13

Refinements to Rate-Based Flow Control with Extensions to Multidrop Applications

S. Pejhan, M. Schwartz and D. Anastassiou
Department of Electrical Engineering, Columbia University
New York, NY 10027, USA, {sassan,schwartz,anastas}@ctr.columbia.edu

Abstract

We present a rate-based, explicit feedback flow control mechanism. Congestion control is distributed among all the nodes on the path from the sender to the receiver. This leads to a more accurate control mechanism and a scalable design. Our mechanism is shown to be effective in scenarios where the bottleneck can vary among the intermediate nodes without leading to buffer overflows during transitional periods. Overhead is kept low by using the same control packet to relay control information and measure internode delays. We analyze both an end-to-end and a hop-by-hop approach, as well as multidrop scenarios.

1 INTRODUCTION

Rate-based congestion control schemes gained popularity with the introduction of fiber optics, and the ensuing development of "light-weight" transport protocols [5]. Distributed multimedia applications and the need for integrated services also gave new impetus to rate-based schemes*, though credit-based schemes are still under serious consideration [9] (mostly for LANs [13]).

Rate-based control uses feedback from the network to adjust the source transmission rate so as to prevent congestion. Earlier schemes proposed having one or more "congestion thresholds" and adjusting the source rate to a "high" or "low" level accordingly (e.g. [16]). The choice of appropriate thresholds and high/low transmission rates in such scenarios often boiled down to intelligent guesses. More sophisticated schemes are described in [2, 17] where a linear increase/exponential decrease algorithm is used to adjust the source rate, again according to threshold based feedback information. Rate-based schemes proposed for ATM networks use Explicit Forward Congestion Indication (EFCI): when a packet encounters congestion at an intermediate node—defined as the queue size at the node buffer exceeding some threshold—it will set the congestion bit to 1. The receiver checks this field and if congestion was experienced notifies the sender [10]. Again, there is the problem of how to define the threshold, but more importantly, by having intermediate nodes relay congestion information through the receiver the system will be slow to respond. Furthermore, the sender has no information regarding the magnitude of the congestion.

A number of publications propose using explicit feedback to capture the magnitude of congestion [8, 7, 1, 6]. The solutions in [8] and [7] follow an end-to-end solution, while a hop-by-hop

*The ATM Forum has selected a rate-based flow control mechanism to support ABR service [4].

L. Mason et al. (eds.), *Broadband Communications*

approach has been proposed in [11]. Closed-loop rate control mechanisms may also be used in conjunction with open-loop mechanisms [17].

In Section 2, we propose an end to end flow control scheme, stating the controller objectives and deriving the control mechanism. Simulation results compare the proposed scheme with others proposed in the literature. Section 3 describes the hop-by-hop version of the proposed schemes. Extensions of the control mechanism to a multidrop scenario are then described (Section 4), followed by concluding remarks (Section 5).

2 END TO END APPROACH

2.1 Network Model and Controller Objectives

Our model consists of the sender, the receiver and the network cloud. The bottleneck is defined as the node with the lowest processing rate and could be any of the intermediate nodes between the sender and the receiver, or even the receiver itself. The bottleneck node could vary with time. The parameters of interest are as follows:

- $\lambda_s(t)$: The source transmission rate. We assume that the sender is able to match the transmission rate to the rate computed by the control mechanism.[†].
- $\lambda_R(t)$: The rate at which packets enter the bottleneck. Note that this is *not* the same as $\lambda_s(t)$ above, although it is related to it. We shall elaborate on this later on.
- $x(t)$: The buffer occupancy at the bottleneck. This is the variable that we want to control through the sender's transmission rate. It is bounded by 0 and some maximum buffer size, B. We are assuming that each flow is allocated its own separate buffer.
- $\mu_b(t)$: The processing rate allocated to the connection of interest at the bottleneck. This is a parameter over which we assume no control[‡], but varies within bounds (§2.4).

The design strategy that is to aim for a target queue size, x^*, at the bottleneck. Congestion control strategies aim at increasing throughout, or reducing packet delay and packet loss. As pointed out in [8], the choice of the set-point reflects a trade-off between these three parameters: the lower the set-point, the lower the packet delay and packet loss, but the more underutilized the bandwidth. We shall not concern ourselves with the choice of the set-point[§]. It is usually set to $B/2$, half the maximum buffer size.

