Optimal Feedback Control for ABR Service in ATM

Paolo Narváez Kai-Yeung Siu *

Abstract

The efficient support of data traffic over ATM networks requires congestion control, whose objectives include maximizing throughput, minimizing switch buffer requirement, and attaining a fair bandwidth allocation. With available bit rate (ABR) service in ATM, congestion control is achieved by requiring data sources to adjust their rates based on the feedback from the network. The difficulties in providing effective ABR service are caused by the burstiness of data traffic, the dynamic nature of the available bandwidth, as well as the feedback delay. Using a new design methodology described in our previous paper, we present here a congestion control algorithm that is provably stable and is optimal in the sense that it has the shortest possible transient response time. Moreover, our algorithm achieves fair bandwidth allocation among contending connections and maximizes network throughput. It also delivers good performance for switches that use a FIFO queuing discipline. Essentially, the algorithm implicitly measures the round-trip delay using resource management cells, and establishes an observer (in the control theoretic sense) to control the flow of data in the network. Rigorous analysis as well as simulation results are presented to substantiate our claims.

1 Introduction

1.1 Objective

Most data applications usually require a service that allows all competing active connections to dynamically share the available bandwidth and minimizes data loss due to buffer shortage during network congestion.To satisfy the service requirement of data traffic, the ATM Forum has defined the available bit rate (ABR) service.

ABR service specifies a rate-based feedback mechanism for congestion control. The general idea of the mechanism is to adjust the input rates of sources on the basis of the current level of congestion along their VCs. A number of different rate-based congestion control schemes (e.g. [3, 4, 8, 9]) have been proposed in the literature. Depending on the specific switch mechanisms employed, different congestion control algorithms yield different transient performance.

While many proposed algorithms for ABR service have been shown to perform well empirically, their designs are mostly based on intuitive ideas and their performance characteristics have not been analyzed theoretically. Using a control-theoretic framework, we propose in our earlier work [6] a new design methodology for ABR congestion control algorithms based on the principle of separating the controller into two key components: rate reference control (RRC) and queue reference control (QRC). The RRC component computes its rate command by using measurements of the available bandwidth in each link, with the objective of adapting the input source rate to match the output available rate. On the other hand, the QRC component computes its rate command by measuring the queue level at each switch, with the goal of maintaining the queue level at a certain threshold. The desirable source rate is then equal to the sum of the QRC and RRC rate commands. The advantage of this separation principle is that it decouples the complex dynamics of the closed-loop control system into two simpler control problems which are analytically more tractable and can be solved independently.

In this paper, we propose a new algorithm that further decouples the QRC and RRC dynamics. While the algorithms in [6] have separate (QRC and RRC) components at the controller, the feedback does not maintain the separation between the rate commands determined by the two components. Conceptually, our new algorithm incorporates the feedback rate commands into two separate channels. To distinguish this new algorithm from its predecessors, we shall refer to it as bi-channel rate control (BRC). The separate channels allow for several improvements. First, BRC does not need an explicit estimate of the round-trip delay in its controller. Second, unlike its predecessors, the new controller is stable even under time-varying de-

*P. Narváez and K.-Y. Siu are with the d'Arbeloff Laboratory for Information Systems & Technology, Massachusetts Institute of Technology. Please direct all correspondences to K.-Y. Siu at Rm. 3-449C, 77 Massachusetts Ave., MIT, Cambridge, MA 02139. Email: siu@list.mit.edu. This work was supported in part by the NSF Young Investigator Awards Program (Grant # MIP-9696144) and by the Department of Energy (Grant #DE-FG03-95ER25273.)

lays. Third, it stabilizes a queue in the shortest time possible given no knowledge of the round-trip delay.

The robustness of BRC against fluctuations in the network delays also allows for improvements in the queuing discipline of the switch. One such improvement is a mechanism in which all VCs can share a single FIFO queue instead of using per-VC queuing. While the real data is stored in a FIFO queue, the controller uses a *virtual* queue [2, 11, 12] to compute the rate and queue size estimates used in the RRC and QRC.

1.2 Related works

In [1], classical control theory is used to model the closed loop dynamics of a congestion control algorithm. Using the analytical model, it addresses the problems of stability and fairness. However, as pointed out in [5], the algorithm is fairly complex and analytically intractable.

In [5], the congestion control problem is modeled by using a first-order system cascaded with a delay. The control algorithm is based on the idea of the Smith predictor [10], which is used as a tool to overcome the possible instabilities caused by the delays in the network. The Smith predictor heavily relies on knowledge of the network round-trip delay. The objective of the algorithm in [5] is to minimize cell loss due to buffer overflow. However, the algorithm only uses the queue level in determining the rate command, and without knowledge of the service rate, always results in a steady-state error with respect to the desired queue level. When the service rate changes is sufficiently high, the algorithm may lead to under-utilization of the link capacity. The major limitation of the algorithm is that it assumes that the network delay is known and remains constant. If the actual delay differs from the assumed constant delay, the controller response will exhibit oscillations and in some cases instability.

Our previous work [6] extends the work in [5] by incorporating an RRC algorithm, which constantly adapts the source rate of a VC to the bandwidth available to it. In parallel with the RRC component, our controller incorporates a QRC component, which is similar to the algorithm in [5]. We show that the QRC corrects any errors computed by the RRC component using measurements of the queue length, and thus can be viewed as an error observer in the control theoretic sense. The new theoretical insights enable us to design congestion control algorithms with various forms of controller structures that can achieve fair bandwidth allocation and high throughput with small buffer requirement, even in the presence of large delay. However, the limitation of our work in [6], as in [5], is that it still assumes a fixed known network delay. When the actual delay differs from the assumed delay, the *primal* controller structure presented in [6] will exhibit similar types of instability as in [5], whereas the *dual* controller structure in [6] will exhibit steady-state errors (but not instability) with respect to the desired queue level.

