Improving Multirate Congestion Control Using a TCP Vegas Throughput Model*

Anirban Mahanti
Dept. of Computer Science
University of Calgary
2500 University Dr. N.W.
Calgary, AB T2N 1N4 Canada
mahanti@cpsc.ucalgary.ca

Derek L. Eager
Dept. of Computer Science
University of Saskatchewan
57 Campus Dr.
Saskatoon, SK S7N 5A9 Canada
eager@cs.usask.ca

Mary K. Vernon
Computer Sciences Dept.
University of Wisconsin, Madison
1210 West Dayton Street
Madison, WI USA 53706-1685
vernon@cs.wisc.edu

Abstract

This paper describes Adaptive Vegas Multicast Rate Control (AVMRC), an equation-based multirate congestion control protocol that uses a recently proposed TCP Vegas throughput model. The AVMRC protocol exhibits TCP Vegas-like behavior and has the key advantage of operating without inducing packet losses when the bottleneck link is lightly loaded.

The AVMRC protocol incorporates a new technique for dynamically adjusting the Vegas threshold parameters based on measured characteristics of the network. This technique implements fair sharing of network resources with widely deployed versions of TCP such as TCP Reno and might be fruitfully incorporated in TCP Vegas itself to aid in its incremental deployment.

To evaluate the benefits of Vegas-like congestion control, the performance of AVMRC is compared using simulations to that of an analogous protocol that is based on a TCP Reno throughput model. Additional design choices in AVMRC are evaluated along four primary dimensions, namely synchronization policy, delay measurement, data transmission policy, and protocol reactivity.

Keywords: multirate congestion control, equation-based protocols, TCP Vegas, performance evaluation

1 Introduction

In a bulk data or streaming media delivery application using (IP or application layer) multicast, different clients may experience different loss rates, packet delays, and available bandwidth to the server. To provide the best feasible service to each client, the server may transmit data using multiple multicast channels, thus allowing each client to adjust its reception rate by varying the set of channels it receives. For example, a streaming media server can use a cumulative layered encoding, in which a base layer and a number of enhancement layers are each delivered on a different multicast channel. Each client receives the base layer and a number of enhancement layers that depends on its sustainable reception rate.

Protocols that determine each client's (dynamic) reception rate are known as *multirate* congestion control protocols. Note that the set of possible reception rates is constrained by the limited number of multicast channels and the (typically fixed) transmission rate on each channel. In the context of streaming media delivery, for example, the number of channels and their rates are determined by the

*To appear in the **Computer Networks Journal**. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada, and by the National Science Foundation under grant ANI-0117810.

media encoding. Within this constraint, the goal of a "TCP-friendly" multirate congestion control protocol is that each client have reception rate similar to that of a TCP connection under similar conditions [4].

A number of TCP-friendly multirate congestion control protocols have been proposed [7, 8, 14, 16, 18, 23, 29, 31, 32]. In the earliest such protocol, Receiver-driven Layered Multicast (RLM) [23], each client attempts to determine an appropriate reception rate through experimentation, as follows. When congestion is observed, as evidenced by packet loss, a layer is dropped. When there is no observed congestion, the client attempts to increase its reception rate by adding a layer, with the frequency of such "join experiments" tuned according to the specific layer and the client's history of past attempts to add that layer. A "shared learning" mechanism is used, in which other clients are notified of join experiments so as to enable them to determine in some cases that a layer subscription is unsustainable without having to attempt the experiment themselves.

RLM has been found to result in TCP unfairness and instability in some scenarios [15]. Several subsequent protocols, including RLC [32], FLID-DL [7], and STAIR [8], attempt to improve fairness by more closely emulating conventional TCP's additive increase, multiplicative decrease congestion control algorithm. A potential disadvantage of this approach, however, is that it can result in highly variable reception rates, which may be undesirable for applications such as streaming media delivery.

Equation-based congestion control protocols use a TCP throughput model that predicts the throughput a TCP connection would achieve as a function of parameters such as round trip time and packet loss rate. Each client applies the throughput model with measured values of the model parameters to compute its target reception rate. Such protocols have the potential to be TCP-friendly and yet provide clients with smoother reception rates than with an additive increase, multiplicative decrease approach.

Previously proposed equation-based multirate congestion control protocols have used a TCP Reno throughput model [14, 18, 29, 30, 31]. Since TCP Reno relies exclusively on packet loss to signal congestion, a characteristic that is reflected in the throughput model, these previous equation-based protocols have the disadvantage that they will ramp up a client's reception rate until packet loss is induced. An alternative approach, based on use of a "packet pair" technique to estimate the available bandwidth on the path between the server and client, is used in the PLM protocol [16]. Although this protocol does not require packet loss to determine an appropriate target client reception rate, the packet pair technique does require that the routers on the path between the server and the client implement a fair queuing scheduling discipline.

This paper develops a new equation-based multirate congestion control protocol, called Adaptive Vegas Multicast Rate Control (AVMRC). A key goal is to estimate the available reception bandwidth at each client without inducing packet loss, and thus to reduce the packet loss rate as well as the variation in the reception rate of each client. To this end, each AVMRC client uses a recently proposed TCP Vegas throughput model [28] to compute its bandwidth share. Another key goal is to share bandwidth fairly with widely deployed versions of TCP such as TCP Reno. The previously defined TCP Vegas protocol uses two *static* threshold parameters (α and β) that determine the target number of packets queued in the network for a TCP Vegas flow [5, 6]. Depending on the values of these static parameters and the network conditions, the Vegas protocol is generally either too aggressive or not aggressive enough when operating in a network that also has some TCP Reno flows [28]. The AVMRC protocol dynamically varies the values of the Vegas threshold parameters that it uses in the Vegas throughput model, based on the network conditions that are reflected in the measured packet loss rate and queuing delay. With the adaptive parameters, an AVMRC flow shares bandwidth fairly with other types of flows, including TCP Reno flows and other rate-controlled flows, over a wide range of network conditions. This new technique for adapting the threshold parameters might be fruitfully incorporated in TCP Vegas itself to aid in its incremental deployment.

To evaluate the benefits of using the Vegas throughput model for congestion control, the perfor-

mance of AVMRC is compared with that of an analogous protocol that uses a TCP Reno throughput model. Other key design choices in AVMRC are also evaluated, along the four dimensions of synchronization policy, delay measurement, packet pacing policy, and protocol reactivity. The main results of the performance study are:

- AVMRC has significant advantages in some common scenarios, in comparison to TCP Reno-based protocols, including operation without inducing packet losses when the bottleneck link is lightly loaded.
- Dynamically varying the Vegas α and β parameters greatly enhances the TCP fairness of AVMRC.
- Coherency among clients behind a common bottleneck link can be achieved using a "weak synchronization" approach in which the times at which clients can add layers are coordinated, but in contrast to previously proposed schemes [7, 32], the layers that may be added at these times are not. Clients thus have greater freedom in matching their reception rate to the target rate computed from the throughput model.
- Parameterizing the throughput model using the sum of the measured average queuing delay along the path from the server and a fixed "aggressiveness constant", rather than measured round trip times, can yield an attractive combination of fairness, reactivity, and efficiency.
- Explicitly tracking both short-term and long-term averages of loss rate and delay can enable clients to achieve both good reactivity and good stability in their channel subscriptions.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of the foundations of equation-based congestion control. The AVMRC protocol is presented in Section 3. Section 4 outlines our performance evaluation methodology. Section 5 presents performance comparisons between AVMRC, the corresponding protocol based on a Reno throughput model (RMRC), and AVMRC without adaptive choice of the α and β parameters (VMRC). Section 6 evaluates other design choices in AVMRC. Conclusions are presented in Section 7.

2 Foundations of Equation-Based Congestion Control

A number of equation-based congestion control protocols have been proposed [9, 14, 18, 29, 30, 31]. These prior protocols have utilized a TCP Reno throughput model. The most commonly employed Reno throughput model, and the TCP Vegas throughput model on which the proposed AVMRC protocol is based, are reviewed in Section 2.1. Prior techniques for online estimation of model parameters are discussed in Section 2.2.

2.1 TCP Throughput Models

Several models have been proposed that predict the steady state throughput of a long duration TCP Reno flow [12, 22, 25]. In a commonly used model [25], an approximation Λ_{reno} for the throughput of a long duration TCP Reno flow, in packets per second, is given by

$$\Lambda_{reno} = \frac{1}{RTT\sqrt{\frac{2p}{3}} + TO \min\left(1, 3\sqrt{\frac{3p}{8}}\right) p\left(1 + 32p^2\right)}, \tag{1}$$

where RTT is the average round trip time in seconds, TO is the TCP retransmission timeout value, and p is the steady state loss event rate. Here it is assumed that the receiver's buffer space advertisements

are not sufficiently restrictive to limit the congestion window size, and that an acknowledgment is sent for each received packet. These assumptions can be relaxed at the cost of a somewhat more complex model [25].

Unlike TCP Reno, which induces losses to estimate available bandwidth, the TCP Vegas congestion control mechanism attempts to detect congestion in the network before packet loss occurs. TCP Vegas uses increases in queuing delay to detect congestion and attempts to maintain, on average, between α and β unacknowledged packets queued in the network, where α and β are the "threshold" parameters of the protocol. For the case in which packet losses are negligible, an approximation $\Lambda_{vegas}^{no-loss}$ for the throughput of a long duration TCP Vegas flow, in packets per second, is given by

$$\Lambda_{vegas}^{no-loss} = \frac{\beta}{RTT - baseRTT} = \frac{\beta}{\Delta}, \tag{2}$$

where $baseRTT$ is the minimum observed round trip time for the flow and $\Delta = RTT - baseRTT$ [28]. Note that in this case of negligible packet loss, TCP Vegas throughput (at least, as predicted by the above throughput model) does not have any round trip time bias, in the sense that it depends *only* on the average queuing delay observed along the path from the sender.

For the case in which packet losses (and possibly timeouts) are significant, an approximation Λ_{vegas}^{loss} for the throughput of a long duration TCP Vegas flow, in packets per second, is given by

$$\Lambda_{vegas}^{loss} = \frac{(n+1)(\frac{1-p}{p} + W)}{NRTT + TO}, \tag{3}$$

where $W = \frac{RTT(\alpha+\beta)}{2\Delta}$, $n = \frac{1-p_{to}}{p_{to}}$, $p_{to} = \min(1, \frac{p+p(1-p)(1-(1-p)^{W-1})}{1-(1-p)^W})$, and $N = 2logW + (n+1)\frac{1-p}{pW} + (1+\frac{n}{4})\frac{W}{8} + \frac{9n}{8} - \frac{11}{4} + \frac{4-2^{logW}}{W}$ [28]. Note that in this case, the throughput model predicts some degree of round trip time bias. Previous studies have found the bias to be lower than with TCP Reno, however [24, 28].

2.2 Online Estimation of Throughput Model Parameters

Prior equation-based congestion control methods are based on a TCP Reno throughput model, and thus need to obtain estimates of the average round trip time (RTT), the loss event rate (p), and (for the model of equation 1), the TCP retransmission timeout value (TO). The latter quantity is typically estimated as a fixed multiple of RTT.

Although obtaining round trip time measurements is relatively straightforward in the context of a two-way packet flow between a single pair of systems, this task is more difficult to accomplish in a scalable fashion in the multicast context. Several schemes involving control packet feedback to the server have been proposed [29, 30, 31]. Other protocols have used a fixed value in place of a measured average round trip time [7, 32]. This approach has the disadvantage of being either aggressive or not as aggressive as a TCP flow under similar conditions, depending on the round trip times being witnessed by the TCP flow.