The buffer occupancy at the bottleneck is governed by equation (1) below, which states that the buffer occupancy at time $t + t_0$ is equal to that at time t plus what came in and minus what left during the time interval $[t, t + t_0]$:

$$x(t + t_0) = x(t) + \int_t^{t+t_0} (\lambda_R(\tau) - \mu_b(\tau))d\tau \qquad \text{subject to } 0 \leq x(t + t_0) \leq B. \tag{1}$$

[†]In practice, it is difficult to accurately achieve this for images and video, as the produced bit-rate is very much source dependent. An elaborate discussion is beyond the scope of this paper.

[‡]These are required to be controllable in a hop-by-hop design as explained in Section 3.

[§]A more elaborate, quantitative discussion appears in [8].

The objective of the controller is to adjust the transmission rate of the sender, $\lambda_s(t)$, in such a way so that the buffer occupancy at the bottleneck, x, remains close to x^*—the two are related via $\lambda_R(t)$ and equation (1).

2.2 Controlling site

The controller needs to frequently obtain information regarding the buffer occupancy $x(t)$, the incoming packet rate $\lambda_R(t)$ and the processing rate $\mu_b(t)$ of the bottleneck. The simplest, and most accurate, form of obtaining this information is from the bottleneck itself. The controller also has to have a model for predicting the future values of these bottleneck parameters due to the non-zero propagation delays (it takes time for information to propagate from the bottleneck to the sender; it will also take time for any change in the transmission rate to affect the buffer occupancy at the bottleneck). These models are discussed in more detail in later sections.

In [7] the bottleneck measures the buffer occupancy and processing rate, and relays this information to the sender. In [8] a packet-pair is sent by the sender, and by measuring the time differential between the acknowledgements, the sender can estimate the processing rate at the bottleneck. This method is somewhat inaccurate, however, as pointed out by the author. It also adds unnecessary overhead, since four packets are generated for each estimate. In both cases, however, the incoming packet rate at the bottleneck is assumed to be identical to the source transmission rate. Meanwhile, [6] and [1] assume a fixed processing rate for the bottleneck, while assuming the incoming packet rate to be equal to a time-shifted version of the source transmission rate.

In all cases except [1], however, the sender acts as controller. One of the main differences in our model is that the bottleneck is the controlling site. This way the controller can use up-to-date information regarding the three parameters of interest. In [7] and [6], the information is already inaccurate by the time it reaches the source. Unlike these parameters, the source transmission rate is fixed for the duration of a control interval. So this information is still valid when it reaches the bottleneck/controller in our design.

Each intermediate node in our model will assume that *it* is the bottleneck. It will compute its desired source transmission rate according to the control mechanism described in §2.4. When the node becomes the bottleneck, its model for predicting its future processing rate would be valid. The same cannot be said for source-based controllers since when bottlenecks change, the source's model for estimating these parameters will not be valid (initially). This is another argument for locating the controller at the bottleneck.

2.3 Control Packet

The receiver will send a high-priority control packet periodically—at intervals of t_0—to the source, indicating its own processing rate (PROC_RT), as well as the rate at which it wants the source to transmit (DES_RT), as computed by its control mechanism (§2.4). Along the path, a switch will modify those parameters under two conditions only. If its own processing rate is lower than PROC_RT in the control packet, this means that the current switch is slower than all those downstream. Hence it will replace both parameters (PROC_RT and DES_RT) with its own processing and desired source transmission rates, as in [7]. If its processing rate is higher than PROC_RT but lower than DES_RT, it will replace the *latter* only with the value of its current processing rate.

This second condition is one of the key refinements that we have made: there is no point for a downstream 'bottleneck' node to request a source transmission rate that is higher than what

can be delivered by any of the intermediate upstream nodes[¶]. With the schemes described in [7] and [8], there can be temporary buffer overflow at upstream nodes. Our mechanism will prevent these. This will be illustrated in our simulations (§2.5).

The sender does not need to have explicit knowledge about which node is the bottleneck: by the time the control packet arrives at the source, PROC_RT will be set to the processing rate of the slowest node. At the sender, the packet is acknowledged immediately, with the new transmission rate (which is set to DES_RT) indicated in the acknowledgement. Upon receipt of the sender's acknowledgement, intermediate nodes will note the new transmission rate, which they will need to compute the desired transmission rate for the next update. They will pass on the control packet to the neighboring downstream node without modifications except if their processing rate is less than the source transmission rate. This will happen only at the bottleneck, in which case the source transmission rate indicated in the control packet is replaced by the node's processing rate.

2.4 Control Mechanism

Control Law

For each intermediate node an 'interval' begins when it receives a control packet going downstream. The intervals at the different nodes are thus not synchronized. At the beginning of the kth interval, the controller records its buffer occupancy, $x^{(k)}$. Using equation (1), it can then estimate its buffer occupancy, $x^{(k+1)}$, at the end of the current interval. This, however, requires knowledge of the incoming packet rate and node processing rates during the current (kth) interval. The incoming rate is provided by the control packet at the very beginning of the interval.