In this paper, we present a new algorithm that does not assume any knowledge of the network delay. Although it still uses the same control principle in the RRC and QRC controllers as in our previous work [6], the algorithm avoids the explicit use of the network delay in computing the desired rate command for each VC. Though the new results described in this paper draw upon the theory developed in our earlier work [6], our presentation is self-contained. However, the reader is assumed to have basic knowledge of classical control theory in order to understand our results.

Because of space limitation, we will omit detailed proofs and explanations, which can be found in the full version of this paper [7].

2 Transient response and performance

In this section, we explore some of the fundamental limits on the transient response of flow control algorithms. We will show some of the restrictions that the network itself imposes on the controller.

We can think of a communication network as a system which contains states, inputs, and disturbances. The states of this system are the various queue lengths, the number of cells in the channels, and the contents of the resource management (RM) cells traveling in the links. The inputs are the rates at which the sources are sending data. The disturbances are the switch service rates and the number of virtual channels passing through each switch. The purpose of a flow control algorithm is to compute the correct inputs to this system so as to stabilize its states (make them reach steady state) in spite of the disturbances.

If the disturbances change in an *unpredictable* manner, it will take some time for the flow control algorithm to stabilize the network. This *settling time* is the time required for the flow control algorithm to *observe* changes in the system and to *control* them. In other words, it is the time to reach steady-state operation after the last change in the disturbances. The settling time is mainly determined by the network delays.

2.1 Single node network

When the network contains only one node, we can obtain very strong bounds on the settling time. The settling time varies depending on whether the delay of the link between the source and the switch is known or not. The minimum bounds on the time needed to stabilize such systems are stated in the following theorems:

Theorem 1 For a network consisting of a single source and a single switch connected by a link with a *known* delay, the minimum time in which *any* flow control algorithm can stabilize the network after a change in the switch's service rate is **the round-trip delay time (RTD)** between the source and the switch.

Note that this lower bound can only be achieved by a flow control algorithm that can accurately measure the disturbance (switch service rate), predict its effect on the network, and request new rates instantly that will reinstate the switch's queue size. This can only be achieved if the round-trip delay is known. Otherwise, the following theorem holds:

Theorem 2 For a network consisting of a single source and a single switch connected by a link with an *unknown* delay, the minimum time in which *any* flow control algorithm can stabilize the network after a change in the switch's service rate, given that the queue does not saturate, is **twice the round-trip delay time (2 RTD)** between the source and the switch.

From a control-theoretic viewpoint, we can interpret this lower bound as the algorithm needing at least 1 RTD to *observe* the network and 1 RTD to *control* it.

3 Algorithm

3.1 Concepts

The BRC algorithm uses two independent controllers that work in parallel and deal with different aspects of the flow control problem. The RRC algorithm matches the source rate with the available bandwidth of the VC. The QRC algorithm tries to stabilize the queue size of the slowest switch.

The QRC algorithm can be seen as RRC's *error observer*. As an observer, the QRC needs to simulate the behavior of the real system. For this purpose, it uses information carried in the RM cells to simulate a hypothetical data flow.

In the multi-node case, both the RRC and QRC control are distributed among the different switches that implement the BRC algorithm. All the switches that are part of a particular VC path can contribute to controlling the data flow of that particular VC. Not every switch in the VC path is required to follow the BRC algorithm. The only thing that every switch is required to do is to transmit the backward and forward RM cells it receives. The switches that implement the full BRC algorithm will be free of congestion. Therefore, although not mandatory, every switch can benefit from implementing the algorithm in order to reduce its congestion. In this section we also assume that the switches use a per-VC queuing discipline, i.e., a separate queue is implemented for each VC.

3.2 Feedback implementation

The first mechanism that must be established for our flow control algorithm is a feedback loop that carries information around the network. This is implemented by resource management (RM) cells which are circulated continuously between the source and the destination. While in the backward direction (from the destination to the source), the RM cells are routed out-band (i.e., using a separate channel), and we assume that their propagation delay is constant. Moreover, in the backward direction, the time interval between consecutive RM cells will be denoted as Δ_{rm}. This time interval does not have to be constant, but should be small enough to ensure a fast response of the system.

When the RM cell reaches the source, it is forwarded to the destination along the same VC path in a forward direction. This time, the RM cells are routed along with the data cells of the corresponding VC. Therefore, the delay of these forward RM cells is the same as the delay of regular data cells of the same VC.

In the backward direction, each RM cell contains four fields of information in which control data can be transferred between the various switches and the source within the same VC path. The first field is used exclusively by the RRC controllers and the other three are used exclusively by the QRC controllers. The four fields are:

- RRC rate: RRC_{rm}
- QRC rate: QRC_{rm}
- Time interval: $Time_{rm}$
- QRC request: Req_{rm}

The RRC and QRC rate fields contain the rate requests used by the independent RRC and QRC algorithms, respectively. The time interval field contains the time when the next RM cell is expected to arrive. This information is used by the QRC controller to calculate its gain. The QRC request field is used to keep track of how many cells each QRC controller has requested.

