Queue Management for Explicit Rate Based Congestion Control

Qingming Ma
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
qma@cs.cmu.edu

K. K. Ramakrishnan
AT&T Labs. Research
600 Mountain Avenue
Murray Hill, NJ 07974, USA
kkrama@research.att.com

Abstract

Rate based congestion control has been considered desirable, both to deal with the high bandwidth-delay products of today's high speed networks, and to match the needs of emerging multimedia applications. Explicit rate control achieves low loss because sources transmit smoothly at a rate adjusted through feedback to be within the capacity of the resources in the network. However, large feedback delays, presence of higher priority traffic, and varying transient situations make it difficult to ensure *feasibility* (i.e., keep the aggregate arrival rate below the bottleneck resource's capacity) while also maintaining high resource utilization. These conditions along with the "fast start" desired by data applications often result in substantial queue buildups.

We describe a scheme that manages the queue buildup at a switch even under the most aggressive patterns of sources, in the context of the Explicit Rate option for the Available Bit Rate (ABR) congestion control scheme. A switch observes the buildup of its queue, and uses it to reduce the portion of the link capacity allocated to sources bottlenecked at that link. We use the concept of a "virtual" queue, which tracks the amount of queue that has been "reduced", but has not yet taken effect at the switch. We take advantage of the natural timing of "resource management" (RM) cells transmitted by sources. The scheme is elegant in that it is simple, and we show that it reduces the queue buildup, in some cases, by more than two orders of magnitude and the queue size remains around a desired target. It maintains max-min fairness even when the queue is being drained. The scheme is scalable, and is as responsive as can be expected: within the constraints of the feedback delay. Finally, no changes are needed to the ATM Forum defined source/destination policies.

1 Introduction

The Available Bit Rate (ABR) service defined by the ATM Forum supports data applications and emerging rate-adaptive multimedia sources in Asynchronous Transfer Mode (ATM) networks. Its operation relies on an effective congestion control mechanism. Rate based congestion control has been considered desirable to achieve high performance over high speed networks that are characterized by large bandwidth-delay products. Its potential to achieve low loss by maintaining a smooth flow of data with its source rate being adjusted through a feedback mechanism [14] allows intermediate systems (switches) to have relatively small buffers while still maintaining high utilizations. In contrast, although window flow control has undergone several refinements to also maintain a smooth even flow of data, there are several conditions during which window flow control results in bursty arrivals into the network. The smooth flow of data packets in response to the arrival of acknowledgements is disturbed in cases where there is ack-compression [16, 15], new flows starting up, or when multiple flows recover from packet loss, and go into slow-start [6]. Moreover, allowing the applications to "fast-start", i.e., to transmit as fast as reasonable upon startup, is desirable for many data applications.

Rate based congestion control seeks to allocate the source rates so as to achieve high resource utilization, while maintaining *feasibility* (i.e., the capacity of any resource in the network—primarily link bandwidth—is not exceeded at any time). With even a small excess in the source rate, it can cause a substantial queue buildup. The ABR service can ensure a particular notion of fairness—max-min fairness [4, 12], which requires a distributed computation [3]. An incremental computation that scales with the number of connections is described in [9].

The incremental computation of source rates can potentially result in the rates being infeasible for short intervals (often one round-trip time). Varying feedback delays that result in asynchronous updates to source rates make the control of this period of infeasibility difficult to predict. Several other practical situations also make it difficult to ensure feasibility: changes in the capacity due to the presence of higher priority traffic; changes in the number of users as they arrive or depart; desire of sources to opportunistically utilize available bandwidth, since once left unused, it is lost forever; and desire of data applications to ramp up as fast as possible on startup. All of these cause transient situations that can result in queue buildups which are sometimes substantial.

One way to ensure feasibility is to force a source being allowed to increase its rate to delay any increase until all other sources have received and implemented their decreases. Thus, the aggregate rate at a given link will never exceed its capacity [4]. This method introduces considerable delay to sources when they start up or when they are asked to increase their rate, thus impacting applications (and user-perceived performance) adversely. It also may lead to underutilization of resources. In addition, when the bandwidth in the network changes, there is a certain time taken to provide feedback to the sources so that they may change their source-rates accordingly. The build-up of the queues during this transient period cannot be avoided even by schemes that are extremely conservative in ensuring feasibility.

Unlike the scheme proposed in [4], the ATM Forum's ABR service attempts to maintain feasibility and avoid a large queue buildup by picking conservative values for the "increase parameter", RIF and the initial cell rate ICR [14]. We show that even a small ICR and RIF, can still result in substantial queue buildups.

We first study the queue behavior in detail and show how and to what degree queues can build up. This motivates the use of a mechanism that manages queue buildup in concert with the rate allocation mechanism for the operation of explicit rate based congestion control. We want to ensure that we operate the network with small queues, so as to minimize packet loss, provide reasonably low (not a strict bound) delay, and keep the feedback delay small (especially if RM cells are processed in-band). Related work has been described in the recent past to manage queue build-up [8]. In broad terms, most of the techniques suggested achieve a low buffer occupancy by keeping the utilization of the link below the full capacity (e.g., having a target utilization of 95%). We seek to maintain the link utilization at the maximum (i.e., 100%), except when we need to reduce the buildup queue. Moreover, we attempt to do this while maintaining exact max-min fairness and the feature of constant-time computation of max-min fairness described in [9].

The next section describes the operation of the feedback control mechanism, for both the source and switch policies. Section 3 motivates the problem of queue buildup under a variety of scenarios. Subsequently, in Section 4 we describe the policy we propose for managing the queue, and examine the performance under the same set of scenarios that we used to motivate the problem. Finally, we conclude in Section 6.

2 Operation of the Feedback Control Mechanism

The explicit rate control scheme depends on a set of co-operating sources periodically probing the network for the appropriate transmission rate. Its goal is to respond to incipient congestion, and to allocate rates to the competing sources in a fair manner, while ensuring feasibility.

Each source of a virtual circuit (VC) periodically transmits a special *resource management* (RM) cell to probe the state of the network. It specifies a "demand" or desired transmit rate in an *ER-field* in each RM cell. In addition, to the currently allowed rate (ACR), which is the rate at which queued cells are transmitted out of the network interface is transmitted in the CCR field of the RM cell.

Each switchcomputes the rate it may allocate to each VC, and overwrites this allocated rate in the ER-field if the computed rate is lower than what was in the received RM cell. As the RM cell progresses from the source to destination, the ER-field value reflects the smallest rate allocated by any of the switches in the path for the VC.

On reaching its destination, the RM cell is returned to the source, which now sets its transmit rate based on the ER-field value in the returned RM cell and a specified policy.

2.1 The Source Algorithm

The source policies are a simplified version of [14], where the primary properties of the feedback control loop have been implemented, without incorporating all the issues relating to the boundary and failure cases. Sources maintain a *DEMAND* (for data sources this may be the outgoing link's rate), used for requesting a rate from the network. When an RM cell returns with an allocated rate ER, the source's allowed rate is changed as follows:

```
if (ACR <= ER)
     ACR <- max(min(ER, DEMAND), MCR)
else
     ACR <- max(min(ACR + (RIF * PCR), ER), MCR)
```

Notice that a request to decrease the rate takes effect immediately. On the other hand, when ER received by the source is higher than the current ACR at the source, ACR is increased additively by a step size of $RIF * PCR$. The increase factor RIF is a negotiated parameter, and PCR is the peak cell rate for the connection. ACR always remains above the minimum source rate MCR.

When an RM cell is transmitted, the ER-field is set to $max(DEMAND, ACR)$. RM cells are periodically transmitted, once every Nrm data cells (e.g., Nrm=32). A large RIF results in reaching the network allocated rate, ER, quickly, but with the potential for some transient overload on the network. Motivated by a desire to keep queues small, RIF is often chosen to be small (we make observations on this "typical" choice later).

2.2 The Switch Allocation Algorithm

There are several switch algorithms proposed for computing the rate to be allocated to a VC [3, 13, 9, 7]. Switches compute an allocated rate for each VC i, based on its requested rate (value in the ER-field) A_i. VCs are classified as being either "satisfied" (in set $\mathcal{S}$) or "bottlenecked". The capacity C of the link is allocated to bottlenecked VCs as:

$$A_B = \frac{C - \sum_{i \in \mathcal{S}} A_i}{N - ||\mathcal{S}||} \qquad (1)$$

A global procedure performing the maxmin computation is described in [2, 4]. Since we perform an incremental calculation upon the arrival of an RM cell, the A_i for VC_i is equal to the "demand" seen in the ER field of the RM cell. The allocated rates for the other VCs are the allocations made when the computation was performed when their forward RM cell was processed.