If the buffers at upstream nodes are not empty, the incoming packet rate for a particular node will depend on the processing rate of its upstream nodes. Each node will assume that its incoming packet rate is equal to the transmission rate indicated in the control packet (the propagation delay is accounted for by the fact that the control interval of the intermediate node is delayed by that much with reference to the control interval of the source). Let us denote the 'indicated' transmission rate by $\lambda_{S'}$ to distinguish it from the actual source transmission rate. The two will be equal for the bottleneck and all preceding nodes, but $\lambda_{S'}$ will equal the bottleneck processing rate for all nodes downstream of the bottleneck. When the buffer occupancies at all nodes are 0 except at the bottleneck, this assumption is accurate: for nodes preceding the bottleneck, the incoming rate is indeed a delayed version of the transmission rate, while for nodes located after the bottleneck the incoming rate is a delayed version of the bottleneck's processing rate.

During the 'transition' periods (i.e. at session start-up and just after changes in the location of the bottleneck), this assumption will underestimate the actual incoming packet rate (for those nodes located downstream of the old bottleneck). As long as the buffers at the different nodes are equal in size, however, this will not pose a problem (i.e. buffer overflows will not occur): the new bottleneck can accommodate all the packets held by the previous bottleneck, after which the system will reach a 'steady state'. Again, this will be illustrated by the simulations.

Using equation (1), and the foregoing discussion, we can estimate the buffer occupancy at the beginning of the next $(k+1)$ interval using a discrete-time model (since the source transmission rate is piece-wise constant):

$$\hat{x}^{(k+1)} = x^{(k)} + t_0 \left[\lambda_{S'}^{(k)} - \hat{\mu}_b^{(k)}\right] \qquad \text{subject to } 0 \leq \hat{x}^{(k+1)} \leq B \tag{2}$$

[¶]This situation will occur if the bottleneck node's buffer occupancy is significantly below its target.

$\hat{\mu}_b^{(k)}$ is estimated as described in section 2.4. The next task is to set $x^{(k+2)}$ equal to x^* and solve for the desired source transmission rate for the interval $k+1$, $\lambda_S^{(k+1)}$.

$$x^* = \hat{x}^{(k+1)} + t_0 \left[\lambda_{S'}^{(k+1)} - \hat{\mu}_b^{(k+1)}\right] \tag{3}$$

We estimate $\hat{\mu}_b^{(k+1)}$ using the method of section 2.4. We then derive the desired value of $\lambda_{S'}^{(k+1)}$ using equation (3),

$$\lambda_{S'}^{(k+1)} = \frac{x^* - \hat{x}^{(k+1)}}{t_0} + \hat{\mu}_b^{(k+1)} \tag{4}$$

The control mechanism—equation (4)—will require the source to transmit at a higher rate than the current processing rate if the buffer occupancy is below the target, and vice versa. The control mechanism as it stands now can lead to instability (see [8] for proof). To rectify this problem, we need to add a gain factor, δ, to equation (4), as shown below:

$$\lambda_{S'}^{(k+1)} = \delta \left(\frac{x^* - \hat{x}^{(k+1)}}{t_0} + \hat{\mu}_b^{(k+1)} \right) \tag{5}$$

where $0 < \delta < 1$. If δ is too small, the system will be too slow to respond. A value of 0.9 seems to be a good choice for δ [8].

Estimating the Processing Rate at the Bottleneck

The processing rates at intermediate nodes vary because new connections are established and old ones broken continuously. [8] and [7] both use basically the same model to estimate the future processing rate. The processing rate during the kth interval, $\mu^{(k)}$, is estimated by equation (6):

$$\hat{\mu}^{(k)} = (1-\theta)\tilde{\mu}^{(k-1)} + \theta\hat{\mu}^{(k-1)} \tag{6}$$

where $\tilde{\mu}^{(k-1)}$ is the last observation of the processing rate‖. Both cited references concur that θ should be a variable. A small value of θ would attach too much importance to the last observation made, which could be corrupted by either observation noise, or uncharacteristic spikes in the processing rate. A large value of θ will make the system slow to respond. In both schemes θ is defined so as to track abrupt changes in the service rate, but not be affected by small changes or noise. While [8] uses a fuzzy predictor to this end**, a simpler scheme is proposed by [7]:

$$\begin{aligned} \theta &= 0.25E^2/\sigma^{(k+1)} \\ E &= \mu^{(k)} - \hat{\mu}^{(k)} \\ \sigma^{(k+1)} &= 0.25E^2 + (1-0.25)\sigma^{(k)} \end{aligned} \tag{7}$$

where E and σ are the estimation error and the estimate for the squared estimation error, respectively. The number 0.25 was derived empirically. Both references indicated that this model performed satisfactorily. Hence we shall use the same model as in [7].