The Exponential Weighted Moving Average (EWMA) technique is widely used for obtaining estimates of average round trip time. After each round trip time measurement, an updated estimate of the average round trip time (RTT_{new}) is computed as a weighted average of the previous estimate (RTT_{old}) and the new measurement (RTT_i),

$$RTT_{new} = \alpha RTT_{old} + (1-\alpha)RTT_i, \tag{4}$$

where α is a parameter of the technique. Note that with EWMA, the weight given to a round trip time measurement decays exponentially with the age of that measurement.

Estimates of the loss event rate p at a client can be obtained using the Average Loss Interval (ALI) method [9]. This method computes p as the inverse of the average length in number of packets of a loss free interval (i.e., interval between loss events). The latter quantity is estimated using a weighted average of the sizes of recent loss free intervals, specifically as $\max(\bar{s}, \bar{s}_{new})$, where

$$\bar{s} = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i} \qquad \bar{s}_{new} = \frac{\sum_{i=0}^{n-1} w_i s_i}{\sum_{i=0}^{n-1} w_i}. \tag{5}$$

Here s_i denotes the number of packets received in the i^{th} most recent loss free interval, s_0 denotes the number of packets received since the most recent loss event, and the weights w_i are chosen such that the most recent loss free intervals receive equal weight while the weights of older intervals gradually decrease to zero. Previous work has used $n = 8$, with weights $1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2$ [9]. Note that the number of packets received since the most recent loss event impacts the estimated loss rate only when this number is relatively large and would decrease the estimated loss rate.

An important issue with both ALI and EWMA concerns the weights that are given to the most recent samples. Better noise attenuation can be achieved by using a high value for the parameter α in EWMA and by using a large n and/or more balanced weights in ALI, but at the cost of diminished reactivity. Placing greater weight on the most recent samples can yield better reactivity, but may result in oscillatory target reception rates when the resulting values are used in a TCP throughput model for equation-based congestion control.

3 The AVMRC Protocol

The main innovation in AVMRC is its use of the TCP Vegas throughput model described in Section 2.1 to compute the target reception rate at each client, with the values of the Vegas threshold parameters α and β dynamically varied so as to achieve fair bandwidth sharing with other types of flows.

Other innovations in AVMRC concern the rules by which clients change their channel subscriptions. First, AVMRC uses both *short-term* and *long-term* averages of delay and loss event rate measures when computing target reception rates using the TCP Vegas throughput model. This is motivated by the observation that clients should generally be cautious in changing their reception rate (i.e., their set of subscribed channels), so as not to overreact to transient changes in network conditions and to avoid oscillation, and yet be quick to react when an increase in their reception rate causes congestion.

Second, rather than using an average of measured round trip time values for the *RTT* parameter in the TCP Vegas throughput model, AVMRC uses the sum of the measured average queuing delay along the path from the server and a fixed "aggressiveness constant" (so named since it has some modest impact on the aggressiveness of the protocol). Using measured round trip times has the disadvantage that clients behind the same bottleneck link, but at differing network distances from the server, may compute significantly different target reception rates and sets of channels to listen to, resulting in somewhat inefficient use of the bottleneck link bandwidth. At the other extreme, using a simple fixed value can result in poorer fairness and reactivity.

Third, AVMRC uses a light-weight approach to coordinate clients behind a common bottleneck, termed here *weak synchronization*. Coordination is desirable so as to limit the churn in channel subscriptions that might result as failed attempts by one client to increase its reception rate causes congestion at the bottleneck that prompts other clients to unnecessarily drop their reception rates. Prior protocols have used more complex mechanisms to achieve this coordination [7, 23, 32].

The technique used in AVMRC to adaptively set the Vegas threshold parameters α and β is described in Section 3.1. Section 3.2 describes the delay and loss event rate measurements employed in AVMRC and the method by which short-term and long-term averages are calculated. The rules for changing

Table 1: AVMRC Protocol Notation

Description	Name	Default Value
Time slot duration (protocol activation granularity)	τ	0.1 seconds
Minimum time slots between channel additions	n_{add}	20 *slots*
Threshold for quick drop	n_{drop}	20 *slots*
Start-up phase duration	$n_{startup}$	50 *slots*
Add hysteresis parameter	γ_1	0.5
Drop hysteresis parameter	γ_2	0.5
Aggressiveness constant	R	0.1 seconds
Range of TCP Vegas threshold parameters	$[\beta_{min}, \beta_{max}]$	
Short-term average one-way queuing delay	δ_{st}	
Long-term average one-way queuing delay	δ_{lt}	
Short-term average loss event rate	p_{st}	
Long-term average loss event rate	p_{lt}	
Average reception rate	B_{recp}	
Expected reception rate	B_{curr}	

channel subscriptions are described in Section 3.3. Additional protocol rules are needed for use when a client first joins a multicast session. These are outlined in Section 3.4. Required notation is summarized in Table 1.

3.1 Adaptive Setting of the TCP Vegas Threshold Parameters

Depending on the choice of the TCP Vegas threshold parameters α and β, TCP Vegas can be either too aggressive, or not aggressive enough, when competing against TCP Reno flows for network bandwidth share [28]. The AVMRC protocol attempts to achieve fair sharing by adaptively setting these parameters to the most aggressive values that would still allow a TCP Vegas flow to achieve *stable backlog*; i.e., to reach a stable throughput between loss events. The stable throughput value is a function of β and can be computed from equation 2. An analytic condition for stable backlog to be achievable has been derived as [28]:

$$p \leq \frac{32}{7W^2 - 36W + 32}, \tag{6}$$

where W is the congestion window size (in packets) that corresponds to the stable throughput value (i.e., $W = \frac{\beta}{\Delta} RTT$). AVMRC adaptively chooses the value of β so that equality is achieved in relation 6, when feasible. The value of α is simply chosen equal to β, as has been suggested in prior work [28].

More precisely, each AVMRC client initializes β to a parameter β_{min}. Each client times its activation of the AVMRC protocol according to a "time slot" of duration τ (not only for computation of new β values, but also for other operations described subsequently). Every τ seconds, the client computes a β value such that equality is achieved in relation 6, using its long-term averages of delay and loss event rate. The client then updates its β value using a weighted average of the computed and the old values, within the constraints of the minimum value β_{min} and a maximum value β_{max}. The simulation implementation for which results are reported here uses 0.05 as the weight for the computed value of β and 0.95 as the weight for the old value, and somewhat arbitrarily sets the values of β_{min} and β_{max} to the number of packets transmitted during a time τ on the channels received by a client with the minimum possible reception rate (channel subscription), and the maximum possible reception rate, respectively.

3.2 Estimating Average Delay and Loss Event Rate

TCP throughput models (including the TCP Vegas model) typically define the loss event rate such that multiple packet losses from a single window of packet transmissions correspond to a single loss event, as the TCP sender will usually only reduce its congestion window size once in reaction to these losses. Thus, in AVMRC, closely-spaced packet losses should also constitute only a single loss event. Experimental results suggest that this notion can be made precise using the same time scale used elsewhere in the protocol. The simulation implementation for which results are reported here considers any packet loss that is observed at the client within time τ of the first loss of a loss event to be part of the same loss event. The long-term average loss event rate p_{lt} is computed using the ALI technique with n value and weights as used in [9], and with a loss free interval defined as the period from the last packet loss of one loss event, until the first packet loss of the next loss event. The short-term average loss event rate p_{st} is simply taken as the inverse of the number of packets received in the most recent loss free interval, s_0 (with p_{st} set to 1 when $s_0 = 0$).

The delay parameters required by the TCP Vegas throughput model used in AVMRC are RTT, TO, and Δ, where RTT is the average round trip time, TO is the TCP retransmission timeout duration, and Δ is the portion of the average round trip time due to packet queuing delay. As in prior work [9], TO is approximated by $4RTT$, so only RTT and Δ need be determined. The AVMRC client does not attempt to estimate either of these quantities as defined. Instead, in place of Δ, AVMRC uses an estimate of the average one-way queuing delay from the server to the client, denoted here by δ. Assuming that it is most important to react to congestion along the path from server to client, rather than also on the reverse path, this choice should yield reasonable behavior, and has the advantage of being readily measured at the client, as described below. In place of RTT, AVMRC uses the sum of δ and a fixed value R, thus facilitating the computation of a common target reception rate by clients behind a common bottleneck, while also retaining the TCP Vegas throughput sensitivity to congestion along the forward path.

The short-term and long-term average one-way queuing delays, δ_{st} and δ_{lt}, are computed from short-term and long-term averages, respectively, of the *time stamp difference* (TSD). For each packet received by a client, TSD is defined as the difference between the value of a timestamp inserted in the packet by the server, reflecting the (server's clock) time at which the packet was transmitted, and the (client's clock) time at which the packet was received. A short-term average TSD is computed using the EWMA technique. In the simulation implementation for which results are reported here, weights of 0.25 and 0.75 are used for the most recent sample and for the previous value of the average, respectively. A long-term average TSD is computed by applying EWMA to the values of the short-term average, once every τ seconds. For the results reported here, weights of 0.05 and 0.95 are used for the current short-term average TSD and for the previous long-term average value, respectively. Values for δ_{st} and δ_{lt} are computed by taking the difference between the short-term average and long-term average TSD values, respectively, and the minimum TSD observed by the client over all packets. Note that any difference between the server and client clocks is canceled out when this subtraction is taken, at least under the assumption that these clocks have constant difference. To ameliorate the impact of changing clock skew as well as of topology changes in the path between server and client, for long-duration sessions it may be desirable to compute the minimum TSD over a moving window.

3.3 Changing Channel Subscriptions

Three key issues in AVMRC concern 1) the frequency with which clients should be able to add/drop channels, 2) coordination of the channel subscription changes of clients behind a common bottleneck, and 3) the conditions under which channel subscription changes should occur.

With respect to the first of these issues, the objective is to achieve both good responsiveness and

good stability.

In AVMRC, a client can add at most one channel every n_{add} time slots, except during a start-up phase as described in Section 3.4. To retain quick responsiveness to increases in congestion, channel drops are allowed to take place at every time slot. Allowing frequent changes in channel subscriptions may result in oscillation, particularly since only coarse-grained reception rate changes are possible, and therefore only an approximate match can be achieved between the target and actual reception rates. The AVMRC protocol uses a *weak synchronization* approach to address the second issue of coordination of clients behind a common bottleneck. Specifically, the server inserts markers in the data stream, or uses a packet field, so as to identify each time slot, and, in particular, the time slots (every n_{add}'th) at which channel additions are permitted. Note that in general different clients will not receive a time slot marker at exactly the same time, owing to varying network delays. As long as $n_{add}\tau$ is relatively large compared to the likely network delay variations, however, additions of channels will be clustered relatively closely together in time. In previously proposed schemes that also use a time slotting mechanism, each channel has a different, statically determined frequency of time slots at which a client can add the channel [7, 32]. In AVMRC, in contrast, a client can add *any* channel at each of the n_{add}'th time slots. This design choice decouples the issue of how frequently clients should be able to change their channel subscriptions, from the issue of how subscription changes should be constrained so as to ensure TCP-friendly behavior, with the latter issue being handled in AVMRC solely using the target reception rates computed from the TCP Vegas throughput model.

