

# Scalable Service Differentiation using Purely End-to-End Mechanisms: Features and Limitations

Thyagarajan Nandagopal Kang-Won Lee Jia-Ru Li Vaduvur Bharghavan  
Coordinated Science Laboratory  
University of Illinois at Urbana-Champaign  
1308 W. Main Street, Urbana, IL 61801  
Email: {thyagu, kwlee, juru, bharghav }@timely.crhc.uiuc.edu

## Abstract—

We investigate schemes for achieving service differentiation via *weighted end-to-end congestion control mechanisms* within the framework of the *additive increase/multiplicative decrease (AIMD) principle*, and study their performance as instantiations of the TCP protocol.

Our first approach considers a class of *weighted AIMD algorithms*. This approach does not scale well in practice because it leads to excessive loss for flows with large weights, thereby causing early timeouts and a reduction in throughput.

Our second approach considers a class of *loss adaptive weighted AIMD algorithms*. This approach scales by an order of magnitude compared to the previous approach, but is more susceptible to short-term unfairness and is sensitive to the accuracy of loss estimates.

We conclude that adapting the congestion control parameters to the loss characteristics is critical to scalable service differentiation; on the other hand, estimating loss characteristics using purely end-to-end mechanisms is an inherently difficult problem.

## I. INTRODUCTION

The underpinning principle of congestion control in the Internet has been the use of end-to-end window/rate adaptation, which is predicated on the assumption of cooperating end hosts sharing the network. As the Internet evolves from a free-use model to a pay-for-use model, the principle of cooperation must be augmented with another principle, namely *differentiation*, in order to support increasingly heterogeneous application requirements and user characteristics. While the cooperative nature of congestion control assumes that sources will jointly adapt their traffic to relieve network congestion, differentiating between flows requires that each flow should receive its “weighted fair share” of the network bandwidth in accordance with some commonly enforced definition of “weighted fairness” (e.g. users who pay more should receive correspondingly higher bandwidth). Recognizing that network-level fair rate allocation/packet dropping mechanisms can aid in achieving end-to-end fairness, network designers are increasingly considering enhanced router mechanisms such as fair queueing [1], random early packet dropping [2], and more recently, quality of service architectures such as Diffserv [3]. However, because different network routers will continue to have diverse mechanisms and fairness properties, and because end hosts must still react to feedback from the network to adapt their traffic according to dynamic network characteristics, end-to-end window/rate adaptation will continue to play a fundamental role in achieving congestion control in the Internet.

The starting point of this work is therefore to *examine mechanisms for achieving service differentiation within the framework of existing end-to-end congestion control algorithms in the Internet*, without making limiting assumptions about special-

ized functionality (e.g. fair queueing) at the network routers. The dominant congestion control principle, embodied in TCP [4], is based on *additive increase* of the window/rate of a flow for probing additional network bandwidth, and *multiplicative decrease* of the window/rate for throttling the bandwidth consumption of a flow upon detecting congestion<sup>1</sup>. TCP performs window-based AIMD in its “congestion avoidance” phase, wherein each flow has the same increase/decrease parameters (e.g. additive increase  $\alpha$  of 1 and multiplicative decrease  $\beta$  of 0.5 in the congestion avoidance phase). Within this framework, we investigate achieving service differentiation by tuning the increase/decrease parameters of each flow according to the pre-specified “weight” of the flow.

We use a simple fairness model for service differentiation: the expected throughput of a flow with a weight  $w$  should equal the sum of the expected throughputs of  $w$  replacement flows along the same path, each with weight 1. In other words, from a throughput perspective, a flow with weight  $w$  is “equivalent” to  $w$  flows with weight 1. To achieve this service model, we investigate a general AIMD-based congestion control algorithm wherein the increase and decrease parameters are functions of the weight of the flow. It turns out that if the increase/decrease weight functions are designed appropriately, then in theory we should be able to achieve the service differentiation objective. Unfortunately, when we instrumented TCP with this algorithm, we found that it does not scale beyond  $w = 10$ , because a flow with a large weight bursts out more packets and reacts less aggressively to loss, both of which cause timeouts to kick in early and cause the flow to come out of congestion avoidance. It turns out that the issue is more fundamental than just one of redesigning the increase/decrease weight functions in the congestion control algorithm. AIMD inherently induces losses in proportion to the *square* of the number of contending flows, and thus flows with larger weights induce significantly more losses in the network, as well as suffer from frequent and bursty losses. Unfortunately, TCP mechanisms are inherently susceptible to poor performance under bursty loss. These observations led us to investigate alternate service differentiation mechanisms, still within the AIMD framework, that induce fewer losses and scale better for larger weights.

A key lesson that we learned in the first phase of our investigations is that when there is a lot of contention for a bottleneck link (e.g. as the number of flows sharing the link increases, or

<sup>1</sup>Newly emerging transport protocols and middleware, such as RAP [5] and Congestion Manager [6], also use the general principles of AIMD congestion control. The discussions in this paper are equally applicable to these protocols.

if some flows have very large weights), the vanilla AIMD approach induces a large loss probability, which causes TCP to time out and enter its “slow start” phase more often. This is detrimental to the performance of TCP in general, and service differentiation in particular, because all flows experience similar timeout and slow start behavior<sup>2</sup>. Thus, the goal is to make the AIMD congestion control algorithm *loss sensitive*. We try to achieve this goal by making the *increase/decrease parameters dependent on the perceived loss characteristics of a flow*. This leads to a “loss adaptive weighted AIMD” congestion control algorithm<sup>3</sup>, wherein the increase/decrease parameters are now functions of both the flow weight  $w$  and the estimated loss probability  $p$ . We investigate the characteristics of such a loss adaptive AIMD algorithm, and conclude that designing appropriate increase/decrease functions can, in fact, significantly reduce the network loss characteristics and enable the service differentiation algorithm to scale by at least an order of magnitude. On the other hand, loss adaptive AIMD is also much more sensitive to the fairness characteristics of the routers in the network and to the accuracy of the estimation of the loss probability. More subtly, we show that the loss adaptive nature of the congestion control algorithm changes the fairness model from a “min-potential delay fairness” model [7] to a “proportional fairness” model [8].

To summarize, this paper explores features and limitations of two classes of AIMD-based end-to-end congestion control algorithms for achieving service differentiation.

- We show that a “weight dependent AIMD” approach achieves service differentiation in a robust and fair manner for small weights, but does not scale well for large weights because it induces excessive losses.
- We show that a “loss adaptive weighted AIMD” approach achieves scalable service differentiation but is sensitive to the accurate estimation of the loss characteristics of the flow, which is an inherently difficult task.
- We discuss both the theoretically expected behavior of these algorithms as well as the practically observed behavior of their instantiations in TCP, and explain the discrepancy that sometimes arises between the two.

It is important to note that we are not necessarily promoting the use of any particular algorithm for end-to-end service differentiation - instead, our goal is to conduct a dispassionate study of two possible approaches. The choice between these, and possibly other approaches, will depend on the range of weights one seeks to support, the fairness characteristics of the routers, etc. Finally, our concern in this paper is solely on the mechanisms for achieving weighted fairness using end-to-end congestion control. For practical deployment, such mechanisms must be associated with other components such as the allocation of weights, policing against dishonest increase of weights, etc.

The rest of the paper is organized as follows. In Section II, we describe a general framework for AIMD-based window adaptation to achieve service differentiation. We apply this general framework to extend TCP and analyze its performance and

<sup>2</sup>It turns out to be impractical to weight the slow start mechanism while still maintaining its exponential behavior, as we will discuss later.

<sup>3</sup>Of course, the traditional AIMD algorithm is also “loss adaptive” in the sense that it adapts flow rate to the estimated loss. By “loss adaptive AIMD”, we mean that not only the rate, but also the AIMD parameters themselves, are adaptive to the estimated loss.

properties in Section III. In Section IV, we investigate a loss adaptive mechanism for service differentiation and evaluate its properties. Section V further compares the performance of the two mechanisms. In Section VI, we discuss related work. In Section VII, we discuss some unresolved issues and conclude the paper.