Given the current knowledge of the allocations to the VCs, we identify those flows that appear to be satisfied, then determine the total bandwidth consumed by those flows, and divide the remaining bandwidth equally between the flows bottlenecked locally. A straightforward computation of the local maxmin fair share (denoted A_B) can be described as follows [12]:

1. Initialization: $\mathcal{S} = \emptyset$, $j = 0$.
2. $A_B^j \leftarrow C/N$, $\forall i \in \{i \mid A_i < A_B^j \;\&\; i \in \mathcal{S}\}$.
3. $A_B^{j+1} \leftarrow (C - \sum_{i \in \mathcal{S}} A_i)/(N - ||\mathcal{S}||)$.
4. **If** $\exists i \notin \mathcal{S}$, $A_i < A_B^{j+1}$ **then** $j \leftarrow j + 1$; go to step 2.
5. $A_B \leftarrow A_B^j$.

Subsequently, the allocation for the VC i, whose RM cell was received, is marked as follows: If $A_B \leq min(ER_i, CCR_i)$, then $A_i = A_B$, and the VC is marked bottlenecked. Otherwise, $A_i = min(ER_i, CCR_i)$, where ER_i is the value in the ER field in the RM cell, and CCR_i is the value in the CCR ("current cell rate") field of the RM cell.

3 Queue Behavior

In this section, we present a detailed study of the queueing behavior of the explicit rate based congestion control to motivate our work on queue reduction mechanisms. We consider both a common queue and a per-VC based queueing structure at the switches. The corresponding scheduling mechanisms would be first-in-first-out (FIFO) or Round-Robin (RR).

The rate allocation performed at any given link for a VC can only take effect after the RM cell has returned back to the source. During this time the queue can be built up if a different VC that has a shorter feedback delay ramps up its rate based on an indication to increase. The larger the differences in the delay, the worse the build up of the queue.

To understand the queueing behavior better in relationship to the feedback delay, we built a cell-level event-driven simulator. Much of our study is with a simple network configuration that contains two switches. Each switch is attached with a few (ranging from 3 to 500) host nodes with links having different propagation delays. There are 2 types of VCs in the configuration (Figure 1) shown:

- Long VCs: with the first link having an 8 millisecond propagation delay.
- Short VCs: with the first link having a 0.8 μsecond delay.

There are 3 source policies we examine, that progressively add small amounts of functionality at connection setup:

- The ATM Forum-based policy for the source, where the source's initial rate a "statically picked" ICR, and the increase, when allowed, is by a factor RIF.
- The policy marked "RATE", where the source sets the initial rate ICR based on computing the max-min fair share, when the connection setup message arrives at the switch. However, the allocation is made at the switch for the VC only upon encountering the first RM cell from the VC when it begins transmitting data.
- The policy marked "RM", where the initial rate ICR is set based on computing the max-min fair share when the connection setup message arrives at the switch. Further, the connection setup is treated as an RM cell so that the allocation is performed at the switch when the connection setup message arrives. The connection setup provides an early "warning" of a new VC's arrival to all of the others.

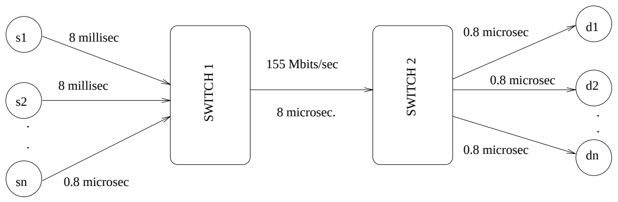


Figure 1: A simple configuration of two-switch network

3.1 Impact of Initial Cell Rate

In practice, a small ICR is preferred because it avoids having a large burst load on the network occuring suddenly when a VC starts up. Our observation is that a small ICR does not always result in a smaller queue length at the bottleneck. This is particularly true when there is significant disparity between the round trip times of the various connections, as in the configuration shown in Figure 1.

A large ICR helps applications. For example, RPCs that have a burst of small messages may benefit from high ICR, rather than wait one or more round trips to ramp up to the max-min fair rate. Secondly, if the network is idle or not fully-utilized, starting with high initial rate can avoid under-utilization during the startup phase. The workload used on the configuration in Figure 1 is as follows: one long VC starts at $t = 0$; two short VCs start at $t = 400$ and $t = 800$ milliseconds. All the VCs are greedy, and have infinite amounts of data to send in this example.

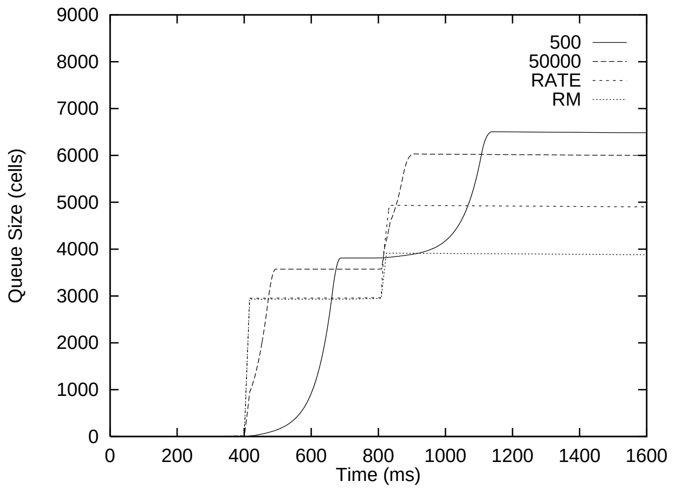
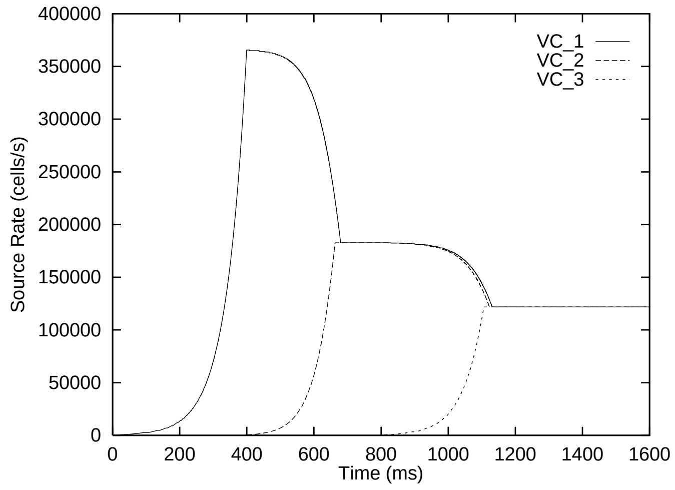


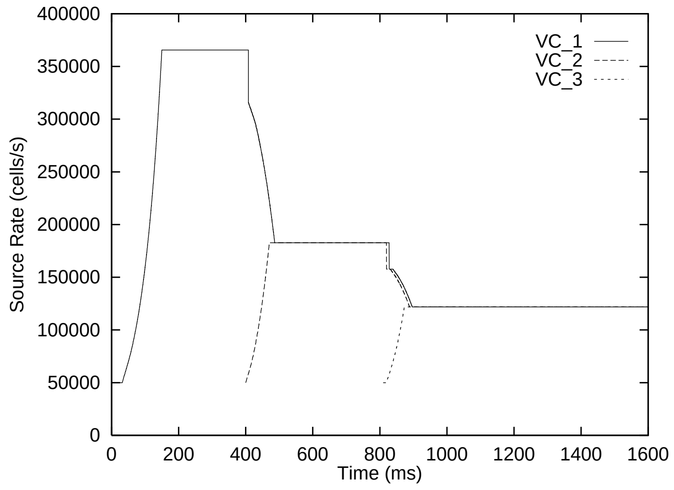
Figure 2: Queue size for different ICR with FIFO, $RIF = 1/512$, and Nrm = 32

3.1.1 Behavior with FIFO Queues

In Figure 2, we show the behavior of the growth of the buffer size with time, for 4 different cases. All of these were with a



(a) $ICR = 500$ cells/sec.



(b) $ICR = 50000$ cells/sec.

Figure 3: Source rate for different ICR with FIFO, $RIF = 1/512$, and Nrm = 32

conservative rate increase factor (RIF) of 1/512. The four cases are: ICR set to 500 cells/sec. for each source starting up; ICR set to 50000 cells/sec. for each source; the RATE option; and the RM option.

When the short VC arrives at $t = 400$ milliseconds, this causes a queue to begin building up. This is because the short VC encounters several steps of increase, by small amounts, for each decrease by the long VC. The decrease by the long VC is commensurate with the amount that the short VC has increased from its initial ICR, at the time the long VC's RM cell is processed. During the feedback delay for the long VC, the short VC can now increase by several steps (many RM cells are feedback to it). The further arrival of another "short VC" at 800 milliseconds causes a further queue to buildup.