Specifically, clients determine target reception rates Λ_{st} and Λ_{lt} by using the TCP Vegas throughput model with the short-term and long-term averages, respectively, of delay and loss event rate. Assume a cumulative subscription approach wherein a client subscribes to a channel i only if the client also subscribes to all channels j for $1 \leq j < i$. Let B_i denote the bandwidth required to support subscription to channels 1 through i. When channel additions are permitted, a client subscribed to channels 1 through i adds a subscription to channel $i+1$ if $\min[\Lambda_{st}, \Lambda_{lt}] \geq B_{i+1} + \gamma_1(B_{i+1} - B_i)$. Using the minimum of the target reception rates allows channel additions only when there is a high probability of the addition being successful. That is, the target reception rate computed using the long-term averages indicates that there is available bandwidth, and that computed using the short-term averages indicates that no new congestion has been observed recently.

The rule for decreasing the subscription level is designed to allow early channel drops following a channel addition that is deemed unsustainable, and/or if the network parameters reflect persistent congestion. Ascertaining whether or not an increase in the subscription level results in congestion is particularly important due to the coarse granularity of the possible bandwidth changes. Therefore, if the client added a new channel in the recent n_{drop} time slots without an intervening channel drop, the subscription level is dropped to $i-1$ if $\min[\Lambda_{st}, \Lambda_{lt}] < B_i - \gamma_2(B_i - B_{i-1})$; otherwise, the condition for dropping a channel is $\max[\Lambda_{st}, \Lambda_{lt}] < B_i - \gamma_2(B_i - B_{i-1})$. The hysteresis parameters γ_1 and γ_2 are used to further decrease or eliminate oscillations in the subscription level.

An important design criterion for the add/drop rules is that all clients behind the same bottleneck should use the same information when making changes in their channel subscriptions, to the extent possible, particularly with respect to channel additions, and when dropping a channel that the client has not recently added. Mechanisms based heavily on use of local information can impede and even prohibit coherency among clients behind a common bottleneck link, as clients may invoke them at widely differing times. An example of a mechanism based on local information is the previously proposed use of a *dead period* following a channel drop, during which time a client does not make any further subscription changes in an attempt to avoid unnecessary multiple drops [32].

3.4 Start-up Behavior

When a client first joins a multicast session, reliable estimates of average delay and loss event rate are not initially available, and the throughput model cannot be used to determine a target reception rate. Instead, AVMRC uses a *slow start* approach analogous to that used in TCP. Assuming again a cumulative subscription approach, a client initially listens only to channel 1. The client is permitted to increase its subscription level by adding one channel per time slot provided the previous channel addition appears to be sustainable; specifically, provided that the average reception rate in the previous time slot, B_{recp}, is at least 95% of the total rate of the current subscribed channels, B_{curr}.

Slow start is terminated when the above test fails (B_{recp} is less than 95% of B_{curr}), or when packet loss is observed, or when the maximum subscription level is reached. In either of the former two cases, the reception rate is reduced by dropping the highest subscribed multicast channel. When slow start is terminated prior to packet loss, a reliable estimate of the average loss event rate is not available, and, therefore, equation (2) is used to obtain target rate estimates, until packet loss is observed.

Recall that during steady state, a client may attempt channel additions only once every n_{add} time slots. Thus, if the slow start phase is terminated before a client reaches its optimal subscription level, there may be a long delay before it can reach this level. To alleviate this potential problem, a start-up phase is defined covering the first $n_{startup}$ slots after a client has joined a multicast session, during which the above restriction is not enforced. f

4 Performance Evaluation Methodology

Performance evaluation of AVMRC and its various design choices is carried out using the *ns-2* network simulator [27]. To determine the benefits, if any, of using a TCP Vegas throughput model, the performance of AVMRC is compared with that of a protocol essentially identical to AVMRC excepting for its use of a TCP Reno throughput model instead of a TCP Vegas model. This *Reno Multicast Rate Control* (RMRC) protocol is summarized in Section 4.1. The performance of AVMRC is also compared with that of the protocol (VMRC) identical to AVMRC excepting for its use of static, rather than adaptive, values of the TCP Vegas threshold parameters in the throughput model. Sections 4.2, 4.3, and 4.4 describe the application model, network models, and the performance evaluation metrics, respectively, both for the AVMRC/RMRC/VMRC comparison, and for the subsequent evaluation of AVMRC along its other key design dimensions.

4.1 The RMRC Protocol

RMRC uses the same protocol rules as AVMRC, excepting for the throughput model used, and an additional mechanism for computing an average loss event rate to use in the TCP Reno throughput model when a client exits slow start prior to packet loss being observed. In AVMRC, such a client uses equation 2 to compute its target reception rate until the first packet loss is observed and thus an estimate of the average loss event rate can be computed. The RMRC protocol, however, requires an estimate of the average loss event rate immediately upon termination of slow start, as there is no variant of the TCP Reno throughput model that does not require a measure of the loss event rate. Therefore, in RMRC the client computes an average loss event rate that is consistent with the current reception rate, by solving for p in equation 1 with the current reception rate substituted on the left-hand side of this equation (i.e., the rate after slow start is terminated). The inverse of this average loss event rate value is then used to seed the loss interval length history used in the ALI method.[1]

[1] In the simulations, this computed loss interval is assigned to s_1 and n is set to 1. The values assigned to the loss history array s and the number of loss events considered for the loss event rate computation n are updated as loss events occur.

Note that as in AVMRC, in place of the *RTT* parameter in its throughput model, RMRC uses the sum of the measured average one-way queuing delay δ and a fixed value R that determines the aggressiveness of the protocol.

4.2 Application Model

An application with cumulative channel subscription is assumed, such as layered video multicast. The number of channels is fixed at nine, with the data rate of channel 1 fixed at 256 Kbps and the rate of channel i for $i > 1$ fixed so that the total reception rate when subscribed to channels 1 through i is equal to 1.5 times that when subscribed to channels 1 through $i - 1$, yielding possible total reception rates of 256, 384, 576, 864, 1296, 1944, 2916, 4374, and 6561 Kbps.

Server transmissions of packets carrying the application data are scheduled periodically, with a default period as used in most experiments of 100 milliseconds. A bursty packet transmission process such as that assumed here is likely to be used by real servers owing to timer, scheduling, and other overheads. A maximum packet size of 1 KB is used, but smaller packets are also transmitted depending on the amount of data that needs to be sent on each channel, as determined by the data rate of the channel and the scheduling period duration.

4.3 Network Models

The simulations whose results are reported here use a simple dumbbell network topology with a common bottleneck link between all sources and sinks. Each source/sink is connected to the bottleneck link by a high bandwidth access link. The propagation delays of the access links are varied to obtain the desired round trip propagation delay between each source/sink pair. The routers use FIFO queuing with drop-tail queue management. Our experiments generally used a buffer size at the bottleneck router that would allow a maximum of between 50 and 250 milliseconds queuing time. This simple network topology clearly does not capture the complexity of the Internet, but is sufficient to enable the performance comparisons of most interest here. Consistent with the recommendations in [11, 26], a wide spectrum of scenarios with varying types and amounts of background traffic are considered.

Congestion in the Internet can occur at low bandwidth low multiplexing links, such as access links, as well as at relatively high bandwidth high multiplexing interior links, such as at the connection points between Internet access providers. To reflect these cases, both the bottleneck link capacity and the intensity/type of background traffic are varied. For the case in which the bottleneck link is close to the client, a low bandwidth bottleneck link (usually at most 6 Mbps in our experiments) with little or no background traffic is simulated. For the case of interior link bottlenecks, a relatively high bandwidth link (usually at least 10 Mbps in our experiments) with varying intensities of background load consisting of a mix of long duration FTP and on/off HTTP sessions is simulated.

The simulated HTTP sessions use a model similar to that used in previous studies [17, 28]. Each HTTP client sends single packet requests to a dedicated HTTP server. After the requested Web object is received from the server, the client delays for a think time that is exponentially distributed with a mean of 500 milliseconds, and then generates another request. The HTTP response sizes are drawn from a Pareto distribution with mean 48KB and shape 1.2, consistent with the heavy-tailed size distribution of HTTP responses that has been observed in measurement studies [2, 3, 19, 21].

Both HTTP and FTP sessions use the NewReno variant of TCP with a default maximum congestion window size of 64 packets. The maximum packets size is 1 KB, with smaller packets used only for the HTTP requests and potentially for the last packet of each HTTP response. Unless stated otherwise, the round trip propagation delays of the background sessions are uniformly distributed between 20 and 460 milliseconds, consistent with previous studies [1, 11, 13, 28].

In simulations in which multiple long duration flows share a single bottleneck link, systematic discrimination against some connections has been observed [10]. Such *phase effects* rarely occur in the real Internet and are usually considered to be an artifact of unrealistic simulation scenarios, such as having only long duration flows, identical length packets, and no reverse direction traffic [11]. Simulations that include a mix of long and short duration flows with heterogeneous round trip propagation delays, such as those in this study, seldom experience this problem. As an additional precautionary measure, however, all flows are started at slightly different times.

The *ns-2* simulator does not model multicast group join/leave latencies. Thus, clients are able to add/drop channel subscriptions instantaneously. This is not expected to impact the results of the performance comparisons in this paper. Note that the dynamic layering scheme of FLID-DL [7] could be added to AVMRC to cope with large multicast group leave latencies.

4.4 Evaluation Metrics

A multirate congestion control protocol can be assessed on the basis of several criteria such as fairness, use of the available bandwidth, smoothness of the reception rate at each client, induced packet loss, coherency of clients behind a common bottleneck link, and responsiveness to changes in network conditions. Seven performance metrics are used to evaluate the considered protocols with respect to the above criteria:

- **Convergence time**: For clients behind a common bottleneck link that join a multicast session at close to the same time, convergence time is computed as the difference between the time at which the last client joined the session and the time at which all clients reach and sustain the optimal channel subscription level. A longer convergence time implies less efficient use of the available bandwidth.

- **Transient packet loss**: The congestion control protocols considered here allow clients to add a channel once every time slot when operating in the start-up phase. This allows clients to quickly attain the optimal subscription level. The transient packet loss for a multicast session is defined as the percentage of session packets dropped at the bottleneck link, as measured from when a client (or the first of a closely-spaced group of clients) joins a multicast session, until the client(s) exit the start-up phase. This metric quantifies the impact of the protocol start-up behavior on network congestion.

- **Steady state packet loss**: The steady state packet loss of a traffic source is defined as the percentage of packets from that source that are dropped at the bottleneck link, as measured from when all multicast clients have exited their start-up phase. For a multicast session, this metric together with the achieved subscription level and its variability over time is an indicator of the client perceived steady state performance.

- **Average session throughput**: The average session throughput is defined as the average total bandwidth obtained by a multicast session at the bottleneck link as measured from when all clients of the session have exited their start-up phase. This metric can be used to assess fairness as well as the ability of a protocol to exploit the available bandwidth.

- **Average client reception rate**: This metric is used in experiments in which multicast session clients attain differing subscription levels, and is defined as the average reception rate of a client as measured from when the client has exited its start-up phase.

- **Session throughput over time**: For this metric, the average total bandwidth obtained by a multicast session at the bottleneck link is calculated over a moving window of size one second for

the entire simulation run. This metric is used to judge the variability in the subscription level at each client in simulations in which all clients converge to the same subscriptions, and also to illustrate protocol responsiveness to sudden changes in network conditions.

- **Session packet loss over time**: The percentage of session packets dropped at the bottleneck link over a moving window of size one second is calculated for the entire simulation run. This metric can yield insight into the burstiness of packet loss.