## II. GENERAL FRAMEWORK FOR WINDOW ADAPTATION

Our goal is to allow for weighted service differentiation (in terms of throughput) within the framework of end-to-end congestion control. While our general approach is applicable for both window and rate control, the rest of the paper will assume window control only. We will instantiate our algorithms in TCP in order to understand the practical impact of our approach in simulation environments. While the analytical characterization of our algorithms will require assumptions about the behavior of packet loss in the network, this is no different from the wealth of TCP analysis in related literature [9], [10], [11]. Indeed, we have closely followed the analytical framework of Padhye et al [11], where the authors demonstrated closely conforming behavior with practical experiments.

In our environment, each flow has a weight  $w$ . In a large network with many flows traversing through bottleneck links, a flow with weight  $w$  attains roughly  $w$  times the expected throughput of a flow with weight 1 along the same path. For other scenarios, e.g. when the slow dial-up access link is the bottleneck for a flow, higher weights may not translate to increased throughput. In any case, our goal is for a flow with weight  $w$  to achieve the same throughput behavior as the sum of the throughputs of  $w$  replacement flows of weight 1 traversing the same path.

A first-cut solution for achieving this service model is to physically treat a flow with weight  $w$  as a bundle of  $w$  parallel flows of unit weight, and demultiplex data from the composite flow into the component flows. While the analogue of this approach has been used at the link layer in routers [14], implementing this approach efficiently in the context of end-to-end protocols that seek to provide sequencing and reliability is extremely difficult, and the composite flow typically becomes as slow as the slowest component flow, even for small weights. This limitation motivates the need to eschew the approach of physically bundling flows and instead investigate approaches for achieving equivalent throughput through weighted congestion control mechanisms.

In AIMD window control, a flow increases its window by an additive constant  $\alpha$  to probe for additional bandwidth in the absence of loss, and decreases its window by a multiplicative constant  $\beta$  for each packet loss. We generalize this window adaptation by making the increase and decrease constants to be functions of the flow weight. Thus, a flow with a weight  $w$  increases by a weighted additive constant  $f_{inc}(w)\alpha$ , and decreases by a weighted multiplicative constant  $\beta/f_{dec}(w)$ , where the functions  $f_{inc}$  and  $f_{dec}$  are designed to achieve weighted service differentiation. Specifically, the window  $W(t)$  at time  $t$  evolves according to the following equation:

$$\begin{aligned} W(t+1) &\leftarrow W(t) + f_{inc}(w)\alpha, & \text{if no loss.} \\ W(t+1) &\leftarrow W(t)[1 - \frac{\beta}{f_{dec}(w)}], & \text{for each loss.} \end{aligned}$$

It is easy to see that this is a generalization of AIMD adaptation [15] with  $f_{inc}(1) = f_{dec}(1) = 1$ . For  $\alpha = 1$  and  $\beta = 0.5$ , we obtain a weighted version of the window adaptation algorithm of TCP in the congestion avoidance phase. In the past, researchers have derived the expected average throughput of TCP based on this algorithm to be  $\frac{1}{RTT} \sqrt{\frac{3}{2p}}$  for a small packet loss probability  $p$  [9], [10]. Following the same approach, we now derive the throughput for the weighted window adaptation algorithm, which will allow us to construct the general form of the functions  $f_{inc}$  and  $f_{dec}$ <sup>4</sup>.

**A. Expected Throughput of Weighted AIMD:** Analyzing pure AIMD behavior is equivalent to considering an idealistic scenario wherein TCP always stays in the congestion avoidance and fast retransmit phase. Let a “loss event” denote the detection of a loss by the sender (e.g. following three duplicate acknowledgements in TCP). We make the same packet loss assumptions as in [11], wherein packet losses across “epochs” (or round-trip times) are independent, and the probability of observing a loss-event is  $p$ . The expected number of packets transmitted between successive loss-events is thus  $1/p$ . For simplicity, we assume that the round-trip time is fixed.

Adapting the results in [11], let  $i$  denote the  $i$ -th loss-event. Consider the time period  $A_i$  between two loss-events. Let the number of epochs in this period be denoted by  $X_i = A_i/RTT$  and the number of packets transmitted in this period be denoted by  $\Phi_i$ . The window size at the end of this time period is  $W_i$ . Assuming  $X_i$  and  $W_i$  to be mutually independent sequences of i.i.d. random variables, we have

$$W_i = (1 - \beta/f_{dec})W_{i-1} + \alpha f_{inc}X_i$$

$$E[W] = \alpha f_{inc} \frac{f_{dec}}{\beta} E[X] \quad (1)$$

The expected number of packets sent in this period is given by

$$E[\Phi] = \frac{E[X]}{2} (E[W](1 - \frac{\beta}{f_{dec}}) + E[W]) = \frac{1}{p} \quad (2)$$

The expected throughput is

$$B = \frac{E[\Phi]}{RTT \cdot E[X]} = \frac{1}{RTT} \sqrt{\frac{1}{2p} \frac{\alpha}{\beta} f_{inc} (2f_{dec} - \beta)} \quad (3)$$

As we mentioned above, for  $f_{inc} = f_{dec} = 1$ ,  $\alpha = 1$  and  $\beta = 0.5$ , Equation (3) reduces to the standard TCP throughput formula of  $\frac{1}{RTT} \sqrt{\frac{3}{2p}}$  for small values of  $p$  [9]. We now use Equation 3 to derive the condition for achieving rate service differentiation. Note that under our service model, the throughput for a flow with a weight of  $w$  is equal to the throughput of  $w$  flows with weight of 1. For a flow with weight of 1,  $f_{dec} = f_{inc} = 1$ , and hence we have<sup>5</sup>

$$B(w) = \sqrt{\frac{1}{2p} \frac{\alpha}{\beta} f_{inc} (2f_{dec} - \beta)} = w \sqrt{\frac{1}{2p} \frac{\alpha}{\beta} (2 - \beta)}$$

Squaring and canceling terms on both sides gives the following condition:

$$f_{inc} (2f_{dec} - \beta) = w^2 (2 - \beta) \quad (4)$$

<sup>4</sup>In the rest of the paper,  $f_{inc}$  and  $f_{dec}$  are assumed to be functions of  $w$ .

<sup>5</sup>We assume the large scale model wherein the loss probability is the same for the LHS and RHS scenarios.

This formula tells us that in order to achieve service differentiation, *the product of the increase and decrease functions should be proportional to the square of the weight of the flow*, i.e.  $f_{inc} f_{dec} \propto w^2$ . There is a wide range of functions for  $f_{inc}$  and  $f_{dec}$  that will satisfy the above condition, and the manner of evolution of congestion windows varies for each choice. We now explore the nature of congestion avoidance for a few different choices of  $f_{inc}$  and  $f_{dec}$ .

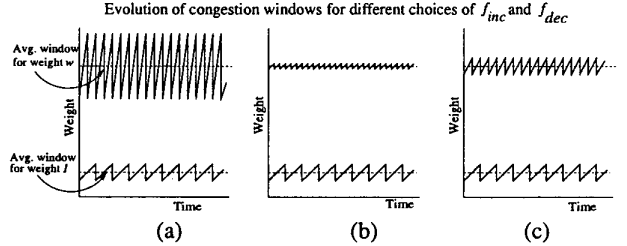


Fig. 1. Congestion window variation for various choices of  $f_{inc}$  and  $f_{dec}$ : (a)  $f_{inc} \propto w^2$  and  $f_{dec} = 1$ ; (b)  $f_{inc} = 1$  and  $f_{dec} \propto w^2$ ; (c)  $f_{inc} \propto w$  and  $f_{dec} \propto w$

- $f_{inc} \propto w^2$ ;  $f_{dec} = 1$

A flow with a large weight introduces large bursts whenever there are no losses in an epoch. When there is a loss, the window reduces to half. This causes large fluctuations in the window, with the band of variation proportional to the weight of the flow (see Figure 1(a)). For example, when  $\alpha = 1$  and  $f_{inc} = 0.1w^2$ , for a weight of 100, the congestion window will increase by 1000 in just one epoch! The bursty nature of the flow induces more burst loss and causes timeouts to kick in earlier, thereby limiting the amount of scaling that can be achieved.