At the destination, the RM fields are reinitialized as follows: RRC_{rm} is set to the maximum rate the destination can receive; QRC_{rm} and Req_{rm} are set to zero; $Time_{rm}$ is set to the time interval between the current and previous RM cell's arrival times. The first RM cell can have any value in the $Time_{rm}$ field.

Every time that an RM cell passes through a switch, the switch obtains the information contained in that cell, performs some computation, and incorporates new information in the cell.

3.3 RRC

The RRC controller is the simpler of the two controllers. For a particular VC, the RRC at each switch is constantly probing the VC's bandwidth. More formally, we can define the *available bandwidth* or *available rate* (r_j) of a VC_j (with respect to an output link of a switch) as the input rate to the switch that would keep the queue of VC_j at a constant level given the present incoming rates of the other VCs. There are various ways of calculating the available rate of a VC. Since we assume that the switch is servicing data in a round-robin per-VC queuing discipline, we can use information on the round-robin scheduling operation to perform our calculation. The following method was proposed in our earlier work [6] for estimating the available rate:

Method Over a time interval T, the switch counts the number of cells that are serviced for each VC. We will denote the number of cells serviced for channel i as x_i. The switch also counts the number of times N that it cycles around the queue round-robin during interval T. Let $\Delta = T/N$ be the time it takes to service one cell. Our estimate of the available rate ($\hat{r}_j$) for channel j can be obtained progressively with the following iteration:

$$\boxed{\begin{aligned}
T_j^0 &= [\sum_i x_i]\Delta & x_i^0 &= x_i & (1)\\
T_j^1 &= [\sum_{i\neq j} x_i^0 + N]\Delta & x_i^1 &= \min[N, \frac{T_j^1}{T_j^0} x_i^0] & (2)\\
&\vdots & &\vdots & \\
T_j^n &= [\sum_{i\neq j} x_i^{n-1} + N]\Delta & x_i^n &= \min[N, \frac{T_j^n}{T_j^0} x_i^0] & (3)
\end{aligned}}$$

$$\hat{r}_j = \frac{N}{T_j^n} \qquad (4)$$

The variable T_j^n is the nth order approximation of the time it would take to service N cells from VC_j if its queue were always non-empty, while x_i^n is the number of cells that would be serviced from $VC_{(i\neq j)}$ in time T_j^n if these VCs maintain their current incoming rates. If $x_j = N$ (the queue was always non-empty), the zeroth order iteration will give a correct result. If $x_j < N$ (at some point the queue of VC_j was empty), T_j will increase slightly at each iteration, reflecting the fact that more time would be needed to perform N round-robins if VC_j were always non-empty.

It can be shown that the above iteration always converges. The result of the iteration gives us the rate that a particular VC would use *if* its queue were always nonempty and the other VCs were to continue transmitting data at the current rate. This is equivalent to the available bandwidth that we defined above.

In order to get an exact measurement of the rate using this method, we would need an infinite number of iterations. However, in practice, only a limited number of iterations is necessary to get a good approximation. When there is a large number of VCs passing through the switch, the zeroth order iteration (1) is sufficiently accurate. Furthermore, the small error can be corrected by the QRC algorithm. In the simulations presented in section 6, we only use the first order iteration (2) to calculate the available rate without noticing any loss in performance.

By using this method (with whichever order of iteration we choose), the RRC can have an updated approximation of the available service rate of the switch for each VC. Every time a backward RM cell passes through the switch, the RRC will incorporate into the cell's command field the *minimum* of the locally calculated rate and the rate contained in the RM field RRC_{rm}. When the backward RM cell reaches the source, the RRC command field will contain the minimum available rate of all the switches on the VC path.

By itself, the RRC command can ensure that the sources are sending data at the correct *average* rate. However, this scheme by itself is unstable and cannot control the queue size. If there is an average error in the measurement of the available rate at some switch, the queue at that switch might empty out or overflow. In order to stabilize the queue size, we need to incorporate the QRC algorithm.

3.4 QRC

The QRC algorithm runs in parallel with the RRC algorithm. It uses feedback from measurements of the queue sizes to calculate its rate command. The QRC algorithm works independently from the RRC and corrects its errors. In this paper, we carry the separation of the two algorithms into the feedback process. This allows us to design a QRC algorithm which does not assume any knowledge of the round-trip delay.

The key functionality of the QRC algorithm is to estimate the *effective* queue size of a VC. The effective queue size of a VC at a switch is the total number of cells in the VC's queue plus all the cells that have been previously requested by the QRC for that VC and are bound to arrive. The measurement of these cells that are "on the fly" requires communication between the various switches along the VC path. The method described below is the multi-node extension of the accounting scheme described in our earlier work [6].

3.4.1 Estimating the effective queue size.

The QRC in each switch maintains a register (Q_{reg}) indicating the *residual* queue size for each VC. The residual queue size consists of all the cells that are "on the fly" and have not entered the real queue yet. The sum of the real and residual queue sizes is the effective queue size.

Whenever a backward RM cell passes by the switch, the previous QRC command is multiplied by the minimum of $Time_{rm}$ and $k_0 \cdot Time_{old}$, where $Time_{rm}$ is the time interval field contained in the current RM cell, and $Time_{old}$ is the time interval field contained in the previous RM cell. (The constant k_0 will be explained later.) The product is the number of cells that were requested in the previous RM time interval (Req_{old}), which is then added to the register Q_{reg}. In this way, cells that have been requested in the previous feedback interval can be added to the effective queue size estimate.