Thus, we see a larger queue at the switch with a small $ICR = 500$ cells/sec., compared to having a larger ICR of 50,000 cells/sec (this is likely to be true only as long as the ICR is smaller than the final max-min fair share of the VC). A further problem is the lower utilization of the bottleneck link when the source starts up with a small ICR. The behavior of the queue can be further explained by observing the source rates of the VCs, as they start up, shown in Figure 3. The rate for the existing long VC gradually decreases as the new short VC ramps up its rate. Looking closely at the time when the sources all converge to the new rate, there is a brief period where the short VC reaches its final max-min fair share, while the long VC has still not reduced down to its steady state due to its feedback delay. This infeasibility results in the queueing that we observe in Figure 2. A larger ICR (Figure 3 (b)), for the "short VC"(VC 2 and VC 3) rapidly withdraws allocation from the existing "long VC"(VC 1). This steeper decrease reduces the interval over which there is an overallocation of the capacity, reducing the queue buildup. Figure 2 also shows the behavior of the queue with the RATE and RM alternatives for startup behavior that we believe are more attractive. The RATE alternative brings down the queue from almost 6K cells to a little less than 5K cells in this particular example. With the "RM" option, this reduces the queue even more: down to nearly 4K cells. This reduction in the queue size is also accompanied by the same reduction in underutilization we observed when starting at a large ICR. With the "RM option", the source rates of the new sources start at what will be their max-min fair share, and the rates of the existing sources drops immediately to its corresponding fair share as well. As a result, the period of infeasibility is minimal (at most one round trip time for the long VC). This results in only a small amount of queueing.

Thus, a small initial cell rate does not always result in a small queue size. Particularly with algorithms that compute the allocation on an incremental basis, having an ICR that is high allows for a quicker correction of the rate for long VCs, and thus a lower queue length at the switch.

3.1.2 Behavior with Per-VC Queues

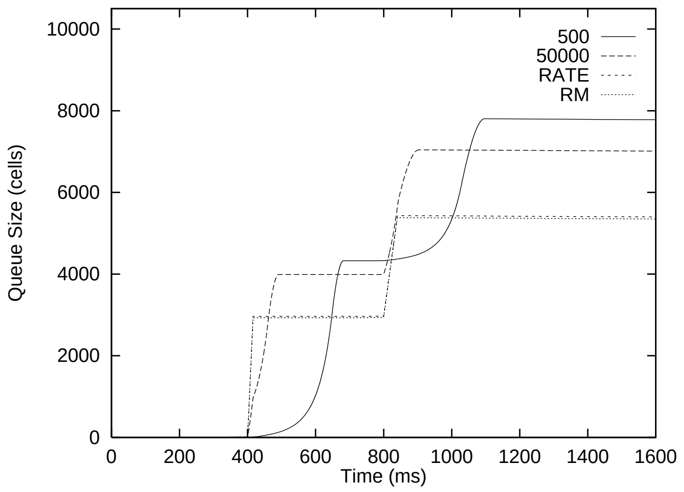


Figure 4: Queue size for different ICR with Per-VC queues, $RIF = 1/512$, and Nrm = 32

Traditional perception has been that per-VC queueing provides isolation and therefore results in generally better queue and delay characteristics. However, we observe one characteristic of per-VC queueing here, that results in larger aggregate queues at the switch. When a cell (let us consider it to be the "marked" arrival) arrives at a non-empty queue subsequent arrivals to different queue can cause additional delay for the marked arrival before it is served. This behavior is similar to that observed in [5]. In our situation, the additional delay experienced by RM cells of an existing long VC which already has a reasonable queue at a switch is of concern. Since the queues are served in round-robin fashion, this RM cell (of the long VC) has to wait it's turn to be served. If in the meantime, a short VC starts up, and rapidly ramps up its rate, it contributes to the delay of the long VC's RM cell. This increased feedback delay (in comparison to FIFO queues) results in additional queueing (from 6500 cells for FIFO to about 7800 cells for per-VC queues). In fact, the effect of a smaller ICR on the growth of the queue is even more with per-VC queues, as seen in Figure 4. What is interesting also is that the difference in the queue sizes with the "RATE" and "RM" cases nearly disappears. This is because the additional delay in the "RATE" option waiting for the first RM cell to perform the allocation is negligible compared to the additional feedback delay introduced for the long VC because of the effect of "future" arrivals with per-VC queueing.

Thus, while per-VC queueing is typically expected to provide much better queue and delay behavior, it is not always the case. To get back the desirable characteristic of per-VC queueing, we see the need to introduce additional mechanisms that manage the queue, and keep it small.

3.2 Impact of Rate Increase Factor

Similar to the situation with a small ICR, a small RIF also leads to a substantial queue. An RIF of 1/512 results in a queue of 6500 cells. Increasing RIF to 1 reduces the queue to 5000 cells. We believe that as long as ICR is small compared to the final max-min fair rate of the source, larger values of RIF result in a smaller eventual queue build up. Figure 5 shows the behavior with an ICR of 500 cells/sec., with FIFO queues for varying values of RIF. The queue reduces slightly with the "RATE" option, compared with the ATM Forum policy with an RIF of 1. When using the "RM" option, the queue drops down further, to almost 4000 cells.

We observe a similar behavior for the queue with per-VC queueing as well. The aggregate queue is generally larger with per-VC queueing.

3.3 Dynamic changes to available bandwidth

One of the major concerns with end-to-end rate control mechanisms is the responsiveness to dynamic changes in the available bandwidth. When the available bandwidth to the ABR service class reduces, it takes a certain amount of time for the sources to react to the reduction in bandwidth (the switch has to perform the reallocation as subsequent RM cells arrive, and then feedback reaches the sources). During this period, the aggregate rate of the ABR VCs may exceed the capacity, resulting in queueing at the bottleneck. The

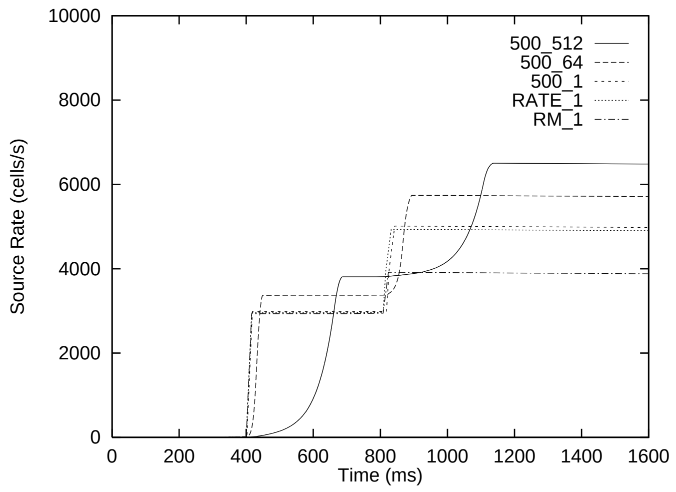


Figure 5: Queue size for different RIF: FIFO

larger the amount of bandwidth reduction, the higher the queue buildup, potentially leading to cell loss.

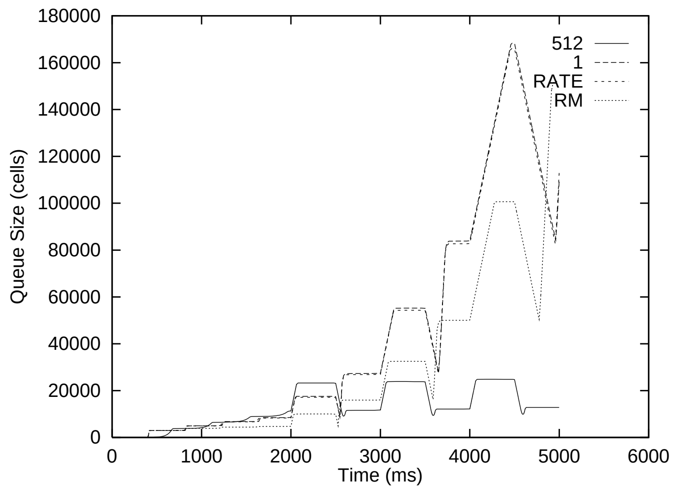


Figure 6: Queue size for dynamic bandwidth changes: FIFO

The model we use for change in the link bandwidth is for a constant bit rate (CBR) VC to turn ON and OFF, periodically. We first have a set of four VCs using the ABR service arrive one after the other. At about 2 seconds, we introduce the CBR VC, which takes 50 % of the bandwidth away from the existing VCs. This results in a queue buildup. The CBR VC remains ON for 500 milliseconds, and then is OFF for another 500 milliseconds. This ON/OFF pattern repeats for the CBR VC. We observe from Figure 6 that the queue build up can be substantial. Using a small initial cell rate of 500 cells/sec. and a conservative increase factor (RIF) of 1/512, results in a queue buildup to about 20K cells each time the CBR VC turns ON, and drains back to about 10K cells when it goes OFF. However, when we use an RIF of 1, there is a substantial queue buildup, reaching almost 180K cells. Using the "RATE" and "RM" options, (with RIF= 1), the queue buildup is almost identical to the case with an RIF of 1. When the link bandwidth changes, this has less impact for the "RM" option also, because the "early warning" is provided only when ABR VCs startup. It is clear that the large RIF causes a substantial impact, resulting in the queue buildup. The behavior with per-VC queueing is similar.

