General AIMD Congestion Control *

Yang Richard Yang, Simon S. Lam

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
E-mail: {yangyang,lam}@cs.utexas.edu

Abstract

Instead of the increase-by-one decrease-to-half strategy used in TCP for congestion window adjustment, we consider the general strategy such that the increase value and decrease ratio are parameters. That is, in the congestion avoidance state, the window size is increased by α per window of packets acknowledged and it is decreased to β of the current value when there is congestion indication. We refer to this window adjustment strategy as general additive increase multiplicative decrease *(GAIMD). We present the (mean) sending rate of a GAIMD flow as a function of α, β. We conducted extensive experiments to validate this sending rate formula. We found the formula to be quite accurate for a loss rate of up to 20%. We also present in this paper how to control flow parameters so that flows with different parameters can achieve different sending rates. In particular, we present a simple relationship between α and β for a GAIMD flow to be* TCP-friendly*, that is, for the GAIMD flow to have approximately the same sending rate as a TCP flow under the same path conditions. We present results from simulations in which TCP-friendly GAIMD flows ($\alpha = 0.31$, $\beta = 7/8$) compete for bandwidth with TCP Reno flows and with TCP SACK flows, on a DropTail link as well as on a RED link. We found that the GAIMD flows were highly TCP-friendly. Furthermore, with β at 7/8 instead of 1/2, these GAIMD flows have reduced rate fluctuations compared to TCP flows.*

1. Introduction

In a shared network, such as the Internet, end systems should react to congestion by adapting their transmission rates to avoid congestion collapse and keep network utilization high [9]. The robustness of the current Internet is due in large part to the end-to-end congestion control mechanisms of TCP [14]. In particular, TCP uses an *additive increase multiplicative decrease* (AIMD) algorithm [5]; the TCP sending rate is controlled by a congestion window which is halved for every window of data containing a packet drop, and increased by one packet per window of data acknowledged.

Today, a wide variety of new applications such as streaming multimedia are being developed to satisfy the growing demands of Internet users. Many of these new applications use UDP because they do not require reliable delivery and they are not responsive to network congestion [27]. There is great concern that widespread deployment of such unresponsive applications may harm the performance of responsive TCP applications and ultimately lead to congestion collapse of the Internet.

To address this concern one approach is to entice these applications to use reservations [7] or differentiated services [6] that provide QoS. However, even if such services become available, we expect that many new applications will still want to use best-effort service because of its low cost. A second approach is to promote the use of end-to-end congestion control mechanisms for best effort traffic and to deploy incentives for its use [9]. However while TCP congestion control is appropriate for applications such as bulk data transfer, many real-time applications would find halving the sending rate of a flow to be too severe a response to a congestion indication, as it can noticeably reduce the flow's user-perceived quality [26]. Furthermore, it will be interesting if we can control service differentiation by using end-to-end mechanisms, such as by controlling the parameters of an end-to-end congestion control protocol.

In the past few years, many unicast congestion control schemes have been proposed and investigated [13, 17, 29, 30, 24, 4, 19, 23, 26, 21, 10, 2]. The common objective of these studies is to find a good alternative to the congestion control scheme of TCP. Since the dominant Internet traffic is

*Research sponsored in part by National Science Foundation grant No. ANI-9977267 and grant no. ANI-9506048. Experiments were performed on equipment procured with NSF grant no. CDA-9624082. An early version of this paper appears in *Proceedings of ICNP 2000*, Osaka, Japan, November 2000.

TCP-based [28], it is important that new congestion control schemes be *TCP-friendly*. By this, we mean that the sending rate of a non-TCP flow should be approximately the same as that of a TCP flow under the same conditions of round-trip time and packet loss [17].

The congestion control schemes investigated can be divided into two categories: AIMD-based [13, 24, 4, 23, 19] and formula-based [17, 29, 30, 26, 21, 10]. Roughly speaking, AIMD-based schemes emulate the increase-by-one and decrease-to-half window behavior of TCP. Formula-based schemes use a stochastic model [17, 18, 20] to derive a formula that expresses the TCP sending rate as a function of packet loss rate, round-trip time, and timeout. Essentially, all of these schemes are based upon the increase-by-one and decrease-to-half strategy of TCP. We observe that decrease-to-half is not a fundamental requirement of congestion control. In DECbit, also based on AIMD, a flow reduces its sending rate to 7/8 of the old value in response to a packet drop [16].

In this paper, we consider a more general version of AIMD than is implemented in TCP; specifically, the sender's window size is increased by α if there is no packet loss in a round-trip time, and the window size is decreased to β of current value if there is a triple-duplicate loss indication, where α and β are parameters. Since the name AIMD is often used in the literature to refer to TCP Reno congestion control (with $\alpha = 1$ and $\beta = 1/2$), we call our approach *general additive increase multiplicative decrease* (GAIMD) congestion control.

GAIMD was first considered by Chiu and Jain [5]. Their study is mainly about stability and fairness properties. They showed that if α and β satisfy the following relationships,

$$\begin{cases} 0 < \alpha \\ 0 < \beta < 1 \end{cases} \quad (1)$$

then GAIMD congestion control is "stable" and "fair." However, their study only considered the case when all flows using the same α, β parameters. Also, they provided no quantitative study of the effects of α and β on performance metrics. In our study, we consider in detail the relationships between various performance metrics and the parameters α and β, assuming that α and β satisfy (1). In the balance of this paper, we assume that α and β satisfy (1) unless otherwise stated.

In particular, we are interested in the sending rate as a steady state metric, and responsiveness, aggressiveness and rate fluctuations as transient metrics. In this paper, we report results on the GAIMD sending rate. Our results on transient behavior will be reported in [33].

Our first result is a formula showing the GAIMD (mean) sending rate as a function of the control parameters, α and β, the loss rate, mean round-trip time, mean timeout value, and the number of packets each ACK acknowledges. We have conducted Internet experiments and extensive simulations to validate this formula. The results show that the formula is accurate over a wide range of α and β values for a loss rate of up to 20%.

With the formula, we investigate how to choose α and β such that flows with different parameters can achieve different sending rates. In particular, we obtain our second result: a simple relationship between α and β for a GAIMD flow to be TCP-friendly, that is, for the GAIMD flow to have approximately the same sending rate as that of a TCP flow. The relationship between α and β to be TCP-friendly is

$$\alpha = \frac{4(1-\beta^2)}{3}$$

This relationship offers a wide selection of possible values for α and β to achieve desired transient behaviors, such as responsiveness and reduced rate fluctuations. For example, we can choose β to be $\frac{7}{8}$ so that a GAIMD sender has a less dramatic rate drop than that of TCP given one loss indication. For this choice of β, if we use $\alpha = 0.31$, the GAIMD flow is TCP-friendly.

