Implementation of ATM Endpoint Congestion Control Protocols

Prashant R. Chandra, Allan L. Fisher, Corey Kosak and Peter A. Steenkiste
School of Computer Science and Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

We describe the implementation of two ATM congestion control mechanisms, credit-based flow control and ATM Forum-style ABR rate control, on the Credit Net experimental ATM LAN testbed. This document focuses specifically on the implementation choices and performance tradeoffs available to the designer of a network interface card. We conclude that although credit protocols are more efficient and easier to implement on current hardware, efficient rate implementations are also possible with modest hardware support.

1. Introduction

In order to minimize data loss due to congestion while achieving high link utilization for bursty traffic, switched networks need some means of applying backpressure, or congestion control, to data sources. In the ATM context, two main approaches have been developed for managing congestion in available-bit-rate (ABR) traffic. The credit approach operates on a hop-by-hop basis, providing each hop's transmitter with information on receiver buffer space. The rate approach is based on estimating an end-to-end rate at which each network terminal may send.

In this paper, we briefly describe the Credit Net host interface card, the platform for our implementations and experiments. We then summarize in turn the structure and performance of several credit implementations and a rate implementation, running at both OC-3 and OC-12 speeds. Along the way, we point out some of the design options and cost/performance tradeoffs available to the network interface designer. In particular, we consider implementations that require varying levels of hardware support. We conclude that although credit protocols are more efficient and easier to implement on current hardware, efficient rate implementations are also possible with modest hardware support.

2. Credit Net Adapter

The architecture of the Credit Net adapter is shown in Figure 1. The adapter contains an ATM ASIC, designed by Intel Architecture Labs, that handles all of the ATM AAL5 segmentation and reassembly in hardware. The ASIC also has a programmable microcontroller which can schedule connections based on available credit. Also associated with the adapter is a i960 coprocessor board that is intended to be used for experimentation. The i960, ASIC microcontroller and the host processor can access the SRAM, which stores connection and segmentation/reassembly state, and can communicate with each other. The adapter plugs into a standard PCI bus, and two different versions operate at OC-3 and OC-12 rates. A detailed discussion of the design of the Credit Net adapter can be found in a separate paper [3].

This research was sponsored by the Defense Advanced Research Projects Agency under contract number F19628-92-C-0116.
The authors can be reached at {prashant, alf, kosak, prs}@cs.cmu.edu.

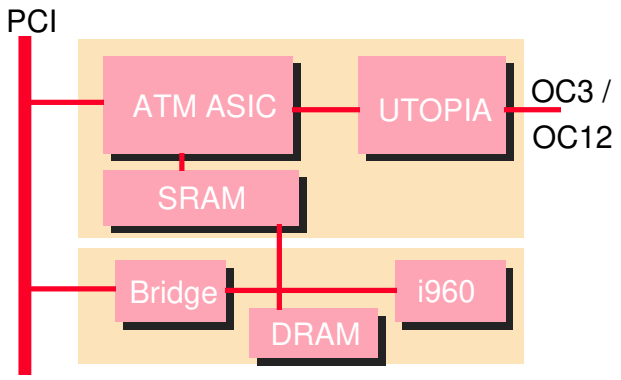

Figure 1: Credit Net adapter architecture.

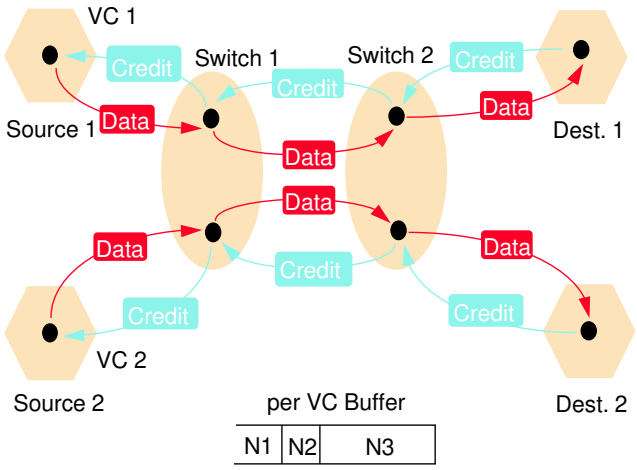

Figure 2: Credit-based flow control concept.