- $f_{inc} = 1$ ;  $f_{dec} \propto w^2$

All flows increase by 1 each epoch, but flows with large weights decrease negligible upon packet loss, and hence start to look “unresponsive” to congestion. This may cause potential congestion collapse, as well as frequent losses (see Figure 1(b)) which can trigger timeouts (we have observed both phenomena). The first behavior affects network stability while the second affects scalability of the flow weights.

- $f_{inc} \propto w$ ;  $f_{dec} \propto w$

The increase and decrease weights are proportional to the weight of the flow. This approach strikes a balance between the two extremes. A desirable feature is that the band of window variation is independent of the flow weight (see Figure 1(c)). This is because the expected window of a flow with weight  $w$  is  $wc$ , where  $c$  is the expected window of a flow with weight 1; thus the decrease in window for a loss-event is  $wc \times \beta/w$ , which is  $c\beta$ . It turns out that this approach scales the best among the different choices for  $f_{inc}$  and  $f_{dec}$ , from our observations in a variety of simulation settings.

This algorithm follows the same principles as proposed in MulTCP [12].

**B. Packet Loss Probability:** Let us now briefly consider the impact of weighted flows on packet losses. From Equation 3, we see that the throughput of a flow is inversely proportional to the square root of its loss-event probability. Consider  $n$  unit weight flows traversing a single link with capacity  $C$ , with equal round-trip times. The fair share rate of each flow is equal to  $C/n$ , and

this rate is proportional to  $1/\sqrt{p}$ . This tells us that the loss-event probability  $p \propto n^2$ , i.e. it is proportional to the square of the number of flows traversing the bottleneck link. Extending this to the case when each flow has a weight  $w_i$ ,

$$p = \left(\frac{\sum_j w_j}{\tilde{C}}\right)^2 \frac{\alpha}{2\beta} f_{inc}(2f_{dec} - \beta) \\ = \left(\frac{\sum_j w_j}{\tilde{C}}\right)^2 \frac{\alpha}{2\beta} (2 - \beta) \quad (5)$$

using Equation (4), where  $\tilde{C} = RTT \cdot C$ . Basically,  $p \propto (\sum_j w_j)^2$ , which means that even if there are a few flows, as their weights increase, the packet loss probability will start to grow very quickly. Unfortunately, we know that for a large loss-event probability, timeouts kick in and TCP comes out of the congestion avoidance phase early. As we will see in the next section, this phenomenon severely limits the ability of the weighted AIMD approach to scale for large weights.

### III. INSTANTIATION OF WEIGHTED AIMD IN TCP CONGESTION CONTROL

We instrumented TCP-SACK with the weighted AIMD congestion control algorithm. We did not modify the behavior of timeout, RTT estimation, and slow start. While we did experiment initially with a weighted slow start algorithm, it turned out to be ineffective for reasons that are intuitively obvious - slow start already introduces highly bursty traffic behavior and adding weighted bursts does not enable flows to exit the slow start phase in proportion to their weights in a reliable and repeatable manner.

In this section, we will present some simulation and experimental results for TCP-SACK, and TCP-SD, which is the corresponding weighted version. However, in order to understand why the tests do not match the analytical behavior of the previous section for larger weights, let us first consider the impact of timeouts within the analytical framework.

#### A. Impact of Timeouts

From Equation 3, for purely AIMD behavior with  $\alpha = 1$  and  $\beta = 0.5$ , the expected throughput of a TCP flow is

$$B = \frac{1}{RTT} \sqrt{\frac{1}{2p} f_{inc}(4f_{dec} - 1)} \quad (6)$$

Following the timeout analysis in [11], the probability that a packet loss results in a timeout is approximately

$$Q = \min\left(1, \frac{3}{E[W]}\right) \quad (7)$$

where  $E[W]$  is calculated from Equation (1). The expected throughput of a TCP connection, taking timeouts into account, is thus given by

$$B = \frac{\frac{1}{p} + \frac{Q}{1-p}}{E[X]RTT + \frac{g(p)T_0Q}{1-p}}$$

where  $g(p) = 1 + p \sum_{i=0}^5 (2p)^i$  and  $T_0$  is the initial timeout period. Substituting  $\alpha = 1$  and  $\beta = 0.5$ , we obtain the throughput

relation for TCP-SD.

$$B \approx \frac{1}{RTT \sqrt{\frac{2p}{f_{inc}(4f_{dec}-1)} + T_0 Q p(1+32p^2)}} \quad (8) \\ \text{where } Q = \min\left(1, 3\sqrt{\frac{p(4f_{dec}-1)}{f_{inc}f_{dec}^2}}\right)$$

While the specifics of the analysis apply to TCP Reno in [11], the general form applies to both TCP Reno and TCP SACK. In fact, some studies have shown that SACKs prevent timeouts in only 4% of all timeout cases [16], and so the form of the equation, if not the precise constants, is fairly accurate. In the above equation, the first term of the denominator corresponds to the AIMD phase while the second term corresponds to the timeout phase. As loss-event probability increases ( $p \geq 1\%$ ), the timeout component starts to dominate, and the expected throughput starts to decrease sharply. This phenomenon is illustrated in Figure 2.a for the full range of loss-event probabilities, and in Figure 2.b by zooming in on the loss-event probability range of 0 to 0.1.

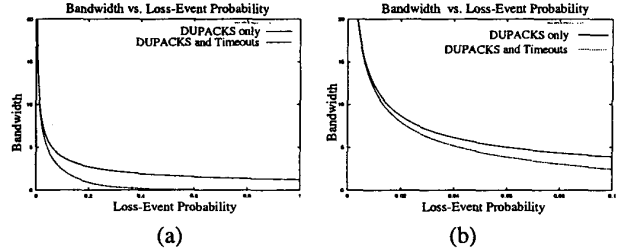


Fig. 2. Bandwidth vs. Loss Probability

Consider this behavior in the context of Equation 5. As the cumulative weights of flows in the system increases, the loss-event probability increases progressively more sharply, which causes timeouts to kick in early and breaks down the scalability of the service differentiation model of weighted AIMD.

#### B. Simulation Results

We simulated 10 TCP flows in the single bottleneck topology where one flow is TCP-SD with a weight of  $w$  and the other nine flows are TCP-SD flows with unit weight (Figure 3). Notice that a TCP-SD flow with unit weight is the same as a TCP-SACK flow. The bottleneck link capacity is set to 10 Mbps and round-trip time delay is approximately 240 msec. Routers are RED with buffer sizes that are large enough to accommodate the maximum burst size of all the flows<sup>6</sup>, so that the drop-tail behavior at the router is minimized. We see that this setup roughly emulates the multiplexing of a large number of flows at the backbone routers of the Internet where in each packet experiences approximately the same probability of loss.

We measured the throughput and loss of each flow while varying the weight of TCP-SD from 1 to 100, with 10 simulation runs per configuration, and 600 seconds per run. As mentioned in Section II, we have measured various combinations

<sup>6</sup>We set the buffer size to  $10 \text{ Mbps} \times 240 \text{ msec} \approx 300 \text{ KBytes}$ , which is approximately 295 packets of size 1 KB each.

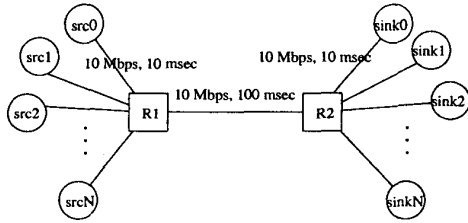


Fig. 3. Single link configuration

of  $f_{inc}$  and  $f_{dec}$ , and found that the throughput performance of TCP-SD is best when  $f_{inc} \simeq f_{dec}$ . Thus in what follows, we only present the result for  $f(2f - 0.5) = 1.5w^2$  where  $f_{inc} = f_{dec} = f$ . Figures 4 and 5 present the result of the measurements.