‖In [7] θ is defined the other way around.

**A random walk model is also proposed for the processing rate, but is discarded in favor of the fuzzy predictor.

In our model, since each intermediate node carries out its own estimation for its future processing rate, θ will have a different value for each of them. With the source-based controller in [7] and [8], if the bottleneck changes, their model for predicting the future processing rate of the bottleneck will be incorrect during some transition period since the information regarding past values of $\mu^{(k)}$, E and $\sigma^{(k)}$ is inaccurate and pertaining to a different node.

Control frequency

Our analysis assumes that the receiver sends control packets once per round-trip time. Increasing the frequency of control makes the system respond faster. On the other hand, this implies that the current incoming packet rate has not yet been affected by the few most recent changes in the source transmission rate. Conversely, the node cannot change the next few changes in the transmission rate, and has to aim for affecting it several steps down the road. That, in turn, requires estimating λ_S, μ and x for several steps down the road. Those estimates will have to rely on other estimates, rather than actual values, which carries the danger of propagating errors.

2.5 Simulation results

The simulation model, consisting of a sender, a receiver and three intermediate nodes is shown in Figure 1. The propagation delay for each link is also shown.

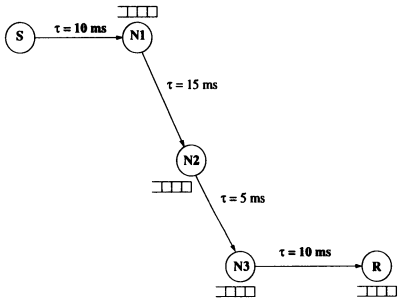

Figure 1 End to end simulation scenario

The simulation parameters are summarized in Table 1. The inter-packet delay at the source was assumed to be 0 (i.e. the source was assumed to always have packets to send), and packets were assumed to have fixed packet sizes (on the order of ATM cell sizes). Congestion was induced by introducing abrupt changes in the processing rates of the intermediate nodes and the receiver at the times shown in Table 2 (the bottleneck node for each time interval is highlighted). The numerical values chosen (in the order of 1000–2000 packets/s or 0.5–1.0 Mbits/s assuming ATM cells) could represent MPEG-1 streams for example.

Figures 2 and 3 show the buffer occupancies at nodes N1 through R for our scheme and that of [7], respectively. The source transmission rate for the two schemes is compared in Figure 4. There are no buffer overflows during transition periods with the proposed scheme—hence no packets are lost. With the scheme proposed in [7], buffer overflows resulted in 23 packets being dropped at node N1 and 159 packets at node N3. This was despite the fact that a relatively low gain factor (0.5) was used. Higher gain factors led to larger oscillations in the transmission rate and higher packet losses.

During the first period (0 to 4.0 seconds) the buffer at the bottleneck, R, rises very quickly

Simulation Period	Control Frequency	Buffer Size	x^*	δ	gain factor for [7]
30 seconds	once every 80 ms	100	50	0.9	0.5

Table 1 Parameters for end to end simulation

time (s)	0.0	4.0	7.6	12.0	16.0
N1	1000	1000	1000	750	750
N2	1200	1200	1200	**720**	950
N3	800	**800**	800	800	**650**
R	**700**	900	**780**	780	780

Table 2 Processing rates (packets/s) for end to end simulation

to its target level (Figure 2). The incoming rate to our bottleneck is limited to 800 packets/s by node R, hence the net rate of increase in buffer occupancy is 100 packets/s at most. During the third period (4.0 seconds to 7.6 seconds), when R again becomes the bottleneck, it rises to its target level at a much slower pace. This is because its processing rate is just under that of $N3$ (780 vs 800 packets/s), hence its buffer is increasing at the net rate of only 20 packets/s (thus the almost 2.5 second rise time to its target level). Now this is true in both schemes. The difference is that in our proposed scheme, the transmitter does not unnecessarily transmit at a higher rate than 800 packets/s whereas it shoots to over 1000 packets/s in the scheme proposed in [7]. This is why we get buffer overflow at $N3$ with the [7] scheme at $t = 7.6$ seconds despite the fact that $N3$ is *not* the bottleneck during this period. The same phenomenon can be observed at $t = 12$ seconds, when $N2$ becomes the bottleneck and we have buffer overflow at $N1$.

