An Implementation Study of ABR EPRCA

Brad Karp* — ISI, Harvard University
Ann Demirtjis† — Sprint
Bill Edwards‡ — Sprint
H.T. Kung§ — Harvard University
Allison Mankin¶ — ISI

April 12, 1996

Abstract

This contribution is purely informational and does not contain any motions.

In this contribution, we present our experience in developing a software-only implementation of ATM Forum ABR with EPRCA for an experimental ATM host adapter and ATM switch. This experience suggests that ABR is possible on hosts without hardware beyond that already typically found on commodity ATM host adapters, but that special-purpose hardware for ABR is essential on the switch side. The former result is noteworthy as the prevailing view has so far been that host adapters will require special-purpose ABR hardware. We describe our host and switch implementations, characterize their performance, note which ABR parameters require attention when doing ABR in software, and conclude with suggested changes to the ABR specification that will make ABR more practical to implement.

Introduction and Motivation

The specification and revision of ABR[1] have been conducted almost exclusively in the absence of a working implementation on real host and switch hardware. The work we describe herein began in the summer of 1995 with the following goals:

- to reveal the behavior of ABR *in vivo*, with real application-generated traffic;
- to create a rigorous test environment for evaluating possible enhancements or fixes to ABR;
- to reveal the challenges in implementing ABR for real network elements.

Moreover, we set about implementing ABR in software *only* for the host and switch, partially out of necessity (the scope of the project was not to include fabrication of hardware), but mainly with the following further goals:

- to determine the feasibility of a purely software ABR implementation for today's commodity ATM hosts and switches, and analyze the software system's performance in detail;
- to produce a system with maximal flexibility (no hardware changes required to make ABR algorithm changes);

*Pierce Hall 110, 29 Oxford St., Cambridge, MA 02138; karp@eecs.harvard.edu.
†9300 Metcalf, MS: KSOPKB0803, Overland Park, KS 66212; ann.demirtjis@qm.sprintcorp.com.
‡9300 Metcalf, MS: KSOPKB0801, Overland Park, KS 66212, bill.edwards@qm.sprintcorp.com.
§Pierce Hall 110, 29 Oxford St., Cambridge, MA 02138; kung@das.harvard.edu.
¶4350 N. Fairfax Drive, Suite 620, Arlington, VA 22203; mankin@isi.edu.

[1]Throughout this document, "ABR" refers to ATM Forum ABR as described in the ATM Forum TM Sepcification Version 4.0, 95-0013R7.1, dated August 18th, 1995, with the EPRCA extension.

- to reveal where a software-only ABR system would deviate from the current ABR specification, and which ABR service parameters are affected by the choice to implement the system exclusively in software.

Our system, SABRA (Software-Only ABR for ATM), runs on 133 MHz Pentium hosts that run NetBSD UNIX and use a research OC-3 ATM host adapter developed by Carnegie Mellon University in cooperation with an industrial partner. SABRA's switch side runs on the CreditNet OC-12 switch co-designed by Harvard University and Nortel. Neither the ATM host adapter nor the switch that are the substrate for SABRA have special-purpose hardware for performing any ABR-specific functions.

The results presented in this contribution suggest that ABR is readily implementable in software to run atop the limited hardware commonly found on commodity ATM host adapters, but that the switch side of ABR likely requires dedicated hardware. The former result is contrary to the commonly espoused view that host adapters require a hardware "ABR engine," while the latter is more in line with expectation. In the remainder of this document, we describe the switch and host implementations and relevant difficulties encountered in their creation, present performance measurements of the software system, and conclude with suggestions for changes to the ABR specification that make ABR more practical for implementation on real host and switch hardware.

Implementation: Host

We briefly outline here the work assigned the host by the ABR specification, assuming familiarity with the ABR parameters and their names[2]:

- Every Nrmth cell sent on a VC must be a forward RM cell.
- The time between transmissions of forward RM cells must be measured; this measurement provides an estimate of the data transmit rate, used to reduce CCR when the host sends at a rate less than CCR.
- CCR must be reduced to comply with the ER field in returning backward RM cells.
- CCR must be reduced when RM cells do not return within a round-trip time.
- CCR must be increased when it is less than PCR and backward RM cells return in a timely fashion.