At the same time, the QRC in each switch maintains a register (Ex_{reg}) for all the unsatisfied cell requests from downstream switches (other switches in the direction of the destination). These "excess" cells are the cells that have been requested from downstream but cannot be sent because of a bottleneck at the node. The requested cells at the node Req_{old} are subtracted from the cells requested at the previous node Req_{rm}. This difference (which represents the number of QRC cells that cannot be requested at this stage) is added to Ex_{reg} . Then Req_{old} is written into Req_{rm}. The calculated QRC_{new} is then written into QRC_{rm} before the RM cell leaves the switch. We will explain later how the algorithm computes the new QRC for the switch.

When a *forward* RM cell leaves a switch, Ex_{reg} is added to Req_{rm}. Ex_{reg} is then reset to zero. We can think of Req_{rm} as the number of cells that have been requested previously by the switch downstream. When a forward RM cell reaches a switch, Q_{reg} is decremented by Req_{rm}. This operation maintains a count of how many cells requested from the switch are still in the network.

The following pseudocode summarizes the operations performed by the QRC algorithm to estimate the effective queue size:

```
Backward RM cell arrives at the switch:
    QRC_BACK_RM()
        $\mathsf{Req_{old}} = \min(\mathsf{Time_{rm}}, \mathsf{k_0 Time_{old}}) \times \mathsf{QRC_{old}}$
        $\mathsf{Q_{reg}} = \mathsf{Q_{reg}} + \mathsf{Req_{old}}$
        $\mathsf{Ex_{reg}} = \mathsf{Req_{rm}} - \mathsf{Req_{old}}$
        $\mathsf{Req_{rm}} = \mathsf{Req_{old}}$
        $\mathsf{QRC_{rm}}$ =CALCULATE_NEW_QRC()
        $\mathsf{QRC_{old}} = \mathsf{QRC_{rm}}$
        $\mathsf{Time_{old}} = \mathsf{Time_{rm}}$
Forward RM cell arrives at the switch:
    QRC_FORW_RM()
        $\mathsf{Q_{reg}} = \mathsf{Q_{reg}} - \mathsf{Req_{rm}}$
        $\mathsf{Req_{rm}} = \mathsf{Req_{rm}} + \mathsf{Ex_{reg}}$
        $\mathsf{Ex_{reg}} = 0$
```

Note that if the propagation delays for the backward RM cells vary or if some RM cells are lost, some error will be introduced in our estimate of the effective queue size. In the first case (time-varying propagation delay), the error can be corrected by using an extra field in the next RM cell. This extra field records the error in the QRC request of the previous RM cell and then corrects the effective queue size at the affected nodes. Likewise, in the second case (RM cell loss), the accumulated error can be corrected by periodic RM cells that compare the total number of QRC requests that each node has made and received. The discrepancies can be used to correctly update the effective queue sizes. The implementation details of this error recovery scheme are beyond the scope of this paper.

3.4.2 Computing the new QRC rate.

In order to calculate the new QRC rate, we use a proportional controller which takes the effective queue size as its input. The effective queue size is the sum of the residual and real queue sizes ($Q_{reg} + Q_{real}$). We use the effective queue rather than the real queue to perform our calculations because there is no delay between the time a cell is requested and when it enters the effective queue. This eliminates the instability associated with delays in feedback control loops. The gain of the controller is determined by the time interval supplied by the backward RM cell. In pseudocode the function to calculate the QRC rate is:

```
CALCULATE_NEW_QRC()
    $\mathsf{QRC_{temp}} = (\mathsf{THRESH} - \mathsf{Q_{reg}} - \mathsf{Q_{real}}) \div (\mathsf{k_0 Time_{rm}})$
    $\mathsf{rate} = \mathsf{QRC_{rm}} + \mathsf{RRC_{rm}} - \mathsf{AvailableRate}$
    Return $\max[\min(\mathsf{rate}, \mathsf{QRC_{temp}}), -\mathsf{AvailableRate}]$
```

THRESH is the desired queue size. It should be set to a small positive constant. Any queue size which will not cause significant queuing delay is adequate. In steady state, the queue of the slowest switch in the VC path will be of this size. The choice for THRESH is not a real design consideration since any small value (relative to the total queue capacity) will do.

The parameter k_0 is a constant gain factor. Its value should be greater than 1 and less than approximately 10. The proportional controller will calculate the rate so that $1/k_0$ of the queue error will disappear in one feedback step. Trying to eliminate the entire queue error in one step can create oscillations due to measurement errors. Setting the goal for less than one tenth of the error unnecessarily slows down the system. Simulations show that a value of about 5 gives smooth and fast results. Again, the exact choice of k_0 is not critical, and any value within a wide range is good.

The rate command calculated by the controller (QRC_{temp}) is limited so that the total rate command (QRC+RRC commands) is not larger than the total rate command downstream *and* so that the total rate command is not negative.

3.5 Source behavior

When the source receives a backward RM cell at time t_{arrive}, it sets its data sending rate to be the sum of the QRC and RRC commands ($QRC_{rm} + RRC_{rm}$). If the source cannot or does not want to comply with this rate request, it can send data at any rate below the specified rate.

The backward RM cell is then forwarded to the data destination. The data fields in the RM cell are left untouched. Even if the source sends less data than the requested rate, the effective queue size estimates will be correctly updated, and the overall algorithm still works.