3.4 Scalability - Behavior with a Large Number of VCs

In Figure 7, we examine the effect of having a large number of VCs. A total of 500 VCs startup, each starting up in a staggered manner, 20 milliseconds apart. The first 250 VCs that startup are "long VCs". Then, the next 250 VCs that startup are "short VCs". We see from the figure that there is a dramatic queue buildup, both with FIFO queues as well as with per-VC queues. With FIFO queues, and an initial rate of 500 cells/second, and an increase factor, RIF, of 1/512, the queue builds up to about 400,000 cells. When RIF is increased to 1, the queue builds up to a little over 1.1 Million cells. With the "RATE" option, the queue builds up to nearly 1.6 Million cells. The behavior with per-VC queues appears to be somewhat better. The "RM" option on the other hand has a substantially smaller queue buildup. The dramatic reduction in the queue is due to the "early-warning" provided to the other VCs when a new VC starts up. But the scale is misleading: even for the "RM" option, the queue buildup is relatively large with both FIFO and per-VC queues (about 12000 cells for FIFO and 45500 cells for per-VC queues).

We believe that it is essential that a queue reduction mechanism be incorporated to complement even the best rate allocation mechanism that is used in the switches, to reduce the queue buildup in all of the scenarios we have observed here.

4 Algorithm for Queue Management

We have seen significant queue buildups because of feedback delay, presence of higher priority traffic, and varying transient situations. This suggests a need of a queue management mechanism that works in a complementary fashion with the rate allocation mechanism. In the following sections we propose such a mechanism and show that the queue buildup can be dramatically reduced in all of the scenarios we have observed up to now.

The hop-by-hop mechanism in [10] adjusts the upstream node's rate in response to queue occupancy information. Its goal is to maintain a target buffer-occupancy (set-point) at each node. The scheme takes advantage of per-flow queueing and information on the service rate seen by the flow (VC) to determine the sending rate at the upstream switch. With a FIFO queues, it is difficult to know the contribution to the queue by each flow (that per-VC queueing provides), and the task of managing the queue becomes more difficult. Another alternative is to allocate only a portion of the actual available link capacity to the requesting VCs, and maintaining a reserve for draining the queue [7]. Firstly, the capacity available is potentially changing frequently, and even maintaining a reserve does not guarantee feasibility at all times. Secondly, we believe a goal of utilizing all of the available bandwidth is a desirable one, especially when there is little or no queueing in the network.

This suggests the need for us to look at an alternative mechanism, where the algorithm for queue reduction is more "centralized". By centralized, we mean that the reduction needed for a given queue is known and controlled at that queue. The overall algorithm is still distributed, in that each queue is independently controlled at that queueing point. The control of the transmission rates is still the same as with the distributed explicit rate control mechanism. An additional attractiveness is that there is no change required to the current specification for the ATM Forum specified source-destination policies [14].

4.1 Design Goals

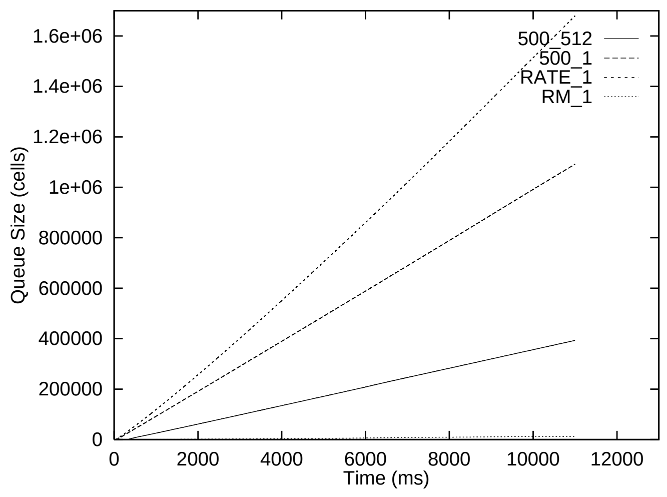
We manage the queue at an intermediate switch instead of at the end system, since this is where the queue is built up. The broad goals that we have for the algorithm to manage queue are the following:

- Allow sources to start-up aggressively. We believe it is desirable to allow sources to start transmitting at a reasonable rate (we interpret this to be the max-min fair rate) close to the steady-state value as soon as possible after connection setup.
- Allow sources to aggressively ramp-up to the newly allocated rate. When a source is allowed to increase from its current rate, it is desirable to allow the source to quickly increase its sending rate to the new value.
- Drain queues as quickly as possible when they build-up, while minimizing link under-utilization, and avoid oscillations in the source rate and the queue size while draining the queue.
- Maintain Max-Min Fair share allocation consistently, even during the periods when the queue is being drained.

4.2 Details of the Algorithm

We use the concept of a target set-point [10] for the queue size at an individual link, above which we begin to perform the function required to reduce the queue. The queue reduction is achieved by modifying the proportion of the link capacity available to be allocated among the competing sources. Reducing the capacity available for all ABR VCs results in our causing a reduction in the allocations only for those VCs that are bottlenecked at this link. The actual queue we see does not reflect the number of cells we should drain once draining starts taking effect. If we decide how much capacity to use based solely on the size of actual queue, it causes under-utilization, oscillations of the queue or takes a very long time to drain the queue because of a very conservative setting of the parameters. This distinguishes our work from [7].

The approach we use exploits the fact that all the sources participate in the algorithm that maintains max-min fair rates, so that the aggregate rate remains "feasible" (i.e., does not exceed the capacity of any resource in the network). Thus, when the queue does build up, we try to keep the reduction in the rates for the VCs to be relatively small. This is to minimize underutilization, and excessive oscillation of the rates. Since the reduction of the queue can only



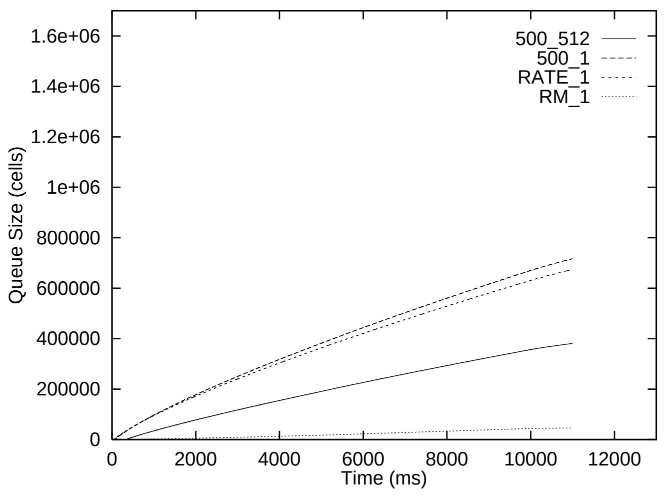


Figure 7: Queue size for 500 VCs: FIFO (left) and Per-VC queues (right)

begin when the feedback reaches the source, the dominant parameter for reducing the queue is the feedback delay. A large reduction of the capacity of the resource for the purposes of reducing the queue build up does not help as much, especially in wide-area networks, and in situations where the feedback delay is widely different. As long as the number of cells queued is not unreasonably large (which would cause too much delay and possibly cell loss), the gain in reducing the capacity by a large fraction does not provide substantial additional help.

4.2.1 Smooth Reduction

Once the queue builds up, we want to reduce the capacity used for all ABR VCs, so that the aggregate arrival rate will be reduced. The first thing to decide is how much capacity to reduce, given the size of the queue. We want the shape of the rate reduction function to provide a "smooth" reduction in the capacity, as a function of the queue size. As the queue size rises substantially above the *set-point*, the reduction in the capacity increases more rapidly. Finally, when the queue size reaches a multiple of the set-point, the amount of reduction in the capacity reaches a ceiling. We choose a conservative ceiling so as to ensure that not too substantial a fraction of the bandwidth is utilized for the reduction of the queue.

Let C be the total capacity used for all ABR VCs. The amount of reduction in the capacity is given by the following function:

$$R = a(x-S)^2 \; + \; b(x-S) + c \qquad (2)$$

The capacity $C - R$ is the amount allocated to all the flows making a request. Let M be the maximum value allowed for the rate reduction R. S is the set-point for the queue, and x is the instantaneous queue size at the link. a, b and c are parameters for the equation solved using the following three points:

$$(x, R) = \{(S,\ 0),\ (2S,\ M/4),\ (3S,\ M)\}$$

on the curve (Figure 8). The motivation is to keep a smooth reduction of the capacity just around the set-point, S, and have a more rapid reduction as the queue builds up.

