFlowMaster* in the DEC ATM MIB.
rvc_support	UINT32	M	If set to ‘1’, the sender supports RVCs. If set to ‘0’, the sender *does not* support RVCs.
RVC Information			
broadcast_vci_out[1]	UINT32	R	Refer to the descriptoin of *adpBroadcastVCI* in the DEC ATM MIB.
broadcast_vci_in[1]	UINT32	R	Refer to the descriptoin of *adpBroadcastVCI* in the DEC ATM MIB.
arp_vci_out[1]	UINT32	R	Refer to the descriptoin of *adpArpVCI* in the DEC ATM MIB.
arp_vci_in[1]	UINT32	R	Refer to the descriptoin of *adpArpVCI* in the DEC ATM MIB.
home_vci	UINT32	R	Refer to the description of *adpHomeVCI* in the DEC ATM MIB.
FLOWmaster Information			
resync_vci[1]	UINT32	F	Refer to the description of *adpResyncVCI* in the DEC ATM MIB.
abr_buffers	UINT32	F	Refer to the description of *adpReceiveBuffers* in the DEC ATM MIB.
abr_buffers_per_vc	UINT32	F	Refer to the description of *adpMaxReceiveBufferCounter* in the DEC ATM MIB.
VCI Ranges			
rvc_vci_min[2]	UINT32	O	Refer to the description of *adpRVCmin* in the DEC ATM MIB.
rvc_vci_max[2]	UINT32	O	Refer to the description of *adpPVCmax* in the DEC ATM MIB.
rvc_vci_min[2]	UINT32	O	Refer to the description of *adpSVCmin* in the DEC ATM MIB.
svc_vci_max[2]	UINT32	O	Refer to the description of *adpSVCmax* in the DEC ATM MIB.
pvc_vci_min[2]	UINT32	O	Refer to the description of *adpPVCmin* in the DEC ATM MIB.
pvc_vci_max[2]	UINT32	O	Refer to the description of *adpPVCmax* in the DEC ATM MIB.

[1] The value used on the link must be the value sent by the dominant endpoint. On a link between a switch and a host, the switch is the dominant endpoint. On a link between a host and a host, the host with a lower UID is the dominent endpoint.

[2] For this field, if both the sender and receiver specify VC ranges that are not equal, then the intersection of the ranges should be used.

Table 3 Key for Escape Message Elements

Key	Description
M	This field must be contain the correct value.
R	If the field *rvc_support* = 1, this field must contain the correct value. Otherwise, this field must contain the value '0'.
F	If the field *flow_master_support* =- 1', this field must contain the correct value. Otherwise, this field must contain the value '0'.
O	This field can contain either the correct value or the value '0'.

7. Initialization

This section describes the initialization of the FLOWmaster flow-control for transmitter and receiver.

To determine if FLOWmaster can be supported on the link, the device must read the DEC MIB of the device at the other end of the link. The device uses either ILMI or the "escape mechanism" to read the MIB. FLOWmaster can be used on the link only if:

- Each side's MIB variable *adpFlowMaster* is equal to 2.
- Each side supports the same type of credit resynchronization.

If it is determined that FLOWmaster flow-control can be used, the device must open up a PVC whose VCI is specified by the MIB object *adpResyncVCI*. This VC carries FLOWmaster control messages.

Next the device should issue a SYNC_BOTH message for all defined PVCs (it is acceptable to issue the message for all VCs).

The device can now use the PVCs and begin to open SVCs.

Note that it is up to the device whether or not closed VCs are configured to return credits. However, if closed VCs do not return credits, then a SYNC_BOTH message must be sent when VCs are opened.

The remainder of this section assumes that both sides have agreed to use FLOWmaster flow-control.

7.1 Transmitter

The transmitter can determine how many total credits it has for all VCs by examining the downstream MIB variable adpReceiveBuffers. The transmitter can partition the credits across its VCs in any way that it wants, as long as no VC receives more than *adpMaxReceiveBufferCounter* credits.

The transmitter must resynchronize any FLOWmaster flow-controlled VCs before using them. It can do this by following an algorithm similiar to the following:

```
send a SYNC_BOTH message (vci_range, seq_num)
seq_num    += 1
retry_cnt   = 0
success     = FALSE

WHILE ((retry_cnt != 4) && (!success))
    IF (SYNC_ACK message with (seq_num - 1) is received before timeout)
        set the vci state for each VCI in vci_range to “okay to xmt”
        success = TRUE
    ELSE IF (SYNC_NACK msg with (seq_num - 1) is rcvd before timeout)
        resend the SYNC_BOTH message
        retry_cnt += 1
    ELSE IF (no response received withing timeout period)
        resend the SYNC_BOTH message
        retry_cnt += 1
END WHILE
```

Software must decide what to do if some of the VCs cannot be resynchronized. The recommended behaviour is that the device should retry synchronization at some later time.

