General AIMD Congestion Control *

Y. Richard Yang, Simon S. Lam
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
{*yangyang,lam*}*@cs.utexas.edu*

TR-2000-09
May 9, 2000

Abstract

Instead of the increase-by-one decrease-to-half strategy used in TCP Reno for congestion window adjustment, we consider the general case 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, loss rate, mean round-trip time, mean timeout value, and the number of packets acknowledged by each ACK. 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 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.

*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.

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 [8]. The robustness of the current Internet is due in large part to the end-to-end congestion control mechanisms of TCP [13]. In particular, TCP uses an *additive increase multiplicative decrease* (AIMD) algorithm [4]; 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 [5] 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 [8]. 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 [25].

In the past few years, many unicast congestion control schemes have been proposed and investigated [12, 16, 28, 23, 3, 18, 22, 25, 20, 29, 9]. 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 TCP-based [26], 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 [16].

The congestion control schemes investigated can be divided into two categories: AIMD-based [12, 23, 3, 22, 18] and formula-based [16, 28, 25, 20, 29, 9]. 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 [16, 17, 19] 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 [15].

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 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 [4]. Their study is mainly about stability and fairness properties. They showed that if α and β satisfy the following relationships,

$$\begin{cases} 0 < \alpha \\ 0 < \beta < 1 \end{cases} \tag{1}$$

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 and rate fluctuations as transient metrics. In this paper, we report results on the GAIMD sending rate. Our results on transient behavior, as well as integration of coarse-grained GAIMD window size control and fine-grained round-trip time control will be reported in [31].

Our first result is a formula showing the (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 extensive experiments to validate this formula. The experiments 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 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 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 [13, 14, 24, 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 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 [24]). 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, 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 [19]. 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 future topic will be to study the variance of the sending rate which is beyond the scope of this paper.
- 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 [19].

2.3 Sending rate formula

The following analytic expression for the average GAIMD sending rate has been derived (see Appendix for derivation):

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

Observe that the formula has the following two terms in the denominator:

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

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

From the derivation in the Appendix, 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. From these expressions, 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

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

Note that Q is the middle term of $TO_{\alpha,\beta}$. From the derivation in the Appendix we know that Q approximates the probability of a loss being a timeout. From the above expression of Q 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, 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, 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 [16, 17]. 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 [19]. Therefore, our formula subsumes these other formulas as special cases.

2.4 Formula validation

We have tested the formula in Equation (2) extensively using the *ns* network simulator. We have also conducted some experiments in LAN environments. 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%.

The purpose of our validation experiments, 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?

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 experiments to be discussed in this section, the bottleneck link bandwidth is fixed at 15Mbps and its propagation delay at 50ms. We have also

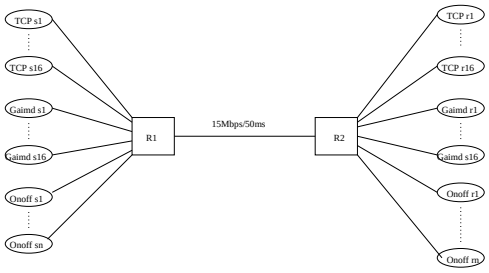

Figure 1: Simulation topology

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 experiments. 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 [21] that WWW-related traffic tends to be self-similar in nature. In [30], 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 [10], 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.