SABRA only makes use of host adapter features ubiquitous in commodity ATM host adapters. It provides the above functionality on the host as follows:

- To trigger the sending of a forward RM cell every Nrm cells on each VC, SABRA maintains a count of cells sent per-VC. When outbound packets complete transmission, the host adapter interrupts the host CPU. It is at this time that SABRA records the number of transmitted cells. When the count for a VC exceeds Nrm, SABRA schedules a forward RM cell for that VC, and subtracts Nrm from the transmitted cell count for that VC.

  Because the host adapter only provides information about the departure of data when an entire packet has left, Nrm must be greater than or equal to the number of cells in a typical outbound packet. If Nrm is less than the typical outbound packet length in cells, SABRA will always learn too late that a forward RM cell must be generated. The MTU for IP over ATM is 9180 bytes, or 192 cells. Thus, to ensure that Nrm is greater than or equal to the outbound packet size, SABRA uses an Nrm of 192. Note that even when Nrm is greater than or equal to the typical outbound packet size, the upper bound on the gap between the "ideal" placement of a forward RM cell and SABRA's placement of that RM cell will be Nrm-1.

[2] Note that CCR and ACR are interchangeable in the ABR specification document; ACR is stored in the host, but is renamed CCR when written into a forward RM cell. For simplicity, we use CCR for both CCR and ACR in this document.

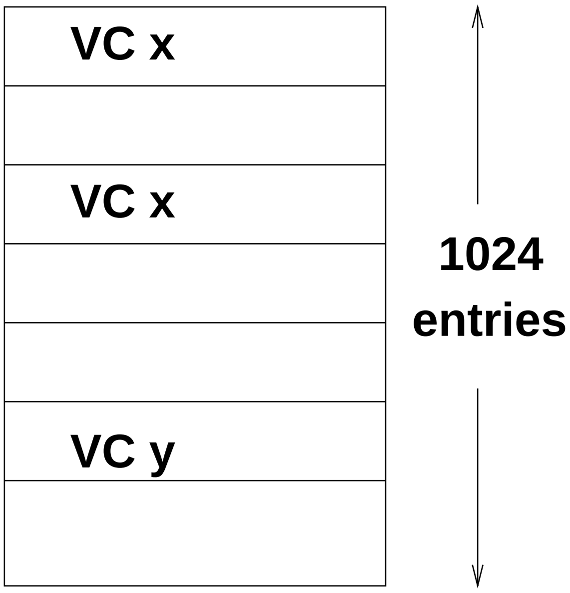

Figure 1: SABRA Host Adapter Scheduler Table

- To measure the interval between transmissions of forward RM cells for each VC, SABRA maintains timestamps (available from the real-time system clock) for the most recent pair of forward RM cell transmissions on each VC. The difference of these two timestamps provides the desired interval timing.

  According to TOF, SABRA reduces the CCR of VCs that are not transmitting forward RM cells sufficiently frequently. Note that because SABRA occasionally sends RM cells up to 2*Nrm $-$ 1 cells apart, SABRA can incorrectly judge that a VC is not sending at its CCR for TOF values less than 2; when TOF < 2, a gap between forward RM cells of 2*Nrm $-$ 1 will cause a CCR reduction. This difficulty is purely an artifact of ABR's estimation of the true outbound data rate from the gap between immediately successive forward RM cells. A more sensible approach could be to use the real cell transmission counts for each VC and real-time system clock to keep a smoothed average of each VC's true outbound data rate; the smoothing would alleviate sampling error from the occasionally late forward RM cell. Deciding when to reduce CCR due to TOF would then amount to direct comparison of CCR and the measured smoothed true outbound data rate.

- To enforce CCR for each VC, SABRA assumes the most widely supported (and minimal) host adapter scheduling hardware: a table-based scheduler. Figure 1 depicts the table for such a scheduler. Each slot of the table identifies a VC and corresponds to a single cell time, and a single VC can occupy anywhere from zero to all the table's slots. The host adapter hardware cycles through the table's slots endlessly, one slot per cell time, transmitting a cell for the VC named in the slot if one is available. SABRA's host adapter employs a 1024-slot table, so each table slot corresponds to 146.25 Kbps. SABRA writes the entries of this table to enforce the CCRs allotted each VC, spacing the slots for each VC as evenly as possible throughout the table.