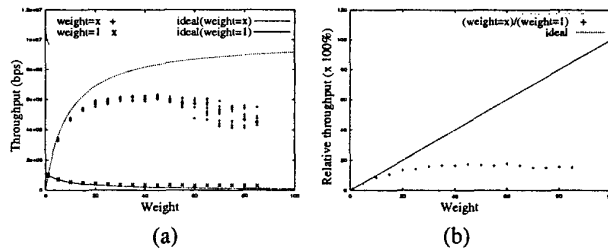


Fig. 4. Performance of TCP-SD: (a) Throughput of TCP-SD ( $w = x$ ) in comparison TCP-SACK, (b) Relative throughput gain of TCP-SD ( $w = x$ ) over TCP-SACK ( $w = 1$ )

In Figure 4, we plot throughput as a function of the weight of the TCP-SD flow. Figure 4(a) shows the absolute throughput values obtained for TCP-SD and TCP-SACK from the individual simulation runs in addition to the desired throughput curve for the weighted and unweighted flows. Figure 4(b) shows the desired relative throughput and the attained relative throughput averaged over the 10 runs. Note that the performance of TCP-SD starts to deviate (and progressively diverge) from the ideal behavior at around  $w = 10$ , where the resulting network utilization is around 60%. Further, as the frequency of timeouts increases, the relative throughput gain of TCP-SD becomes less predictable, as one would expect.

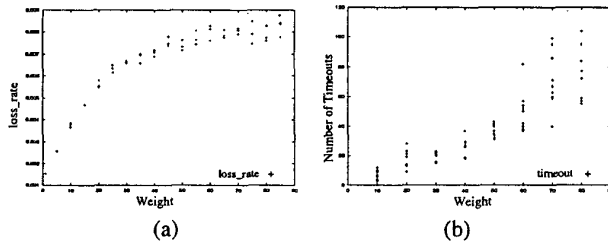


Fig. 5. Loss Measurement: (a) Average loss rate measured at the sender, and (b) Number of timeouts observed by TCP-SD

Figure 5 shows the loss probability observed by the flows and the number of timeouts observed by TCP-SD. While the loss probability is reasonably low, the number of timeouts observed by the TCP-SD sender is very high (up to 100 timeouts when

$w = 80$ ) because of its bursty nature at large weights. TCP-SD starts to lose scalability around  $w = 10$ , when the loss-event probability is around 0.4%. This result conforms reasonably well with the divergence of the throughput with timeouts in Figure 2. In essence, all our measurements point to the need for reducing loss probability in order to enable the scalability of service differentiation. On the other hand,  $p \propto n^2$  is a fundamental behavior of AIMD, which implies that weighted AIMD algorithms have an inherent scalability limitation.

#### IV. LOSS ADAPTIVE AIMD

##### A. The Case for Loss Adaptive AIMD

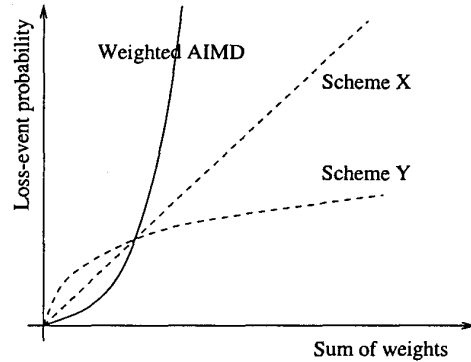


Fig. 6. Loss probability  $p$  versus the sum of weights: AIMD, Scheme X, and Scheme Y

Figure 6 illustrates the relationship of the loss probability  $p$  and the sum of weights for weighted AIMD. As expected,  $p$  increases very rapidly as the sum of weights increases. Ideally we want the loss probability to grow sub-linearly or at most linearly with the sum of weights of the flows traversing a bottleneck link, as illustrated for the hypothetical Schemes X and Y in Figure 6.

This motivates a congestion control algorithm that adopts the AIMD behavior and is also adaptive to loss. We call this *loss adaptive AIMD*, by which we mean that the AIMD parameters (in addition to the rate) are adaptive to the estimated loss.

In its general form, the window evolution under loss adaptive AIMD is as follows:

$$W(t+1) \leftarrow W(t) + f_{inc}(w)h_{inc}(p), \text{ if no loss.}$$

$$W(t+1) \leftarrow W(t)[1 - \frac{h_{dec}(p)}{f_{dec}(w)}], \text{ for each loss.}$$

where  $f_{inc}(w)$  and  $f_{dec}(w)$  are the weighting parameters as in weighted AIMD, and  $h_{inc}(p)$  and  $h_{dec}(p)$  are the increase/decrease parameters of the adaptation. When  $h_{inc}(p) = \alpha$  and  $h_{dec}(p) = \beta$ , the adaptation algorithm reduces to weighted AIMD.

Among the possible choices of  $h_{inc}(p)$  and  $h_{dec}(p)$ , we consider the case when  $h_{inc}(p) = \frac{\alpha}{p^a}$  and  $h_{dec}(p) = \xi p^b$ , in this paper.

In this section, we first investigate the properties of loss adaptive AIMD congestion control, and describe the loss rate estimation mechanism. Then we present simulation results for TCP-LASD, which is an enhancement of TCP-SACK with loss adaptive AIMD.

### B. Model for Loss Adaptive AIMD

Let us consider the loss adaptive AIMD algorithm motivated in the previous section. The new algorithm is essentially in the same form as AIMD with  $\alpha$  and  $\beta$  replaced by  $\frac{\zeta}{p^a}$  and  $\xi p^b$ , respectively.

$$W(t+1) \leftarrow W(t) + f_{inc}(w) \frac{\zeta}{p^a}, \text{ if no loss.}$$

$$W(t+1) \leftarrow W(t) \left[ 1 - \frac{\xi p^b}{f_{dec}(w)} \right], \text{ for each loss.}$$

We can calculate the loss incurred by the loss adaptive AIMD following the same derivation as in Equation 5. Equation 9 presents the loss-event probability of the system, where all flows in the network adopt the loss sensitive AIMD.

$$p = \left( \frac{\sum_j w_j}{\bar{C}} \right)^2 \frac{\zeta}{2\xi p^{a+b}} (2 - \xi p^b) \quad (9)$$

Thus  $p \propto (\sum_j w_j)^{\frac{2}{a+b+1}}$ . When  $a + b + 1 \geq 2$  the loss-event probability grows linearly or sub-linearly with the sum of the weights. In this paper, we consider the specific case of  $a + b + 1 = 2$  because we will see later that the practical instantiations of this algorithm are very sensitive to the accuracy of the loss measurement  $p$ , and for larger values of  $a + b + 1$  the algorithm may become highly unstable in terms of short-term fairness across flows. Given this restriction, the simplest combinations for the increase/decrease parameters for the case are  $(\alpha, \xi p)$  and  $(\frac{\zeta}{p}, \beta)$  pairs.

Although both choices ought to work equally well in theory, loss-adaptive decrease (the former combination) does not work well in practice, especially with small congestion windows, because it often degenerates into the timeout/slow start regime. Thus, from now on, we focus on the loss adaptive AIMD with increase/decrease parameter of  $(\frac{\zeta}{p}, \beta)$  pairs.

$$W(t+1) \leftarrow W(t) + f_{inc}(w) \frac{\zeta}{p}, \text{ if no loss.}$$

$$W(t+1) \leftarrow W(t) \left[ 1 - \frac{\beta}{f_{dec}(w)} \right], \text{ for each loss.}$$

where  $f_{inc}$  and  $f_{dec}$  are related to the weight  $w$  of the LASD flow by Equation 4. The average throughput that can be obtained under this model is given by

$$B = \frac{1}{RTT} \frac{1}{p} \sqrt{\frac{1}{2} \frac{\zeta}{\beta} f_{inc}(2f_{dec} - \beta)} \quad (10)$$

In this model, note that  $\zeta$  is a design parameter, which represents the loss probability for which the loss adaptive AIMD becomes equivalent to the vanilla AIMD algorithm. When the loss probability  $p$  is greater than  $\zeta$ , then the flow increases *less aggressively* thereby contributing to the reduction in the successive increase of loss probability. Depending on the network conditions, it is possible for  $p$  to be less than  $\zeta$ , in which case the increase is more aggressive than AIMD. On the other hand, as shown in Figure 7, we wish to ride the loss adaptive AIMD curve for  $p > \zeta$ , and the vanilla AIMD curve for  $p < \zeta$ . To this end, the increase parameter is set to  $\alpha = \min(1, \zeta/p)$ , where  $W(t+1) \leftarrow W(t) + f_{inc}\alpha$  if no loss is observed. Effectively, we bound the increase function to the increase of vanilla AIMD. Note that this model involves measuring loss probabilities frequently, and updating  $\zeta/p$  at regular intervals. We discuss the mechanism for loss estimation in the next section.