The source will also set an expiration interrupt for time $t_{arrive} + k_0 \times Time_{rm}$. If the next RM cell arrives before the expiration time, the rate values of the new RM cell will become the current data sending rates. On the other hand, if a new RM cell does not arrive by the expiration time, the source stops sending data. When a new RM cell arrives, the source restarts transmitting data at the rate specified by the RM cell.

4 Analysis

In this section, we will analyze the behavior of the control algorithm. We first look at the single node case and see how the minimum time requirements are satisfied. Then we will show its stability and fairness in the multi-node case. In this section, we still assume that all the switches use a per-VC queuing discipline.

4.1 Single node case

In figure 1, we present a block diagram representation of the feedback loop for the single node case. We use a discrete-time dynamic system model. The time interval between events is the inter-arrival time between RM cells at the switch (Δ_{rm}). At time n, $r[n]$ is the number of cells serviced, $d[n]$ the error in measuring the available rate, $x[n]$ = `THRESH` the desire queue level, and $q[n]$ the real queue size. For convenience, we shall

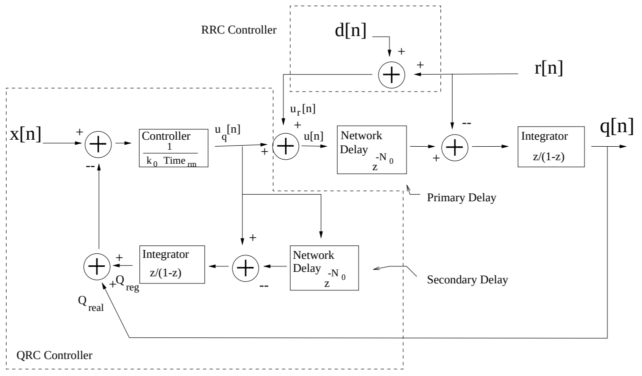

Figure 1: Single node block diagram.

express the RTD in terms of integral multiples of Δ_{rm}, which is assumed to be small compared with the RTD.

This model is similar to the continuous time model presented in [6]. The main difference is that the secondary delay (the lower of the two in the figure 1) used in the QRC is not an artificial delay as used in the Smith Predictor but the real delay of the network. If the real round-trip delay varies, the secondary delay in the QRC will vary simultaneously. Conceptually, the primary delay models the delay in carrying the real data, whereas the secondary delay models the delay in carrying the RM cells which describe the QRC traffic. We showed in [6] how the QRC algorithm can be thought of as an observer of the RRC error. The command $u_q[n]$ is an observer estimate of the difference between the output and input rates at the queue, *if* the RRC were to function by itself. With the new algorithm, the QRC uses the information contained in RM cells to *simulate* the flow of the real data cells it has requested, and thus being able to observe and correct the error of the basic RRC algorithm.

Using z-transform methods, we can calculate the transfer functions from $x[n]$, $d[n]$, and $r[n]$ to $q[n]$:

$$H_x(z) = \frac{Q(z)}{X(z)} = \frac{Kz^{-N_0}}{(K+1) - z^{-1}} \tag{5}$$

$$H_r(z) = \frac{Q(z)}{R(z)} = \frac{(-1 + z^{-N_0})}{1 - z^{-1}} + \frac{K(z^{-N_0} - z^{-2N_0})}{(1 + K - z^{-1})(1 - z^{-1})} \tag{6}$$

$$H_d(z) = \frac{Q(z)}{D(z)} = \frac{z^{-N_0}}{1 - z^{-1}} - \frac{Kz^{-2N_0}}{(1 + K - z^{-1})(1 - z^{-1})} . \tag{7}$$

N_0 is the number of discrete steps in the round-trip delay, and K is the gain of the proportional controller, $1/(k_0 Time_{rm})$.

From the transfer functions, we can see that every input-output relation is stable. A bounded error in the switch rate measurement will only cause a bounded error in the real queue size. A change in the service rate will only affect the queue size for some transient time.

From equation (6), we can calculate how soon $q[n]$ will reach steady state after $r[n]$ reaches steady-state. This is what we referred to in section 2 as the settling time or time to stabilize the queue. By using inverse z-transforms, we can find the impulse response $h_r[n]$ between $r[n]$ and $q[n]$:

$$h_r[n] = -u[n] + \left(2 - (1+K)^{-(n+1-N_0)}\right) u[n-N_0] - \left(1 - (1+K)^{-(n+1-2N_0)}\right) u[n-2N_0] \ , \quad (8)$$

where $u[n]$ is the step function. Since $K = 1/(k_0\ Time_{rm})$, as the interval between RM cells decreases (the feedback frequency increases), the gain K increases and $h_r[n]$ approaches zero for $n > 2N_0$ or $2RTD$. Furthermore, $\sum_{n=0}^{\infty} h_r[n] = 0$. This leads us to the following theorem:

Theorem 3 Suppose a single switch is connected to a single source and the bi-channel rate control (BRC) algorithm is used. At two round-trip delay time (RTD) after the last change in the available rate of the VC, its queue length at the switch can be made arbitrarily close to the reference level by increasing the feedback frequency. Moreover, given a fixed feedback frequency, the error between the queue length and the reference level will converge to zero exponentially fast after 2 RTD following a change in the available rate.

Note that the value of $h_r(2RTD)$ decreases *exponentially* with the feedback frequency. Therefore, the difference between $q[n]$ and `THRESH` at 2 RTD will also decrement exponentially with frequency. A small increase in feedback frequency will result in a large decrease in the queue size error at 2 RTD. A feedback period that is an order of magnitude smaller than the RTD will be enough to ensure that the queue size at 2 RTD will be within a few percentages of the reference level. Theorem 2 establishes that the minimum time to stabilize a queue without knowledge of the round-trip delay is 2 RTD. Therefore, the BRC algorithm approaches this limit exponentially fast as the feedback frequency increases.