3. Credit-based Flow Control

Of the several approaches proposed for ABR flow control, credit-based flow control was the first to be proposed and built. The most important goal of ABR flow control is to prevent data loss due to congestion. As shown in Figure 2, the credit-based approach achieves this in a straightforward manner by sending explicit hop-by-hop feedback to the sender, in the form of "credit" messages, regarding the buffer space available at the receiver. The sender can transmit data only when it has sufficient credit. The credit based approach requires each connection to have its own separate buffer at the switch and if the switches serve connections in a round-robin fashion, fair sharing is achieved.

There are several variants of credit-based flow control [2, 4, 5]; the particular algorithm described in this document is the N123 algorithm [4]. In this scheme each connection's buffer is divided into three zones: N1, N2 and N3. The N1 area is proportional to the credit round trip time (which includes the physical round trip time plus delays due to credit processing at the sender and the receiver) in the feedback loop. The N2 parameter controls the frequency of credit messages and allows a tradeoff between credit cell overhead and the timeliness of feedback. An N2 value of 10 means that a credit cell is sent upstream for every 10 cells of data forwarded to the downstream node. The N3 area prevents data underflow so that the connection can sustain its targeted bandwidth. The credit cell reflects the number of empty cell buffers in the N2 + N3 area (also called N23).

3.1. Credit Implementations

The flexible architecture of the Credit Net adapter allows several different implementations of credit-based flow control. The implementations can be classified along two dimensions: (a) based on where the credit cell processing is implemented, the implementations can be classified as *host-based* or *i960-based*; (b) based on the granularity of scheduling quantum the implementations can be classified as *coarse grain* or *fine grain*. Note that the implementations that use the i960 could be implemented in hardware in a production system. Hence we have four different implementations of credit-based flow control that are described in greater detail below.

Host Coarse Grain Implementation

In this case the host processor handles credit cell processing in software. The scheduling granularity is on the order of packets, i.e. the host does not schedule data for transmission until an entire packet's worth of credit is available. The great advantage of this implementation is that it is a purely software implementation and hence can be implemented on any adapter without requiring special purpose hardware. The downside to this is that the buffer allocation at the switch (N23) has to be at least as large as the maximum packet size.

Host Fine Grain Implementation

In this case the host processor handles credit cell processing in software as before. However the scheduling granularity is on the order of cells. The ASIC microcontroller in the adapter is used to schedule cells based on availability of credit. The advantage of the fine grain approach is that the per connection cell buffers at the switch need be no larger than the feedback delay to prevent underflow.

i960 Coarse Grain Implementation

In this case the processing of incoming credit cells is off-loaded to the i960 coprocessor. The i960 polls for incoming credit cells and continuously updates a credit table that is shared with the host processor. The advantage of this implementation is that off-loading the credit cell processing to the i960 saves host CPU cycles which can lead to increased network bandwidth for applications. Another advantage is the reduced latency in handling incoming credit cells and hence faster response to feedback. As before this implementation also requires at least one MTU worth of cell buffers per connection at the switch.

i960 Fine Grain Implementation

The i960 fine grain implementation offers the best performance of all implementations at the cost of extra hardware (i960 or custom hardware). Fine grain scheduling allows the per-connection buffers at the switches to be small, and the i960 credit cell processing minimizes processing latency for incoming credit cells. The feedback loop is closer to the network in this case and hence dynamic response is much faster.

3.2. Performance Comparison

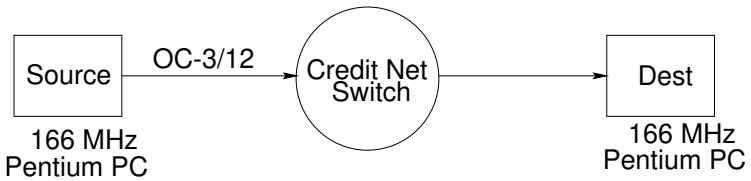

Figure 3: Experimental setup for comparing different credit implementations

In this section we compare the performance of the different implementations and study the tradeoffs involved. The experimental setup is as shown in Figure 3. The hosts involved are two Pentium 166 MHz PCs equipped with the Triton PCI chipset and running NetBSD 1.1. In this experiment a single UDP stream is established between the two PCs and the throughput is measured using the application *ttcp*, sending 8192-byte packets.