Now let us look at the loss curve of the loss adaptive AIMD. From Equation 9, we get  $p \propto \zeta \sum_j w_j$  using  $a = 1, b = 0$ . Figure 7 presents the loss curve of AIMD and the loss sensitive AIMD.

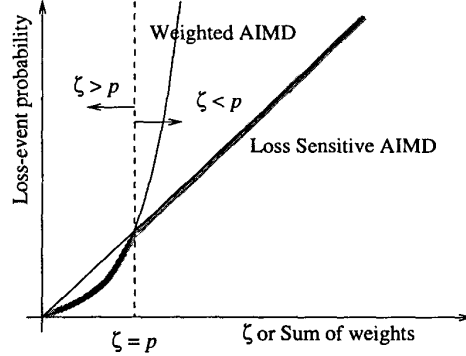


Fig. 7. The relationship of the loss rate  $p$  versus the sum of weights

We can interpret this graph as the relationship between the loss rate and the sum of weights assuming  $\zeta$  is a constant. But, at the same time, we can interpret the graph as the relationship between the loss rate and the design parameter  $\zeta$  assuming that the sum of weights is given. We see that weighted AIMD and loss adaptive AIMD behave in the same manner when  $\frac{\zeta}{p} = 1$ . By choosing  $\alpha = \min(1, \frac{\zeta}{p})$ , the loss curve of the modified loss adaptive AIMD moves along the weighted AIMD curve in  $\zeta < p$  region, and the loss adaptive AIMD curve in  $\zeta > p$ . This is a desired behavior since now the loss injected by the modified loss adaptive AIMD is always the minimum of the two (grey line).

### C. TCP-LASD: An Instantiation of Loss Adaptive AIMD

We have instantiated the loss adaptive AIMD mechanism in TCP-SACK, which we call TCP-LASD (TCP with loss adaptive service differentiation). The key challenge of designing TCP-LASD is to devise an accurate loss rate estimation algorithm. The estimated loss rate is used for adapting the additive increase factor  $\alpha = \min(1, \zeta/p)$ , and the accuracy of the loss rate estimation is critical to the performance TCP-LASD.

First let us consider the following simple algorithm. The sender maintains a variable  $S_N$  that tracks the average number of packets sent per loss-event, over the last  $N$  loss-events. The loss-event probability is estimated to be  $p = 1/S_N$ . In other words, if  $n(i)$  is the number of packets sent between loss-events  $i-1$  and  $i$ , then the loss-event probability  $p = \frac{N}{\sum_{k=i-N+1}^{i-1} n(k)}$ .

This simple estimation algorithm however can lead to a poor rate fairness among flows for the following reason. When a flow experiences a packet loss after transmitting only a small number of packets (after the last loss event), then its loss estimate will be higher than the actual loss-event probability. Overestimating the loss-event probability adversely affects the loss estimates in the future, since now the sender will send out packets at a lower rate than the other flows and will take a long time to recover.

One possible solution to overcome this problem is as follows: when the sender predicts that the actual loss rate is lower than its

current loss estimate, then it increases  $\alpha$  by a multiplicative constant  $\mu$ . Thus, a flow with an overestimate of  $p$  has a chance to increase its congestion window rapidly and catch up with flows with correct estimates of  $p$ . The modified loss estimation algorithm is

1. After sending a packet
2.  $num\_sent \leftarrow num\_sent + 1$
3. *if* ( $num\_sent > num\_thresh$ )
4.  $\alpha = \min(1, \mu \times \alpha)$
5.  $num\_thresh \leftarrow \mu \times num\_thresh$
6. Upon packet loss
7.  $n(i) \leftarrow num\_sent$
8.  $i \leftarrow i + 1; num\_sent \leftarrow 0$
9.  $num\_thresh \leftarrow 1/p$
10.  $p \leftarrow \frac{N}{\sum_{k=i-N+1}^{i-1} n(k)}$
11.  $\alpha = \min(1, \zeta/p)$

The key to the modified algorithm is to predict if the current loss estimate is higher than the actual loss rate of the connection. A new variable called *num\_thresh* now contains the expected number of packets sent out per loss-event according to the current loss estimate. It is initially set to  $1/p$  (line 9). After sending packets, the sender compares the number of sent packets with *num\_thresh*. If the number of sent packets is greater than *num\_thresh*, then the sender predicts the current loss estimate is higher than the actual rate and increases  $\alpha$  (lines 3, 4). Then it also increases the *num\_thresh* in order to determine the next increase of  $\alpha$  (line 5). The constant  $\mu (> 1)$  is a multiplicative increase factor for  $\alpha$  and *num\_thresh*. In the current implementation of TCP-LASD, it is set to 2.

Obviously, the performance of TCP-LASD is critically dependent on the accuracy of the loss estimation algorithm. Since  $W \propto \frac{1}{p}$ , where  $W$  is the expected congestion window,  $\frac{dW}{dp} \propto \frac{1}{p^2}$ , which is very large. Therefore, once the loss estimation deviates from the correct value, it further adversely affects the performance of TCP-LASD. It turns out estimating the loss rate using purely end-to-end mechanism is inherently a difficult problem which discuss later in Section VII.

In a related work [19], TFRC uses a weighted full average loss interval method. The average loss rate is computed over the last  $n$  intervals (where  $n = 8$  by an empirical choice), with recent samples being weighted more than the older samples. In TCP-LASD, we have not adopted the weighted average loss estimator since we found that it is highly sensitive to the accuracy of recent samples. Preliminary evaluation seems to indicate that our mechanism has a lower variance but tends to over-estimate the loss probability. Fortunately, this is not a serious problem because flows adapt their increase parameters after not observing a loss for a threshold time (line 4 in the algorithm). In fact, the AIMD principle of our approach enables us to ride through some of the inaccuracies in the loss estimations, though TCP-LASD is still very sensitive to accurate loss probability estimation. Comparison with the weighted TFRC work for achieving service differentiation is currently ongoing work, for which we expect to have results in the next month.

#### D. Simulation Results

In this section, we present the results of TCP-LASD simulations, done using the ns-2 simulator [13]. We consider the same network setting as in Figure 3. The network routers perform

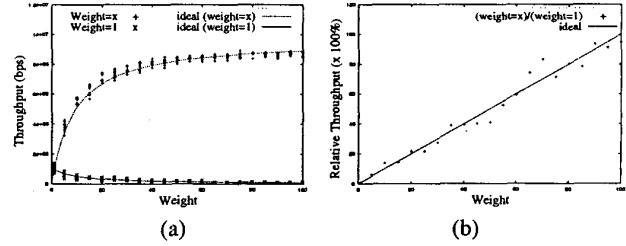


Fig. 8. Performance of TCP-LASD: (a) Throughput of the weighted flow ( $w = x$ ) in comparison with the other flows ( $w = 1$ ), (b) Relative throughput gain of the weighted flow ( $w = x$ ) over the other flows ( $w = 1$ )

RED algorithm with a large buffer to minimize the tail drop effect. The bottleneck link capacity is 10 Mbps and the round-trip delay is approximately 240 msec.

There are ten TCP-LASD flows where one flow has weight of  $w$  and the other nine flows have weight of 1. We measured the throughput and loss-event rate of each flow, varying  $w$  from 1 to 100. For each weight  $w$ , 10 simulation runs were executed. Each simulation was run for 600 simulation seconds.

In Figure 8, we plot the measured throughput with respect to weight. The x-axis represents the weight of the TCP-LASD flows, and the y-axis represents the average throughput. We have superposed the ideal throughput performance curve on the simulation results. From the figure, we observe that TCP-LASD scales up to a weight of 100, which is an order of magnitude improvement over TCP-SD. Figure 8(b) plots the relative throughput gain of the TCP-LASD connection with weight  $w$  over the other TCP-LASD connection with weight of 1. From the figure we clearly see that the throughput of TCP-LASD with weight  $w$  is approximately  $w$  times the throughput of TCP-LASD with weight 1 for the range of weights  $w = 1 - 100$ .