Note also the behavior of the source transmission rate at $t = 4.0$ seconds, when $N3$ becomes

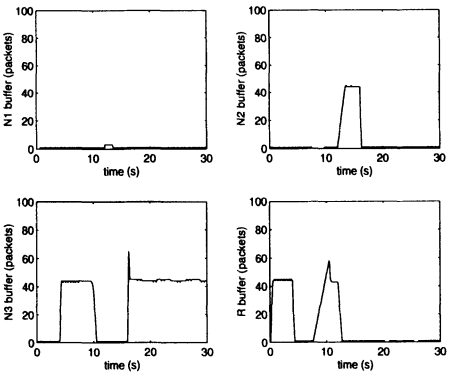

Figure 2 Buffer occupancies with proposed end to end scheme

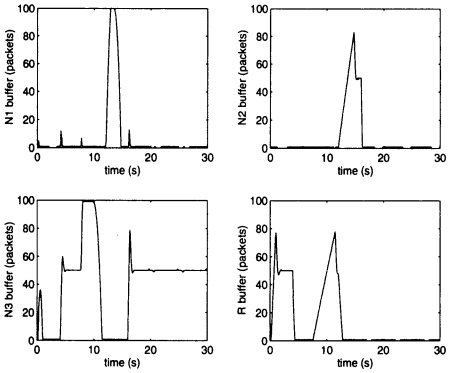

Figure 3 Buffer occupancies with scheme proposed by [7]

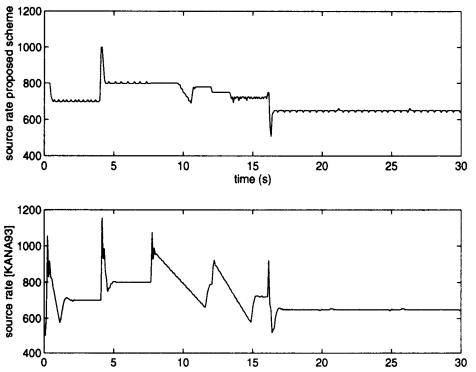

Figure 4 Source rate variations for the proposed scheme and that of [7]

the bottleneck. Its buffer is empty so it asks for a high transmission rate (close to 1200 packets/s as shown in the curve corresponding to the scheme in [7]). In our scheme, however, the source transmission rate is limited to 1000 packets/s by node $N1$.

At $t = 10$ seconds, the occupancy of the bottleneck, R has reached its target level, but it is still increasing at a net level of 20 packets/s because $N2$, which had previously been the bottleneck, is still transmitting the packets left in its buffer at 800 packets/s. At $t = 10$, $N2$ is finally done, but $N3$ has overshot its target, so the source transmission rate is lowered and then adjusted back to 780 packets as the target is reached (upper graph in Figure 4).

A second set of simulations was run using the same configuration as before, but with the processing rates increased by an order of magnitude (representing MPEG-2 rates, for example). The only other difference was that the buffer sizes were also increased by an order of magnitude. In general, the buffer sizes should be on the order of the number of packets generated during one control interval. Smaller buffer sizes lead to heavy packet losses for both schemes. The results were very similar to the first set. Again, packet loss due to buffer overflow was 0 for the proposed scheme while 139 packets were lost at node N3 for the scheme in [7] (see [12] for details).

3 HOP BY HOP APPROACH

One of the main drawbacks of end-to-end schemes using a feedback mechanism is the problem of feedback implosion at the source, under a large multicast scenario. This has led many researchers to reject closed-loop schemes altogether (e.g. [15]). Others have used a combination of probabilistic querying (receivers send feedback information with some probability), randomly delayed responses (receivers randomly delay their feedback control messages) and expanding scoped search (transmitter gradually increases the scope of its control messages) to deal with this problem ([3]). We will develop a hop-by-hop approach (based on the end-to-end scheme devised earlier) because it can be applied to large scale multicast applications and can respond to changes faster, as the propagation delay between adjacent nodes is smaller than that between the two end-points of a connection [11, 9].

In a hop-by-hop model, the sender and the receiver are connected via a number of intermediate nodes. The major difference is that the processing rates at the intermediate nodes (μ_i) are controllable. The maximum processing rate, μ_i^{max} say, is allocated depending on the number of connections going through the node. This maximum value corresponds to μ_i in the end-to-end approach. In the hop-by-hop approach, the actual processing rate can be less than or equal to this value depending on the rates requested by the downstream node (this is elaborated below).

3.1 New control mechanism

The sender and every intermediate node adjusts its transmission rate to the requests made by the immediately downstream node. Separate control packets will be exchanged between adjacent nodes. For each pair of nodes, one control packet is exchanged per round-trip time.