With the exception of the last two metrics, which show performance as a function of time over a single simulation run, unless stated otherwise the graphed results for each metric include the average from 10 simulation runs, each using a different seed for the random number generator, along with the observed minimum and maximum values (shown using a vertical bar centered on the mean).

5 Performance Comparisons of AVMRC, RMRC, and VMRC

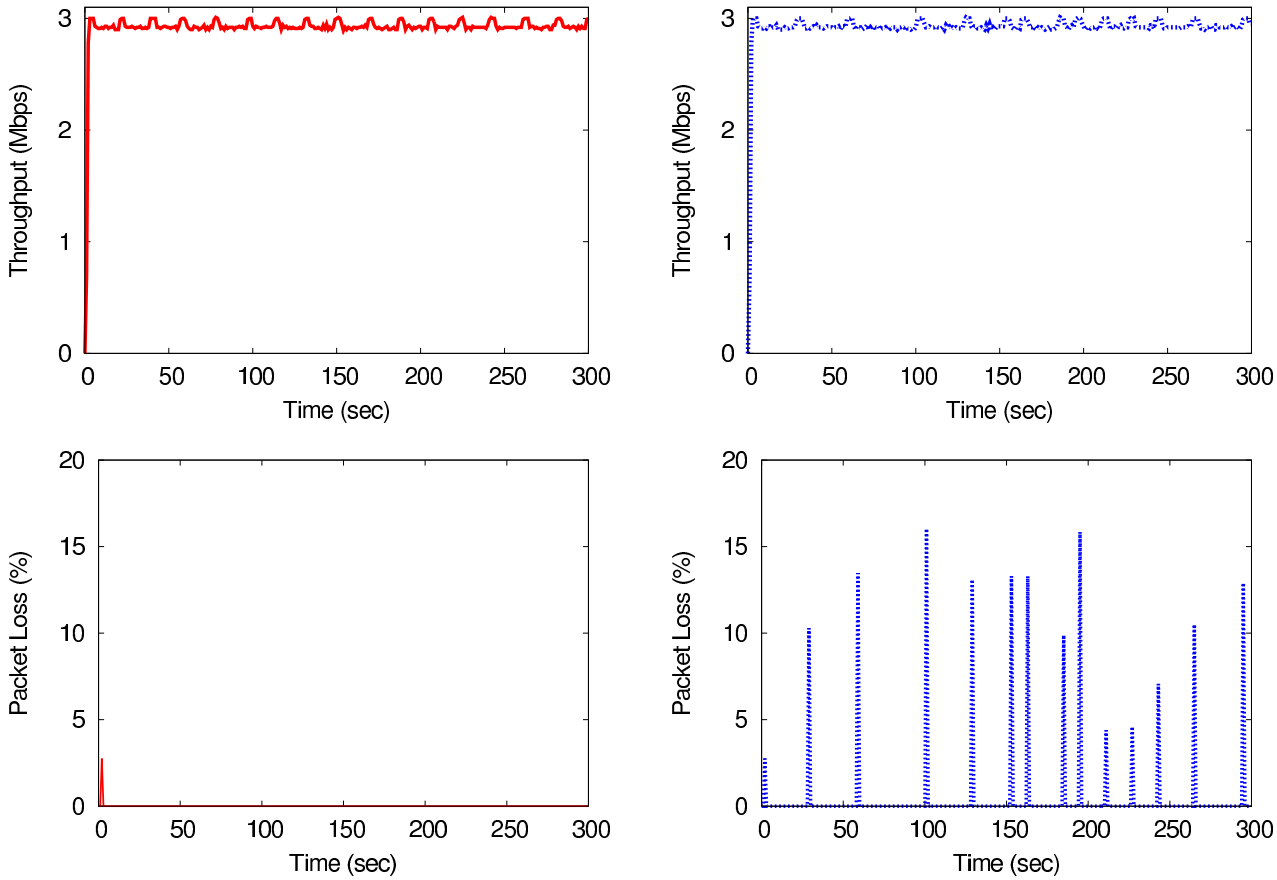
The AVMRC and RMRC protocols are compared using a sequence of increasingly complex scenarios, beginning in Section 5.1 with the simplest case of a single multicast session and no competing background traffic. Section 5.2 considers the case of multiple multicast sessions each using the same protocol (either AVMRC or RMRC), but no competing traffic of other types. Sections 5.3 and 5.4 compare the performance of these protocols when there is competing UDP traffic, and competing TCP traffic, respectively. Section 5.4 also includes comparative results for VMRC.

5.1 No Background Traffic

Protocol performance when there is no competing traffic is important both because it may be indicative of actual performance in low contention environments, and because it can yield insights into protocol behavior that are more difficult to obtain using more complex scenarios.

The workload used in the first experiment consists only of a single multicast session with a single client. The bottleneck link has a capacity of 3 Mbps and router buffer size of 80 packets. The round trip propagation delay between the client and the server is 100 milliseconds. Results from a representative simulation run are shown in Figure 1. With both AVMRC and RMRC, the single client quickly attains the highest sustainable subscription level, and maintains this subscription for the duration of the simulation excepting for periodic attempts to add a channel. The AVMRC controlled session only experiences packet loss at the end of slow start; otherwise, the ability of AVMRC to quickly detect congestion when an attempt is made to increase the reception rate beyond the bottleneck link capacity allows it to operate without inducing packet losses. The RMRC controlled session, however, periodically experiences episodes of high packet loss in which as many as 16% of the packets transmitted over a one second interval are dropped. This is due to the fact that an unsustainable reception rate can only be recognized in RMRC through packet loss.

The next experiment studies performance with varying numbers of clients. The bottleneck link capacity and router buffers are configured as in the previous experiment. The round trip propagation delay from the server to each client is uniformly distributed between 40 and 100 milliseconds. This is a smaller range than is used by default for the background traffic of later experiments, and models a scenario in which the multicast session clients are moderately clustered rather than spread out across the Internet. Clients join the multicast session at uniformly distributed random times between 0 and 5 seconds. All clients attain the highest subscription level possible given the bottleneck link capacity. Figure 2 shows the average, minimum, and maximum convergence times, transient loss rate, and steady state loss rate, from 10 simulation runs, as functions of the number of clients.

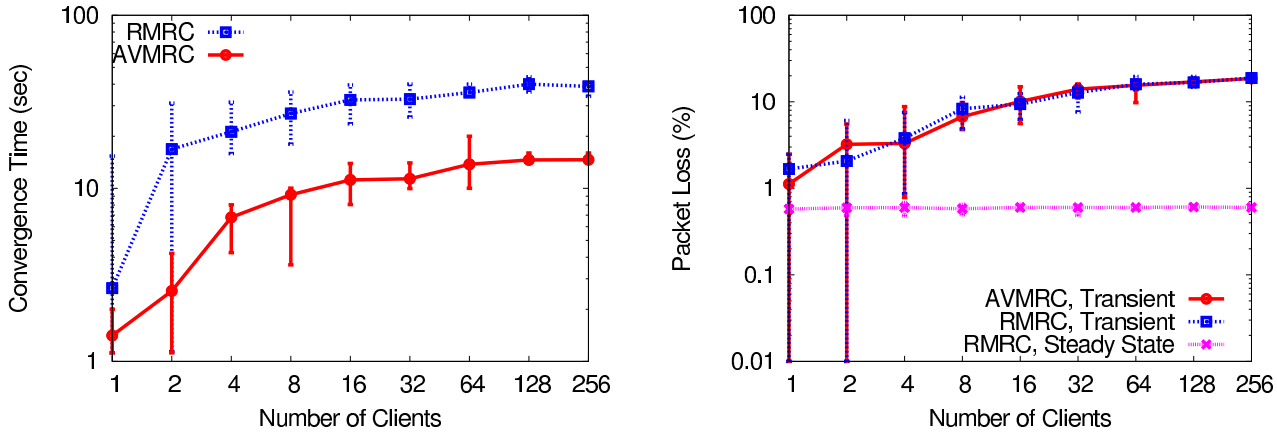


(a) AVMRC (b) RMRC

Figure 1: Throughput and Packet Loss of a Single Client Multicast Session (no background traffic, bottleneck link of capacity 3 Mbps with a router buffer of 80 packets, round trip propagation delay of 100 milliseconds)

As shown in Figure 2(a), the convergence time with AVMRC is about a third of that with RMRC. When slow start is terminated prior to the first observed packet loss, with AVMRC the throughput model of equation (2) is applied with delay estimates that should be similar for clients behind a common bottleneck link. In contrast, in RMRC an estimate of the loss event rate is required, and clients behind a common bottleneck link may use quite different estimates since an estimate is computed based on the client's subscription level at the time slow start was terminated. The ability of AVMRC to more quickly detect when an increase in reception rate is unsustainable, owing to the impact of increased queuing delay on the target reception rate as computed from the TCP Vegas throughput model, also helps reduce convergence time. Note that although the convergence time initially increases with both protocols with increasing numbers of clients, it eventually stabilizes, since multiple clients adding the same channel at roughly the same point in time has much the same impact as a single client adding that channel.

Figure 2(b) shows that the transient packet loss is similar with both AVMRC and RMRC. However, with AVMRC the *steady state* loss percentage is zero (i.e., once all clients have exited the start-up phase, no packet loss is experienced), while with RMRC there is a steady state loss percentage of about 0.6%, independent of the number of clients. As in the single client experiment, the non-zero steady state loss with RMRC reflects the fact that in this protocol an unsustainable reception rate can be recognized only



(a) Convergence Time (b) Packet Loss

Figure 2: Scalability of a Single Multicast Session with Increase in the Number of Clients (no background traffic, bottleneck link of capacity 3 Mbps with a router buffer of 80 packets, round trip propagation delay of clients uniformly distributed between 40 and 100 milliseconds)

once packet loss occurs.

A variety of other experiments with no background traffic have yielded results similar to those described above, including experiments with higher bandwidth bottleneck links [20]. One such experiment scales up the bottleneck link capacity in increments of 1 Mbps, increasing the buffer size by 30 packets with each increment. The multicast channel rates are correspondingly scaled up. The steady state loss percentage with AVMRC remains zero (i.e., no packet loss once all clients have exited the start-up phase), while that with RMRC decreases with increasing transmission rates as can be predicted from the TCP Reno throughput model. Again, all clients reach the optimal subscription level with both protocols, although the convergence is faster with AVMRC than with RMRC.

5.2 Multiple Multicast Sessions

Experiments with multiple multicast sessions with no other traffic provide insight into how AVMRC and RMRC controlled sessions share bandwidth with other such sessions, and the impact of competition among multiple sessions on packet loss rates. In the experiment for which results are shown here, the number of competing multicast sessions is scaled up in increments of two, with the bottleneck link capacity and buffer size increasing by 3 Mbps and 80 packets, respectively, with each increment. The initial configuration has two multicast sessions sharing a bottleneck link of capacity 3 Mbps with a buffer size of 80 packets. Each multicast session has a single client, which joins its respective session at a uniformly distributed random time within the first 5 seconds of the simulation. The round trip propagation delay between each client and its server is uniformly distributed between 40 and 440 milliseconds.

With both AVMRC and RMRC, a fair division of bandwidth is achieved between the competing multicast sessions, with the average throughput of a session being within a factor of two of the average throughput of the other competing sessions [20]. Figure 3 shows the overall average steady state loss percentage as well as that of each session. Unlike in the previous single session experiments, the steady state packet loss with AVMRC is non-zero. This may be mostly due to the bursty data transmission policy used by the servers. However, with AVMRC the packet loss percentage is about a factor of two less than with RMRC.