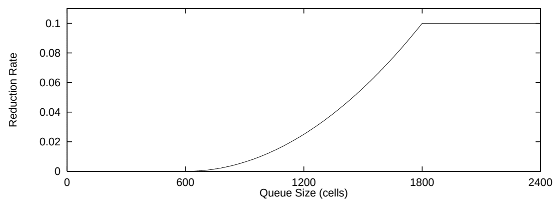


Figure 8: Rate reduction function, for a set-point, $S = 600$ cells.

S was chosen to be 600 cells in our experiments, and we chose to bound the total capacity reduced for queue reduction, M, to be 10%. Given the amount of rate reduction as a function of the queue size using the quadratic function in equation (2), we apply this rate reduction for a time period until the queue drains.

It is possible to use other reduction functions, with a different shape for the reduction in the capacity, compared to the one shown in Figure 8. For example, we also used cubic and exponential functions. We observed that, by using a smoother function when the queue size is close to setpoint, it helps avoid oscillations. But, the queue drained excessively slowly. Reducing the interval over which the queue is drained (changing from $(S, 3S)$ to $(2S, 3S)$) increases the oscillations of the queue. An initial choice of the setpoint to be 600 cells was based on having a reasonable target of 2 milliseconds worth of delay contributed per hop, and buffering a small number of IP packets (9K bytes each) before impacting the capacity on a $155Mb/s$ link.

4.2.2 Virtual Queue

When we begin to drain the queue, the current size of the queue is not an accurate estimate of the aggregate buildup over time of the difference between the service rate at the resource and the arrival rates from the sources. Basing the reduction on the instantaneous queue size may result in oscillations of both the queue and the source rates. This is particularly true in the presence of large feedback delays. If the rate reduction is applied until the queue completely drains, the feedback delay to the source will result in under-utilization of the link subsequently, when the aggregate rate of the sources becomes less than the capacity for an additional round-trip time. This effect in the oscillation of the queue size has been examined using differential equations by [1]. The reduction of the rate of the sources needs to be implemented such that we do not overcorrect and experience underutilization of the link.

To enable draining the queue quickly without causing under-utilization, we introduce a concept of a "virtual queue" by tracking the size of queue that has to be drained, but which has not yet taken effect on the real queue. The virtual queue is the difference between the maximum queue achieved in the current "regeneration cycle" and the aggregate size of the queue that has been reduced so far by the rate reduction mechanism during this cycle. This virtual queue is used to determine how much longer the rate reduction has to be applied. A regeneration cycle for the queue length is similar to that discussed in [11]. The start and end points of the regeneration cycle are when the queue transitions the set point, S (and we also start a new cycle when the actual queue exceeds the virtual queue).

We reduce the allocated rate on an individual VC basis. VCs that are not bottlenecked at this switch are allocated a rate lower than the rate for bottlenecked VCs. Although they may contribute to queueing on a transient basis (due to jitter and other events that may cause cell bunching), the rate reduction mechanism does not (and may not be able to) address the effects of these transients. Thus, to reduce the queue built up, we control the allocations of the VCs bottlenecked at this link. Since all the bottlenecked VCs at a link have an equal share, we only need to examine a simple boolean state variable of whether the VC is bottlenecked or not.

The state maintained at each link is the following:

- On a per port (p) basis, we maintain the following variables: the maximum queue seen in this regeneration cycle,($p.max\_q$); the amount of capacity ($p.reducing\_cap$) reduced for the purposes of queue reduction, the size of queue drained so far ($p.reduced\_q$), and the queue length at the time an RM cell was received on the port ($p.prev\_q$).
- On a per VC (vc) basis, we maintain the time the last RM cell was received from that VC ($vc.prev\_rm\_time$,t_{vc}), and the bandwidth reduction that this VC has contributed ($vc.reducing\_cap$,vc_r). Thus, we can track the total capacity reduction for the link over the number of bottlenecked VCs using the link.

At any time t when a RM cell is received, if the VC is bottlenecked by the link, the number of cells being reduced (or the size of virtual queue being reduced) between this and the previous RM cells for this VC is:

$$\text{vc.q\_reduced}(t) = vc_r * (t - t_{vc}) \tag{3}$$

The instantaneous queue size at a link is an integration over time of the difference between the input and the output rates, $r_i(t)$ and $r_o(t)$ as seen at the link. That is,

$$x(t_1) \; = \; \int_{t_0}^{t_1} (r_i(t) \; - \; r_o(t))\, dt \tag{4}$$

The amount of rate reduction therefore can also be determined knowing the service rate and the virtual queue size. Since the queue is an integrator of the difference in the rates, the relevant information is the largest queue size achieved ($p.max\_q$) during this regeneration cycle, which has to be drained. Let the capacity of the link be C, and the reduced capacity allocated amongst all the bottlenecked VCs be $c(t)$ (which is the value of R in equation (2) as a function of time). Let us assume that the queue crossed the value of "set-point", S, at time t_i. If t_j is the time until which the rate is being reduced, then the amount of queue reduced, $Q_r(t_j)$,

$$Q_r(t_j) \; = \; \int_{t_i}^{t_j} (C - c(t))\, dt \tag{5}$$

The arrival of RM cells from each source gives us the necessary timing information that we may exploit to determine the amount of "queue-reduction" we have achieved by (5). The amount of reduction of the queue contributed by individual VCs is maintained at the switch by knowing the time since the last RM cell arrived for that VC, according to equation (3).

We seek to determine, dynamically, the time t_j at which the queue reduction has to stop in a given regeneration cycle. Simplistically, t_j occurs when the total amount of queue reduction achieved, Q_r (by equation (5)) has reached $Q_r = (p.max\_q - S)$. We stop the reduction when $(p.max\_q - Q_r) \leq S$. However, this ignores the fact that there is feedback delay involved, and waiting till this amount of the accumulated queue is drained is inadequate. There is still one half-round trip time's (RTT) worth of cells sent at the source's incorrect rate, to be accounted for. These cells were transmitted from the source from the time when the switch stopped applying the rate reduction up to one half RTT. Thus, the queue built up due to this excess rate has to be drained. We apply a correction factor to reduce the queue for a slightly longer time period, t_k.

A simple approximation for t_k we have chosen, determines the current buffer size, b_{rem} at the time t_j, when the queue reduction has reached the value $(p.max\_q - S)$. This b_{rem} is now used as the amount of queue to be further reduced. We then iteratively use equation (5) to determine how much further we should apply the queue reduction. Since the queue size that is now used to determine the amount of rate reduced is now relatively small, the amount of reduction in the allocated capacity is also relatively small. There is a more "gradual" reduction in the queue, but this is not harmful, since we are operating in the region where the amount of queue built up is also small.

4.2.3 Slow Recovery

When the queue has been reduced to the setpoint, the bandwidth has to be released to the existing VCs. In a distributed rate allocation scheme that tries to take advantage of unused capacity by other VCs, allocation decisions are based on what other VCs are currently using. However, with the presence of a large number of VCs, multiple existing VCs may arrive at the same conclusion, which leads to transient over-allocation. Oscillations in the queue size with therefore occur. To reduce the amount of oscillations that may result (even if the amount of bandwidth released is relatively small), we recover the capacity of a VC to its max-min fair share rate gradually by allocating a share of the increased capacity to each of the bottlenecked VCs. The rate increase allowed for a VC is based on the maximum reduction in the rate we allowed and the number of bottlenecked VCs. When the allocation to a VC is increased, a ceiling function is applied to this increase. for an interval of up to one maximum round-trip time interval.

The pseudo code for the entire mechanism is shown in Figure 14.

5 Simulation Results

In this section, we examine the ability of our queue management algorithms to control the queue size for the various scenarios we examined earlier in Section 3, with the same options for the startup and increase factors at the sources.

5.1 Behavior with Variation in ICR and RIF

Figure 9 shows the behavior of the queue at the 1st switch in the topology shown in Figure 1, when a "long VC" starts at time $t = 0$, and two "short VCs" arrive at 400 and 800 milliseconds respectively (all with an $RIF = 1/512$). Both the "RATE" and "RM" options result in a queue buildup initially to 3000 cells, just as we observed for this case without rate reduction. However, we are able to rapidly bring down the queue to our target setpoint of 600 cells in 5 to 6 round-trip times. There is no continuing buildup of the queue as we observed in the cases without a queue reduction mechanism (Section 3.1), where the queue size built up to over 7000 cells even with just these 3 VCs (because of the arrival of the two short VCs). A smaller ICR results in a slightly smaller initial peak, of about 2400 cells. When the third VC comes on at 800 milliseconds, since there are more VCs now, there is a smaller queue buildup due to this incoming VC with all the options. The differences between the various options for the peak queue size is not as significant; the primary difference with a small ICR is the additional time it takes for the queue to buildup. We also observe that the difference in the queue behavior between FIFO queues and per-VC queues is not that substantial. There is a small additional queue with per-VC queues when the ICR is preset (50000 or 500) compared to using FIFOs.