The first experiment studies the credit cell processing overhead at the host. In this case the output link from the switch runs at full speed, with N1 set to 44 and N3 set to 200. At OC-3 speeds, as shown in Figure 4, host implementations suffer from the burden of credit cell processing, especially at low values of N2. For N2 greater than 10, the throughputs of all four implementations are similar. The i960 outperforms the

Figure 6: Photograph of the comp

to its origin.

- **Scheduling of cells on ABR VCs:** Given the rates at which each VC is allowed to transmit, a host scheduler must multiplex them over the outgoing link.

On average, each of the RM cell processing operations (generation, interpretation, reflecting) will happen once every 32 cell times. At OC-3 rates, this corresponds to some operation being performed every 30 microseconds. Transmit scheduling decisions need to be made at an even finer resolution. A key benefit of the rate control approach is that switches are not required to do such scheduling, although doing so may help; the load falls instead on the sending terminal. If the host processor is expected to do useful work, these tasks will need to be performed by some other agent, e.g., a coprocessor or the adapter itself.

4.1. Implementation

In our implementation, the transmit scheduling algorithm is implemented on the ASIC's microcontroller. While the various RM cell processing operations could conceivably also be implemented on the microcontroller, severe code space constraints caused us to implement them on the i960 coprocessor instead. In the following sections we discuss the design and implementation of the transmit scheduling algorithm and each of the RM cell processing operations.

Generation of Outgoing RM Cells

The RM cell format contains several flags and three rate fields, protected against bit error by a 10 bit CRC. Although the CRC computation would be straightforward in custom hardware, a software implementation on the i960 would perform poorly. Luckily, the only field that varies between RM cells is the current cell rate (CCR) field. The other two are set to constant values: the explicit rate (ER) field is set to the link peak cell rate, and the minimum cell rate (MCR) field is set to zero. Hence the CRC for the RM cell depends only on the CCR field. Furthermore, since the ATM Forum rate format[1] can describe only 16K+1 different values, the i960 implementation is able to precompute the CRCs corresponding to all possible CCRs and store them in a table of modest size. Our current implementation, however, does not include this capability. The scheduling of outgoing RM cells is easily incorporated into the scheduler described below.

Interpretation of Incoming RM Cells

An incoming RM cell affects the rate of its corresponding VC by either triggering a multiplicative decrease in rate (in the case of congestion), or an additive increase (in the case of no congestion). Further, it imposes an upper bound, via the Explicit Rate (ER) field, on the VC's rate. Because arithmetic on values expressed in the rate format is somewhat awkward, best performance is again achieved by precomputing and tabulating appropriate result values for all possible rates. The CRC can be checked by comparing it against a tabulated value in the manner described above.

Similar processing occurs when various timeouts, on the order of 100 milliseconds, are exceeded. These timeouts can be efficiently implemented on the i960 coprocessor.

Reflecting Far-End RM Cells

At the destination, a forward RM cell is modified and reflected toward its source. The modification consists of setting the cell's direction bit and indicating any rate constraints imposed by the destination. If the destination is capable of receiving cells at link rate, no such constraint is necessary, as it will be imposed by the edge switch. If it is necessary or desirable to use the ABR mechanism to indicate dynamic congestion at the destination, the simplest mechanism is to set the RM cell's congestion bit. The RM cell's new CRC can be looked up in a table.

Scheduling of Cells on ABR VCs

A simple implementation of a rate scheduler would maintain a priority queue of VCs, sorted by their next valid transmit time. Each time a VC were scheduled, it would be requeued with a priority value of the current time plus an interval whose value is *(link rate/VC rate)*. Unfortunately, both the maintenance of a priority queue and the arithmetic complexity involved make this implementation unattractive. Worse, it has the consequence that under conditions where allowed cell rates oversubscribe the outgoing link, it slows

[1] Rates are expressed as $2^e(1 + m/512)nz$, where e is a 5 bit exponent, m is a 9 bit mantissa, and nz is a one bit nonzero flag.

down all connections in proportion, including those that are bottlenecked elsewhere in the network. We argue that it is preferable to adopt the max-min fairness criterion widely used for the rest of the network: we define a fair share rate such that all VCs with allowed rates below the fair share receive their allowed rates, all VCs with higher allowed rates receive the fair share rate, and all link bandwidth is used as long as some VC is not receiving its allowed rate. This results in contention for resources at the link leaving the host being treated in the same way that contention at links leaving switches is (ideally) treated.