  One slot in Nrm allotted each VC is marked usable *only* by an RM cell. Today's host adapters require a VC to be either in AAL5 or "raw" (single cell) mode. Thus, raw RM cells can't be queued on the same VC as AAL5 outbound data, and RM cells need to be sent on what the host adapter considers a logically separate "RM cell VC" that must be independently allocated schedule table slots. The ABR specification's view of RM cells as consuming a VC's CCR "in-band" requires this dedication of one slot in Nrm for each VC to an RM cell. If a VC's CCR requires a number of slots n not evenly divisible by Nrm, SABRA conservatively dedicates one table slot in the last $n \bmod$ Nrm to the logically separate RM cell VC. Here, "conservatively" refers to the fact that SABRA errs on the side of stealing bandwidth from a VC's data by reserving too much bandwidth for that VC's RM cells.

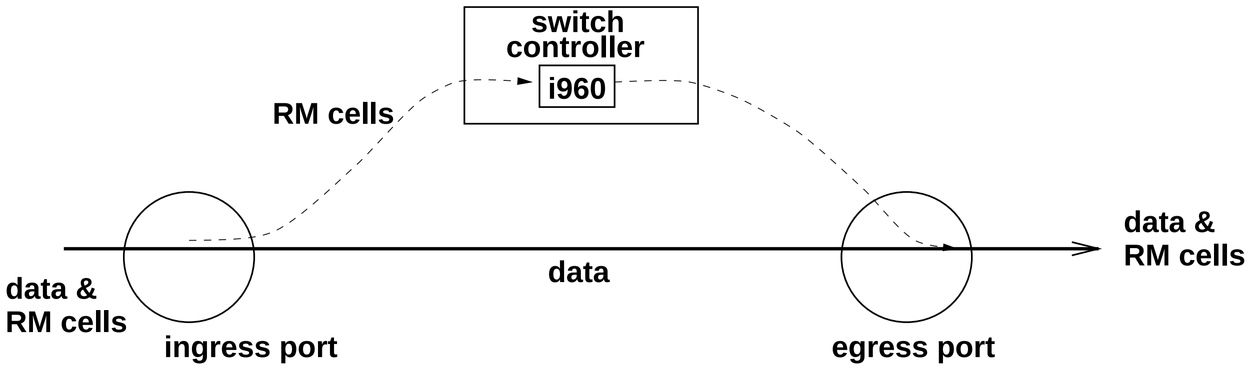

Figure 2: SABRA Switch Implementation

When a rate change is made for a VC, SABRA rewrites the entire scheduler table. This approach is naive and will be improved; a promising improvement is to maintain a freelist of unused table slots and a per-VC list of slots in use. Adding bandwidth to a VC's allocation would then amount to allocating slots from the freelist, filling them in with the identifier of that VC, and appending them to that VC's in-use slot list. Subtracting bandwidth from a VC's allocation would be the reverse of this process. Both adding and subtracting bandwidth would then require work proportional to the magnitude of the rate change, rather than the constant, worst-case work of rewriting the entire table. Over time, such table updates would destroy the uniform spacing of VCs' cell slots; periodic (less frequent than individual rate changes) rewrites of the table from scratch would mitigate this tendency.

- SABRA uses the host adapter's raw cell transmission facility to send RM cells, and arranges for the host adapter to interrupt the host CPU upon receipt of an RM cell. SABRA neither generates nor checks the CRC-10 on RM cells; the SABRA host adapter does not support CRC-10 in hardware, and software implementation of a CRC-10 is beyond the intended scope of our inquiry.
- The quantities manipulated in the arithmetic for ABR's rate calculations have fractional components. Because SABRA is implemented in the NetBSD kernel, inside which floating-point computation is prohibited, and because floating-point operations suffer from long latencies, SABRA contains a fixed-point arithmetic package hand-coded in assembler for efficient ABR rate arithmetic in the kernel.

Implementation: Switch

The SABRA switch uses the CCR field in forward RM cells to compute a smoothed average (MACR) of the rates of the ABR VCs which depart via the same output port in accordance with the EPRCA algorithm. Also in accordance with EPRCA, it monitors the queue length at each output port to determine when output ports are congested, and reduces the ER field in backward RM cells which enter on the input side of congested output ports. The SABRA switch does not check the CRC-10 of RM cells.

Figure 2 depicts one direction of the path of a data and RM cell stream through the SABRA switch. A general-purpose processor (a 33 MHz i960) on the switch control card processes all RM cells (forward and backward, for all ports). The queue lengths of the output ports are available to the i960, and are maintained by hardware on the switch ports themselves.