The control law itself is exactly the same as before. All nodes will aim for a target buffer occupancy x_i^*. An 'interval' begins upon receipt of a control packet from an upstream neighbor. At the beginning of each interval a node will estimate its buffer occupancy at the end of the interval using equation (2), and its desired incoming rate for the next interval using equation (5). It will send this desired rate to the upstream neighbor via the control packet.

The upstream neighbor will adjust its transmission rate according to the following rule: μ_{i-1} will be set to the desired rate if the latter is less than μ_{i-1}^{max}; otherwise it will be set to μ_{i-1}^{max}. Regardless of what it sets its rate to, the ability of the upstream node to deliver packets at that rate depends on its own incoming rate and the number of packets in its buffer. The maximum rate at which the upstream rate can actually deliver is given by $\lambda_{i-1} + x_{i-1}/t_0$ (where λ_{i-1} is the incoming rate of the upstream neighbor). If this quanitity is less than the rate at which it wants to send, this will result in the downstream node overestimating its incoming rate. As the simulations will show, however, this scenario is not likely if the target buffer occupancies are chosen carefully (i.e. targets are not too close to 0, so that there is always a pool of packets that can be used if the downstream node requests an increased rate). Note that only those nodes located upstream of the bottleneck will actually be able to reach their targets (x_i^*).

3.2 Simulations

For ease of comparison, the same network topology as before was used to simulate the hop-by-hop control mechanism, but with the rates and chain of events shown in Table 3. The buffer occupancies and processing rates are shown in Figures 5 and 6. As can be seen, all buffers upstream of the bottleneck maintain their target occupancies. This allows for a faster convergence when the bottleneck parameters change. As before, no packets were dropped.

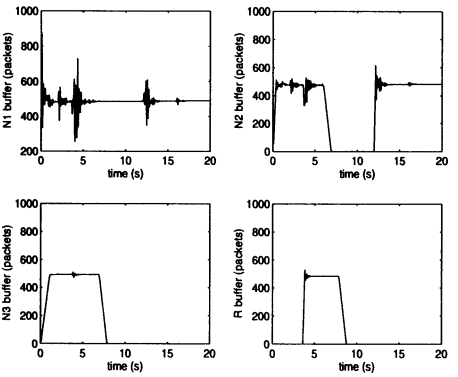

Figure 5 Buffer occupancies for hop-by-hop scheme

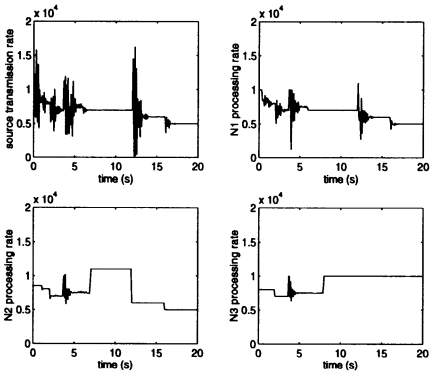

Figure 6 Processing rates for hop-by-hop scheme

Looking at Figures 5 and 6, we see that nodes closer to the source exhibit greater oscillations than those closer to the receiver. Node N3 adjusts its rate to that of the receiver. The latter changes infrequently (once in our simulation). Node 2, however, has to adjust itself to the rate of Node 3, which is slightly more volatile (since that itself is being adjusted to the rate of the receiver), and so on. Each upstream node will be adjusting its rate to a more volatile target. Since delta is less than 1 (0.9), it will not lead to instability though. An interesting instance of this phenomenon occurs at $t = 3.6$ seconds. At this point, R becomes the bottleneck. $N3$ adjusts its rate after a few slight oscillations (in both its processing rate and buffer occupancy). $N2$ has to go through greater oscillations (in both parameters), and $N1$ through even bigger oscillations. This is why there are wide fluctuations in $N1$'s buffer occupancy (and processing rate) at $t = 3.6$ seconds.

Nevertheless, to reduce the oscillations, we gradually reduced delta from 0.9 (receiver) to 0.6 (N1). This resulted in lower oscillations but slightly slower convergence [12].