If the software is trying to resynchronize several VCs at the same time, it might want to optimize the above algorithm by allowing VCs to transmit as soon as their VC is resynchronized. Waiting for the whole range to be resynchronized is not necessary.

7.2 Receiver

When the receiver receives either a SYNC_REQ or a SYNC_BOTH_REQ message, it performs the following algorithm:

```
vci_range   = vci_range from the received message.

DO [for all VCIs in the range]
    IF [vci is CBR] vci = “passed”
    IF [vci has all buffers emptied AND all credits returned]
        vci = “passed”
UNTIL [time T expires] OR [all VCIs in the range pass]

IF (all VCs in the range passed)
    send a SYNC_ACK message with sequence number from request
    set each VC to send credits (even for CBR VCs)
ELSE
    send a SYNC_NACK message with the appropriate vci_range.
END IF
```

7.3 General

If a device detects that the carrier on the link disappeared, it must reinitialize the link. This is the only method for a device to change the state of FLOWmaster flow-control after the link is operational.

If an ABR VC is opened, it must return credits except during a credit resynchronization. CBR circuits may or may not return credits, depending upon the implementation.

8. When to Execute Credit Resynchronization

A device must resynchronize all VCs if it receives *any* corrupted cells. It is also necessary for the device to periodically resynchronize all VCs that transmit data.

The rate at which a device must resynchronize a VC depends upon the bit error rate of the link. Because performing credit resynchronizations on an active VC can degrade its performance, credit synchronization should not be performed more often then necessary.

The following is a list of suggestions concerning periodic credit resynchronization:

- *Do not* resynchronize a VC if its credit balance is equal to the initially assigned number of credits.
- Try to resynchronize a VC when it is not active. This is less intrusive than resynchronizing an active VC.
- Resynchronize a VC if its credit count exceed its initial value.
- When a VC is resynchronized, determine whether or not the credit balance for the VC was incorrect. If not, then consider performing credit resynchronization less often (be careful not to increase the period too much). Conversely, if the credit balance was incorrect, consider performing credit resynchronization more often.

9. FLOWmaster Interaction with Circuit Setup and Release

The following sections discuss the interactions between FLOWmaster and circuit setup and release.

9.1 Circuit Setup

Figure 4 shows an example of a call setup and release between two end-nodes with one intervening switch.