Performance

The SABRA software performance results described here are for the following scenario:

- single VC, OC-3 LAN, no congestion, SABRA host through SABRA switch to another SABRA host
- Nrm = 192, PCR = 353000 (OC-3 link rate)

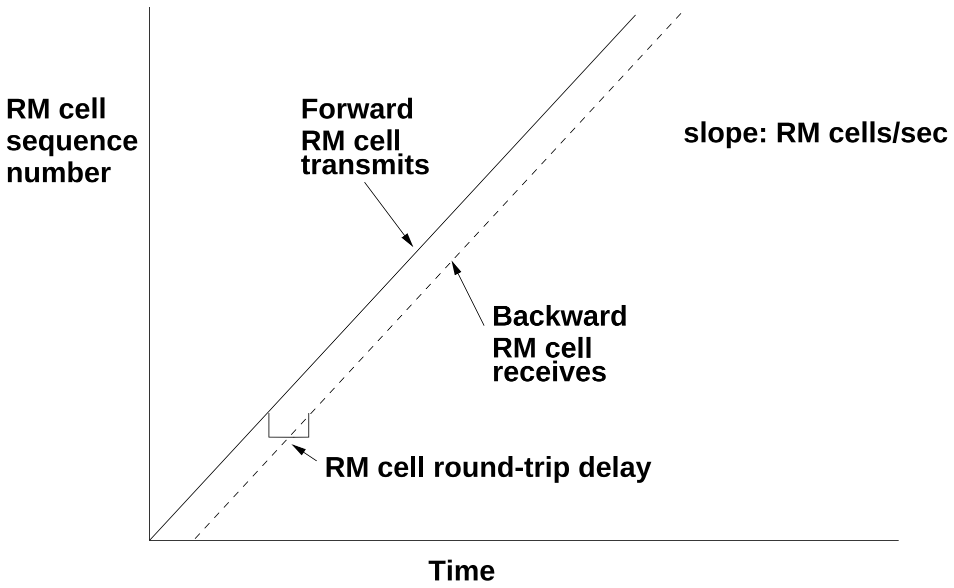

Figure 3: Expected System Behavior: RM cell sequence number *vs.* time

- traffic: 8K (171-cell) UDP packets, sent as fast as possible

A SABRA host can send UDP at 104 Mbps with ABR activated for the transmitting VC. This is within 5.5% of the throughput attainable with ABR turned off, and hence without any of the overhead of ABR-related processing.

We present a series of graphs of RM cell sequence number (y-axis) *vs.* time (x-axis). Note that the sequence numbers are not used by ABR; they were added to SABRA's RM cells purely for instrumentation purposes. The times of the graphed events were measured on the sending SABRA host using the Pentium cycle counter (accurate to less than 1 μs). These graphs track the evolution of SABRA, and reveal the issues which needed to be resolved before SABRA performed well.

First, Figure 3 depicts how an ideal RM cell sequence number *vs.* time plot would appear. The left line plots transmits of forward RM cells; the right plots reception of backward RM cells. Both these lines should show regular departures and arrivals of RM cells. The horizontal gap between the two curves at the same y value represents the end-to-end round-trip delay incurred by the RM cell with sequence number y. This delay should be minimized; delay of RM cells causes out-of-date congestion information to arrive at the sending host. The slope of the left line is the rate of forward RM cell generation in cells per second, while that of the right line is the rate of backward RM cell receipt. These slopes should be equal; if they are not, some element in the path is either introducing jitter into RM cells or is incapable of processing RM cells at the rate at which they are generated.

Figure 4 depicts the initial SABRA system's behavior. The system sends a forward RM cell on with each of several back-to-back outbound packets, then does not for a single outbound packet, and then repeats this pattern. This phase relationship between the packet size (171 cells) and Nrm (192 cells) is expected—two packet completions are required to move the transmitted cell count high enough to trigger the first forward RM cell transmittal. For the second and each of the subsequent eight packets to complete sending, 171 cells are added to the count and 192 are subtracted. At that point, the transmitted cell count falls below 192, and no RM cell is sent for one packet completion. The cycle then completes.

Two undesired effects are immediately apparent from Figure 4:

- Receipt of backward RM cells is mutually exclusive with transmittal of forward ones!
- The worst-case, round-trip delay of RM cells is on the order of 20 ms. The forward RM cells which were transmitted during that 20 ms interval then return "bunched together," until another 20 ms delay occurs.