Theorem 4 Suppose a single switch is connected to multiple sources, some of which may *not* be greedy and have limited data rates, and the bi-channel rate control (BRC) algorithm is used. We let RTD_{max} be the longest round-trip delay of all the VCs passing through the switch and T_s be the time 3 RTD_{max} after the last change in the *total* switch output rate. At time T_s, the queue length of every VC bottlenecked at the switch can be made arbitrarily close to the reference level by increasing the feedback frequency. Moreover, given a fixed feedback frequency, the error between the queue lengths and the reference level will converge to zero exponentially fast after T_s.

Note that Theorem 4 still applies even if we are using a low order approximation of the available rate.

4.2 Multiple node case

4.2.1 Transient response. The behavior of our control algorithm is not very different in the multiple node case from the single node case. The slowest or most congested node of a VC path controls the dynamics of the VC flow. This node (the *active* node) imposes its rate commands over the other nodes. The dynamics of the VC queue at the active node are similar to those for the single node case. As in the single node case, the multi-node algorithm will stabilize effectively the queue level of its active node. In steady-state, the active node will have a queue size equal to `THRESH`. The other nodes in that VC will have empty queues.

After the available rate of a particular switch in a multi-node network achieves steady-state, it will take 2 RTD to stabilize the queue of the switch. We are assuming that the feedback frequency is sufficiently high. During the first RTD, the new RRC command is being sent to the source and the switch is expecting this new rate. At 1 RTD, the switch starts receiving data at the correct RRC rate. Since the effective queue is now stable, the switch stops requesting a QRC command and the total command sent by the switch achieves steady-state. After another RTD, the arrival rate at the switch reaches steady-state and the whole system is in steady-state. As pointed out in section 2, this is the minimum time for a flow control algorithm without knowledge of link delays to stabilize a VC.

4.2.2 Fairness. In steady-state, the source follows the command of the slowest or active node of the VC path. Therefore this node will have a queue size equal to `THRESH`. The other nodes in that VC will have empty queues. We can then make the following statement on the algorithm's allocation of rates to the different VCs.

Theorem 5 The bi-channel rate control (BRC) algorithm provides a max-min fair allocation of bandwidth among the different virtual channels.

5 Virtual Queuing

In the preceding sections, we have discussed the implementation of our flow control algorithm in network switches that use per-VC queuing. The advantage of this queuing discipline is that it decouples the dynamics of one VC from the other VCs. In per-VC queuing, each VC sees the combined effects of the dynamics of the other VCs as simple variations in its service rate.

In this section, we show how to implement our flow control algorithm in network switches that use FIFO queuing. We accomplish this by using the technique of *virtual queuing* [2, 11, 12], which emulates per-VC queuing on a FIFO queue. Simply put, the idea involves keeping track of the queue length on a per-VC basis as if per-VC queuing were actually being used. The operation of per-VC queuing can be summarized in the following pseudo-code. $q^i_{virtual}$ is the virtual queue size for VC i, MAXVC is the total number of VCs passing through the switch, and i is a local variable.

```
if a cell from VC j arrives at the switch,
    $q^j_{virtual} = q^j_{virtual} + 1$

if a cell is serviced by the FIFO queue
    do
        {
        $i = i + 1$
        if ($i$ = MAXVC)
            $i = 0$
        }while($q^i_{virtual} = 0$)
    $q^i_{virtual} = q^i_{virtual} - 1$
```

While the real data cells are stored in the FIFO queue, the RRC and QRC algorithms pretend that the cells are stored in the (virtual) per-VC queues. They perform all their calculations as if the switch was still using per-VC queuing. Therefore, the rates the QRC and RRC algorithms request will still be the same as in the per-VC queuing case. After the round-trip delay, the inputs to the FIFO queue will become these rates. After the queuing delay, the output of the FIFO queue will be on average (with averaging window of size greater than the round-robin period) the same as that of a per-VC queue. Therefore, as long as the rate requests are based on a per-VC queue (real or virtual) after a sufficient time period (one round-trip delay and one queuing delay), the behavior of the FIFO queue will be on average (as defined above) the same as that of a per-VC queue.

Note that the virtual queue is not used as an estimation of the real queue, but as a tool to distribute bandwidth fairly among the various VCs. The key idea is to request data from the source as if the switches used per-VC queuing. Once the correct rates are sent from the sources, the VC rates across the network will converge to the equivalent rates under per-VC queuing. The simulation results presented in the next section will show that the virtual queuing technique does behave very similarly to real per-VC queuing.

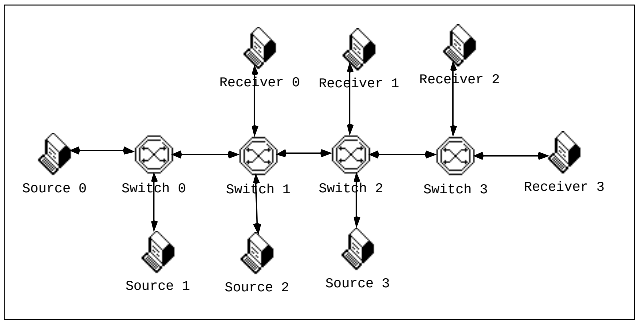

Figure 2: Opnet simulation configuration.