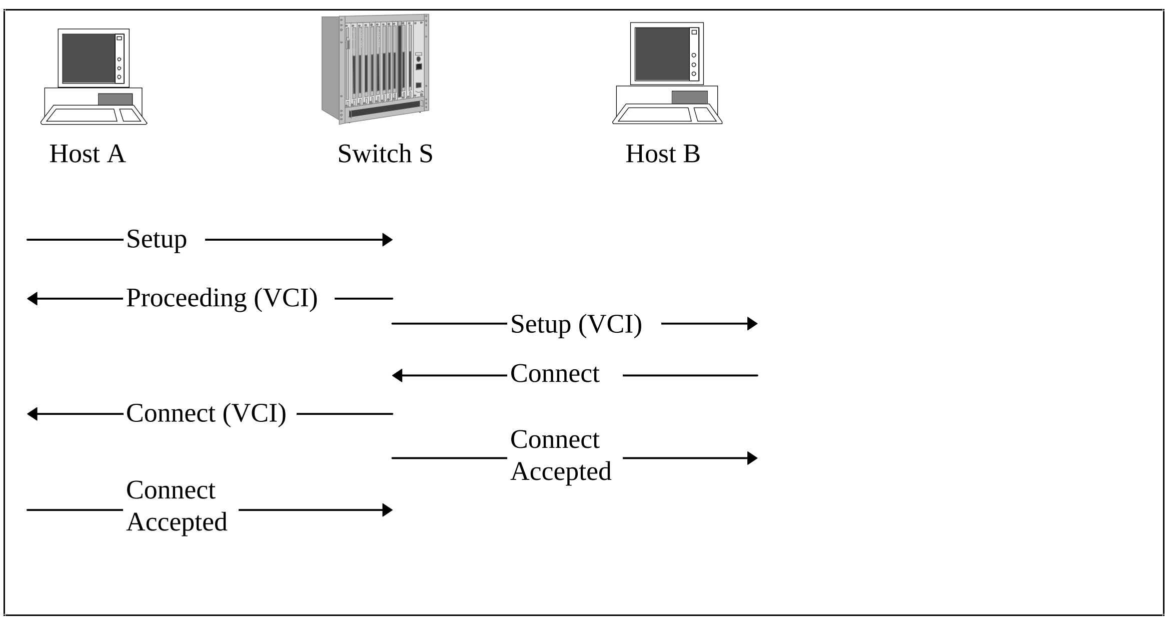

Figure 4 Circuit Setup Example

In this example host A needs to setup a circuit with host B. To do this, host A sends a call setup message to the switch that requests that a VC be setup between host A and B. When the switch receives the request, it must send a call setup message to host B. Optionally, it can also send a “call proceeding” message back to host A. This enables host A to know the VCI of the circuit to host B before the connection is completely setup.

When host B receives the call setup message, it can accept the call setup by sending a connect message back to the switch. The switch then sends the “connect accept” message back to host B and forwards the connect request message to host A. Host A can then respond with a “connect accepted” message.

1. *The responder (Host B in this example) must not start transmitting on the newly created VCI until it receives the “connect accepted message”.* The responder is allowed to either buffer or discard the data.

2. *The initiator (Host A in this example) must not start transmitting on the newly created VCI until it receives the “connect” message.*

3. *Switches are recommended to support the “call proceeding” message.* This gives the initiator time to initialize the newly created VC before it receives data. If the switch did not send this message, then the initiator could not setup the VC until it received the connect message. Since it is possible for the responder to send data on the VC and have it pass the connect message to the initiator, the initiator might drop any “early” data transmitted by the responder.

4. *The initiator must enable the newly created VCI to return credits before sending the “connect accepted message”.* This is necessary to ensure that the VCI is ready to return credits when it receives data.

Note that it is possible for the VCI on host A to receive data from host B before the VCI has been opened. This can occur because the data path from the switch to host A may be faster than the control path. Therefore, the first cells that B transmits can be lost. Additionally, the not-yet-open VCI on host A might not return credits to the switch. As a result, the switch would not be able to transmit at full link rate on that VCI. To help alleviate this situation, host A should send the switch a “SYNC_BOTH” message to give a hint to the switch that it should perform a credit resynchronization in the reverse direction.

9.2 Circuit Release

Figure 5 shows an example of the release of a circuit. In this example, host A initiates the release of the circuit by sending a “release” message to the switch. The switch fowards the release message to host B, and sends a “release complete” message to host A when it tears down the VC between the switch and host A. When host B completes closing the VC, it sends a “release complete” message to the switch.

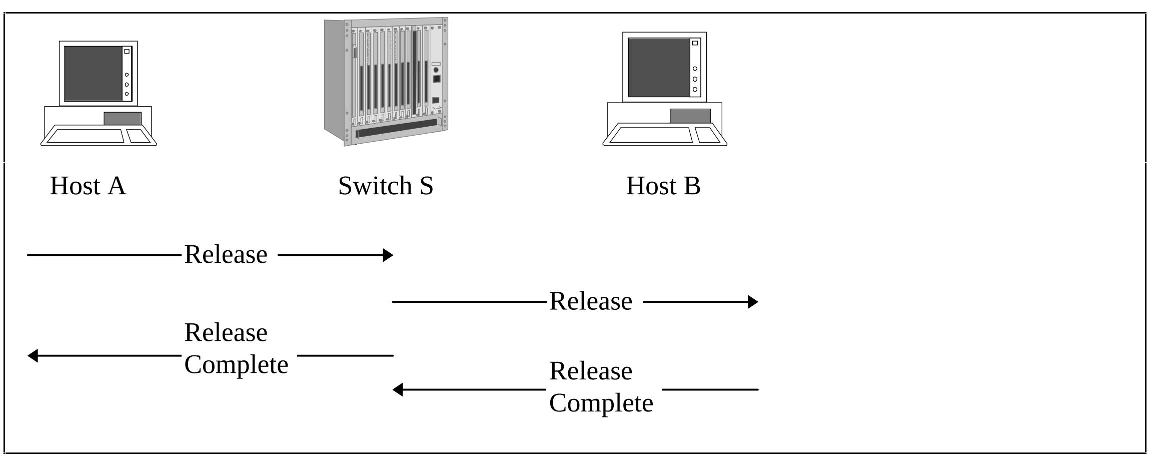

Figure 5 Call Release

1. The initiator cannot send a “release” message for a VCI until it has blocked further transmissions on the VCI, and either:

- *Flushes all queued packets on the VCI,*

- *Waits for all queued packets to be transmitted.*

2. *The initiator must continue returning credits on a VC until it receives the "release complete" message, and it has sent all queued credits.*
3. *Before sending a "release complete" message, the responder must:*
    - *Return all queued credits on a VC, and*
    - *Disable further transmissions*
    - *Wait for all queued packets to be sent or flushed.*

10. Format of Messages to Read Credit Resync Object

This section describes the format of the requests and respsonses for reading the credit resynchronization object from the DEC MIB. The formats are provided here so that devices which do not support SNMP can use the "raw" formats.

```
0x30 0x28                       ! SEQUENCE, message
  0x02 0x01 0x00                !   INTEGER, SNMP version = 0
  0x04 0x04 0x49 0x4C 0x4D 0x49 !   OCTET STRING, Community = "ILMI"
  0xA2 0x1E                     !   CHOICE, Get Response PDU
  0x02 0x01 0x00                !   INTEGER, request ID = 0
  0x02 0x01 0x02                !   INTEGER, error status = no such name
  0x02 0x01 0x01                !   INTEGER, error index = 1
  0x30 0x10                     !   SEQUENCE of objects
    0x30 0x11                   !     SEQUENCE, first object
    0x06 0x0C 0x2B 0x06 0x01 +  !     OBJECT ID, adEscapeSupport
              0x04 0x01 0x24 +  !        1.3.6.1.4.1.36.2.18.17.1.2.0
              0x02 0x12 0x11 +  !
              0x01 0x02 0x00    !
    0x05 0x00                   !     NULL, value is NULL
```

Figure 6 Request Message Format

```
0x30 0x29                          ! SEQUENCE, message
  0x02 0x01 0x00                   !   INTEGER, SNMP version = 0
  0x04 0x04 0x49 0x4C 0x4D 0x49 !   OCTET STRING, Community = "ILMI"
  0xA2 0x1E                        !   CHOICE, Get Response PDU
  0x02 0x01 0x00                   !   INTEGER, request ID = 0
  0x02 0x01 0x00                   !   INTEGER, error status = no error
  0x02 0x01 0x00                   !   INTEGER, error index = 0
  0x30 0x13                        !   SEQUENCE of objects
    0x30 0x11                      !     SEQUENCE, first object
    0x06 0x0C 0x2B 0x06 0x01 +  !     OBJECT ID, adEscapeSupport
              0x04 0x01 0x24 +  !        1.3.6.1.4.1.36.2.18.17.1.2.0
              0x02 0x12 0x11 +  !
              0x01 0x02 0x00    !
    0x02 0x01 0x01                 !    INTEGER, value is 1 (SUPPORTED)
```

Figure 7 Response Message, No Error

```
0x30 0x29                          ! SEQUENCE, message
  0x02 0x01 0x00                   !   INTEGER, SNMP version = 0
  0x04 0x04 0x49 0x4C 0x4D 0x49 !   OCTET STRING, Community = "ILMI"
  0xA2 0x1E                        !   CHOICE, Get Response PDU
  0x02 0x01 0x00                   !   INTEGER, request ID = 0
  0x02 0x01 0x00                   !   INTEGER, error status = 0
  0x02 0x01 0x00                   !   INTEGER, error index = 0
  0x30 0x13                        !   SEQUENCE of objects
    0x30 0x11                      !     SEQUENCE, first object
    0x06 0x0C 0x2B 0x06 0x01 +  !     OBJECT ID, adEscapeSupport
              0x04 0x01 0x24 +  !        1.3.6.1.4.1.36.2.18.17.1.2.0
              0x02 0x12 0x11 +  !
              0x01 0x02 0x00    !
    0x02 0x01 0x02                 !     INTEGER, value 2, NOT SUPPORTED
```