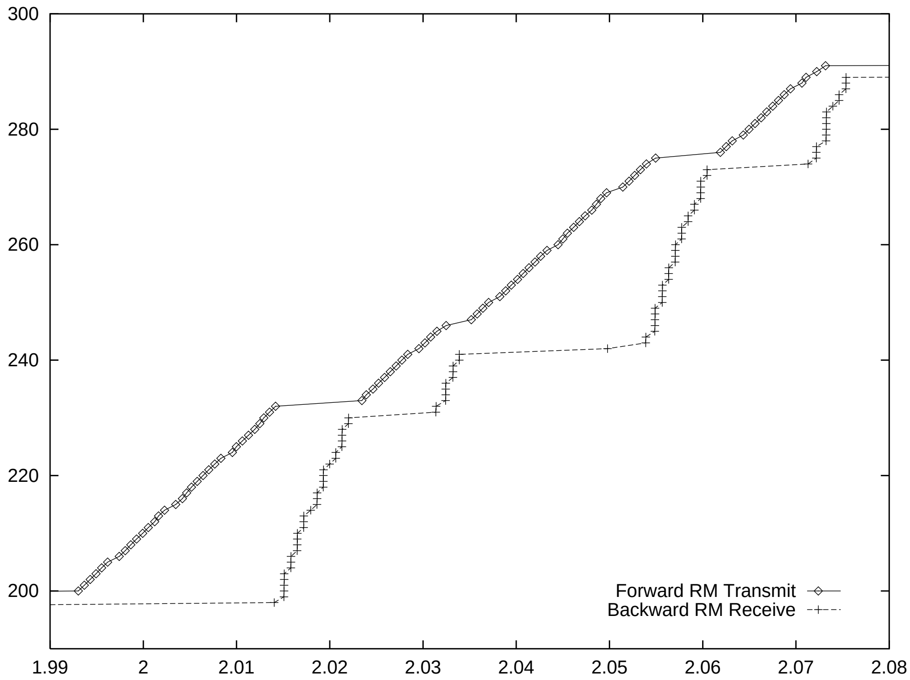

Figure 4: Initial SABRA System Behavior: RM cell sequence number *vs.* time

We start with the first of these two observations. Figure 5 is the same as Figure 4, with two additional event types recorded: packet transmit completions are shown as squares, and schedule table rewrites are shown as X's. It is apparent that packets are not being transmitted, either, while backward RM cells are received.

The explanation for the starvation of the host adapter's transmit hardware is in the schedule table rewrites. Figure 5 clearly shows that the schedule table is being rewritten for *each* backward RM cell which arrives. Investigation of the host adapter architecture reveals that there is a single bus to the SRAM in which the schedule table lives. This bus is *shared* by the host CPU, which writes entries into the table, and the host adapter scheduler hardware, which reads entries from it. The arbitration discipline enforced by the adapter is to give strict priority to host CPU writes over the scheduler's reads. Thus, the scheduler starves and stalls all cell transmission while the host CPU touches the table, which happens more or less continuously while backward RM cells are received.

This arbitration problem is an artifact of the particular SABRA host adapter; there exist other host adapters with *two* schedule tables. In such adapters, the scheduler can read one simultaneously with the host's writing the other. Changing a value in a register permits instant toggling back and forth of the scheduler between the two tables. The lesson here is that a software ABR implementation requires a host adapter which permits simultaneous data transmission and schedule changes.[3]

Other fixes will ameliorate this problem on the SABRA host adapter, too. As mentioned during the discussion of the table writing algorithm, an implementation which generates many fewer writes per rate change is achievable. Moreover, it is wasteful to rewrite the scheduler table unless a CCR has actually changed. In this particular scenario (a single VC without congestion), modifying SABRA so that the schedule table is rewritten *only* after a rate change eliminates *all* table rewrites once the sending VC has ramped up to its PCR. Figure 6 depicts SABRA's behavior with the addition of this change.

SABRA now correctly continues transmitting forward RM cells (and data) while receiving backward RM cells. The problem of the 20 ms worst-case round-trip delay for RM cells remains, however. This time, the SABRA switch was to blame: the switch control processor had no mechanism for asynchronous notification

[3]Another possibility for supporting concurrent data transmission and schedule changes is simply to use a dual-ported RAM for the schedule table; in this way, the writer and reader each have exclusive access to their own port.