The balance of this paper is as follows. In Section 2, we present the sending rate formula for a GAIMD flow. Experiments to validate the formula are also presented in this section. In Section 3, we use the formula to derive conditions under which a GAIMD flow is TCP-friendly. In Section 4, we present experimental results for the TCP-friendliness conditions. We give a summary of related TCP-friendly congestion control schemes in Section 5. Conclusion and future work are presented in Section 6.

2. Modeling Sending Rate

The motivation of this paper is to consider a general class of congestion window adjustment policies. Window adjustment policy, however, is only one component of a complete congestion control protocol. Other mechanisms such as congestion detection and round-trip time estimation are needed to make a complete protocol. Since TCP congestion control has been studied extensively for many years, GAIMD adopts these other mechanisms from TCP Reno [14, 15, 25, 1]. In the next subsection, we give a brief description of the GAIMD congestion window adjustment algorithm. All other algorithms are the same as those of TCP Reno.

2.1. GAIMD congestion window adjustment

A GAIMD session begins in the *slowstart* state. In this state, the congestion window size is doubled for every window of packets acknowledged. Upon the first congestion indication, the congestion window size is cut in half and the session enters the *congestion avoidance* state. In this

$$T_{\alpha,\beta}(p, RTT, T_0, b) = \frac{1}{RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p} + T_0 \min\left(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}p}\right)p(1+32p^2)} \quad (2)$$

state, the congestion window size is increased by α/W for each new ACK received, where W is the current congestion window size. For convenience, we say that the window size is increased by α per round-trip time. So far we have assumed that the receiver returns one new ACK for each received data packet. Many TCP receiver implementations send one cumulative ACK for two consecutive packets received (i.e., delayed ACK [25]). In this case, the window size is increased by $\alpha/2$ per round-trip time. GAIMD reduces the window size when congestion is detected. Same as TCP Reno, GAIMD detects congestion by two events: *triple-duplicate ACK* and *timeout*. If congestion is detected by a triple-duplicate ACK, GAIMD changes the window size to βW. If the congestion indication is a timeout, the window size is set to 1.

2.2. Modeling assumptions

In the Appendix of [34], we derive an analytic expression for the sending rate of a GAIMD sender as a function of α, β, p (loss rate), RTT (round-trip time), T_0 (timeout value), and b (number of packets acknowledged by each ACK). The derivation is a fairly straightforward extension of a similar formula derived for TCP by Padhye, Firoiu, Towsley, and Kurose [20]. Various assumptions and simplifications have been made in the analysis which are summarized below:

- We assume that the sender always has data to send (i.e., a saturated sender). The receiver always advertises a large enough receiver window size such that the send window size is determined by the GAIMD congestion window size.

- The sending rate is a random process. We have limited our efforts to modeling the mean value of the sending rate. An interesting topic will be to study the variance of the sending rate which is discussed in [33].

- We focus on GAIMD's congestion avoidance mechanisms. The impact of slowstart has been ignored.

- We model GAIMD's congestion avoidance behavior in terms of rounds. A round starts with the back-to-back transmission of W packets, where W is the current window size. Once all packets falling within the congestion window have been sent in this back-to-back manner, no more packet is sent until the first ACK is received for one of the W packets. This ACK reception marks the end of the current round and the beginning of the next round. In this model, the duration of a round is equal to the round-trip time and is assumed to be independent of the window size. Also, it is assumed that the time needed to send all of the packets in a window is smaller than the round-trip time.

- We assume that losses in different rounds are independent. When a packet in a round is lost, however, we assume all packets following it in the same round are also lost. Therefore, p is defined to be the probability that a packet is lost, given that it is either the first packet in its round or the preceding packet in its round is not lost [20].

2.3. Sending rate formula

The analytic expression of Equation (2) for the average GAIMD sending rate T has been derived (see Appendix of [34] for derivation):

We first observe that the denominator of the formula is the summation of the following two terms:

$$TD_{\alpha,\beta}(p, RTT, b) \triangleq RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p} \quad (3)$$

$$TO_{\alpha,\beta}(p, T_0, b) \triangleq T_0 \min\left(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}p}\right)p(1+32p^2) \quad (4)$$

From the derivation, we know that the denominator consists of only the first term $TD_{\alpha,\beta}$ if all congestion indications are triple-duplicate ACKs; note that $TD_{\alpha,\beta}$ does not depend on T_0. The second term $TO_{\alpha,\beta}$ is added when congestion indications can be both timeouts and triple-duplicate ACKs; note that $TO_{\alpha,\beta}$ does not directly depend on RTT. Comparing these two terms, we observe that when loss rate p is small, $TD_{\alpha,\beta} = O(p^{0.5})$ and $TO_{\alpha,\beta} = O(p^{1.5})$, therefore, $TD_{\alpha,\beta}$ dominates $TO_{\alpha,\beta}$, and the sending rate is mainly determined by $TD_{\alpha,\beta}$. However, as p increases, $TO_{\alpha,\beta}$ becomes larger. Define

$$Q \triangleq \min\left(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}p}\right)$$

We notice that Q is the middle term of $TO_{\alpha,\beta}$. From the derivation we know that Q approximates the probability of a loss being a timeout. From the expression of Q we note that when p is small, the probability of timeout is low. However, as p increases, the probability of timeout increases rapidly to 1.

We next consider how the sending rate varies with the parameters, RTT, T_0, α, β. It is obvious from Equation (2) that the sending rate decreases with increasing RTT or T_0. If β is increased towards 1, both $TD_{\alpha,\beta}$ and $TO_{\alpha,\beta}$ will decrease, leading to a higher sending rate. Also if α is increased, both $TD_{\alpha,\beta}$ and $TO_{\alpha,\beta}$ will decrease, leading to a higher sending rate. Furthermore, we observe that β must be less than 1 for the sending rate formula to be valid. If α approaches 0, the denominator in Equation (2) goes to infinity and the sending rate goes to 0.

Lastly, we note that Equation (2) reduces to other well-known TCP formulas when we instantiate it with $\alpha = 1$ and $\beta = \frac{1}{2}$. First, if we ignore the $TO_{\alpha,\beta}$ term, we obtain

$$T_{1,\frac{1}{2}}(p, RTT, b) = T_{TCP}(p, RTT, b) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}}$$

which is the formula derived in [17, 18]. Next, if we include the $TO_{\alpha,\beta}$ term, we have

$$T_{1,\frac{1}{2}}(p, RTT, T_0, b) = \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}$$

which is the formula derived in [20]. Therefore, our formula subsumes these other formulas as special cases.