4 MULTIDROP SCENARIO

time	0.0 s	2.0	3.6	6.0	12.0	16.0
N1	10000	10000	10000	**7000**	11000	5500
N2	8500	11000	11000	11000	**6000**	**5000**
N3	**8000**	**7000**	10000	10000	10000	10000
R	10000	10000	**7500**	7500	7500	7500

Table 3 Processing rates (packets/s) for hop-by-hop simulation

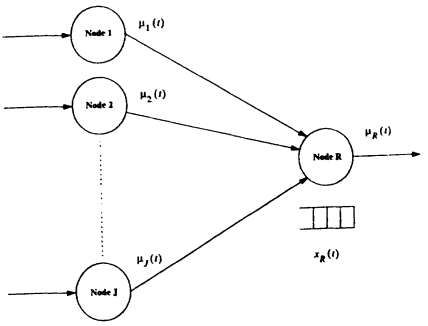

Figure 7 Model for Multidrop Scenario

Figure 7 depicts the case where an intermediate node has more than one upstream neighbor (in the hop-by-hop case) or more than one source transmitting to it (in the end-to-end case). There are J streams flowing into the same buffer at node R. To calculate the buffer occupancy at node R, we need to modify equation (2) as shown:

$$x_R(t+t_0) = x_R(t) + \int_t^{t+t_0} (\sum_{j=1}^{J} \mu_j(\tau - d_j) - \mu_R(\tau))d\tau$$

$$\hat{x}_R^{(k+1)} = x_R^{(k)} + \sum_{j=1}^{J} d_j \mu_j^{(k-1)} + \sum_{j=1}^{J} (t_0 - d_j)\hat{\mu}_j^{(k)} - t_0 \hat{\mu}_R^{(k)} \quad , 0 \le x_R^{(k+1)} \le B_R \tag{8}$$

d_j is the propagation delay between nodes R and j. We are assuming that the control period is longer than (or equal to) all d_j. Note the explicit inclusion of the delays in equation (8). If the delays between all sources and the bottleneck were assumed to be equal (as in [14]) then indeed we could use the same logic as before and equation (8) would reduce to equation (2). But given the unequal delays, the length of the interval is now set to that of the farthest away source. The bottleneck's interval will thus be synchronized with that of the farthest away source, and overlap two control periods for the other sources.

The control packet can be used to measure the propagation delay between intermediate nodes and the sender. In [7], it is only stated that the delay (measured in multiples of the control frequency interval) is computed using time-stamps included in the control packets, while [8] and [6] assume that the delay is given.

In our mechanism, each intermediate node computes the delay between itself and the source. Upon receipt of the control packet, the node registers the time it passes the packet upstream. As the sender's acknowledgement is then received by each node, it can compute the round-trip (propagation) delay between itself and the source. This is another advantage of having the controller at the bottleneck. It would not be practical to have the source measure the round-trip delays, as it would a) not be a scalable mechanism, and b) introduce too much overhead since the source would have to send a control packet separately to each intermediate node, receive its corresponding acknowledgment, and compute the delay using time-stamp information.

We proceed as before and set $x_R^{(k+2)}$ equal to x_R^*.

$$x_R^* = x_R^{(k+1)} + \sum_{j=1}^{J} d_j \mu_j^{(k)} + \sum_{j=1}^{J} (t_0 - d_j)\hat{\mu}_j^{(k+1)} - t_0 \hat{\mu}_R^{(k+1)} \tag{9}$$

which leads to a single equation with J unknowns:

$$\sum_{j=1}^{J} (t_0 - d_j)\hat{\mu}_j^{(k+1)} = x_R^* - x_R^{(k+1)} - \sum_{j=1}^{J} d_j \mu_j^{(k)} + t_0 \hat{\mu}_R^{(k+1)} \tag{10}$$

To solve equation (10), we can apply a fairness criterion and have all J nodes transmit at the same rate, say $\mu_f^{(k+1)}$, and then solve for this single unknown, which, after replacing $x_R^{(k+1)}$ using equation (8), yields:

$$\mu_f^{(k+1)} = \frac{x_R^* - x_R^{(k)} - \sum_{j=1}^{J} d_j \mu_j^{(k-1)} + t_0[\hat{\mu}_R^{(k+1)} + \hat{\mu}_R^{(k)} - \sum_{j=1}^{J} \mu_j^{(k)}]}{\sum_{j=1}^{J} (t_0 - d_j)} \tag{11}$$

The problem with the fairness criterion is that it assigns equal transmission rates to all J nodes regardless of their capabilities or requirements. A more sophisticated scheme is to have each of the J nodes designate a desired target transmission rate, say λ_j^*, selected according to some criteria[††]. The target values could be communicated to the downstream controller in the control packet. The controller would then attempt to solve equation (10) with the aim of minimizing the difference between the allocated transmission rate and the target transmission rate over all J nodes. We can express this mathematically as:

$$Min\left(\sum_{j=1}^{J} (\lambda_j^* - \mu_j^{(k+1)})^2\right) \qquad \text{subject to satisfying equation (10)}$$

The above term is minimized (see [12] for proof) for:

$$\mu_j^{(k+1)} = \lambda_j^* - \frac{(t_0 - d_j)K}{\sum_{j=1}^{J} (t_0 - d_j)} \tag{12}$$

where K is equal to the right hand side of equation (10).