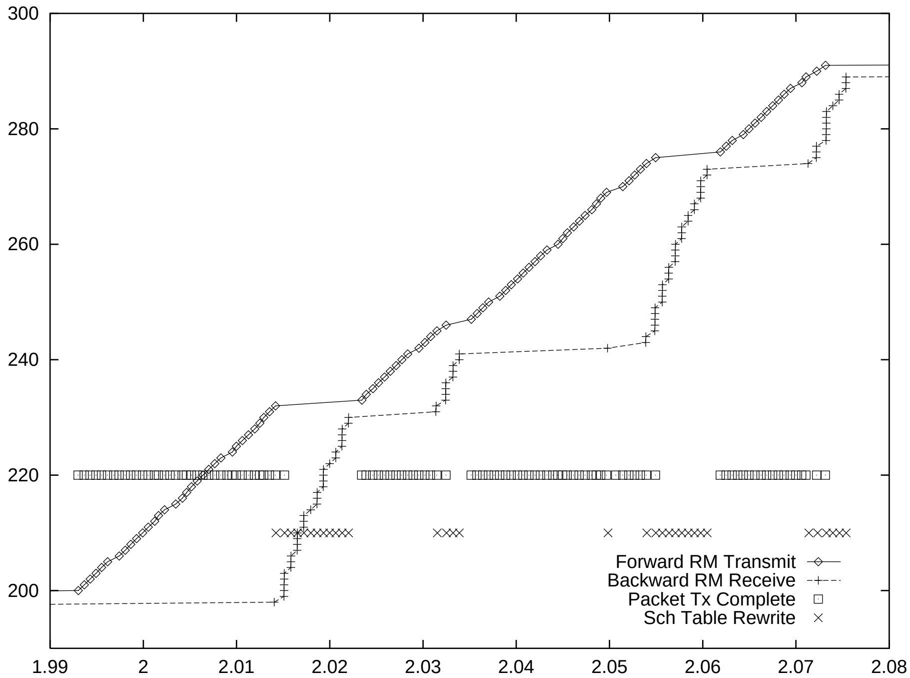

Figure 5: Initial SABRA System Behavior: Show Packet Transmits and Table Rewrites