2.4. Formula validation

Because of the simplicity of GAIMD, we have implemented GAIMD in both NetBSD and Linux kernels, and conducted some experiments in a LAN environment. We have also tested the formula in Equation (2) extensively using the *ns* network simulator. In all cases, the accuracy of the formula is respectable over a wide range of α and β when the loss rate is less than 20%. In this section, we report our simulation validations.

The purpose of our validations, presented in this section, is to answer the following questions:

- Is the formula accurate? Over what range of loss rate p is it accurate?
- Since it is a statistical mean, when do sending rate variations become significant?
- What is the general trend when the formula loses accuracy?

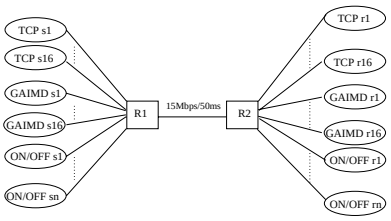

Figure 1. Simulation topology

2.4.1 Simulation setup

The simulation topology we chose to present results is the well-known single bottleneck ("dumbbell") as shown in Figure 1. We have also conducted simulations for other topologies; the results are similar.

In all of the simulations to be discussed in this section, the bottleneck link bandwidth is fixed at 15Mbps and its propagation delay at 50ms. We have also conducted experiments with other link bandwidths and propagation delays; the results are similar. In all simulations, the access links are sufficiently provisioned to ensure that packet drops/delays due to congestion occur only at the bottleneck link from $R1$ to $R2$.

We included three types of flows in the simulations. The first type is GAIMD flows. To see sending rate variations, we placed 16 GAIMD flows. For comparison purposes, we also placed 16 TCP Reno flows. Since the dominant traffic on the Internet is web-like traffic, we believe that it is important to model the effects of competing web-like traffic (short TCP connections, some UDP flows). It has been reported in [22] that WWW-related traffic tends to be self-similar in nature. In [31], it has been shown that self-similar traffic can be created by using several ON/OFF UDP sources whose ON/OFF times are drawn from a heavy-tailed distribution such as the Pareto distribution. Therefore, we chose ON/OFF UDP flows as the third type of traffic. In these experiments, we set the mean ON time to be 1 second, and the mean OFF time to be 2 seconds. During ON time each source sends at 500Kbps. The shape parameter of the Pareto distribution is set to be 1.5. In our experiments, we varied the number of ON/OFF sources from 10 to 70 to create a loss rate from about 1% to about 30%.

Another aspect of the simulations worth mentioning is how we start the flows. To avoid phase effects [11], we assign small random propagation delays to the access links and start the flows randomly.

In all experiments in this section, each simulation is run for 120 seconds. The loss rate is approximated by dividing the number of times a GAIMD flow or TCP flow reduces its window size by the total number of packets it sends. Notice that this estimation of loss rate is a lower bound for the loss rate that we defined in model derivation. Consequently, we

will see that the formula will overestimate and give an upper bound of the sending rate.

2.4.2 Predication accuracy

We first evaluate the predication accuracy of the formula. A good measure of the accuracy is the ratio of the predicated sending rate and the actual sending rate. The closer this ratio to 1, the better the predication accuracy. To test the validity range of the formula, for each β, we vary α from 0.1 to 1.0. For each α, β pair we vary the number of ON/OFF flows from 10 to 70 to create a loss rate from about 1% to about 30%.

Figures 2, 3, 4 demonstrate the predication accuracy for $\beta = 0.5, 0.75, 0.875$. The bottleneck link is a drop-tail link. In these three figures, the averages of the loss rates, round-trip times, and timeouts of the 16 GAIMD flows in each experiment are used to calculate a predicated sending rate for the experiment. Then the actual sending rates of the 16 GAIMD flows are averaged to obtain an average actual sending rate. What the figures show are the ratio between the calculated average sending rate using Equation (2) and the actual average sending rate. We observe from the figures that for a wide range of α, β, the formula predications are pretty close to the actual sending rate when the loss rate is less than about 20%. Next, we consider the impact of loss patterns on the accuracy of the formula. In the analytic model, we assume that (i) losses in different rounds are independent, and (ii) losses in the same round are correlated, i.e., when one packet is lost, all packets following it in the same round will also be lost. For a drop-tail router, this correlated-loss assumption is quite reasonable. To see the potential impact of loss patterns, we repeat the above experiments for a RED link. Figure 5 repeats the experiment in Figure 4 but uses a RED link. Comparing Figure 4 and 5, we see that loss patterns do not have a large impact on the accuracy of the formula.

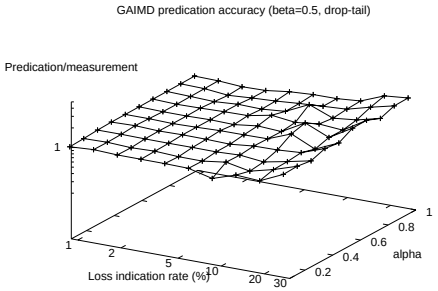

Figure 2. Accuracy for $\beta = 0.5$ and drop-tail

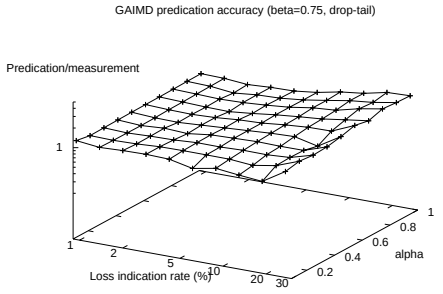

Figure 3. Accuracy for $\beta = 0.75$ and drop-tail

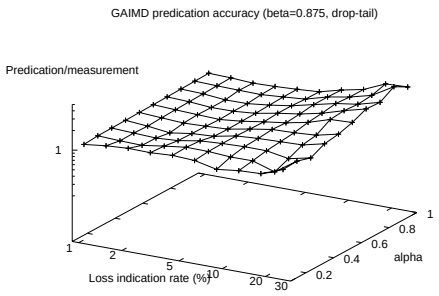

Figure 4. Accuracy for $\beta = 0.875$ and drop-tail

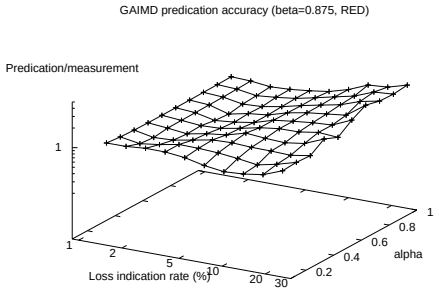

Figure 5. Accuracy for $\beta = 0.875$ and RED

2.4.3 Sending rate variation

Since what we derived is the mean value of the sending rate as a random process, we expect to see higher variations in the sending rate when loss rate increases. We illustrate this effect in this subsection. In addition to plotting the predication accuracy, Figures 6, 7 show the predication accuracy for each of the 16 GAIMD flows, for $\alpha = 0.5$, $\beta = 0.5$ and $\alpha = 0.4$, $\beta = 0.75$, and for both drop-tail and RED links. Observe from the figures that with increasing loss rate, sending rate variations increase. However, from both figures we can see that when the loss rate is 10% or less, the predication is accurate and the sending rate variance is reasonably small.