Equation (12) is biased towards more remote transmitters. The denominator of the second

[††] In the hop-by-hop case, the targets would simply be μ_i^{max}.

term on the right is the same for all transmitters, as is the term K, but $t_0 - d_j$ is smaller for more remote transmitters, hence their allocated rates are closer to their desired target rates. This is a positive feature, since more remote receivers are slower to adapt to network changes at node R anyway. So it is beneficial to have them deviate less from their desired targets, and have the closer, faster responding, transmitters do most of the work to achieve the desired buffer occupancy.

5 CONCLUSIONS

We presented a rate-based congestion control scheme, with both an end-to-end approach and a hop-by-hop approach. The scheme differed from traditional techniques in that control was distributed among all the hosts on the path from the source to the receiver (as opposed to a source-based approach). This led to a more accurate and scalable design. Compared with similar schemes, our design was better able to deal with varying bottlenecks, preventing buffer overflow during the transition periods. We also showed how the same control packet could be used to relay control information and serve as a means of measuring the inter-node delay for all intermediate nodes. Finally, we showed how the scheme could be extended to multipoint scenarios.

REFERENCES

[1] L. Benmohamed and S. M. Meerkov. Feedback Control of Congestion in Packet Switching Networks: The Case of a Single Congested Node. *IEEE/ACM Transactions on Networking*, 1(6):693–708, December 1993.

[2] J. Bolot and T. Turletti. A Rate Control Mechanism for Packet Video in the Internet. In *Proceedings of the IEEE INFOCOM*, pages 1216–1223, Toronto, Canada, June 1994.

[3] J. Bolot, T. Turletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. In *Proceedings of the ACM SIGCOMM*, pages 58–67, London, UK, August 1994.

[4] F. Bonomi and K. Fendick. The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service. *IEEE Network Magazine*, 9(2):25–39, March/April 1995.

[5] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister, H. Rudin, and R. Williamson. A Survey of Light-Weight Transport Protocols for High-Speed Networks. *IEEE Transactions on Communications*, 38(11):2025–2039, November 1990.

[6] Kerry W. Fendick, Manoel A. Rodrigues, and Alan Weiss. Analysis of a Rate-Based Control Strategy with Delayed Feedback. In *Proceedings of the ACM SIGCOMM '92*, pages 136–148, Baltimore, MD, August 1992.

[7] H. Kanakia, P.P. Mishra, and A. Reibman. An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport. In *Proceedings of the ACM SIGCOMM '93*, pages 21–31, San Fransisco, CA, September 1993.

[8] Srinivasan Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of the ACM SIGCOMM '91*, pages 3–15, Zürich, Switzerland, September 1991.

[9] H. T. Kung and K. Chang. Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks. In *Proceedings of the IEEE INFOCOM*, pages 239–252, Boston, MA, April 1995.

[10] D. E. McDysan and D. L. Spohn. *ATM: Theory and Applications*. McGraw Hill, 1995.

[11] P.P. Mishra and H. Kanakia. A Hop by Hop Rate-based Congestion Control Scheme. In *Proceedings of the ACM SIGCOMM '92*, pages 112–123, Baltimore, MD, August 1992.

[12] S. Pejhan. *Protocols for Multipoint, Multimedia Communications.* PhD thesis, Columbia University, 1995.

[13] K. K. Ramakrisnan and P. Newman. Integration of Rate and Credit Schemes for ATM Flow Control. *IEEE Network Magazine*, 9(2):49–56, March/April 1995.

[14] C. Roche and N. T. Plotkin. The Converging Flows Problem: an Analytical Study. In *Proceedings of the IEEE INFOCOM*, pages 32–39, Boston, MA, April 1995.

[15] R. Yavatkar and L. Manoj. Optimistic Strategies for Large-Scale Dissemination of Multimedia Information. In *ACM Multimedia '93*, pages 12–20, Anaheim, CA, August 1993.

[16] N. Yin and M.G. Hluchyj. A Dynamic Rate Control Mechanism for Source Coded Traffic in a Fast Packet Network. *IEEE Journal on Selected Areas in Communications*, 9(7):158–181, September 1991.

[17] N. Yin and M.G. Hluchyj. On Closed Loop Rate-Control for ATM Cell Relay Networks. In *Proceedings of the IEEE INFOCOM*, pages 99–108, Toronto, Canada, June 1994.