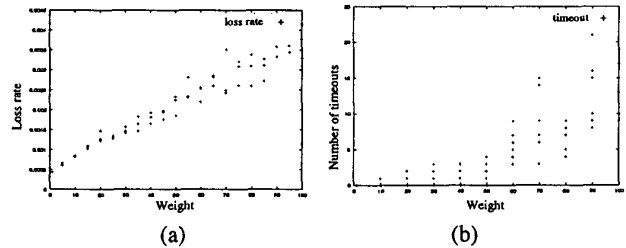


Fig. 9. Loss Measurement: (a) Average loss-event probability observed by the flows with respect to weight  $w$ , (b) Number of timeouts observed by the weighted flow

Figure 9 plots the loss probability measured at the sender and the number of timeouts. In general, we observe that the loss incurred by TCP-LASD is lower than the TCP-SD case, which was the motivation and design goal for TCP-LASD. From the low probability it naturally follows that the number of timeouts experienced by the weighted TCP-LASD flow is smaller than that of the weighted TCP-SD flow. Thus, the reduction of the loss rate and the number of timeouts contributes to the order of magnitude improvement in the scalability of TCP-LASD.

When we tested the same configuration with a lower bottleneck bandwidth of 1 Mbps, then the scalability achieved by TCP-LASD is up to  $w = 20$ . There are two reasons for this

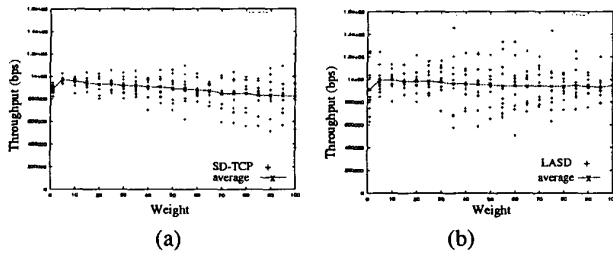


Fig. 10. Throughput performance (all flows with the same weight): (a) TCP-SD, and (b) TCP-LASD

downscaled saturation: (a) the loss probability increases up to 0.4–0.6% when  $w = 20$ , and (b) the flows with  $w = 1$  have already arrived at the minimum congestion window. In this case, the window decrease upon packet loss is negligible and they essentially act like an unresponsive sender.

Nevertheless, there is a significant drawback with TCP-LASD control. Since it adapts the additive increase phase of window control according to loss measurement in the past, and the expected window is very sensitive to an inaccurate measurement of loss rate as pointed out in the previous section, the resulting rate allocation of TCP-LASD flows is far less fair than that with TCP-SD. For example, in the above simulation in the single bottleneck configuration, we have, at times, observed that among unit weight TCP-LASD flows, the maximum throughput is twice the minimum throughput. This unfairness problem is more aggravated in a multiple link network as we will show in the next section.

## V. COMPARISON OF WEIGHTED AIMD AND LOSS ADAPTIVE AIMD

In this section, we compare the performance of TCP-SD and TCP-LASD. We consider the following scenarios: (a) when all flows have large weights, (b) when there are network dynamics due to introduction of new flows, and (c) when flows are sharing a multiple link network.

### A. All flows with weights

In Sections III and 4, we have considered a case when only one flow has a varying weight whereas all the other flows have unit weights. In this section, we simulate the following cases: (a) when all flows in the network have the same weight, and (b) when flows have different weights, e.g., some flows have weight of 50, and the others have weight of 100. We have considered the single bottleneck link configuration in Figure 3.

When all flows have the same weight, ideally we want the performance to be the same as when all the flows have unit weights. However, we know that having a flow with weight  $w$  is equivalent to having  $w$  flows with unit weight and thus from the analysis in Section III, it incurs large number of packet loss. Therefore, we expect the performance of both TCP-SD and TCP-LASD to degrade with the weight.

Figure 10 shows the throughput performance of TCP-SD and TCP-LASD when all the flows in the network has the same weights. There were ten flows sharing the bottleneck link shown in Figure 3. The dots in the figure represent the throughput of individual flows and the lines represent the average of them.

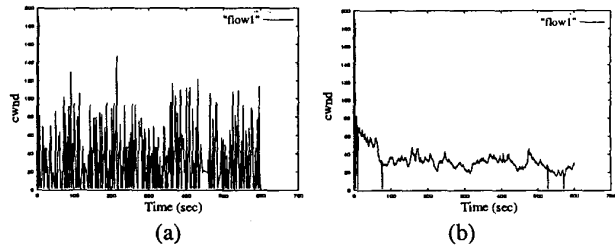


Fig. 11. Congestion window evolution of a flow (all flows with weight 20): (a) TCP-SD, and (b) TCP-LASD

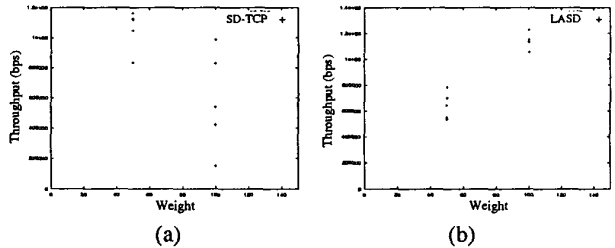


Fig. 12. Throughput performance (5 flows with weights 50, and the other 5 with weights 100): (a) TCP-SD, and (b) TCP-LASD

To our surprise, we observe that even with very large weight ( $w = 100$ ), the performance of both protocols does not degrade much. The average throughput of TCP-LASD is marginally higher than TCP-SD, but the throughputs of individual flows are much spread out.

Although, the average throughput performance is similar for both TCP-SD and TCP-LASD, the window evolution of TCP-SD and TCP-LASD is very different. Figure 11 presents the window evolution of a TCP-SD sender and a TCP-LASD sender with weight 20. In general, with high weights ( $w > 10$ ) TCP-SD repeats the timeout followed by slowstart all the time. In other words, most loss-events result in timeout and is recovered via slow start. Thus the window evolution is spiky, and result in high average throughput. In case of TCP-LASD, the congestion window evolution is much graceful and operates in congestion avoidance most of the time. For example, with weight of 20, TCP-SD flows experienced 155 timeouts on the average, while TCP-LASD had only 10 timeouts.

Now we are considering a case when the flows have different weights. In the same single link configuration with ten flows, we set five flows to have weight 50 and the other five flows to have weight 100. Ideally, we want the average throughput of the weight 100 flows to be twice that of the weight 50 flows.

Figure 12 presents the result. The x-axis represents the weight and the y-axis represents the throughput. The dots in the figure represents the throughput of the individual flows.

From the figure, we observe that TCP-SD fails to provide the desired service differentiation. Instead, due to the overshoot in the increase phase, the flows with weights 100 see more timeouts and achieves even less throughput. The problem is that TCP-SDs with large weights always operate in the slow start phase and the weighted adaptation during congestion avoidance cannot take effect. This problem cannot be solved by weighting  $\alpha$  in the slow start period. Weighting  $\alpha$  during the slow

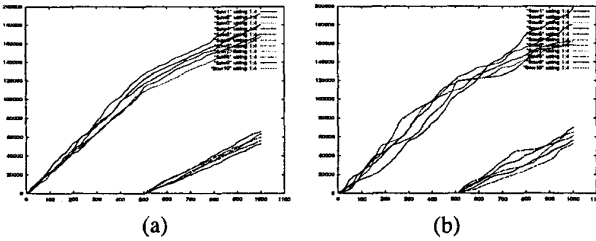


Fig. 13. Throughput performance (all flows with weights 1, five flows starts at 500 sec): (a) TCP-SD, and (b) TCP-LASD

start period results in too much aggressive increase in congestion window, and now the window adaptation behavior is always slowstart and timeout.

On the other hand, we observe that TCP-LASD achieves approximately 1:2 of throughput for weight 50 and 100 flows. Therefore, although the current instantiation of loss adaptive AIMD in TCP-LASD suffers from a severe unfairness problem due to the coarse loss estimation mechanism, there is a motivation for further investigation of this paradigm in the context of service differentiation.

### B. Dynamic Network Environments

So far, we have simulated the case when all the flows start almost at the same time. In this section, we present the performance of TCP-SD and TCP-LASD when the network condition dynamically changes, e.g. a sudden reduction of network bandwidth or introduction of new flows. In this scenario, we are interested in how each congestion control mechanism adapts to the network dynamics.