A major trend we observe from all the figures is that the sending rate formula tends to overestimate when loss rate

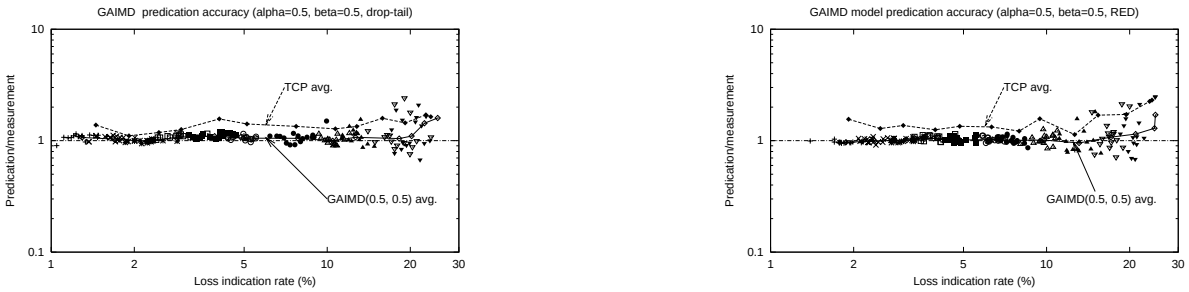

Figure 6. Variations of sending rate for $\alpha = 0.5$, $\beta = 0.5$ with drop-tail and RED

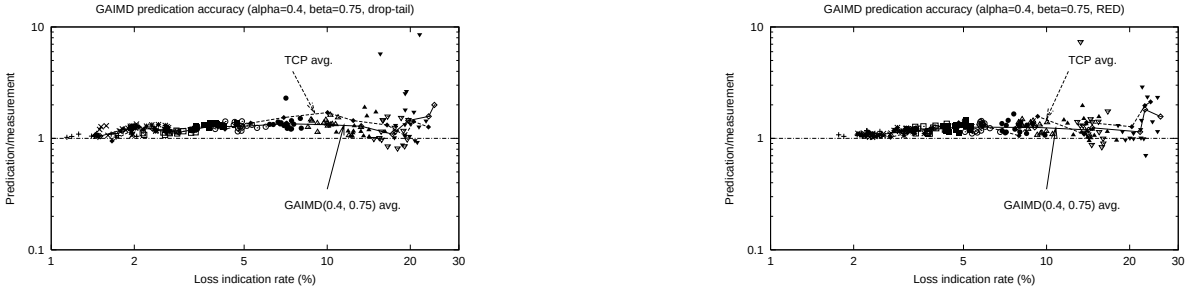

Figure 7. Variations of sending rate for $\alpha = 0.4$, $\beta = 0.75$ with drop-tail and RED

is high or when the α, β parameters are aggressive. Even though we desire an accurate sending rate model, we note that some applications of the formula do not require high accuracy but rather consistency. For example, if the purpose of using the formula is to compare the sending rates of two α, β pairs, then we can apply the formula as long as the amount of inaccuracy is consistent. We are particularly interested in relative predication accuracies between GAIMD and TCP flows because a major objective of ours is to use the formula to derive TCP-friendly GAIMD flows. Therefore, if its predication accuracy for a GAIMD flow is similar to the predication accuracy of a TCP flow, we can still use the formula to compare the sending rates of a GAIMD flow and a TCP flow. In both Figures 6, 7 we have also shown the predication accuracy of the 16 comparison TCP flows. We observe that the overestimates for GAIMD and for TCP are similar for most of the experiments we have conducted.

In summary, the validation experiments show that the formula is reasonably accurate for a wide range of α and β when the loss rate is not too high (up to 20%). For a loss rate of up to 10%, the sending rate variance is also small; thus the formula gives an accurate predication of the sending rate of a GAIMD flow.

3 TCP-friendly GAIMD

From the sending rate formula for a GAIMD flow, we observe that it is possible to control (α, β) pairs to yield the desired relative sending rate. Utilizing Equation (2), we can select the parameters α and β of a GAIMD flow such that the flow achieves d times the sending rate of a TCP flow, i.e.

$$T_{\alpha,\beta}(p, RTT, T_0, b) = d \cdot T_{1,\frac{1}{2}}(p, RTT, T_0, b) \quad (5)$$

Of particular interest are the (α, β) pairs that have (approximately) the same sending rate as that of a TCP flow, i.e. $d = 1$. We call these (α, β) pairs the *TCP-friendly curve*. Hereafter, we focus our attention on the condition for α and β to be TCP-friendly.

Note that p is a free variable in Equation (5). One way to derive the TCP-friendly α for a given β is to have p in the derived expression. However, this implies measuring p. To select α and β values such that equality holds for all p, we will have two equations: one for $TD_{\alpha,\beta}$ and one for $TO_{\alpha,\beta}$. In this case, the only solution is $\alpha = 1$ and $\beta = 1/2$. Therefore, we propose to relax the constraint of trying to match rates for all p. More specifically, we present three methods to determine the TCP-friendly α for a given β.

- TD TCP-friendly curve
  This is the simplest case, as we only try to match the first term $TD_{\alpha,\beta}$

  $$TD_{\alpha,\beta}(p, RTT, b) = TD_{1,\frac{1}{2}}(p, RTT, b)$$

  Canceling the common variables p, RTT and b from both sides, and squaring, we get

  $$\frac{(1-\beta)}{\alpha(1+\beta)} = \frac{(1-0.5)}{1*(1+0.5)}$$

Rearranging, we have

$$\alpha = \frac{3(1-\beta)}{(1+\beta)} \tag{6}$$

(It is interesting to see that according to Equation (6), for $\beta = 1$, we have $\alpha = 0$, and for $\beta > 1$, we have $\alpha < 0$. Even though these are not stable parameters, the pairing makes sense.)