Figure 8 Response Message, Not Supported

```
0x30 0x28                          ! SEQUENCE, message
  0x02 0x01 0x00                   !   INTEGER, SNMP version = 0
  0x04 0x04 0x49 0x4C 0x4D 0x49 !   OCTET STRING, Community = "ILMI"
  0xA2 0x1E                        !   CHOICE, Get Response PDU
  0x02 0x01 0x00                   !   INTEGER, request ID = 0
  0x02 0x01 0x02                   !   INTEGER, error status = no such name
  0x02 0x01 0x01                   !   INTEGER, error index = 1
  0x30 0x10                        !   SEQUENCE of objects
    0x30 0x11                      !     SEQUENCE, first object
    0x06 0x0C 0x2B 0x06 0x01 +  !     OBJECT ID, adEscapeSupport
              0x04 0x01 0x24 +  !        1.3.6.1.4.1.36.2.18.17.1.2.0
              0x02 0x12 0x11 +  !
              0x01 0x02 0x00    !
  0x05 0x00                        !    NULL, value is NULL
```

Figure 9 Message Response, No Such Name

```
0x30 0x28                        ! SEQUENCE, message
  0x02 0x01 0x00                 !   INTEGER, SNMP version = 0
  0x04 0x04 0x49 0x4C 0x4D 0x49 !   OCTET STRING, Community = "ILMI"
  0xA2 0x1E                      !   CHOICE, Get Response PDU
  0x02 0x01 0x00                 !   INTEGER, request ID = 0
  0x02 0x01 0x05                 !   INTEGER, error status = gen err
  0x02 0x01 0x00                 !   INTEGER, error index = 0
  0x30 0x10                      !   SEQUENCE of objects
    0x30 0x11                    !     SEQUENCE, first object
    0x06 0x0C 0x2B 0x06 0x01 +  !     OBJECT ID, adEscapeSupport
              0x04 0x01 0x24 +  !        1.3.6.1.4.1.36.2.18.17.1.2.0
              0x02 0x12 0x11 +  !
              0x01 0x02 0x00    !
  0x05 0x00                      !    NULL, value is NULL
```

Figure 10 Message Response, Gen Error with Error Index = 0

```
0x30 0x28                        ! SEQUENCE, message
  0x02 0x01 0x00                 !   INTEGER, SNMP version = 0
  0x04 0x04 0x49 0x4C 0x4D 0x49 !   OCTET STRING, Community = "ILMI"
  0xA2 0x1E                      !   CHOICE, Get Response PDU
  0x02 0x01 0x00                 !   INTEGER, request ID = 0
  0x02 0x01 0x05                 !   INTEGER, error status = gen err
  0x02 0x01 0x01                 !   INTEGER, error index = 1
  0x30 0x10                      !   SEQUENCE of objects
    0x30 0x11                    !     SEQUENCE, first object
    0x06 0x0C 0x2B 0x06 0x01 +  !     OBJECT ID, adEscapeSupport
              0x04 0x01 0x24 +  !        1.3.6.1.4.1.36.2.18.17.1.2.0
              0x02 0x12 0x11 +  !
              0x01 0x02 0x00    !
  0x05 0x00                      !    NULL, value is NULL
```

Figure 11 Message Response, Gen Error with Error Index = 1

Appendix A - Cell Header Formats

A.1 Cell Types

The types of cells discussed in the following section are: idle, null, data and OAM.

Idle cells

Because the physical layer (for example the SUNI chip) inserts and removes idle cells, they cannot carry ATM layer data. This means that idle cells cannot carry credit information. Idle cells use the unassigned cell format.

Null cells

The ATM layer inserts and removes Null cells. Null cells carry credit information, but do not contain any user data.   When the device has a credit to send, but no user data on which to piggy-back the credit, it may generate a cell.   This generated cell does not contain any data, but differs from an idle cell in that the part of the VPI and VCI fields contain credit information. Null cells must be sent on VCI 0.

Data Cells

Data cells contain both credit information and user information. The VC specified in the VCI field of the cell header receives the user information.

OAM Cells

FLOWmaster does not apply flow-control to OAM cells. OAM cells do not consume credits when transmitted, and do not return credits “forwarded”. They can, however, carry credit information.

A.2 Cell Header Formats

FLOWmaster flow-control uses fields in the cell header to contain credit information. Depending upon the type of link that carries the cell, there are two different formats of the cell header. The following tables show the formats of the cell headers:

Table 4 Format of a Cell Header at the UNI Interface

<table>
<tr><th>8 5</th><th colspan="2">4 1</th></tr>
<tr><td>GFC</td><td colspan="2">Credit[11:8]</td></tr>
<tr><td colspan="3">Credit[7:0]</td></tr>
<tr><td colspan="3">VCI</td></tr>
<tr><td>VCI</td><td>PT</td><td>CLP</td></tr>
<tr><td colspan="3">HEC</td></tr>
</table>