6 Simulation results

In this section we present some simulation results using our flow control algorithm in a network with per-VC queuing and in networks with FIFO queuing. Both scenarios were simulated using the Opnet simulation package.

6.1 Simulation scenario

The network configuration that we are simulating is depicted in figure 2. Sources 0, 1, 2, and 3 are sending data to receivers 3, 0, 1, and 2 respectively. All the sources have always data to send and will always comply to the rate request. The propagation delays between the switches is 10 msec, and that between the switches and users is 0.5 usec. Switches 0, 1, 2, and 3 can transmit data at each of their ports at rates 7000, 8000, 9000, and 10,000 cells/sec respectively. The RM cell frequency is set to (1/64) of the maximum data rate or 1562.5 Hz, and the QRC gain is set to $1/(5\Delta_{rm})$ or 312.5 s^{-1}. The desired queue level THRESH is 50 cells for each of the VC queues.

At initialization, none of the sources are sending any data. The receivers are continuously sending RM cells to the sources at the fixed frequency. At time 0.05 sec, source 0 starts sending data to receiver 3 on VC 3. Because switch 0 is the slowest node, it becomes the bottlenecked and active node. At time 0.15 sec, source 3 starts sending data to receiver 2 on VC 2. Because switch 2 has to transmit cells belonging to VCs 2 and 3, it becomes the next active node. At time 0.3 sec, source 2 starts sending data to receiver 1 on VC 1. This causes switch 1 to become the next active node.

Finally, at time 0.4 sec, source 1 starts sending data to receiver 0 on VC 0. Switch 0 becomes once again the active node.

6.2 Switches with per-VC queuing

When the network has a per-VC queuing discipline, our algorithm uses the per-VC queues to perform its RRC and QRC calculations. Our new algorithm, without using knowledge of the round-trip delay, can achieve practically identical results to the algorithm simulated in [6], which did use knowledge of the round-trip delay.

In figure 3, we can observe the RRC commands that the switches compute for the sources. The data samples are recorded at the time when the RM cell with the command arrives at the corresponding source. Initially, the RRC command for source 0 is 70,000 cell/sec, which is the rate of the slowest link in the path. When source 3 starts sending data, the RRC command for both sources becomes 45,000 cells/sec, which is half the rate of the node they share (switch 2). When source 2 starts transmission, the RRC commands for sources 0 and 2 become 40,000 cells/sec, since that is half the capacity of the node they share (switch 1). Since source 0 is only using 4/9 of switch 2's capacity, the RRC command for source 2 is increased to 5/9 of switch 2's capacity or 50,000 cells/sec. Finally, when source 1 starts sending data, the RRC command for sources 0 and 1 becomes 35,000 cell/sec (half of switch 0's capacity), and sources 2 and 3 increase their rates to 45,000 and 55,000 cells/sec respectively to utilize the unused bandwidth in switches 1 and 2.

The RRC commands shown in figure 3 are max-min fair to all the sources. However, we can also see that it takes some time (delay between switch and source) for a new RRC command computed at the switch to be followed by the source. For example, at time 0.15 sec, switch 2 send a lower RRC command to source 0. However, the command reaches source 0 at time 0.17 sec. Between times 0.15 and 0.19 sec, switch 2 is receiving from sources 0 and 2 more cells that it can service. This error cannot be corrected by the RRC controller since latency is an inherent characteristic of the network. However, this error can be corrected at a later time by the QRC controller.

In figure 4, we can observe the QRC commands that arrive at each source. Initially, when each source starts sending data, the initial QRC command is high in order for the source to send an initial burst of cells that will fill up the VC queue of the active node to the desired size (50 cells in this case). After the initial burst, the QRC source quickly settles down to close to zero.

At time 0.17 sec the QRC command for source 0 goes negative in order to slow source 1 down. This is needed because source 3 has started sending data at time 0.15 sec and switch 2 has become congested. Since the QRC controller in switch 2 can observe the error caused by the delay, it can request the exact QRC rate needed to correct this error. Likewise, at time 0.31 sec, the QRC command for Source 0 becomes negative to reduce the congestion at switch 1 caused by the activation of source 1. Throughout the simulation, the small QRC commands correct the small errors in the RRC due to the errors in the estimation of the actual service rate.

In figure 5, we can observe the queues for VC 3 (the VC between source 0 and receiver 3) at each of the switches. At time 0.05 sec, the queue of the active node (switch 0) fills up to the desired queue size of 50 cells. Notice how the ramp is close to a first order exponential without any oscillations. At time 0.15 sec, when source 3 starts sending data, switch 2 becomes the active node and its queue starts increasing steadily for 0.04 sec (round-trip delay). Notice how it takes 2 round-trip delays **2 RTD** (0.08 sec) to stabilize this queue back to the 50 cell level.

At time 0.3 sec, switch 1 becomes the active node, and its queue increases for 0.2 sec (round-trip delay). The QRC command forces the queue size to decrease to the desired level. Again, it takes **2 RTD** (0.04 sec)

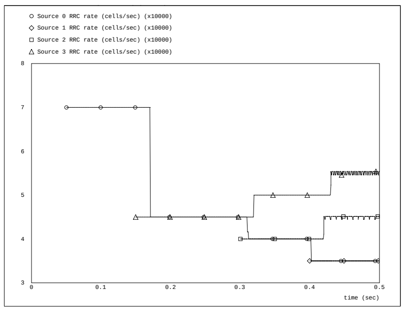

Figure 3: RRC rates at each source.