From both formula derivation and validation, we know that compared to $TO_{\alpha,\beta}$, $TD_{\alpha,\beta}$ becomes less important when p increases towards 1. Therefore, it may be better to try to match the $TO_{\alpha,\beta}$ term. Thus, a second equation to determine the TCP-friendly α for a given β is obtained as follows.

- TO TCP-friendly curve

$$TO_{\alpha,\beta}(p, T_0, b) = TO_{1,\frac{1}{2}}(p, T_0, b)$$

Canceling the common variables p, T_0 and b from both sides, we have

$$\sqrt{\frac{1-\beta^2}{\alpha}} = \sqrt{\frac{1-0.5^2}{1}}$$

Rearranging, we get

$$\alpha = \frac{4(1-\beta^2)}{3} \tag{7}$$

(Notice that for $\beta = 1$, we have $\alpha = 0$, and for $\beta > 1$, we have $\alpha < 0$, the same pairing as in the previous method.)

- Error minimizing TCP-friendly curve
The two previous approaches are based on considering the two terms in the denominator of Equation (2) one at a time. We next consider both terms and use optimization to find α^* for a given β such that the mismatch between GAIMD and TCP rates is minimized over a range of loss rates. Formally, we define the error function

$$E_\beta(\alpha) = \int_0^1 w(p) \left| \frac{T_{\alpha,\beta}(p)}{T_{1,\frac{1}{2}}(p)} - 1 \right| dp \tag{8}$$

where $w(p)$ is a function which allows loss rates that are important to be given more weight in the optimization. In this paper, we consider a simple function that gives a weight of 1 to any loss rate less than a threshold value; a loss rate higher than the threshold gets a weight of 0. Figure 8 shows the shape of our weight function.

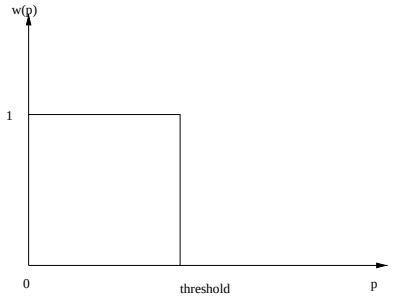

Figure 8. Weight function $w(p)$

Figure 9 shows $E_\beta(\alpha)$ for $\beta = 0.875$, $T_0 = 4RTT$, with the weight function threshold varying from 0.1 to 0.7. Note that $E_\beta(\alpha)$ has a well-defined bottom and the optimal α^* for a given β is easy to find. We observe the trend that as the weight function threshold increases, the optimal α^* increases. In the $\beta = 0.875$ case, α^* increases from 0.26 to about 0.3 when the weight function threshold was changed from 0.1 to 0.3.

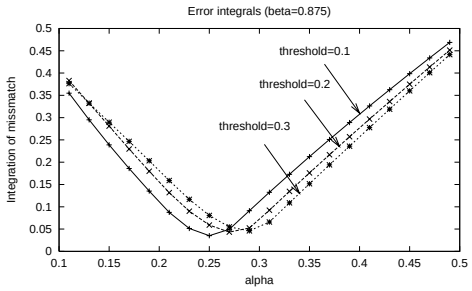

Figure 9. Error integral as a function of α

Figure 10 shows TCP-friendly curves obtained by the three methods described above. There are several interesting observations. First we observe that the curve determined by $TD_{\alpha,\beta}$ is higher than others when β is less than 0.5, and less than others when β is larger than 0.5. Second, we see that the TCP-friendly α determined by $TO_{\alpha,\beta}$ gives an upper bound when β is larger than 0.5, and the curve is also very close to the one determined by optimization if the weight function threshold is above 40%. Therefore, we propose to use Equation (7) to get the TCP-friendly α for a given β whenever we want to do error minimization up to a 40% loss rate.

Figure 11 shows ratios between the sending rates of GAIMD and TCP Reno for different values of TCP-friendly α determined by the three methods; β is fixed at 0.875. We observe from this figure that at a low loss rate a GAIMD flow using the α determined by TO will receive about 20% higher bandwidth than TCP Reno; and the flow using the α determined by TD will receive lower bandwidth.

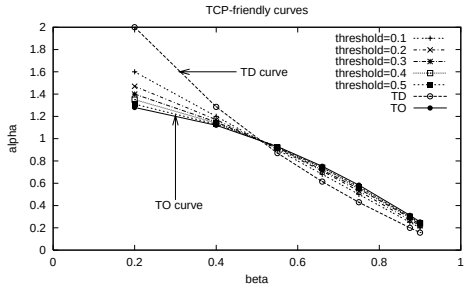

Figure 10. TCP-friendly curves

However, the differences diminish as the loss rate becomes higher. One factor we need to consider when determining α is that we only compared GAIMD with TCP Reno. However, many variants of TCP, e.g. NewReno, SACK [8], and TCP Vegas [3], achieve higher bandwidth than TCP Reno. Therefore, it is reasonable to select the α that is somewhat more aggressive than TCP Reno at a low loss rate[1]. We will see in the next section that TCP SACK does reduce the advantage of GAIMD when we use the α determined by TO.

We also observe from Figure 11 that when loss rate is very high, the ratios converge to one because essentially all loss indications are timeouts, and the parameters α and β no longer play an important role. However, as we will see in the next section, under very high loss rate, TCP receives more bandwidth than GAIMD because of its more aggressive window increasing behavior. This shows that our formula loses accuracy when the loss rate is very high.

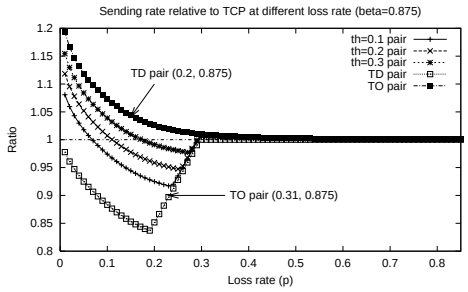

Figure 11. Ratios of the GAIMD flow sending rate and TCP sending rate

3.1 A closer look at TCP-friendliness

In previous subsections, we derived TCP-friendly curves using Equation (2). In this subsection, we provide an intuitive explanation of why a GAIMD flow can be TCP-friendly.

[1]Another possibility is to adaptively change α by measuring loss rate.