Table 5 Format of a Cell Header at the NNI Interface

<table>
<tr><th colspan="2">8 5</th><th colspan="2">4 1</th></tr>
<tr><td>VCI[13:12]</td><td colspan="3">Credit[13:8]</td></tr>
<tr><td colspan="4">Credit[7:0]</td></tr>
<tr><td colspan="4">VCI[11:4]</td></tr>
<tr><td colspan="2">VCI[3:0]</td><td>PT</td><td>CLP</td></tr>
<tr><td colspan="4">HEC</td></tr>
</table>

At the NNI interface, FLOWmaster places the VC to receive the credit in:

- The lower 2 bits of the GFC field, and
- VPI field, and
- The most significant 4 bits of the VCI.

However, because UNI links use the GFC field, FLOWmaster only uses the VPI field to contain the VC that receives the credit.

These cell formats enable FLOWmaster to support 2^{14} VCs on NNI interfaces and 2^{12} VCs on UNI interfaces.

FLOWmaster flow-control uses fields in the cell header to contain credit information. Depending upon the type of link that carries the cell, there are two different formats of the cell header. The following tables show the formats of the cell headers:

Table 6 Format of a Cell Header at the UNI Interface

<table>
<tr><th>8 5</th><th colspan="2">4 1</th></tr>
<tr><td>GFC</td><td colspan="2">Credit[11:8]</td></tr>
<tr><td colspan="3">Credit[7:0]</td></tr>
<tr><td colspan="3">VCI</td></tr>
<tr><td>VCI</td><td>PT</td><td>CLP</td></tr>
<tr><td colspan="3">HEC</td></tr>
</table>

Table 7 Format of a Cell Header at the NNI Interface

<table>
<tr><th colspan="2">8 5</th><th colspan="2">4 1</th></tr>
<tr><td>VCI[13:12]</td><td colspan="3">Credit[13:8]</td></tr>
<tr><td colspan="4">Credit[7:0]</td></tr>
<tr><td colspan="4">VCI[11:4]</td></tr>
<tr><td colspan="2">VCI[3:0]</td><td>PT</td><td>CLP</td></tr>
<tr><td colspan="4">HEC</td></tr>
</table>

At the NNI interface, FLOWmaster places the VC to receive the credit in:

- The lower 2 bits of the GFC field, and
- VPI field, and
- The most significant 4 bits of the VCI.

However, because UNI links use the GFC field, FLOWmaster only uses the VPI field to contain the VC that receives the credit.

These cell formats enable FLOWmaster to support 2^{14} VCs on NNI interfaces and 2^{12} VCs on UNI interfaces.

Appendix B - DEC ATM MIB

FLOWmaster uses the following MIB attributes for initialization and operation. The Digital ATM MIB manages FLOWmaster flow-control through the following objects. For more detail, refer to the MIB.

adEscapeSupport

Indicates whether or not the device supports the escape mechanism described in section 6.

adFlowMaster

Indicates whether or not the device supports FLOWmaster flow-control.

adRVC

Indicates if Resilient Virtual Circuits are supported on this device.

adpFlowMaster

Indicates if FLOWmaster flow-control is supported on this *port*.

adpCreditResync

Indicates the type of credit resynchronization that the device port supports.

adpResyncVCI

Indicates the VCI on which to send credit resynchronization request and response cells.

adpReceiveBuffers

The total number of cells available for FLOWmaster buffering. This total is shared amongst all of the VCs.

adpMaxReceiveBufferCounter

The maximum number of credits that the transmitter can allocate to any given VC.

For more information on these MIB attributes, refer to the DEC ATM MIB.

11. Issues

This section is a log of issues and their resolutions.

Description	Resolution
Should a stuck VC force a resync on the next hop or simply wait for the periodic resync to happen? If resync happens periodically, then more complex mechanisms are not required.	It is the responsibility of the switch to monitor all VCs to verify that progress is being made on each of them. If not, the switch must take corrective actions. The exact actions can be discussed at a later date (the switch may choose to do nothing).

Description	Resolution
What is the recommended rate for credit resynchonization for fiber and copper links? Should this be expressed as an equation based on the bit error rate of the link?	For now, every 10 seconds for both type of links. Eventually a more sophisticated algorithm will be developed.
Should the initial number of credits be 10 or 20?	A variable called the “downstream credit return latency” must be added to the DEC ATM MIB. Implemenations should incorporate this number into the credit calculation (link distance and other factors must also be included).

End

of

Document