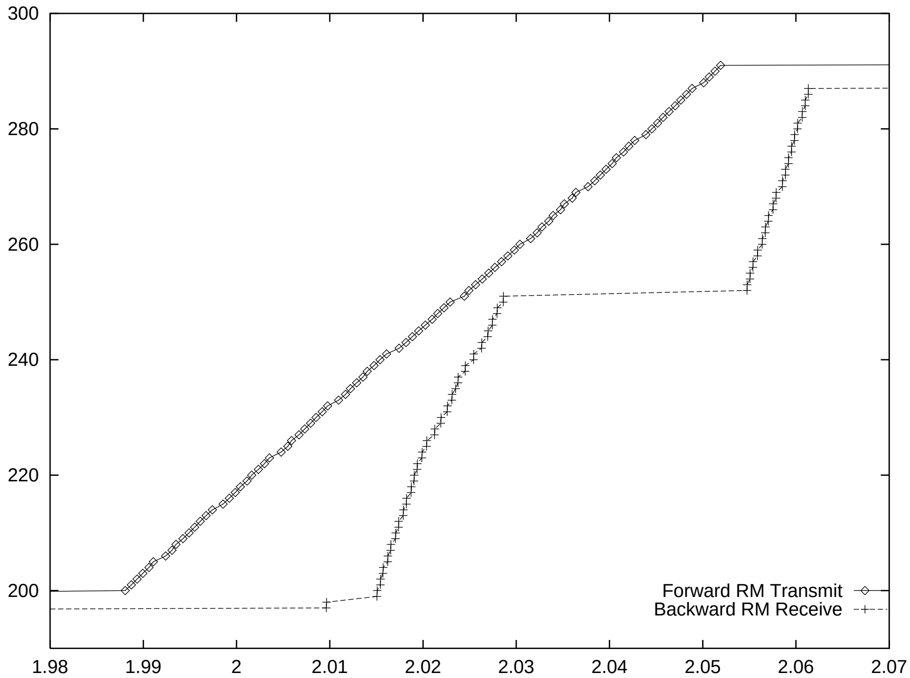

Figure 6: SABRA System Behavior: Change Rates Only When Required

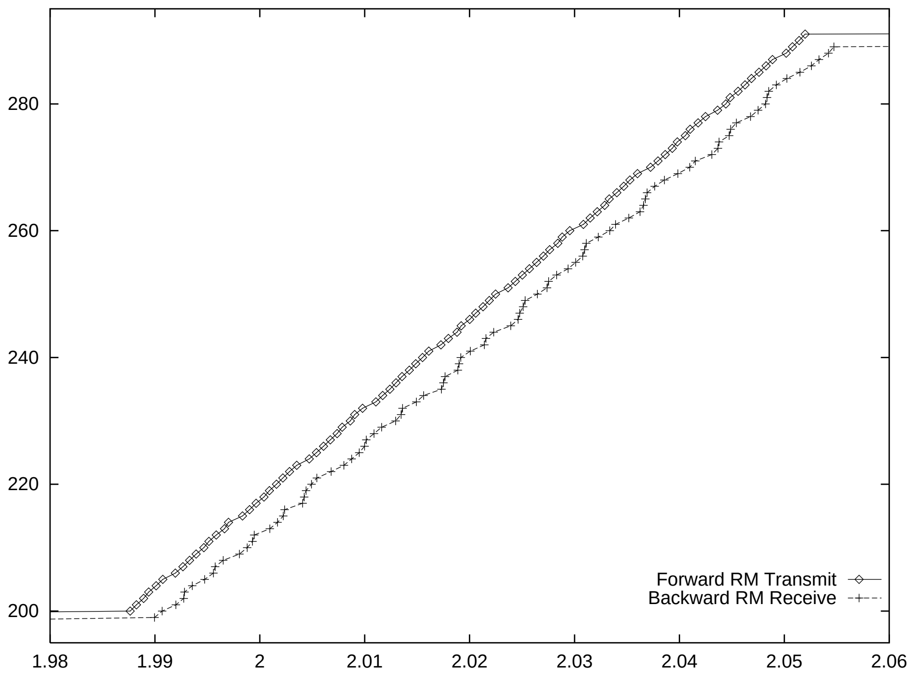

Figure 7: SABRA System Behavior: Added Interrupts on RM Cell Arrival at Switch

that an RM cell had arrived for processing. The UNIX-like kernel which runs on the SABRA switch's control processor implemented RM cell processing as a thread. Because the kernel's timer tick resolution for thread scheduling is 20 ms, an RM cell could "just miss" catching the RM cell consumer thread awake, in which case the thread would sleep, not to awake again for another 20 ms. At that time, the thread would drain the queue of RM cells which accumulated during its 20 ms asleep, only to sleep again after it drained the queue. (This explanation assumes that the switch is capable of processing the RM cells more quickly than they arrive; this assumption is valid because the slope of the backward RM cell receive curve is steeper than that of the forward RM cell transmit curve.)

We modified the firmware of the SABRA switch to present an interrupt to the switch control processor upon RM cell arrival, and rewrote the switch software to be interrupt-driven. Figure 7 shows the system's behavior after these changes.

The slopes of the two lines are now equal. The horizontal gap between the lines at a fixed y value (an RM cell's end-to-end, round-trip delay) is typically 3 ms. Given that these data were taken on a LAN, this figure seems excessive. Examination of the SABRA host adapter's behavior when transmitting and receiving raw (non-AAL5) cells revealed a major contributor to this delay: the adapter does not notify the host CPU of the receipt of a raw cell until the arrival of the *subsequent* raw cell. Each trip an RM cell makes into a SABRA host then adds an inter-forward-RM-cell transmission time to that RM cell's round-trip delay. As there are two such receives along each RM cell's round-trip path (one at the destination system, and another back at the source system), this situation is responsible for about 1.2 ms of the remaining 3 ms round-trip delay to be accounted for. Note that these 1.2 ms are specific to the SABRA host adapter, and not fundamental to SABRA.

Figure 8 provides a map of the latency of an RM cell's round-trip path under SABRA.

The SABRA switch can process 6200 RM cells per second; this number is the slope of the backward-RM cell receive curve in Figure 6, during an interval where the switch is draining an accumulated queue of RM cells. Table 1 details the instruction counts involved in processing forward RM cells; Table 2 has the corresponding figures for backward ones. Note the portion of the instructions in the "processing" phase spent on parsing the non-standard, RM-cell floating-point format. These instructions are effectively wasted work; there is nothing gained from expressing rates (the only fields used in an ABR RM cell) in floating-point. Representing a fraction of a cell per second is pointless, and there is ample room in the RM cells for carrying