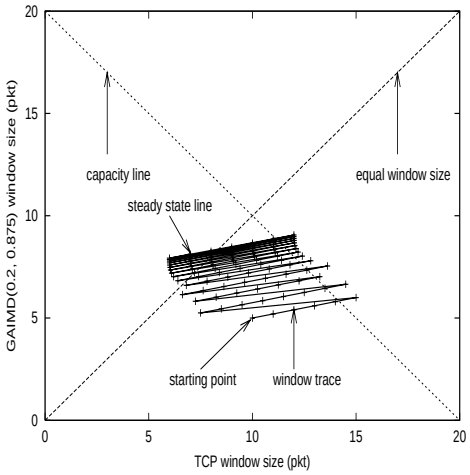

Figure 12. Window size changing trace

Figure 12 shows the evolution of the window sizes of a GAIMD(0.2, 7/8) flow and a TCP flow with the same round-trip time [5]. Since we do not consider timeout, for $\beta = 7/8$, we use the TD curve, and set the value of α as 0.2. The initial window size of the GAIMD(0.2, 7/8) flow is 10, and the initial window size of the TCP flow is 10. Therefore, the starting point of this trace is at (10, 5). When the sum of the window size of the two flows is less than 20, namely the link capacity, the TCP flow will increase its window size by 1 and the GAIMD(0.2, 7/8) flow will increase by 0.2 per RTT. When the sum is greater than the link capacity, the TCP flow reduces its window size to half, and the GAIMD(0.2, 7/8) flow reduces to 7/8 of the previous value. From this figure, we first observe that the trace will not converge to the *equal window size* curve. This means that two TCP-friendly flows with different control parameters will not have equal sending rate at *any* instant of time. We observe, however, that the window size trace crosses the equal window size curve. In particular, when the trace is on the left of the equal window size curve, the GAIMD(0.2, 7/8) flow has a larger window size and therefore will send more packets. On the other hand, when the trace is on the right of the equal window size curve, the TCP flow will send more packets. As a result, in the long run, they will receive about the same bandwidth. We also observe from this figure that when the flows enter steady states, that is, when the trace fluctuates along the top most line, the oscillation range of the GAIMD(0.2, 7/8) flow (projection of the top line on the y-axis) is smaller than that of the TCP flow (projection of the top line to the x-axis). This result indicates that the rate fluctuations of the GAIMD(0.2, 7/8) flow will be smaller.

4 Experimental Evaluation of GAIMD TCP-friendliness

In this section, we present experimental results for one particular GAIMD, namely, for $\alpha = 0.31$ and $\beta = 0.875$. It will be referred to as GAIMD(0.31, 0.875). We will study its performance mainly from the perspective of TCP-friendliness. Results for other TCP-friendly pairs, such as $\alpha = 0.58$ and $\beta = 0.75$, are similar.

For experiments in this section, we used the topology in Figure 1. However, we used only two types of flows: n TCP Reno flows, and n GAIMD(0.31, 0.875) flows. The number n is varied from 1 to 64. Each simulation was run for 120 seconds.

4.1 TCP-friendliness

From the analytic model, we see that loss rate has a major impact on the sending rate. Therefore, we evaluated the TCP-friendliness of GAIMD(0.31, 0.875) for a wide range of loss conditions. There are two experiment parameters we can use to control the loss rate, namely: the number of flows ($2n$) and the bottleneck link bandwidth.

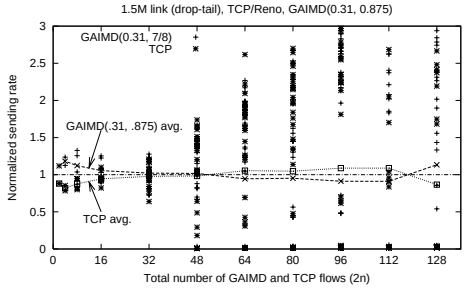

Figure 13. Normalized sending rates for 1.5Mbps drop-tail bottleneck link with Reno

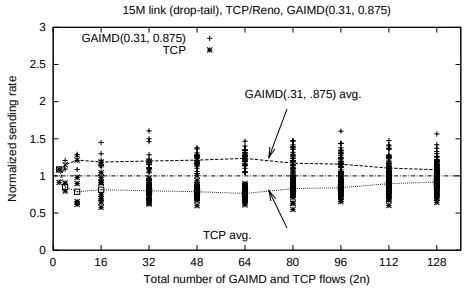

Figure 14. Normalized sending rates for 15Mbps drop-tail bottleneck link with Reno

Figures 13, 14 show for a drop-tail bottleneck link the normalized[2] average sending rates of GAIMD(0.31, 0.875) and TCP flows, as well as the sending rates of individual flows. We observe that at a low loss rate (15Mbps link, or 1.5Mbps link with less than 64 flows), GAIMD(0.31, 0.875) flows receive more bandwidth than TCP flows. This is expected from Figure 11. With a higher loss rate (1.5Mbps link with more than 64 flows), TCP flows receive higher bandwidth than GAIMD(0.31, 0.875) flows. We have seen consistently from all of our experiments that at a high loss rate TCP flows receive higher bandwidth than TCP-friendly GAIMD flows. One explanation is that TCP Reno increases more aggressively under high loss than TCP-friendly GAIMD (i.e., $\alpha < 1$). Whereas GAIMD's smaller reduction (i.e., $\beta > 1/2$) does not play as important a role because the congestion window size is small under high loss.

Another observation we can make from these figures is that the variance of individual flow rates is much higher for the 1.5Mbps link than for the 15Mbps link. This is expected because we have already seen that sending rate variance increases with loss rate increase.

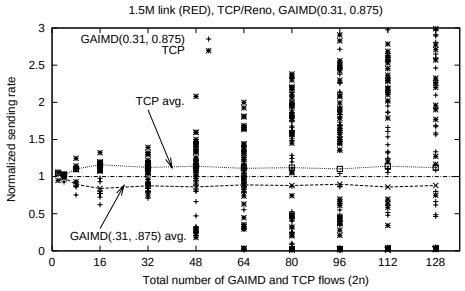

Figure 15. Normalized sending rates for 1.5Mbps RED link with Reno

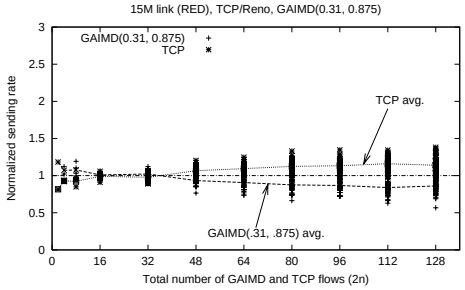

Figure 16. Normalized sending rates for 15Mbps RED link with Reno

[2] such that a fair share of the link bandwidth is 1.