The behavior of the queue with different values of RIF (going from 1/512 to 1/16) is also similar to what we observe in Figure 9.The peak buildup is similar for larger values of RIF, even up to when it is 1, for a fixed $ICR = 500$ cells/sec. and even with the "RATE" and "RM" options (which use an $RIF = 1$, but may have a larger value of ICR). Having a small RIF (1/512) causes the queue to grow slower, and therefore gives our queue-reduction mechanisms to act and bring down the queue (we exclude the figures due to lack of space).

5.2 Effect of Changes to Available Bandwidth

We examine the ability of our proposed queue management mechanisms to control the queue when the available bandwidth changes (e.g., reflecting a CBR VC going ON and OFF), corresponding to the scenario in Section 3.3. When the CBR VC turns ON, the peak queue build up is about 1800 cells ($3{*}S$), considerably smaller than the nearly 170,000 cells we observed without any queue reduction mechanism in place. The case with FIFO queues is shown in Figure 10. When the CBR VC turns OFF, there is a potential for underutilization until the other VCs (especially those with a long RTT) ramp up their source rates. Only the most conservative option of an ICR =500 and $RIF = 1/512$ results in a small period over which the queue goes down to 0, for about 2 round-trip times. The "RATE" and "RM" options do not result in any under-utilization because the queue size drops down to about 250 cells. At this point, the rate of the ABR VCs catch up due to the large increase factor of 1. However, the large RIF does not hurt because we are able to bring the queue back down reasonably rapidly. With per-VC queues (figure excluded), the only difference is with the "RATE" option where the queue drops down to nearly 0 for a brief period, less than a round trip time. Once we bring the queue size down, there is little difference between FIFO and per-VC queueing as far as the queue buildup is concerned.

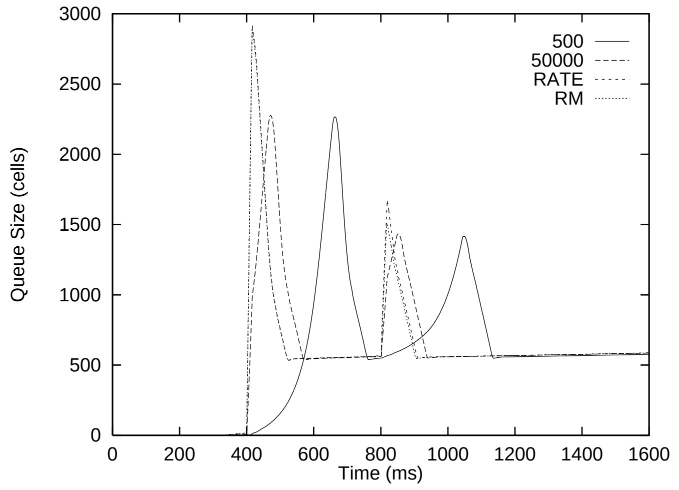
5.3 Scalability: Behavior with a Large Number of VCs

To examine the ability of our queue management mechanisms to truly scale to very large number of VCs, we examined the performance with 500 VCs. The difficulty with a large number of VCs is that the rate for an individual VC is so small that small perturbations to each source's rate beyond the max-min fair share results in considerable queueing. We showed in Figure 7 that without a mechanism to reduce the queue, the queue build up is in fact substantial. There are 250 "long VCs" arriving initially, each staggered 20 milliseconds apart, and then there are another 250 "short VCs" that arrive subsequently, again spaced 20 milliseconds apart. The configuration used is the simple, two-switch topology shown in Figure 1.

The behavior of the queue with our rate reduction mechanism to manage the queue is shown in Figure 11. One of the explicit enhancements we had to include for rate reduction as a result of the large-scale configuration was to recognize that the time between RM cells for an individual VC was larger than the round-trip time. Furthermore, the time estimate for the round-trip delay observed at connection setup, especially for per-VC queues is significantly smaller than when data is actually flowing. Furthermore, the slow recovery described in Section 4.2.3 is critical to ensure that the queue remains manageable. The best behavior is observed with the RM option, where the queue remains close to the setpoint value of 600 cells under all circumstances. In fact, even the conventional option of having an $ICR = 500$ cells/sec and $RIF = 1/512$ shows good behavior (Figure 11).

5.4 Performance with a General Topology

So far, we have focused on a relatively simple topology that emphasized the difference between the feedback delays for the sources. In this subsection we demonstrate that our mechanisms maintain max-min fairness even in a more general topology, where there are multiple bottlenecks. The



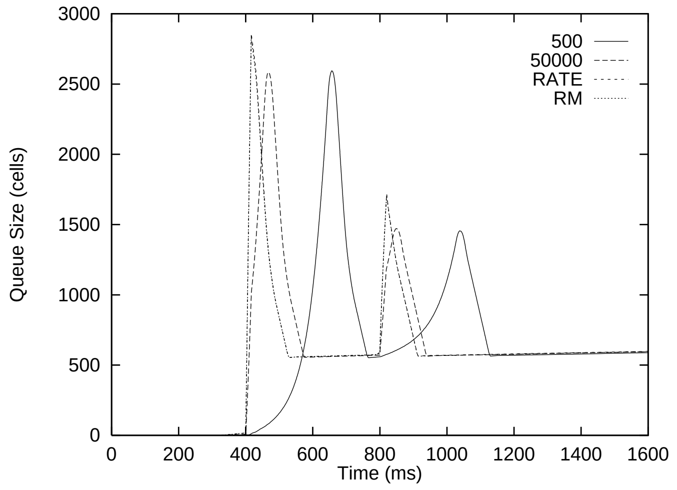
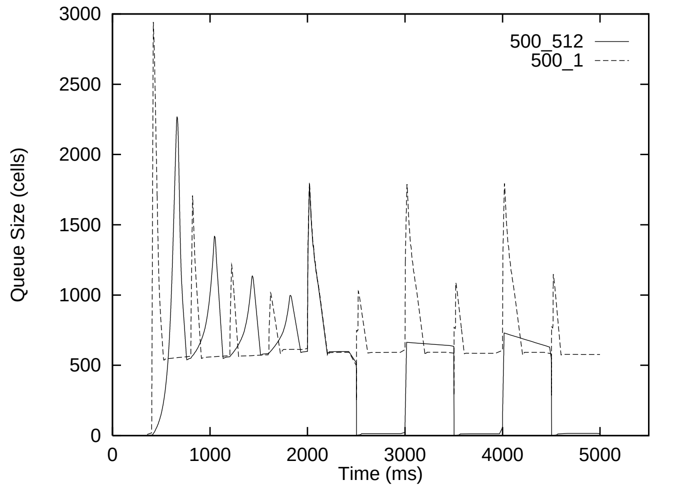


Figure 9: Reduced queue size for different ICR: $RIF = 1/512$: FIFO (left), Per-VC queues (right)



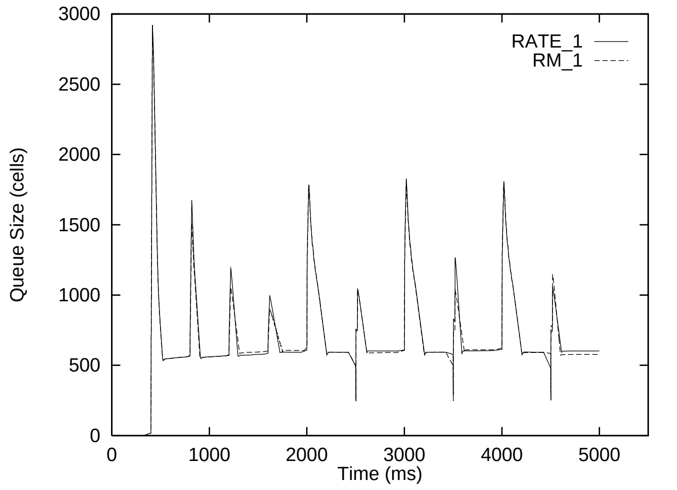


Figure 10: Reduced queue size with dynamic bandwidth with FIFO queues

topology is shown in Figure 12.