In the single bottleneck configuration shown in Figure 3, we start 5 flows initially, and then start 5 flows later (at 500 sec). Figure 13 presents the result plotting the throughput curve of each flow. We set all flows to have unit weights.

In this case, we are primarily interested in how fast each congestion control scheme adapts to the network dynamics and if the adaptation results in a fair manner. From the figure, we observe that both TCP-SD and TCP-LASD adapts fairly quickly to the introduction of new flows and now all the flows get about the same rate. In addition, as we have already observed, TCP-LASD flows results in a larger deviation in the average throughput (or less fair rate allocation) than TCP-SD flows.

### C. Multiple Link Configuration

Now we close the comparison of TCP-SD and TCP-LASD showing their performance in a simple multiple link configuration (see Figure 14).

There are 9 flows sharing the network, which consists of 4 backbone links and 18 access links. All links have 10 Mbps capacity and the round-trip time is set to approximately 200 msec. Among the long flows, two flows (flows 1 and 2) have varying weights and the other flows (flows 3 – 9) have unit weights.

Figure 15 presents the relative throughput gain of the weighted flows over the unit weight flows. From the figure, we observe essentially the same trend as we have observed in the single bottleneck link case: The throughput scalability of TCP-SD is limited to a small weight value ( $w < 10$ ) whereas the

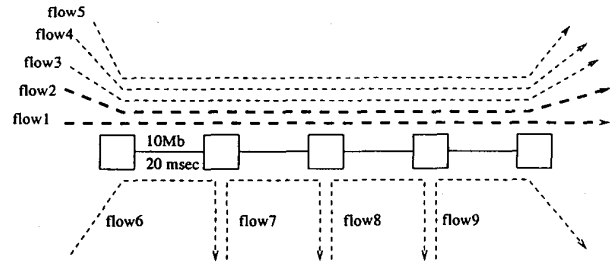


Fig. 14. Multiple Link Configuration

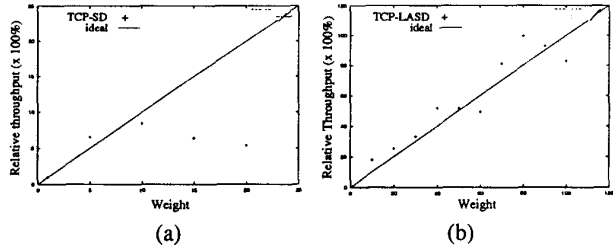


Fig. 15. Relative throughput gain: (a) TCP-SD, and (b) TCP-LASD

scalability of TCP-LASD scales up to a large weight ( $w \approx 100$ ). However, we observe a larger throughput variation with TCP-LASD in case of multiple link configuration than in the single link configuration.

## VI. BACKGROUND AND RELATED WORK

Weighted fair rate allocation among network flows has been considered as a candidate service model for differentiated services [3], [18]. Wang [18] has outlined the characteristics of a user-share differentiation (USD) model based on weighted bandwidth allocation in contrast to conventional diff-serv service models such as profile-based tagging and delay priority differentiation. Although the focus of the document is not to elaborate on the mechanism itself, USD assumes a support from network in order to enable the desired rate service differentiation.

A preliminary approach to achieve rate service differentiation by extending AIMD congestion control was proposed by Crowcroft et.al. [12]. Our results in Section 2 conform to the conclusion of their work.

In [6], the authors have employed AIMD congestion control for a group of flows rather than individual flows. A single congestion window is maintained for a group of flows sharing the same end-points and the flows are scheduled accordingly to share the resulting bandwidth. The aim of such an approach is to enable co-operative congestion management among a group of flows to achieve a high fairness index.

We have made use of the wide body of research that has analyzed the behavior of TCP [9], [10], [11]. The average throughput of TCP has been characterized as a function of the round-trip time and the observed packet loss probability of a flow by Mathis [9] and Mahdavi [10]. This analysis has been improved upon by Padhye et.al [11] by including the effects of timeouts, delayed acknowledgements and receiver flow windows. This characterization models the practical TCP behavior to a great

extent, and we use it in this paper to analyze the algorithms for service differentiation.

Floyd et.al. [10], [19] have proposed using the TCP throughput equation in [11] to perform rate based congestion control. Equation-based congestion control, as it is known, measures the round-trip time and observed loss probability to compute the TCP-friendly rate for a flow. The advantage of such an approach lies in the fact that it provides smooth congestion control, avoiding abrupt variations in sending rate. Unfortunately, it is also susceptible to inaccuracies in the measurement of loss probability. Our ongoing work seeks to compare loss adaptive AIMD to a weighted version of equation-based congestion control.

## VII. DISCUSSION AND SUMMARY

Let us now consider some of the issues with achieving throughput service differentiation using purely end-to-end congestion control algorithms. Broadly, there are issues at three levels: (a) implementation issues such as the accuracy of loss measurement, (b) fairness models and TCP friendliness, and (c) philosophical issues such as the role of end-to-end versus network functionality in achieving service differentiation.

### A. Loss Measurement

Accurate measurement of packet loss probabilities using purely end-to-end measurements without any coordination between the data senders in a network is an inherently difficult problem. In fact, this is a deep research topic in itself, and our goal in this paper is not to develop new loss measurement mechanisms, but rather to adapt what is currently available for our purpose. Even at the router level, there is no unanimous acceptance of a definition for “fair dropping”, particularly when routers do not maintain per-flow state or assume coordination of edge routers. Certainly, most of the fair dropping work in literature has striven to achieve “longer term” (order of a few round trip times) rather than “short term” (order of a packet lifetime) fairness, and the goal has typically been on providing the same expected value of loss probability for all packets rather than the minimizing variance in loss probability that different flows may see over a few round trip times. Pragmatically, routers may be considered to be “fair” if they can provide the same expected loss probability for all packets in a short time span. From the end-to-end perspective, related work on loss measurement has typically sampled the number of packets sent successfully between successive loss events, and then used stochastic models of varying sophistication to predict loss probability. For small loss probabilities, the sample size must be large; however under dynamic conditions, older samples may be worthless. This conflict makes the accurate tracking the varying loss probability inherently difficult.

Consider even a single link that has a packet loss probability of  $p$ . The expected number of packets received between two losses is  $1/p$  for small  $p$ . On the other hand, the probability of receiving  $n$  packets before observing a loss is  $(1-p)^n p$ . Since the *mean* value is different from the *mode* value, simply collecting a few samples and averaging in some form (as we have done in this work, following similar approaches in recent work [19]) may result in large variations of the estimated loss probability. Unfortunately, this severely impacts the short-term fairness

property of the congestion control algorithm for loss adaptive AIMD. To the best of our knowledge, none of the state-of-the-art end-to-end mechanisms for estimating loss probability have claimed to be very accurate, and this is a serious problem that can possibly only be resolved with some addition information from the network routers.

### B. Fairness and TCP friendliness

Recalling our first statement in this paper, the point of end-to-end congestion control is to achieve network stability in a cooperative manner. Cooperative sharing of a common resource automatically leads to the question: “what is the fairness model for arbitrating shared resources among competing/cooperating flows?” Let us digress from the main focus of the paper for a moment, and discuss an analytical model for fairness following the work of Kelly et.al. [8].

In [17], the authors model congestion control as an optimization problem of a function  $J(x) \leftarrow \alpha \cdot U(x) - \beta \cdot p \cdot x$ , where  $U(x)$  is the utility function for the flow,  $x$  is the rate,  $p$  is the loss probability, and  $\alpha$  and  $\beta$  are constants. While we do not have the space to discuss the significance of this optimization problem in detail, we can broadly think of the first term as the “utility” of the rate to a flow and the second term as the “price” paid by the flow. The authors show that for any concave, continuous and differentiable utility function  $U(x)$ , this optimization problem corresponds to an end-to-end congestion control algorithm.