We next consider the effects of loss patterns on GAIMD TCP-friendliness. Figures 15 and 16 repeat the experiments in Figure 13 and 14 with RED links. Comparing the figures, we observe that with RED instead of drop-tail links, TCP receives higher bandwidth than GAIMD(0.31, 0.875). We verified this in some other experiments, and it appears that the random and early dropping of RED does protect TCP traffic from less responsive traffic, such as GAIMD(0.31, 0.875).

In our third set of experiments, the competing TCP flows implement TCP SACK instead of TCP Reno. While it is generally assumed that Reno generates the dominant traffic in the current Internet, many operating systems are beginning to support TCP SACK; for example, Linux kernel supports TCP SACK as its default. Therefore, we think it is important to evaluate the TCP-friendliness of GAIMD when competing with TCP SACK. (We have also experimented with the case that GAIMD is based on TCP SACK instead of Reno. In this case, GAIMD will become more aggressive.)

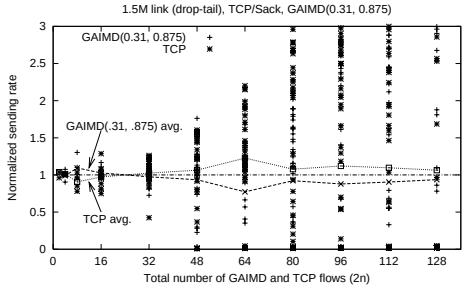

Figure 17. Normalized sending rates for 1.5Mbps drop-tail link with TCP SACK

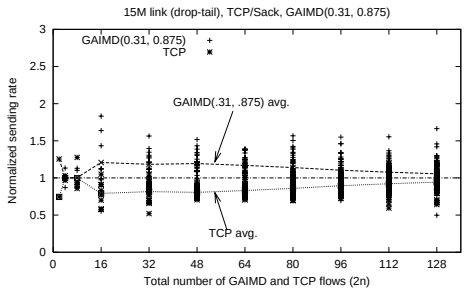

Figure 18. Normalized sending rates for 15Mbps drop-tail link with TCP SACK

Figures 17 and 18 repeat the experiments in Figures 13 and 14 except that the competing TCP flows are SACK instead of Reno. It can be seen that the results are very similar to the cases when the competing flows are Reno. However, we do observe that the crossover point in Figure 17 is at a lower loss rate than the one in Figure 13 (at 24 flows versus 48 flows for a 1.5Mbps drop-tail link).

Figures 19 and 20 repeat the experiments in Figures 15 and 16 except that the competing Reno flows are replaced with SACK flows; we can see that the results are similar to the previous cases.

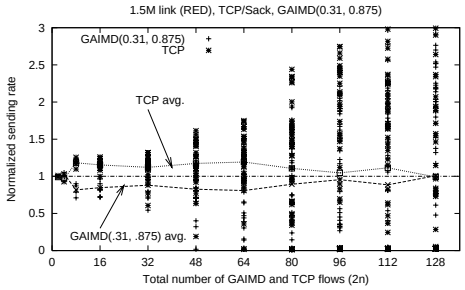

Figure 19. Normalized sending rates for 1.5Mbps RED link with TCP SACK

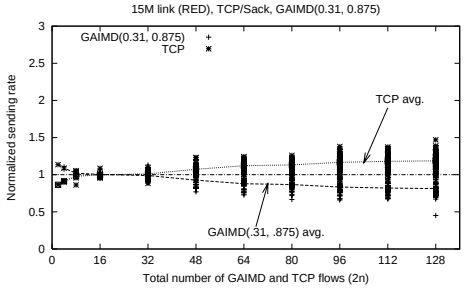

Figure 20. Normalized sending rates for 15Mbps RED link with TCP SACK

To summarize, we see that GAIMD flows compete with both TCP Reno and TCP SACK flows in a highly friendly manner over a wide range of loss rates and for both drop-tail and RED queueing disciplines.

4.2 Rate fluctuations

Having investigated long-term sending rate fairness, we next evaluate the transient behavior of GAIMD. In our study, we are particularly interested in the smoothness of its sending rate, the convergence speed to *fair* state and its response to congestion. We observe that a GAIMD flow with a smaller value of β will have a faster response to congestion, but its rate fluctuation will be higher. However,

due to space limitation, a detailed discussion of our findings is deferred to [33]. Figure 21 shows time traces of the sending rates of one GAIMD(0.31, 0.875) flow and one TCP flow when 4 GAIMD(0.31, 0.875) flows and 4 TCP Reno flows share one RED link with 15Mbps bandwidth and 20ms propagation delay. Each point in the figure is calculated over a time interval of 150ms, about 2 to 3 times the round-trip time. We can observe visually that GAIMD's sending rate is relatively smooth compared to that of TCP. From [33], we know that if we measure smoothness by sending rate coefficient of variations, GAIMD with $\beta = 7/8$ will have about half of the coefficient of variations of TCP at low loss rate.

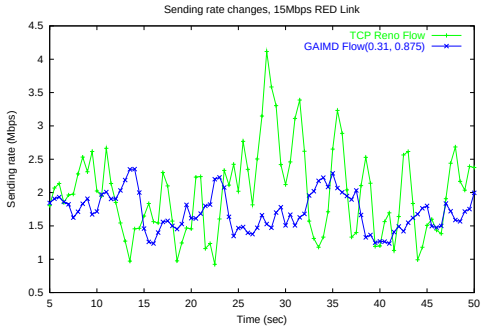

Figure 21. GAIMD and TCP sending rate traces for a 15Mbps RED link

4.3 Implementation

GAIMD is straightforward to implement because we only need to change two parameters in TCP Reno. Note, however, that we need to distinguish the first loss during slow start; in this case, the window size is dropped to half instead of β.

5 Summary of Related Work

AIMD was first proposed by Chiu and Jain in [5]. This design principle was used in DECbit [16] and TCP [14]. One of the first to consider implementing TCP-like congestion control for video services is [13]. However, it uses the standard TCP adjustment rule, and therefore, has the same TCP rapid rate changes.

Ozdemir and Rhee proposed the TEAR protocol (TCP Emulation at the Receivers) in [19]. In TEAR, a receiver emulates the congestion modifications of a TCP sender. To transform from a window-based scheme to a rate-based scheme, an weighted sliding window moving average of the congestion window size is divided by the estimated round-trip time [12]. As we will see in [33], TEAR has some problems in its responsiveness, and aggressiveness behaviors.