There are 8 sources that start up in a staggered manner (the start times for each of the sources is shown in the figure), and each source goes through different sets of resources. The link bandwidths between the switches are different (shown in the figure). The link bandwidth from the sources to the switches are all 155 Mbits/sec. There are 3 sources that have long feedback delays (the long links are marked "L" $= 8$ milliseconds), and the other 5 sources have short feedback delays (marked "S"$= 0.5$ μseconds). The target max-min rates for VC 1 and VC 6 are approximately 20K cells/sec. (8.333 Mbps) and 40K cells/sec. (16.667 Mbps) respectively. Figure 13 shows the behavior of the source rates for VC 1 and 6. VC 1 starts at time $t = 0$, and VC 6 starts at time $t = 900$ milliseconds. We observe that the sources achieve their steady state max-min fair share rates subsequent to the last VC starting up (at time $t = 1500$ milliseconds). Although there is a small reduction of the rates due to the queue buildup at both switch 3 and switch 6, the observation from the figure is that we retain max-min fair rates for the VCs throughout the process of different sources starting up (when the target fair rate for a VC changes).

6 Conclusions

Explicit rate mechanisms used for allocation of link capacity to sources in an end-end rate based congestion control scheme have the desirable property of adjusting source rates to ensure feasibility. However, on a transient basis, we find that the rate is exceeded, causing queues to buildup. When sources place a persistent load and the rate allocation scheme attempts to fully allocate the available capacity, this queue does not drain quickly, or naturally. Furthermore, even a small difference in the aggregate source rate above the link capacity can cause a quick, substantial buildup of the queue. This is especially true when the feedback delays to sources are large and widely varying. It is very desirable to allow sources to opportunistically utilize available bandwidth, since once it is left unused, it is lost forever. This encourages sources to start up aggressively and to ramp up their rate to the final value as fast as possible. All of this results in large queues.

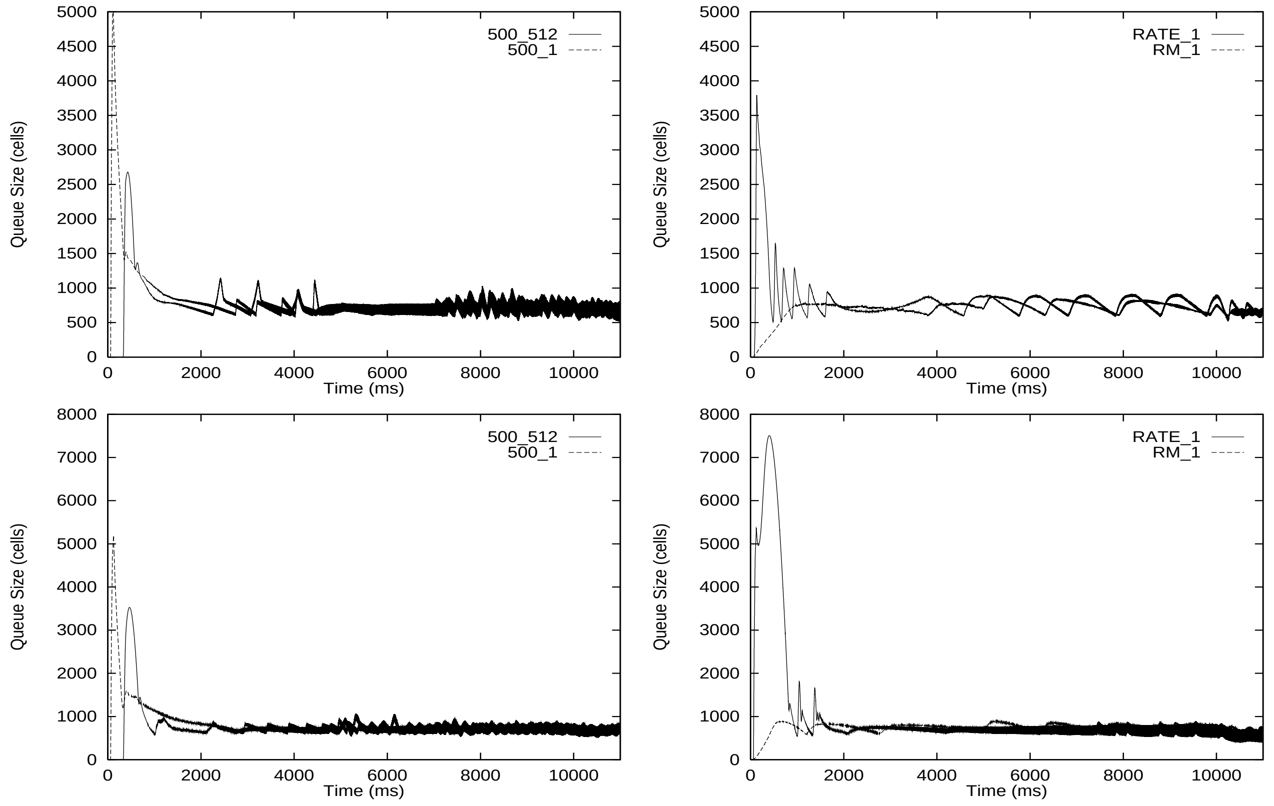


Figure 11: Reduced queue size for 500 VCs: FIFO (top) and Per-VC queues (bottom)

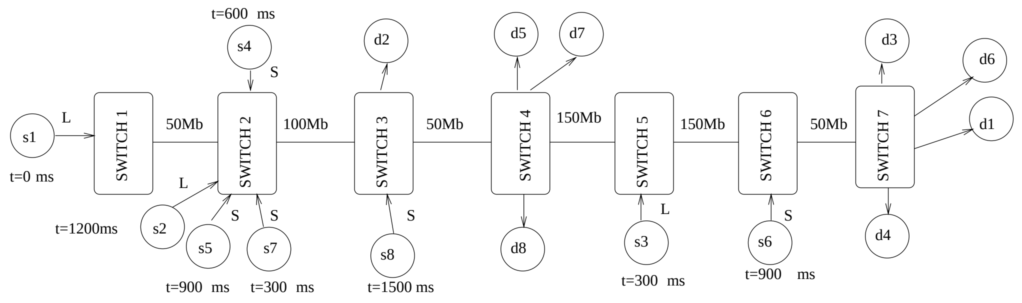


Figure 12: GFC configuration

We showed that having a small ICR and/or small RIF does not always help in keeping the queues small. The queue buildup when the available capacity for ABR changes is also substantial. We also showed that when we have a large number of VCs active (500 VCs), the queue buildup is clearly unreasonable (over a million cells). We also showed that in some cases, the use of per-VC queues in fact results in a higher aggregate queue in comparison to FIFO queueing. With per-VC queues we found that the delay to RM cells caused by future arrival of cells to another competing VC resulted in a higher queue, especially when the increase parameter, RIF was small. Thus, we found that it is essential to have a queue management mechanism in addition to a good rate allocation mechanism in the switch.

This paper described a scheme for managing the queue buildup at switches even under the most aggressive behavior patterns of the sources. The scheme operates in the context of the Available Bit Rate (ABR) congestion scheme specified by the ATM Forum, with switches using the explicit rate option (which we believe has the most promise of maintaining rates close to the feasible value). The mechanism is entirely compatible with the currently specified source and destination policies of the ABR specification.

Switches observe the buildup of the queue (which acts as an integrator of the excess rate over time), and use it to reduce the portion of the link capacity that is allocated to the sources bottlenecked at this link. Rather than the instantaneous queue, we use the concept of a "virtual" queue, which tracks the amount of queue reduced by reducing the allocation to bottlenecked VCs. We do this by taking advantage of the natural timing of "resource management" (RM) cells transmitted by sources, which allows us to measure how much of the buildup queue will be reduced by our reducing the allocation of the capacity to a VC, even though that

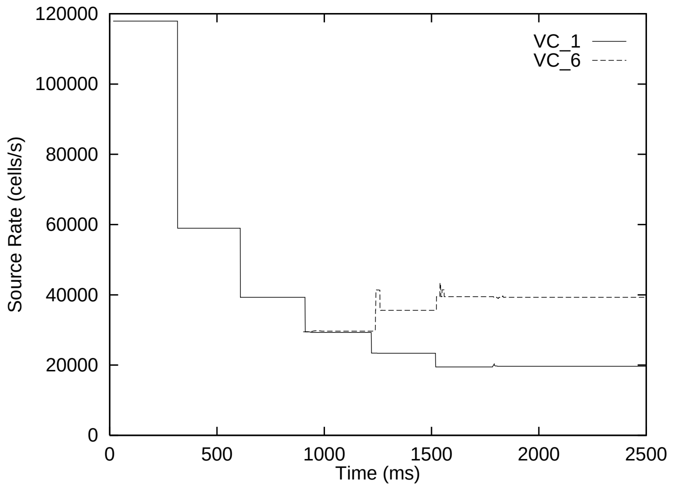


Figure 13: Reduced Rate for two VCs

reduction may not be reflected as yet in the instantaneous queue.