The goal of an end user is to optimize  $J(x)$ , i.e. converge to a rate allocation  $x$  such that  $J'(x) = 0$ . This implies that at the optimal point,

$$\alpha \cdot U'(x) - \beta \cdot p = 0$$

Multiplying by  $x^2$ , at the optimal point  $\alpha \cdot u'(x) \cdot x^2 - \beta \cdot p \cdot x^2 = 0$ . We can thus converge to the optimal rate allocation by setting

$$dx/dt \leftarrow \alpha \cdot u'(x) \cdot x^2 - \beta \cdot p \cdot x^2 = 0$$

It turns out that the congestion control algorithm of AIMD is  $dx/dt \leftarrow \alpha - \beta \cdot p \cdot x^2$ , which is identical to the optimization problem above for  $U'(x) = \frac{1}{x^2}$ . In other words, the AIMD congestion control corresponds to a utility function of  $U(x) = K - 1/x$ . The fairness model achieved with this utility function is called *min potential delay* fairness, because the goal is to minimize  $1/x$ , which is intuitively equivalent to minimizing delay [7], [17].

Now consider the loss adaptive AIMD congestion control function. In this case,  $dx/dt \leftarrow \alpha/p - \beta \cdot p \cdot x^2$ . This solution converges to the rate allocation given by

$$\alpha/p - \beta \cdot p \cdot x^2 = 0$$

Multiplying by  $p$  and factoring, loss adaptive AIMD converges to the rate allocation that maximizes  $J(x)$  at  $\sqrt{\alpha} - \sqrt{\beta} \cdot p \cdot x = 0$ . Now consider the congestion control algorithm

$$dx/dt \leftarrow \alpha \cdot U'(x) \cdot x - \beta \cdot p \cdot x^2$$

This congestion control algorithm has the same form as the loss adaptive AIMD congestion control for  $U'(x) = 1/x$ . In other words, the TCP-LASD congestion control corresponds to a utility function of  $U(x) = \log(x)$ , which results in proportional fairness [8], [17]. This discussion points to the subtle difference

in the fairness model when we make the AIMD parameters “loss adaptive”. In fact, in retrospect it should be intuitively obvious to the reader that the fairness model ought to be different for loss adaptive AIMD (as compared to AIMD), because  $p \propto n^2$  in AIMD, while  $p \propto n$  in loss adaptive AIMD.

Note that the entire preceding discussion has focused on the impact of “loss adaptivity” on AIMD parameters, and not the impact of weighted congestion control. In fact, the weighted version of any congestion control algorithm in the weighted AIMD form we considered in Section 2 will achieve the service differentiation principle in theory, so long as  $f_{inc} \times f_{dec} \propto w^2$ . This leads to a very interesting result: in order to achieve *differentiation* in a scalable way, the *cooperation* principle of AIMD must be modified!

A final concern, critical for the deployment of any congestion control algorithm, is its TCP-friendliness property. We have shown that AIMD and loss adaptive AIMD follow fundamentally different fairness models. However, because of the way that we upper bound the increase factor to 1 in the loss adaptive AIMD algorithm, a flow that uses this algorithm will never increase more aggressively than a contending AIMD flow. In other words, in a network that consists of a mix of TCP and TCP-LASD flows, TCP flows will gain an unfair advantage over TCP-LASD flows and not vice versa.

### C. End-to-end versus Network Functionality

At this point, we have discussed a number of issues dealing with the design, implementation, and properties of the two classes of end-to-end congestion control algorithms: weighted AIMD and loss adaptive weighted AIMD. Let us now step back and reconsider the broader question: “what is the role of end-to-end congestion control in achieving service differentiation?”

At one extreme, one could argue for a quality of service architecture that provides a rate controller for each flow in the access router serving the host. In this architecture, the end host only needs to interact with its access router to discover the flow rate and adapt itself accordingly. At the other extreme, one could argue for the sort of model that we started out with in this paper, wherein we do not assume any special functionality from the network routers - nothing that TCP does not assume for its fair operation currently - and provide the mechanisms for weighted congestion control in the end hosts.

Interestingly, the mechanisms that we have considered in this paper are applicable to both the network models described above. In fact, the same weighted AIMD and loss adaptive weighted AIMD algorithms that we implemented in TCP can be implemented in end host middleware such as a Congestion Manager [6], or edge routers of each network cloud. Of course, the granularity of the “flow” in each case will be different, but the principle of rate or window adaptation will remain the same. So without bias to the specific network model, we can rephrase the question: “can the congestion control mechanisms considered in this paper provide viable service differentiation?”

Our conclusions are mixed. Clearly, we have seen that AIMD has fundamental restrictions that limit its ability to scale for large weights (within the framework that we considered). While loss adaptive AIMD does scale by an order of magnitude, it is very sensitive to loss estimation, which is an inherently difficult problem. This is where the coordination between the network

routers and the end hosts may prove to be extremely useful. If the network routers can indicate their loss probability, then using the mechanisms that we proposed in this paper, one could conceivably achieve scalable *and* short-term fair service differentiation.

To summarize, we believe that the weighted AIMD approach will not scale, but is applicable for limited service differentiation because it is fully compliant with the existing fairness model of TCP. We believe that the loss adaptive weighted AIMD will scale, but it has serious deployment problems for two reasons: (a) it is sensitive to the accuracy of loss measurement, and (b) its fairness model is different from TCP, which may cause TCP-LASD flows to get unfairly poor treatment in a mixed flow composition. We do believe that the loss adaptive AIMD approach can indeed overcome the first problem if there is adequate network feedback in terms of the expected loss probability. Thus, it is potentially a viable candidate for deployment in an autonomous network cloud wherein all the edge routers perform loss adaptive AIMD, while requiring stateless and very simple feedback from the routers.

### REFERENCES

- [1] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queueing algorithm,” in *ACM SIGCOMM*, August 1989.
- [2] S. Floyd and V. Jacobson, “Random Early Detection gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.
- [3] K. Nichols, S. Blake, F. Baker, and D. L. Black, “Definition of the Differentiation Services Field (DS Field) in the IPv4 and IPv6 Headers,” RFC 2474, December 1998.
- [4] V. Jacobson, “Congestion Avoidance and Control,” in *ACM SIGCOMM*, August 1988.
- [5] R. Rejaie, M. Handley, and D. Estrin, “RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet,” in *IEEE INFOCOM*, March 1999.
- [6] H. Balakrishnan, H. S. Rahul, and S. Seshan, “An Integrated Congestion Management Architecture for Internet Hosts,” in *ACM SIGCOMM*, September 1999.
- [7] L. Massoulié and J. Roberts, “Bandwidth Sharing: Objectives and Algorithms,” in *IEEE INFOCOM*, March 1999.
- [8] F. P. Kelly, A. Maulloo, and D. Tan, “Rate control in communication networks: shadow prices, proportional fairness and stability,” *Journal of the Operational Research Society*, vol. 49, pp. 237–252, March 1998.
- [9] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm,” *ACM Computer Communications Review*, vol. 27, pp. 67–82, July 1997.
- [10] J. Mahdavi and S. Floyd, “TCP-friendly unicast rate-based flow control.” Note sent to end2end mailing list, January 1997.
- [11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP Throughput: A Simple Model and its Empirical Validation,” in *ACM SIGCOMM*, September 1998.
- [12] P. Oechslin and J. Crowcroft, “Weighted Proportionally Fair Differentiated Service TCP,” *ACM Computer Communications Review*, vol. 28, July 1998.
- [13] “ns-2 Network Simulator.” <http://www-mash.cs.berkeley.edu/ns/>, 1998.
- [14] H. Adishesu, G. Parulkar, and G. Varghese, “A Reliable and Scalable Striping Protocol,” in *ACM SIGCOMM*, August 1996.
- [15] D.-M. Chiu and R. Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” *Computer Networks and ISDN Systems*, vol. 17, pp. 1–14, 1989.
- [16] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, “TCP Behavior of a Busy Internet Server: Analysis and Improvements,” in *IEEE INFOCOM*, March 1998.
- [17] S. Kunniyur and R. Srikant, “End-to-End Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks,” in *IEEE INFOCOM*, March 2000.
- [18] Z. Wang, “User-Share Differentiation (USD): Scalable bandwidth allocation for differentiated services.” Internet Draft <http://www.bell-labs.com/user/zhwang/diff-serv/usd.txt>, July 1997.
- [19] S. Floyd, M. Handley, J. Padhye, and J. Widmer, “Equation-based Congestion Control for Unicast Applications.” <http://www.aciri.org/tfrc/>, January 2000.