Another type of congestion control is to use additive increase, multiplicative decrease in some form, but not applying it to a congestion window. The Rate Adaption Protocol (RAP) [23] uses an AIMD rate control scheme based on regular acknowledgments sent by the receiver which the sender uses to detect lost packets and estimate RTT. The authors use the ratio between long-term and short-term averages of RTT to fine tune the sending rate on a per packet basis. In addition to the change from a window-based approach to a rate-based approach, RAP also includes a mechanism for the sender to stop sending in the absence of feedback from the receiver. However, RAP does not account for the impact of retransmission timeouts.

Another AIMD protocol is DLA [24] which makes use of RTP reports from the receiver to estimate loss rate and round-trip times.

In equation-based congestion control approaches [17, 26, 21, 10], the sender uses an equation that specifies the allowed sending rate as a function of RTT and packet drop rate, and adjusts its sending rate as a function of those measured parameters. However, the stability of this particular approach is not understood yet. Also, measuring loss rate turns out to be a complex issue, especially when the tradeoff between responsiveness and accuracy has to be considered.

In [2], Bansal and Balakrishnan use Binomial algorithms to generalize TCP-style additive-increase by increasing inversely proportional to a power k of the current window (for TCP, k=0) and TCP-style multiplicative-decrease by decreasing proportional to a power l of the current window (for TCP, $l = 1$). As we will see in [32], the analysis of GAIMD and Binomial can be combined to have a more generalized AIMD congestion control.

6 Conclusion

In this paper, we have considered a general version of AIMD congestion control, where the increase value and decrease ratio in congestion window adjustment are parameters, α and β, respectively. We derived a simple formula for the (mean) sending rate of a GAIMD flow as a function of α, β, loss rate, mean round-trip time, mean timeout value, and the number of packets acknowledged by each ACK. Our extensive experiments showed the formula to be quite accurate for a loss rate of up to 20%. We also found that we can choose the control parameters to implement end-to-end flow service differentiation. In particular, we present in this paper a simple relationship between α and β for a GAIMD flow to be *TCP-friendly*. We presented results from simulations in which TCP-friendly GAIMD flows ($\alpha = 0.31$, $\beta = 7/8$) compete for bandwidth with

TCP Reno flows and with TCP Sack flows, on a DropTail link as well as on a RED link. We found that the GAIMD flows were highly TCP-friendly. Furthermore, with β at 7/8 instead of 1/2, these GAIMD flows have reduced rate fluctuations compared to TCP flows. We are currently investigating tradeoffs among rate fluctuation, responsiveness, and convergence speed. We will report the results in [33].

Acknowledgment

The authors would like to thank Steve Li for valuable discussions.

References

[1] M. Allman, V. Paxson, and W. Stevens. *TCP Congestion Control, RFC 2581*, Apr. 1999.

[2] D. Bansal and H. Balakrishnan. TCP-friendly congestion control for real-time streaming applications. Technical Report MIT–LCS–TR–806, Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A., May 2000.

[3] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGCOMM '94*, Vancouver, Canada, May 1994.

[4] S. Cen, C. Pu, and J. Walpole. Flow and congestion control for Internet streaming applications. In *Proceedings of Multimedia Computing and Networking 1998*, Jan. 1998.

[5] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17, June 1989.

[6] D. Clark and J. Wroclawski. An approach to service allocation in the Internet. work in progress (IETF Internet-Draft), July 1997.

[7] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an Integrated Services Packet Network: architecture and mechanism. In *Proceedings of ACM SIGCOMM '92*, July 1992.

[8] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Communications Review*, 26(3):5–21, July 1996.

[9] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4), Aug. 1999.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM 2000*, Aug. 2000.

[11] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience*, 3(3), Sept. 1992.

[12] J. Golestani and K. Sabnani. Fundamental observations on multicast congestion control in the Internet. In *Proceedings of IEEE INFOCOM '99*, 1999.

[13] S. Jacobs and A. Eleftheriadis. Providing video services over networks without quality of service guarantees. In *Proceedings of World Wide Web Consortium Workshop on Real-time Multimedia and the Web*, Oct. 1996.

[14] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, Aug. 1988.

[15] V. Jacobson. Modified TCP congestion avoidance algorithm. Note sent to end2end-interest mailing list, 1990.

[16] R. Jain, K. K. Ramakrishnan, and D.-M. Chiu. Congestion avoidance in computer networks with a connectionless network layer. Technical Report DEC–TR–506, DEC, Aug. 1987.

[17] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. Note sent to the end2end-interest mailing list, 1997.

[18] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 27(3):67–82, July 1997.

[19] V. Ozdemir and I. Rhee. TCP emulation at the receivers (TEAR), presentation at the rm meeting, Nov. 1999.

[20] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proceedings of ACM SIGCOMM '98*, Vancouver, B.C., Sept. 1998.

[21] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A model based TCP-friendly rate control protocol. In *Proceedings of NOSSDAV '99*, June 1999.

[22] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols and self-similar network traffic. In *Proceedings of IEEE ICNP '96*, 1996.

[23] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *Proceedings of IEEE INFOCOM '99*, volume 3, Mar. 1999.

[24] D. Sisalem and H. Schulzrinne. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. In *Proceedings of NOSSDAV '98*, July 1998.

[25] W. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1997.

[26] W.-T. Tan and A. Zakhor. Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol. *IEEE Trans. on Multimedia*, 1, June 1999.

[27] V. Thomas. IP multicast in RealSystem G2. White paper, RealNetworks, Jan. 1998. Available at http://service.real.com/.

[28] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6), Nov. 1997.

[29] T. Turletti, S. F. Parisis, and J.-C. Bolot. Experiments with a layered transmission scheme over the Internet. Research Report No 3296, INRIA, Nov. 1997.

[30] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of IEEE INFOCOM '99*, volume 3, Mar. 1999.

[31] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high variability: statistical analysis of Ethernet LAN traffic at the source level. In *Proceedings of ACM SIGCOMM '95*, 1995.

[32] Y. R. Yang, M. S. Kim, and S. S. Lam. Analysis of Binomial congestion control. Technical Report TR–00–14, The University of Texas at Austin, June 2000.

[33] Y. R. Yang, M. S. Kim, and S. S. Lam. Transient behaviors of TCP-friendly congestion control protocols. Technical Report TR–00–23, Department of Computer Sciences, The University of Texas, Austin, Texas, U.S.A., Sept. 2000.

[34] Y. R. Yang and S. S. Lam. General AIMD congestion control. Technical Report TR–00–09, Department of Computer Sciences, The University of Texas, Austin, Texas, U.S.A., May 2000.