Our ABR scheduler is based on this notion of fairness and on a single simplifying assumption: we assume that all VCs have an allowed rate of at least some minimum value. Slower VCs can be handled by host software if necessary. This allows us to represent the rates as a small, fixed number of intervals, where the interval is the approximate reciprocal of the rate. Converting rates to intervals is straightforward; best performance is achieved by table lookup. Using intervals has two benefits. First, it allows us to do scheduling arithmetic using only addition on a simple fixed-point representation (our representation comprises a 12 bit integer part and a 4 bit fractional part). Second, it allows us to represent the future in which cells can be scheduled as a small wraparound table. For reasons of implementation convenience, we use intervals of up to 2047 cells. For an OC-3 link, this lets us represent payload rates between 66.4 kbps (a bit higher than an ISDN B channel) and 136 Mbps, the maximum rate for ATM over OC-3. OC-12 rates are four times larger. Note that this scheme introduces a worst-case rate error of 3%; if this is not tolerable in a particular network, a solution is to round each rate downward, with the net effect of shifting bandwidth slightly from bottlenecked to un-bottlenecked VCs.

The algorithm uses three data structures: a round-robin queue, a high-priority queue, and a scheduling timetable. The timetable is large enough to accommodate the largest possible interval. The general idea is that VCs with allowed rates in excess of the fair share will be scheduled from the round-robin queue, while VCs with lower allowed rates will be scheduled from the timetable. The high-priority queue exists to ensure that low-rate VCs are serviced as rapidly as possible, to avoid falling below their allowed rates.

The scheduling algorithm lets VCs shift dynamically between the two populations, as the number of active VCs, their allowed rates, and the fair share change. The fair share itself is never explicitly represented, but arises implicitly. At each time step, the algorithm operates as follows:

1. Append the VCs from the current timetable entry to the high priority queue.
2. If the high priority queue is not empty, remove the VC at the head of the queue, schedule it, and add it to the end of the round-robin queue.
3. Otherwise:
   (a) Remove the VC at the head of the round-robin queue.
   (b) If it is eligible to go, schedule it and return it to the end of the queue.
   (c) Otherwise, add it to the timetable at its earliest eligible time, and repeat from step 3a.

Eligibility is determined by a computation involving the last time the VC was scheduled, the VC's interval, and the Generic Cell Rate Algorithm (GCRA) Limit parameter specified by the network. VCs enter and drop out as they receive or exhaust data to transmit.

Note that although the loop in step 3 may be performed a large number of times, hence potentially missing a time slot, this will be a rare occurrence. Also note that the total number of steps involving a VC between successive scheduling events is bounded, so the total scheduling effort expended is proportional to the sum of the number of time steps taken and the total number of cells scheduled.

4.2. Performance and Complexity

The code for the scheduler achieves high throughput through two additional techniques. First, it allows more than one cell at a time be scheduled, to decrease scheduling overhead. In protocol terms, this requires the GCRA Limit parameter to be greater than or equal to the Interval parameter. Second, it precomputes the next VC to send, so the scheduling computation is pipelined with the action of the segmentation process.

Figure 7 shows the throughput of raw AAL5 transfers, using different burst sizes, for OC-3 and OC-12 links. The OC-3 link can run at line rate (136 Mbps) with a 2-cell burst; at OC-12, using a burst of 8 cells, the scheduler achieves 541 Mbps vs. a line payload rate of 544 Mbps.

Elem

[4] H. T. Kung and Alan Chapman, "The FCVC (Flow Controlled Virtual Channels) Proposal for ATM Networks", *Proceedings of International Conference on Network Protocols*, October 1993, San Francisco, California.

[5] C. Özveren, R. Simcoe, and G. Varghese. Reliable and efficient hop-by-hop flow control. In *Proceedings of the SIGCOMM '94 Symposium on Communications Architectures and Protocols*, pages 89–100, University College, London, UK, October 1994. ACM.