The mechanism takes advantage of the connection setup message to both compute a max-min fair rate for a VC, as well as allocate the rate to the incoming VC so that it serves as an "early warning" for existing VCs (the "RM option"). We showed that the queue reduction mechanism maintains the queue at the resource close to the "setpoint" we chose (600 cells, or about 3 IP packets of 9K), even when the source policy has the most aggressive behavior: the initial cell rate, ICR, is the final max-min fair share of the VC and the rate increase factor, RIF, is the maximum allowed, 1. We showed that even when we have a higher priority CBR VC take away half of the link bandwidth, the queue buildup is not significant. In all of the cases, there is little or no underutilization of the bottleneck, which is very desirable as well. To examine the ability of the mechanism to scale up to large numbers of VCs, we showed that the performance is excellent even when we go up from a few (less than 10) VCs, up to 500 VCs, in a demanding configuration which has 4 orders of magnitude difference in feedback delays. Without the queue reduction mechanism, the queue buildup can in fact reach 10^6 cells with 500 VCs. With the queue reduction mechanism, the queue remains close to 600 cells - a significant improvement.

The scheme is elegant in that it is simple. It is scalable, and is as responsive as can be expected: within the constraints of the feedback delay. One of the most important characteristics of the queue reduction mechanism is that it maintains max-min fairness, even when the queue is being drained. With the distinct queue draining mechanisms in place, the differences in the behavior of the queue between FIFO queueing and per-VC queues also reduces, which we believe is very desirable.

Acknowledgements We would like to sincerely thank Peter Steenkiste of CMU and Chuck Kalmanek of AT&T Research for kindly supporting this work. Thanks also to Partho Mishra for initial discussions on the solution approach.

References

[1] Bolot, J.-C., Udaya Shankar, A., "Analysis of a Fluid Approximation to flow control dynamics", Proc. Infocom '92, Florence, Italy, May 1992.

[2] Bertsekas, D., Gallager, R., Data Networks, ch. 6, Prentice Hall, Englwood Cliffs, N.J., 1992.

[3] Charny, A., Clark, D.D., Jain R., "Congestion Control With Explicit Rate Indication", Proc. ICC'95, June 1995.

[4] Charny A., Ramakrishnan, K.K., and Lauck, A., "Time Scale Analysis and Scalability Issues for Explicit Rate Allocation in ATM Networks" IEEE/ACM Transactions on Networking, August 1996.

[5] Clark, D., Shenker, S., Zhang, L., "Supporting real-time applications in an integrated services packet network: architecture and mechanism", Proceedings of ACM Sigcomm '92, pp., Sept. 1992.

[6] Jacobson, V., "Congestion Avoidance and Control", Proceedings of of ACM Sigcomm '88, pp.314-329, Aug. 1988.

[7] Jain, R., Kalyanaraman, S., Vishwanathan, R., Goyal, R., "A Sample Switch Algorithm", AF-TM 95-0178R1, ATM Forum Traffic Management Working Group, Feb. 1995.

[8] Jain, R., Kalyanaraman, S., Goyal, R., Fahmy, S., Vishwanathan, R., "ERICA Switch Algorithm: A Complete Description," AF-TM 96-11721, ATM Forum Traffic Management Working Group, August 1996.

[9] Kalampoukas, L., Varma A., Ramakrishnan, K.K. "An Efficient Rate Allocation Algorithm for ATM Networks Providing Max-Min Fairness", Proceedings of 6th IFIP International Conf. on High Performance Networking, HPN '95, Palma De Mallorca, Balearic Islands, Spain, Sept. 11-15, 1995.

[10] P. Mishra., H. Kanakia "A Hop by Hop Rate-based Congestion Control Scheme", Proceedings of ACM SIGCOMM '92 Conference, 1992.

[11] K. K. Ramakrishnan, Raj Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer", Proceedings of of ACM Sigcomm '88, Aug. 1988.

[12] K.K.Ramakrishnan, Raj Jain, Dah-Ming Chiu. "Congestion Avoidance in Computer Networks With a Connectionless Network Layer. Part IV: A Selective Binary Feedback Scheme for General Topologies", DEC-TR-510, Digital Equipment Corporation, 1987.

[13] L. Roberts, "Enchanced PRCA (Proportional Rate Control Algorithm)", AF-TM 94-0735R1, August 1994.

[14] Sathaye, S.S., "ATM Forum Traffic Management Specification Version 4.0" (draft), AF-TM 95-0013R10, ATM Forum Traffic Management Working Group, Feb., 1996.

[15] Wilder, R., Ramakrishnan, K. K., Mankin, A., "Dynamics of Congestion Control and Avoidance of Two-Way Traffic in an OSI Testbed", ACM Computer Communication Review, Vol. 21, No.2, pp.43-58, April 1991.

[16] Zhang, L., Shenker, S., Clark, D., "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", Proceedings of ACM Sigcomm '91, pp.133-147, Sept. 1991.

```
Rate_Reduction($p$ : PORT, $vc$ : VC)
    $cap \leftarrow$ ABR_capacity($p$)                          /* get total capacity used for ABR VCs */
    $q \leftarrow$ queue_size($p$)                              /* get the queue size of the out port */
    $t \leftarrow$ Current_Time()                               /* get the current system time */
    **if** ($q < S$) **then**                                   /* if the queue size is smaller than the setpoint */
        **if** ($RTT\_clock \leq 0$) **then**                   /* if the clock for an extra maximal RTT is not started */
            $RTT\_clock \leftarrow 1$                           /* start the extra maximal RTT clock */
        **else if** ($(t - p.maxRTT) > RTT\_clock$) **then**    /* if maxRTT after queue being drained */
            $vc.reducing\_cap \leftarrow 0$                     /* clear the capacity reduced for $vc$ */
            $RTT\_clock \leftarrow 0$                           /* stop ticking */
            **end if**
        **end if**
        $p.reducing\_cap \leftarrow 0$                          /* capacity being reduced for the port */
        $p.reduced\_q \leftarrow 0$                             /* total queue size reduced so far */
        $p.max\_queue,\ p.prev\_q \leftarrow q$                 /* the maximal queue size and the previous queue size */
        $new\_cap \leftarrow cap$                               /* use the full capacity */
    **else if** (bottleneck($vc, p$) $= 0$) **then**            /* if the VC is not bottlenecked by the port */
        $new\_cap \leftarrow cap - p.reducing\_cap$             /* the VC does not contribute to queue build-up */
    **else**
        $rm\_intval \leftarrow t - vc.prev\_rm\_time$           /* time between this and the previous Rm cell */
        $v\_red\_q \leftarrow rm\_intval * vc.reducing\_cap$    /* size reduced for virtual queue between two RM cells */
        $p.reduced\_q \leftarrow p.reduced\_q + v\_red\_q$      /* reduced size of virtul queue by all VCs using the port */
        $r\_red\_q \leftarrow q - p.prev\_q$                    /* change of real queue */
        $v\_q \leftarrow p.max\_queue - p.reduced\_q$           /* update virtual queue size */
        **if** ($r\_red\_q > v\_red\_q$) **and** ($v\_q < q$) **then**   /* if more cells are seen from the real queue */
            $p.max\_queue \leftarrow q$                         /* update virtual queue to real queue */
            $p.reduced\_q \leftarrow 0$                         /* clear reduced queue size */
        **end if**
        **if** ($v\_q \leq S$) **then**                         /* if virtual queue is below set-point */
            $p.max\_queue \leftarrow q$                         /* update virtual queue */
            $p.reduced\_q \leftarrow 0$                         /* clear reduced queue size */
        **end if**
        $p.reducing\_cap \leftarrow$ Reduction($v\_q, cap$)     /* call quadratic reduction function */
        $vc.prev\_rm\_time \leftarrow t$                        /* update the previous RM cell time */
        $num\_b \leftarrow p.bottleneck$                        /* update reducing capacity for the vc */
        $vc.reducing\_cap \leftarrow p.reducing\_cap / num\_b$  /* get the number of bottlenecked VCs */
        $p.prev\_q \leftarrow q$                                /* update the previous queue size */
        $new\_cap \leftarrow cap - p.reducing\_cap$             /* update reducing capacity for the vc */
    **end if**
    $vc.ER \leftarrow$ MaxMin_FairShare($p, vc, new\_cap$)      /* do maxmin fair share rate allocation */
    $inc1 \leftarrow vc.reducing\_cap$                          /* get the capacity reduced by this VC */
    $n \leftarrow$ MAX($1, p.max\_RTT / vc.rm\_intval$)         /* steps to complete rate recovery in maxRTT */
    $inc2 \leftarrow$ MAX_RED$/(num\_b * n)$                    /* the maximal recovery rate in each step */
    $max\_ER \leftarrow vc.prev\_ER +$ MIN($inc1, inc2$)        /* get the maximal allowed ER value */
    **if** ($vc.ER > max\_ER$) **then**                         /* if allocated rate is larger than allowed maximum */
        $vc.ER \leftarrow max\_ER$                              /* use the allowed maximal as new ER */
    **end if**                                                  /* get maxmin fair share */
```

Figure 14: Pseudo-code for the rate reduction algorithm.