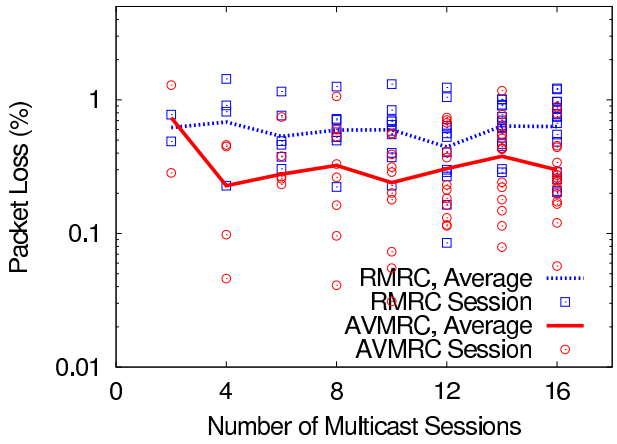
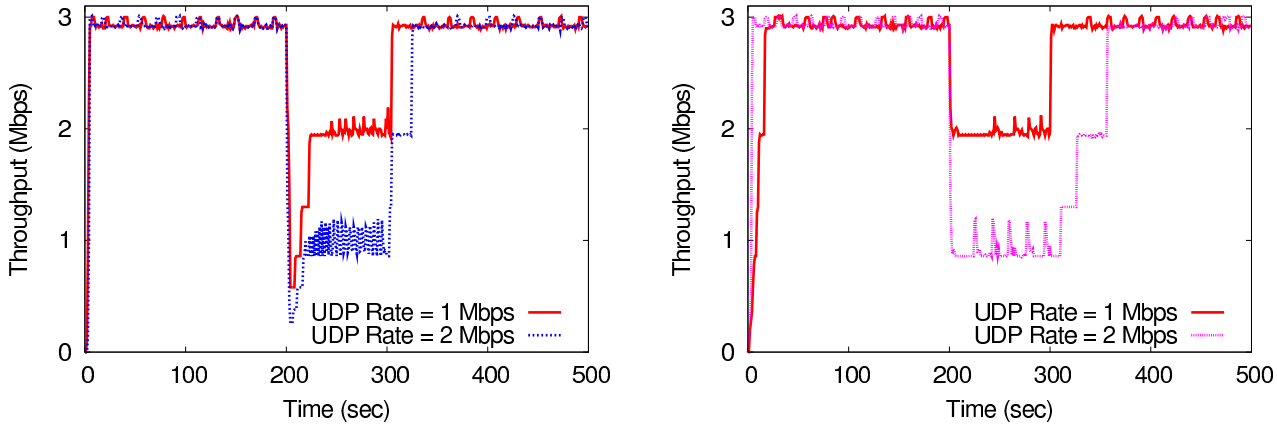


Figure 3: Steady State Packet Loss with Competing Multicast Sessions (no background traffic, bottleneck link of capacity $1.5n$ with a router buffer size of $40n$ packets shared by n multicast sessions, round trip propagation delay between each client and its server is uniformly distributed between 40 and 440 milliseconds)



(a) AVMRC (b) RMRC

Figure 4: Single Multicast Session with a Single Uncontrolled UDP Flow (UDP background traffic, bottleneck link of capacity 3 Mbps with a router buffer of 80 packets, two multicast clients with round trip propagation delay of 50 and 150 milliseconds)

5.3 UDP Background Traffic

In the previous scenarios, all traffic is controlled by the same congestion control protocol, either AVMRC or RMRC. A simple scenario in which this is not the case is that of a single multicast session sharing the bottleneck link with a single UDP flow operating without congestion control.

The first experiment of this type uses a 3 Mbps bottleneck link with an 80 packet buffer. The multicast session has two clients, each of which joins the session at a uniformly distributed random time within the first 5 seconds of the simulation. The round trip propagation delays between the server and the two clients are 50 and 150 milliseconds. The competing UDP flows begin at time 200 seconds, and continues for 100 seconds until time 300 seconds. Figure 4 shows the throughput attained by the multicast session as a function of time, for two different values of the rate of the UDP flow, from one

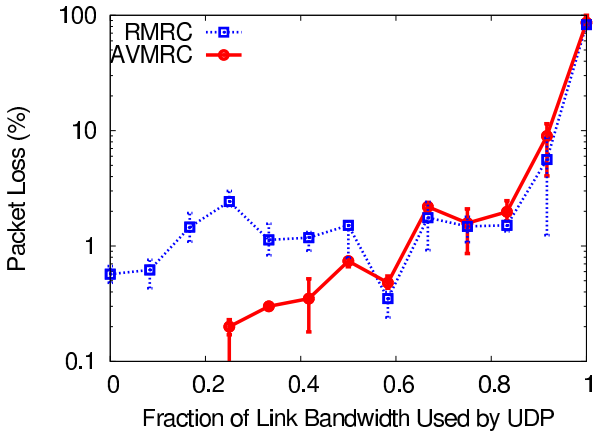


Figure 5: Single Multicast Session Sharing Link with Uncontrolled UDP Flow of Varying Rates (UDP background traffic, bottleneck link of capacity 3 Mbps with a router buffer of 80 packets, round trip propagation delay of clients uniformly distributed between 40 and 100 milliseconds)

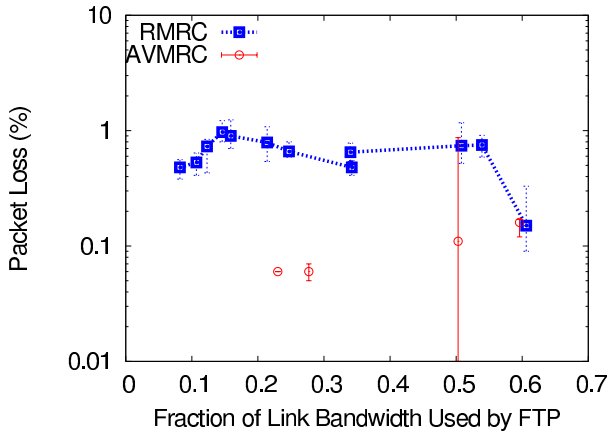


Figure 6: Single Multicast Session with a Single FTP Flow (TCP background traffic, bottleneck link of capacity 3 Mbps with a router buffer of 80 packets, round trip propagation delay of clients uniformly distributed between 40 and 100 milliseconds)

representative simulation run. Although not shown in the figure, both clients of the multicast session are almost always subscribed to the same channels, so the session throughput shown in the figure also indicates the reception rate at each client.

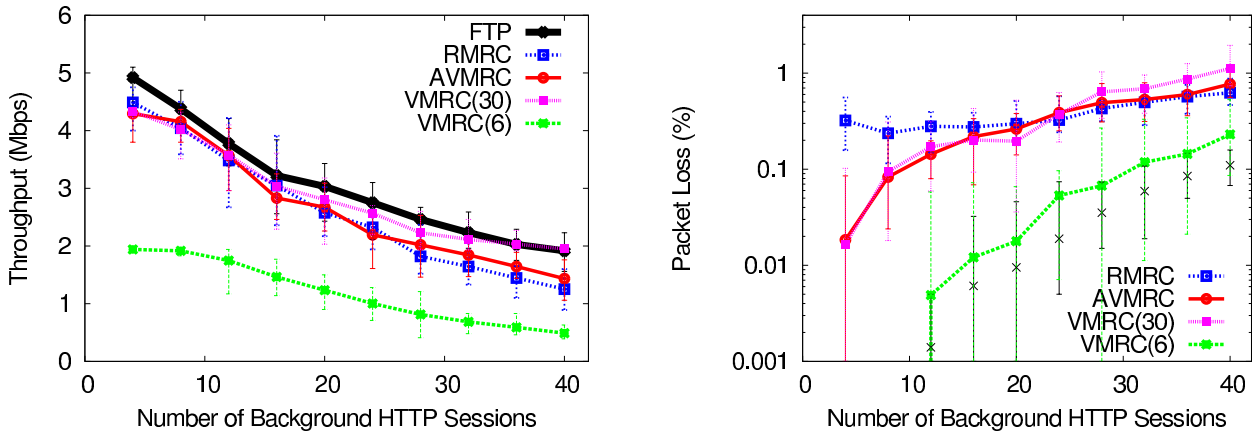
The results illustrate that both AVMRC and RMRC can quickly respond to changes in network conditions. The AVMRC protocol is somewhat more sensitive to such changes since it reacts to both packet loss rate and queuing delay changes. Although queuing delays also impact the target reception rate computed by the TCP Reno throughput model used in RMRC, the impact is much smaller.

A second experiment considers the impact on AVMRC and RMRC performance of the fraction of the bottleneck link bandwidth used by the UDP flow. The bottleneck link is configured as in the previous experiment, and in this case the multicast session has 16 clients each with a round trip propagation delay uniformly distributed between 40 and 100 milliseconds. Each client joins the multicast session at a uniformly distributed random time within the first 5 seconds of the simulation.

With both AVMRC and RMRC, clients successfully achieve the highest reception rate possible given the bandwidth used by the UDP flow [20]. Figure 5 shows the steady state packet loss percentage of the multicast session as a function of the fraction of the bottleneck link bandwidth used by the UDP flow. The curves are quite ragged owing to the coarse granularity of the possible reception rates of the multicast clients. A key observation from this figure is that the steady state packet loss experienced with AVMRC is substantially lower than that with RMRC when the UDP flow is using less than 50% of the bottleneck link bandwidth. In fact, when the UDP flow is using less than 30% of the bottleneck link bandwidth, with AVMRC the multicast session experiences no steady state packet loss. In general, the lower the fraction of the bandwidth used by the UDP flow, the more similar this scenario becomes to that with a single multicast flow and no background traffic as considered in Section 5.1.

5.4 TCP Background Traffic

Most traffic in the Internet is from applications using TCP. A simple scenario with TCP background traffic is the same as that of Figure 5, but with the UDP flow replaced by a single long duration



(a) Throughput Share (b) Packet Loss

Figure 7: Single Multicast Session Sharing a Low Bandwidth Bottleneck Link with Background HTTP Traffic (background HTTP traffic, bottleneck link of capacity 6 Mbps with a router buffer of 100 packets, round trip propagation delay of FTP client is 80 milliseconds and that of multicast clients is uniformly distributed between 20 and 460 milliseconds)

FTP session (i.e., TCP flow). The TCP flow has a round trip propagation delay of 100 milliseconds. Figure 6 shows the packet loss percentage of the multicast session as a function of the fraction of the bottleneck link bandwidth used by the TCP flow. This fraction is controlled by adjusting the TCP receiver-imposed maximum congestion window size. Only results up to a fraction of 0.6 are shown, since the TCP flow does not obtain a larger share of the bottleneck link bandwidth even with no limit placed on the maximum congestion window size. Note that with AVMRC, unlike with RMRC, the multicast session has no packet loss in a majority of the simulations, as clients are able to quickly detect when a reception rate is unsustainable. In the cases in which the multicast session does experience packet loss when using AVMRC, the packet loss appears to be due to some form of synchronization between the bursty transmissions of the TCP source and those of the multicast source.