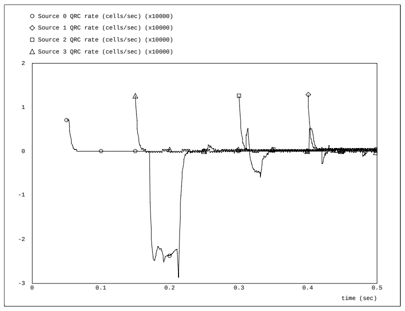

Figure 4: QRC rates at each source.

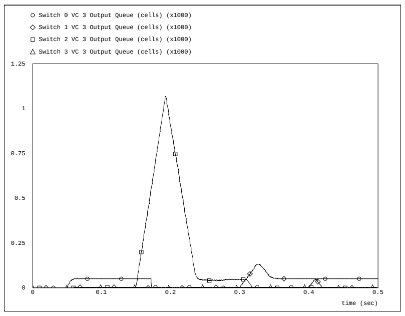

Figure 5: Output queue of VC3 at each switch.

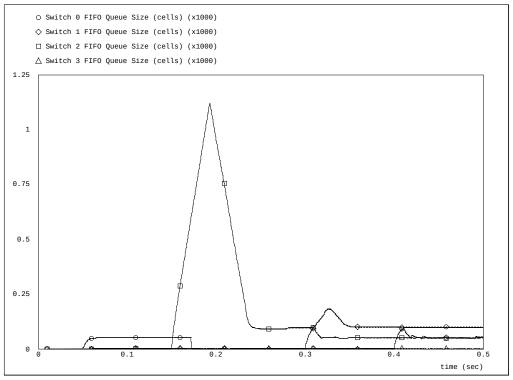

Figure 6: Actual FIFO queues at each switch.

to stabilize this queue. At time 0.4 sec, switch 0 becomes the active node again. Since the delay to source 0 is negligible, there is no overshoot. Note that when a node ceases to be active, its queue empties out. At every moment of time between 0.05 and 0.5 sec there is at least one active node with a non-empty queue.

6.3 Switches with FIFO queuing

When the network uses a FIFO queuing discipline, our new algorithm needs to make use of virtual queuing to obtain information for the RRC and QRC controllers. Surprisingly, the performance of the algorithm with virtual queuing is almost identical to its performance with per-VC queuing. Figure 6 shows how the size of the FIFO queue at each switch is very close to the sum of the VC queues shown in the per-VC queuing case. For more figures see the full version of this paper [7].

7 Concluding remarks

We have presented the design and analysis of a new efficient congestion control algorithms for ABR service in ATM based on a control theoretic approach. This extends our earlier work [6], where we introduced a methodology which separates the controller into two simplified, decoupled, and complementary parts (RRC and QRC). Furthermore, the QRC part (in the primal structure) or the RRC part (in the dual structure) behaves as an error observer of the other.

The results in this paper improve our earlier work in several ways. We first separated the feedback of the QRC and RRC control. This allowed us to implement the QRC observer in the network itself, without having to use artificial delays. We can think of the algorithm as using special information flowing through the network to observe and control the bulk data of the network itself. The idea of virtual queuing is then applied to implement our algorithm with FIFO queues.

References

[1] L. Benmohamed and S. Meerkov, "Feedback Control of Congestion in Packet Switching Networks: The Case of a Single Congested Node," IEEE/ACM Transactions on Networking, Vol. 1, No. 6, 1993.

[2] F. Chiussi, Y. Xia, and V. P. Kumar, "Virtual Queuing Techniques for ABR Service: Improving ABR/VBR Interaction," IEEE Infocom '97.

[3] M. Hluchyj et. al. "Closed-Loop Rate-based Traffic Management," ATM Forum Contribution 94-0438R2, July 1993.

[4] R. Jain, S. Kalyanaraman, and R. Viswanathan. "The OSU scheme for congestion avoidance using explicit rate indication," ATM Forum Contribution 94-0883, September 1994.

[5] S. Mascolo, D. Cavendish, and M. Gerla, "ATM Rate Based Congestion Control using a Smith Predictor: an EPRCA Implementation," IEEE Infocom '96.

[6] P. Narváez and K.Y. Siu, "Design of Feedback Control Algorithms for ABR Service in ATM," Technical Report 11/96, d'Arbeloff Laboratory for Information Systems and Technology, MIT.

[7] P. Narváez and K.Y. Siu, "Optimal Feedback Control for ABR Service in ATM," Technical Report 1/97, d'Arbeloff Laboratory for Information Systems and Technology, MIT.

[8] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara. Rate-based Congestion Control for ATM Networks. Computer Communication Review, vol 25, n.2, 1995.

[9] K.-Y. Siu and H.-Y. Tzeng, "Intelligent Congestion Control for ABR Service in ATM Networks," Computer Communication Review, vol.24, no.5, 1994.

[10] O. Smith, "A Controller to Overcome Dead Time," ISA Journal, Vol. 6, No. 2, Feb. 1959.

[11] H.-Y. Tzeng and K.-Y. Siu, "Performance of TCP over UBR IN ATM with EPD and Virtual Queuing Techniques," Workshop on Transport Layer Protocols over High Speed Networks, IEEE Globecomm, Nov. 1996.

[12] Y. Wu, K.-Y. Siu, and W. Ren, "Improved Virtual Queueing and EPD Techniques for TCP over ATM," to appear in IEEE ICNP'97.