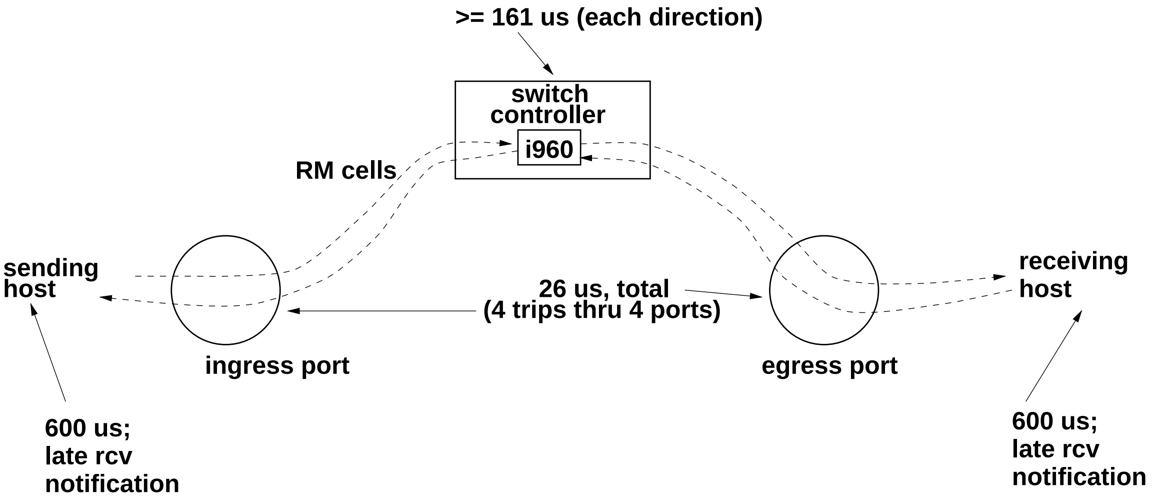

Figure 8: End-to-End RM cell Latency in SABRA

Task	Instruction Count
find cell	38
read cell	50
process cell	195 (39 for FP)
write cell	50
total	333

Table 1: Instruction Counts for Forward RM Cell Processing at SABRA Switch

rates as 32-bit integers.

As a final performance datapoint, note that the average RM cell arrival rate at a 16-port, OC-3 switch is about 29.5K RM cells per second (every 34 μs). The worst case, however, is that there are RM cell arrivals at 16 ports in one cell cycle. This is 177 ns per RM cell, and could occur for back-to-back cell cycles on a WAN switch. While a fast, general-purpose processor might keep up with the long-term average RM cell arrival rate, it could only do so at the risk of delaying RM cells. Moreover, a switch's general-purpose control processor has other work to do besides processing RM cells, including VC setup, monitoring functions, *&c.*

Conclusion

We have presented the implementation of SABRA, a software-only implementation of ABR with EPRCA for a real ATM host adapter and switch, and an analysis of its performance.

Task	Instruction Count
find cell	38
read cell	50
process cell	222 (83 for FP)
write cell	50
total	360

Table 2: Instruction Counts for Backward RM Cell Processing at SABRA Switch

The most important observations drawn from our experience with SABRA are as follows:

- Results thus far are very positive regarding the possibility of implementing ABR for hosts in software only, while special-purpose hardware appears necessary on the switch side.
- Eliminate the 16-bit floating point format for rates from RM cells, and replace it with a 32-bit integer format. The floating-point format must be parsed either in dedicated hardware or in dozens of instructions, and does not improve the functionality of ABR in any way whatsoever.
- Table-based schedulers in host adapters are adequate for enforcing rates, but care must be taken to design them so that they are capable of transmitting data while their contents are updated.
- Scheduling forward RM cells in software requires an Nrm greater than or equal to the output packet size, and introduces a worst-case margin of error for forward RM cell scheduling of Nrm -1 cells. This margin of error in forward RM cell placement can cause the sending host to decide incorrectly (via TOF) that a VC is transmitting at a rate less than its CCR. Measuring the output rate of a VC directly from transmitted cell counts over time rather than the time interval between two consecutive forward RM cells should alleviate this wrinkle.
- Allocating bandwidth for RM cells in-band with data on the same VC would be made much easier with the addition of a small hardware change to host adapters: if the buffers in which data queued for transmit on a VC contained a tag bit identifying them *individually* as AAL5 or raw cells, the host could then queue RM cells in-band on the same VC as data. This feature would require that the CRC-32 computation engine simply pause its computation of the CRC-32 for the outbound AAL5 PDU when a buffer marked as raw occurred in the middle of buffers for the same outbound AAL5 PDU.