Figure 7 explores the performance of AVMRC, RMRC, and also VMRC, when there are multiple TCP flows that start and complete during the simulation run. In this experiment the background traffic consists of on/off HTTP sessions as described in Section 4.3. In each simulation run, in addition to the HTTP traffic there is either one long duration FTP session, or one multicast session with 16 clients using either AVMRC, RMRC, or VMRC with the Vegas threshold parameters both statically set to either 6 or 30. In this and subsequent simulations, background HTTP/FTP sessions begin at uniformly distributed random times within the first 2 seconds of the start of the simulation. The foreground flows begin after a "warm up" period consisting of the first 5 seconds of the simulation. Client(s) of the foreground multicast or FTP session begin at uniformly distributed random times within the 5 seconds following the end of the warm up period. This design allows assessment of the bandwidth share obtained by a multicast session in comparison to that obtained by a long duration TCP flow, when competing against the same HTTP traffic, using the different multicast congestion control algorithms. VMRC is considered as well in this scenario, and in the subsequent two experiments, as these have sufficient background traffic to enable study of the impact of statically setting the threshold parameters. The round trip propagation delay between the FTP sender and receiver is fixed at 80 milliseconds, and each multicast session client has a round trip propagation delay uniformly distributed between 20 and 460 milliseconds. The bottleneck link is of capacity 6 Mbps with a 150 packet buffer.

Figure 7(a) shows the throughput obtained by the foreground session as a function of the number

of on/off HTTP sessions, while Figure 7(b) shows the packet loss percentage. Note that the multicast session achieves a similar throughput with both AVMRC and RMRC, although the packet loss is lower with AVMRC than with RMRC when the bottleneck link is lightly loaded (less than 20 HTTP sessions in this simulation). The achieved throughput is similar to that with a long duration FTP session in place of the multicast session, indicating that both AVMRC and RMRC can be TCP friendly across a range of background traffic loads. With VMRC and a static TCP Vegas threshold parameter value of 30, again the multicast session achieves a throughput similar to that of a long duration FTP session, but with a value of 6 the multicast session obtains less than one half of its fair share of the bottleneck link bandwidth. In contrast, in the subsequent two experiments (results shown in Figures 8 and 9), a value of 6 yields much more equitable behavior than a value of 30, illustrating the difficulty of statically choosing appropriate values for the TCP Vegas threshold parameters.

The remaining two experiments in this section concern scenarios with more complex background traffic consisting of a mix of long duration FTP and on/off HTTP sessions. All background flows have round trip propagation delays uniformly distributed between 20 and 460 milliseconds. As in the previous experiment, in addition to the background traffic there is either one long duration FTP session, or one multicast session with 16 clients using either AVMRC, RMRC, or VMRC with the Vegas threshold parameters both statically set to either 6 or 30, and round trip propagation delays set as in previous experiments.

Figure 8 shows the average throughput obtained by the foreground session as a function of the total number of background flows for a bottleneck link of capacity 45 Mbps with a 250 packet buffer. The background traffic consists of a mix of 90% HTTP and 10% FTP sessions. Note that for less than 50 background flows, there is very little congestion in the network, and the multicast or FTP session obtains its maximum possible throughput (for the multicast session, as determined by the sum of the rates of all channels, and for the FTP session, as determined by the maximum congestion window size of 64 packets). Larger numbers of background flows enable assessment of fairness, and it can be seen that the multicast session obtains a similar bandwidth share as the FTP session, with either AVMRC or RMRC. With VMRC, a TCP Vegas threshold parameter value of 6 yields fairer behavior than a value of 30 over the range of background traffic loads considered in the figure, but results in the multicast flow obtaining substantially less than its fair share of the bottleneck link bandwidth for numbers of background flows in the 70-150 range. The packet loss behavior is similar to that observed in the previous experiment [20].

Figure 9 shows the average throughput achieved by the foreground session as the bottleneck link bandwidth is scaled up from 10 Mbps to 100 Mbps. In this initial configuration, there is a 100 packet buffer at the bottleneck link and 30 background flows. The bottleneck link bandwidth is scaled up in increments of 10 Mbps, with the bottleneck link buffer size increasing by 100 packets and the number of background flows increasing by 30, with each increment. As in the previous experiment, the background traffic consists of a mix of 90% HTTP and 10% FTP sessions. Note that both AVMRC and RMRC achieve reasonably fair share of the bottleneck link bandwidth. With VMRC, although a value of 6 for the TCP Vegas threshold parameters yields better results than a value of 30 for high bottleneck link bandwidths, neither value yields uniformly good results over the entire bandwidth range. In contrast, the adaptive setting of the TCP Vegas threshold parameter value in AVMRC appears to be quite effective.

6 Performance Evaluation of AVMRC along Key Design Dimensions

AVMRC performance depends on a number of other key system design features in addition to its throughput model, including its use of weak synchronization, the approach used for determining the round trip time parameter RTT, the degree of burstiness of packet transmissions at the multicast server, and the time scale of protocol operation (the parameter τ).

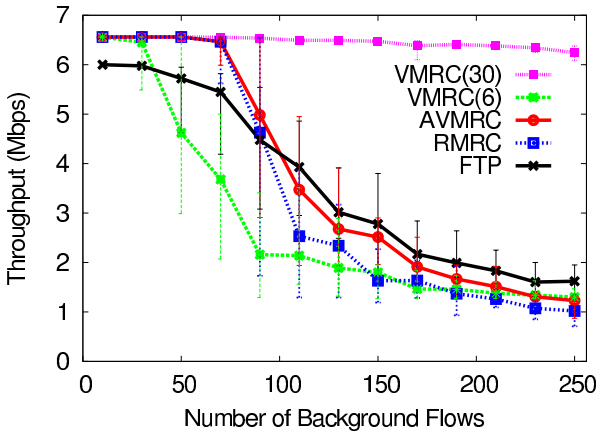


Figure 8: Single Multicast Session Sharing a High Bandwidth Bottleneck Link with Background HTTP & FTP Traffic (background traffic consists of 90% HTTP and 10% FTP sessions, bottleneck link of capacity 45 Mbps with a router buffer of 250 packets, round trip propagation delay of FTP client is 80 milliseconds and that of multicast clients is uniformly distributed between 20 and 460 milliseconds)

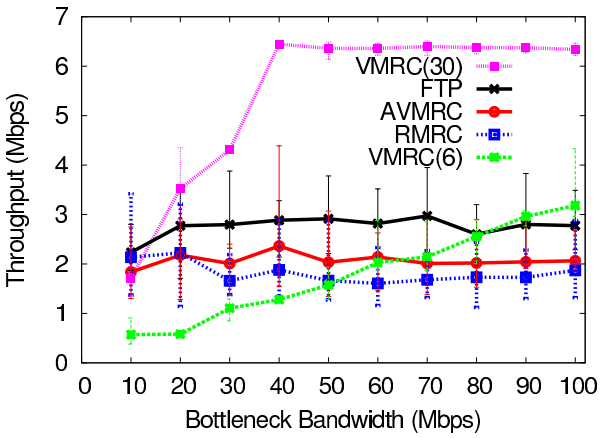
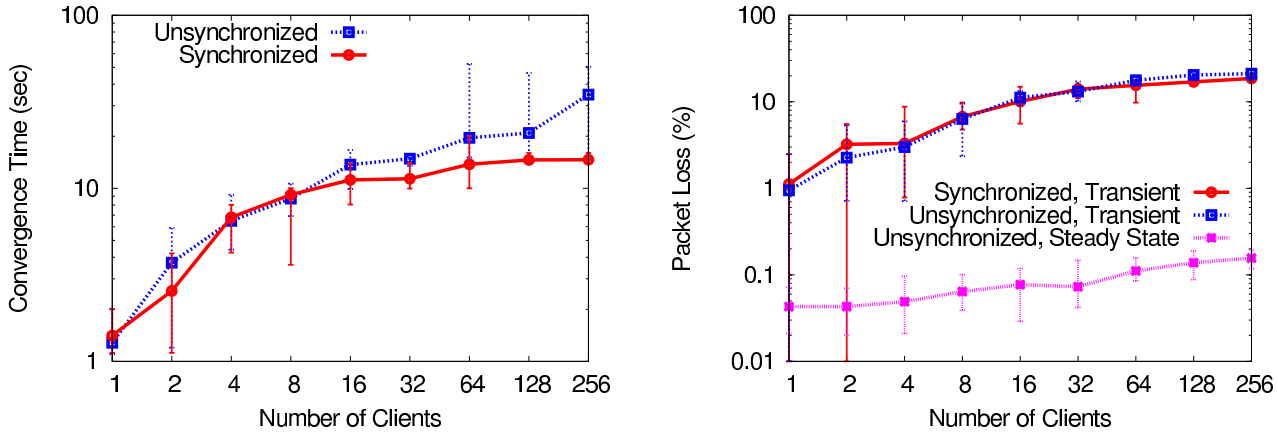


Figure 9: Effect of System Scaling (background traffic consists of 90% HTTP and 10% FTP sessions)

Section 6.1 compares the performance of AVMRC with weak synchronization (i.e., the server periodically signals when clients can attempt channel additions), to that of a variant of AVMRC with no such synchronization. Section 6.2 compares three methods for determining a value for the *RTT* throughput model parameter: the AVMRC approach of using the sum of the estimated queuing delay in the path from the server to the client and a fixed value, using measured round trip times, and using only a fixed value. The performance impact of the frequency at which packet transmissions are scheduled at the multicast server, and thus of the burstiness of multicast packet transmissions, is studied in Section 6.3. Section 6.4 considers the impact on protocol performance of the time slot duration τ.

6.1 Synchronization Approach

In AVMRC, channel additions are permitted only once every n_{add} time slots. The server signals the time slots and in particular signals every n_{add}'th time slot, so as long as $n_{add}\tau$ is relatively large compared to the network delay variations among clients, channel additions will be relatively closely clustered together in time. Figure 10 shows sample results comparing this weak synchronization approach to that of an "unsynchronized" approach in which channel additions are still only permitted every n_{add} time slots, but in which clients time the slots independently (i.e., the server does not signal the time slot boundaries, nor every n_{add}'th slot in particular). The scenario considered in the figure includes a single multicast session with no competing traffic. The round trip propagation delay between the server and each client is uniformly distributed between 40 and 100 milliseconds. The bottleneck link has capacity 3 Mbps and an 80 packet buffer. The clients join the multicast session at uniformly distributed random times between 0 and 5 seconds. As shown in Figure 10(a), for less than 16 clients the convergence times with the unsynchronized and the weak synchronization approaches are similar. For larger numbers of clients, however, the convergence time with the unsynchronized approach is significantly longer than that with



(a) Convergence Time (b) Packet Loss

Figure 10: Impact of the Synchronization Approach with Varying Numbers of Clients (no background traffic, bottleneck link of capacity 3 Mbps with a router buffer of 80 packets, round trip propagation delay of clients uniformly distributed between 40 and 100 milliseconds)

weak synchronization. Also, as seen in Figure 10(b), with the unsynchronized approach there is a non-zero steady state packet loss percentage that increases with the number of clients. These results can be explained by the increased duration of time that at least one client is subscribed to too many channels, with the unsynchronized approach, owing to the fact that the attempts of clients to subscribe to an additional channel are spread out over time rather than being clustered as with weak synchronization.[2]

Other experiments have investigated the impact of the synchronization approach in environments with competing traffic, and have found little difference in performance [20]. It appears that use of weak synchronization can yield substantive performance benefits over the unsynchronized approach only in low contention environments, in which the dynamics of the packet queue at the bottleneck link are dominated by the multicast session dynamics.

6.2 Determination of the RTT Parameter

Sample results illustrating the performance impact of the method of determining a value for the RTT parameter in the TCP Vegas throughput model are given in Figures 11 through 13. Each of these figures considers one or more of three methods, as applied with all other protocol features as in AVMRC. The "fixed R + one-way queuing" method is the AVMRC approach, while the "fixed R" method is to simply use a fixed value R for the RTT parameter. In the "measured round trip time" method, each client is assumed to measure the round trip time during an initial exchange with the multicast server, as well as the value of TSD (the difference between the server's clock time at which a packet was transmitted and the client's clock time at which that packet was received) for that initial exchange. Subsequent round trip time estimates are computed from observed changes in TSD. These are then averaged using the EWMA technique as is done in the AVMRC protocol for obtaining the average one-way queuing delay δ. Note that this method yields accurate estimates of the average round trip time assuming that congestion is on the forward path only, or more generally, that the delay on the path back from the client

[2]For small numbers of clients, including the case of just a single client, the small steady state packet loss percentage that is observed with the unsynchronized approach can be explained as follows. With the synchronized approach as implemented here, all packets from a single transmission burst by the server are marked as belonging to the same time slot. With the unsynchronized approach, in contrast, reception of packets from the same transmission burst may overlap the client-determined time slot boundary, potentially slowing the client's reactivity to increased queuing delay and packet loss.

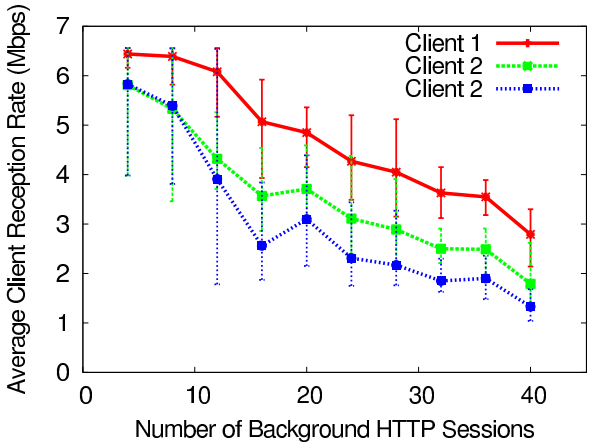


Figure 11: Average Client Reception Rate with the Measured Round Trip Time Method (HTTP background traffic, bottleneck link of capacity 10 Mbps with a router buffer of 150 packets, three clients with round trip propagation delays of 20, 200, 400 milliseconds, respectively)

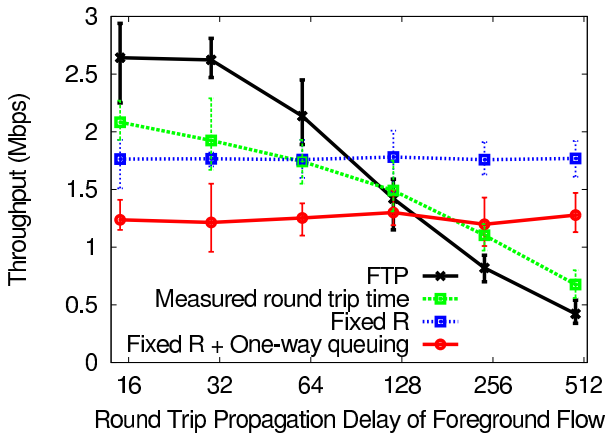


Figure 12: TCP Fairness with Different *RTT* Determination Methods (90 HTTP background sessions, bottleneck link of capacity 10 Mbps with a router buffer of 150 packets)

to the multicast server is similar to that observed during the initial exchange throughout the client's participation in the session.[3]

For Figure 11, a multicast session with three clients and varying numbers of HTTP sessions share a bottleneck link with capacity 10 Mbps and a 150 packet buffer. The round trip propagation delays between the multicast server and the three clients are 20, 200, and 400 milliseconds, respectively. The clients use the measured round trip time method for determining *RTT*. Note how the clients attain differing average reception rates, since they use differing average round trip time values for the *RTT* parameter in the TCP Vegas throughput model. This makes less efficient use of the bottleneck link than if all clients attained the same reception rate.

Figure 12 studies TCP fairness. A similar scenario is considered as for the previous figure, but with a fixed number (90) of HTTP sessions and one of four possible foreground sessions: a single client multicast session using one of the three methods for determining *RTT*, or a long duration FTP session. The figure graphs the achieved foreground session throughput as a function of the round trip propagation delay between the client and server.

Note that the TCP Vegas throughput model is less sensitive to round trip time than the TCP Reno throughput model, and thus even with the measured round trip time method for determining *RTT*, the achieved throughput of the multicast session is less sensitive to round trip propagation delay than is that of the FTP session. Qualitatively similar results are obtained for other background traffic loads [20]. Both the method utilizing a fixed value, and the AVMRC method of using a fixed value plus the average one-way queuing delay, display little or no sensitivity to round trip propagation delay. Thus, with these methods, a multicast session can be less "fair" in the sense that there can be greater discrepancy between the reception rate achieved by a multicast session client, and the throughput of a TCP session passing through the same bottleneck link and with the same round trip propagation delay between client and server. This is the cost of achieving common reception rates among clients behind a common bottleneck, but with differing round trip propagation delays.

[3]In the *ns-2* simulations, this method is simulated without using an initial exchange between server and client, using for the reverse path delay the known delay when there is no queuing.

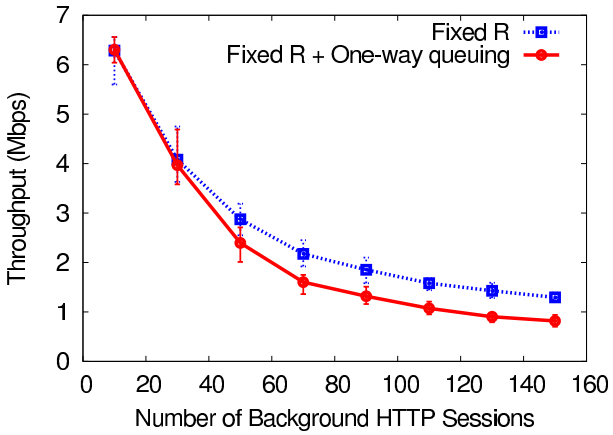
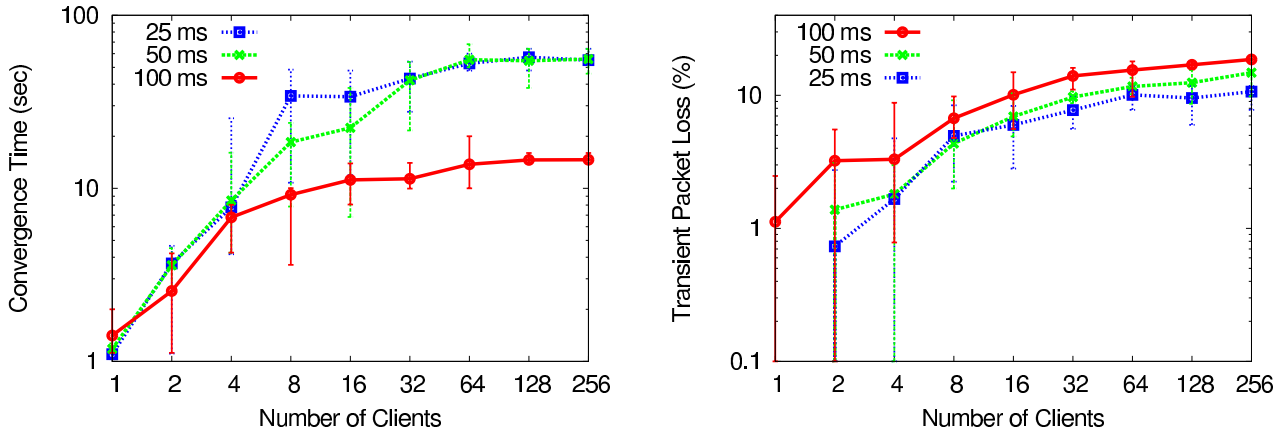


Figure 13: Impact of the RTT Determination Method on Responsiveness to Increased Congestion (HTTP background traffic, bottleneck link of capacity 10 Mbps with a router buffer of 150 packets, round trip propagation delays of clients uniformly distributed between 20 and 420 milliseconds)



(a) Convergence Time (b) Packet Loss

Figure 14: Impact of the Frequency of Packet Transmission Scheduling with Varying Numbers of Clients (no background traffic, bottleneck capacity of 3 Mbps with a router buffer of 80 packets, round trip propagation delay of clients uniformly distributed between 40 and 100 milliseconds)

As illustrated in Figure 13, a key advantage of the "fixed R + one-way queuing" method in comparison to the "fixed R" method is that it provides greater responsiveness to changes in congestion. Here the bottleneck link has a capacity of 10 Mbps and a 150 packet buffer, and the single multicast session has 16 clients with round trip propagation delays uniformly distributed between 20 and 420 milliseconds. Note that with the "fixed R" method, the multicast session retains a larger throughput share (i.e., is less responsive) as the background traffic load increases, in comparison to with the "fixed R + one-way queuing" method.

6.3 Frequency of Packet Transmission Scheduling

The frequency at which packet transmissions are scheduled at the multicast server determines the burstiness of multicast packet transmissions, which can impact queuing behavior in the network, and thus the

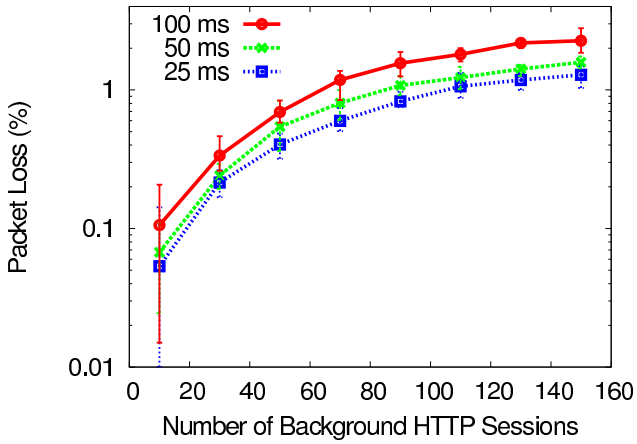
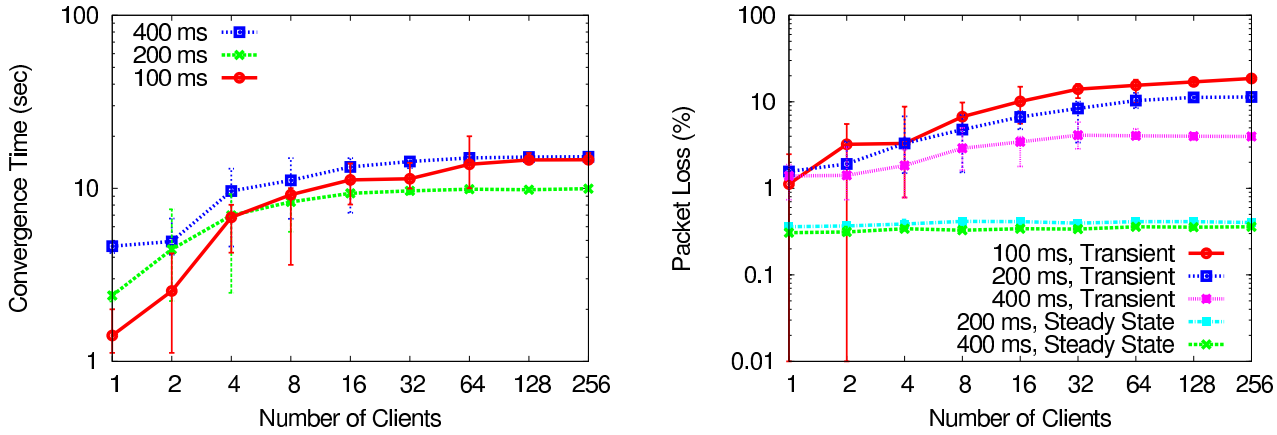


Figure 15: Impact of the Frequency of Packet Transmission Scheduling with Background Traffic (HTTP background traffic, bottleneck link of capacity 10 Mbps with a router buffer of 150 packets, round trip propagation delays of clients uniformly distributed between 20 and 420 milliseconds)

performance properties of a congestion control protocol. Figure 14 shows the convergence time and transient packet loss percentage for three values of the scheduling period (25, 50, and 100 milliseconds), for the same scenario used for Figure 10. In all cases, there are no packet losses in the steady state. The impact of the scheduling period on the steady state packet loss percentage in scenarios with background traffic is illustrated in Figure 15, for the same scenario as used for Figure 13. The steady state packet loss rate is similarly impacted by the scheduling period in scenarios with background traffic, as illustrated in Figure 15, for the same scenario used for Figure 13. At least for the range of values considered here, the scheduling period appears to have minimal impact on fairness [20]. In general, the best tradeoff among packet loss, convergence time, and system scheduling overhead, would be system and application specific.

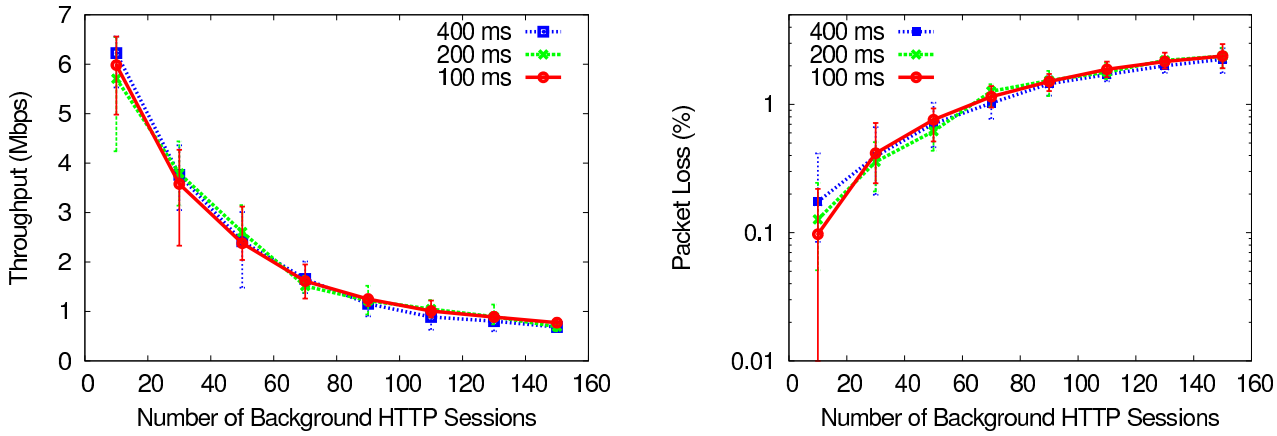
6.4 Time Slot Duration

The time scale at which the AVMRC protocol operates (i.e., the frequency with which channel subscription decisions are made, and the speed with which the protocol can react to changes in network conditions), is controlled by the time slot duration τ. Figure 16 illustrates the impact of the time slot duration on convergence time and packet loss, for the same scenario as considered in Figures 10 and 14. For small numbers of clients (for the results shown in the figure, four or less), a longer time slot duration yields a longer convergence time, while for larger numbers of clients, it appears that there is some intermediate time slot duration that minimizes the convergence time. With respect to packet loss, the default time slot duration of 100 milliseconds yields zero steady state packet loss, in contrast to a steady state loss percentage of approximately 0.4% for time slot durations of 200 to 400 milliseconds. Evidently, these longer time slot durations do not allow the protocol to react to increasing queuing delays sufficiently quickly to completely avoid packet loss, following a client's attempt to increase its reception rate to an unsustainable level. The results for transient packet loss require some care in interpretation. Note that as the time slot duration increases, so does the length of the start-up phase at each client that defines the time period over which the transient packet loss percentage is computed. Specifically, for values of τ of 100, 200, and 400 milliseconds, the client start-up phase length is 5, 10, and 20 seconds, respectively. Thus, although the transient packet loss rate appears to decrease in Figure 16(b) as the time slot duration is increased, in fact this is an artifact of the increasing length of the start-up phase.



(a) Convergence Time (b) Packet Loss

Figure 16: Impact of the Time Slot Duration with Varying Numbers of Clients (no background traffic, bottleneck link of capacity 3 Mbps and a router buffer of 80 packets, round trip propagation delay of clients uniformly distributed between 40 and 100 milliseconds)



(a) Throughput Share (b) Packet Loss

Figure 17: Impact of the Time Slot Duration with Background Traffic (HTTP background traffic, bottleneck link of capacity 10 Mbps with a router buffer of 150 packets, round trip propagation delays of clients uniformly distributed between 20 and 420 milliseconds)

For example, for large numbers of clients, the transient packet loss with $\tau = 100$ milliseconds is approximately 10%, and with $\tau = 400$ milliseconds is approximately 4%. Since the time period over which the transient loss is computed is approximately four times as long with $\tau = 400$ milliseconds, however, the total number of packet losses is actually higher (by about 60%) with $\tau = 400$ milliseconds than with $\tau = 100$ milliseconds.

Figure 17 shows the impact of the time slot duration when there is background HTTP traffic, for the same scenario considered in Figure 13 and 15. In this case, the choice of time slot duration, at least among the values considered in the figure, has little impact on the achieved average throughput and packet loss rate. It may be that the aggressiveness of the AVMRC protocol when competing against background traffic remains roughly constant as the time slot duration is increased, since although the increased time slot duration may cause clients to attempt additions of channels less frequently, it may

also delay channel drops.

7 Conclusions

This paper has proposed and evaluated AVMRC, a simple equation-based multirate congestion control protocol that employs a recently proposed TCP Vegas throughput model. AVMRC requires no feedback to the source, no explicit coordination among clients, and places no constraints on the data transmission policy or the spacings between the feasible client reception rates. Our performance study shows that AVMRC exhibits TCP Vegas-like behavior, and in the event that the bottleneck link is lightly loaded, AVMRC can operate without inducing packet losses.

AVMRC incorporates a new technique for dynamically adjusting the TCP Vegas threshold parameters, so as to achieve fair sharing of network resources with other types of congestion-controlled flows including the widely deployed versions of TCP. Our future research goals include application of this idea in TCP Vegas itself.

References

[1] M. Allman. A Web Server's View of the Transport Layer. *ACM Computer Communication Review*, 30(5):10–20, October 2000.

[2] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30–37, May/June 2000.

[3] M. Arlitt and C. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Trans. on Networking*, 5(5):631–645, October 1997.

[4] B. Braden, D. Clark, J. Crowcroft, B. Davis, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Patridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommmendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, April 1998.

[5] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. ACM SIGCOMM '94*, London, UK, September 1994.

[6] L. Bramko and L. Peterson. TCP Vegas: End-to-End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[7] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. FLID-DL: Congestion Control for Layered Multicast. In *Proc. NGC '00*, Stanford, CA, November 2000.

[8] J. Byers and G. Kwon. STAIR: A Practical AIMD Multirate Multicast Congestion Control. In *Proc. NGC '01*, London, UK, November 2001.

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM '00*, Stockholm, Sweden, August 2000.

[10] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[11] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *Proc. of First Workshop on Hot Topics in Networking*, Princeton, NJ, October 2002.

[12] M. Goyal, R. Guerin, and R. Rajan. Predicting TCP Throughput from Non-Invasive Network Sampling. In *Proc. IEEE INFOCOM '02*, Hiroshima, Japan, March 2002.

[13] H. Jiang and C. Dovorolis. Passive Estimation of TCP Round-trip Times. *ACM Computer Communication Review*, 32(3):75–88, July 2002.

[14] G. Kwon and J. Byers. Smooth multirate multicast congestion control. In *Proc. IEEE INFOCOM'03*, San Francisco, CA, April 2003.

[15] A. Legout and E. Biersack. Pathological Behaviors for RLM and RLC. In *Proc. NOSSDAV'00*, Chapel Hill, NC, June 2000.

[16] A. Legout and E. Biersack. PLM: Fast Convergence for Cumulative Layered Multicast Transmission Schemes. In *Proc. ACM SIGMETRICS '00*, Santa Clara, CA, June 2000.

[17] S. Low, F. Paganini, J. Wang, S. Adlakha, and J. Doyle. Dynamics of TCP/RED and a Scalable Control. In *Proc. IEEE INFOCOM '02*, New York, NY, June 2002.

[18] M. Luby, V. Goyal, S. Skaria, and G. Horn. Wave and Equation Based Rate Control Using Multicast Round Trip Time. In *Proc. ACM SIGCOMM '02*, Pittsburgh, PA, September 2002.

[19] A. Mahanti. Web Proxy Workload Characterisation and Modelling. Master's thesis, Department of Computer Science, University of Saskatchewan, September 1999.

[20] A. Mahanti. *Scalable Reliable On-demand Media Streaming Protocols*. PhD thesis, Department of Computer Science, University of Saskatchewan, March 2004.

[21] A. Mahanti, C. Williamson, and D. Eager. Traffic Analysis of a Web Proxy Caching Hierarchy. *IEEE Network*, 14(3):16–23, May/June 2000.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27(3):67–82, July 1997.

[23] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc. ACM SIGCOMM '96*, Stanford, CA, September 1996.

[24] J. Mo, R. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proc. IEEE INFOCOM '99*, New York, NY, March 1999.

[25] J. Padhye, V. Firioiu, D. Towsley, and J. Kurose. Modelling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM '98*, Vancouver, Canada, September 1998.

[26] V. Paxson and S. Floyd. Difficulties in Simulating the Internet. *IEEE/ACM Trans. on Networking*, 9(4):392–403, August 2001.

[27] NS Project. The Network Simulator: ns-2 (Web site). http://www.isi.edu/nsnam/ns.

[28] B. Samois and M. Vernon. Modeling the Throughput of TCP Vegas. In *Proc. ACM SIGMETRICS '03*, San Diego, CA, June 2003.

[29] D. Sisalem and A. Wolisz. MLDA: A TCP-friendly Congestion Control Framework for Heterogeneous Multicast Environments. In *Proc. 8^{th} Int'l. Workshop on QoS*, Pittsburg, PA, June 2000.

[30] W. Tan and A. Zakhor. Multicast Transmission of Scalable Video using Receiver-driven Hierarchical FEC. In *Packet Video Workshop*, New York, NY, April 1999.

[31] T. Turletti, S. Parisis, and J. Bolot. Experiments with a Layered Transmission Scheme over the Internet. INRIA Tech. Report 3296, November 1997.

[32] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like Congestion Control for Layered Video Multicast Data Transfer. In *Proc. IEEE INFOCOM '98*, San Francisco, CA, April 1998.