Exploring Congestion Control[1]

Aditya Akella[2] Srinivasan Seshan[3] Scott Shenker [4]
Ion Stoica[5]

May 2002
CMU-CS-02-139

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

From the early days of modern congestion control, ushered in by the development of TCP's and DECbit's congestion control algorithm and by the pioneering theoretical analysis of Chiu and Jain, there has been widespread agreement that linear additive-increase-multiplicative-decrease (AIMD) control algorithms should be used. However, the early congestion control design decisions were made in a context where loss recovery was fairly primitive (*e.g.* TCP Reno) and often timed-out when more than a few losses occurred and routers were FIFO drop-tail. In subsequent years, there has been significant improvement in TCP's loss recovery algorithms. For instance, TCP SACK can recover from many losses without timing out. In addition, there have been many proposals for improved router queueing behavior. For example, RED active queue management and Explicit Congestion Notification (ECN) can tolerate bursty flow behavior. Per-flow packet scheduling (DRR and Fair Queueing) can provide explicit fairness.

In view of these developments, we seek to answer the following fundamental question in this paper: Does AIMD remain the sole choice for congestion avoidance and control even in these modern settings? If not, can other mechanism(s) provide better performance?

We evaluate the four linear congestion control styles – AIMD, AIAD, MIMD, MIAD – in the context of these various loss recovery and router algorithms. We show that while AIMD is an unambiguous choice for the traditional setting of Reno-style loss recovery and FIFO drop-tail routers, it fails to provide the best goodput performance in the more modern settings. Where AIMD fails, AIAD proves to be a reasonable alternative.

[1]This research was sponsored by DARPA under contract F30602-99-1-0518. Additional support was provided by IBM. Views and conclusions contained in this document are those of the authors and should not be interpreted as respresenting the official policies, either expressed or implied, of DARPA, IBM or the U.S. government.

[2]Computer Science Department, Carnegie Mellon University (aditya@cs.cmu.edu).
[3]Computer Science Department, Carnegie Mellon University (srini@cmu.edu).
[4]Internet Research Center at ICSI (ICIR), Berkeley (shenker@icir.org).
[5]Computer Science Department, UC Berkeley (istoica@cs.berkeley.edu).

Keywords: Congestion Control, Active Queue Management

1 Introduction

The first sophisticated transport congestion control algorithms, developed almost simultaneously for DECbit [1] and TCP [2], employed Additive-Increase Multiplicative-Decrease (AIMD) window adjustment algorithms. A later theoretical study [3] confirmed, in a simple model with synchronous congestion signals and static bandwidth, that AIMD was the only fair and stable choice among the four linear alternatives AIMD, AIAD, MIMD and MIAD. In the past decade, due to the tremendous success of TCP congestion control and to the enduring persuasiveness of [3], the superiority of AIMD has become a widely accepted and deeply held belief.

As a result, there have been very few research studies advocating, or even exploring, linear schemes other than AIMD. While there have been many papers on congestion control, most of them investigate algorithmic issues that fall well within the AIMD paradigm. Of those departing from the AIMD paradigm, the majority propose either non-linear congestion control algorithms [4] or approaches that differ radically from TCP [5, 6]. Notable exceptions to this statement are [7] (discussed in Section 5) and [8, 9] (discussed in Section 6) which propose linear control algorithms other than AIMD.

In this paper, we revisit the question of whether AIMD is indeed the only reasonable choice among the various linear congestion control schemes. We seek to *explore* these other options for linear congestion control and to see if any of them could theoretically serve as a viable option for a modern congestion control algorithm. As such, we are ignoring all issues of TCP compatibility (or TCP-friendliness) and incremental deployment.

We ask this question because the early development of TCP congestion control was done in the context of Reno-style loss recovery and FIFO drop-tail routers. TCP-Reno reacts fairly *severely* to losses. If a Reno flow incurs more than a few losses within a given window, it times out and restarts. Thus, in the past, it was important that the window adjustment algorithm increase its window conservatively to avoid multiple losses. However, much progress has been made on loss recovery algorithms in the past decade. The more modern loss recovery recovery schemes, like SACK [10, 11], incur only a *gentle* penalty from losses since they can endure many packet drops within a single window without restarting. Hence, there may be less of a need for conservative window adjustment algorithms.

In addition, the drop and scheduling policies at routers have changed significantly from the early days of TCP. The need for Active Queue Management (AQM) is widely accepted and the RED algorithm [12] is widely implemented (although it isn't clear how widely deployed it is). AQM in general, and RED in particular, give flows early congestion signals – well before the physical queue is exhausted – and can better absorb bursts of packets. This reduces the burstiness of the loss patterns and lessens the chance that TCP needs to time-out and restart. Explicit congestion notification (ECN) [1, 13] goes even further, giving congestion signals without losses. Thus there is almost no penalty upon loss when TCP SACK is employed in conjunction with ECN.

In addition to modifying the drop-policies at routers, there have also been calls for routers to adopt per-flow queueing schemes, like Deficit Round Robin (DRR) [14, 15], that explicitly ensure fairness between flows. If such schemes became widely deployed (and some ISPs are now deploying routers with this capability), then one need not require TCP's window adjustment algorithm to provide fairness. While the days of widespread deployment of DRR or similar algorithms are, at best, far in the future, in this paper, we evaluate if such a deployment would allow us to use different window adjustment schemes.

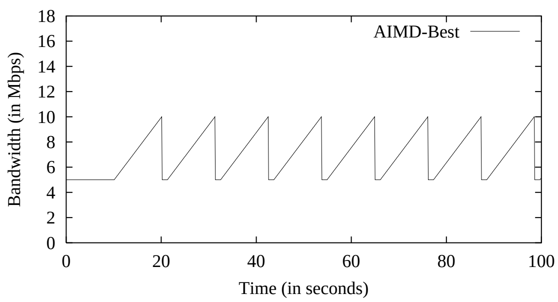
Most flows in the Internet carry only a few packets, and thus the available bandwidth seen by long-lived flows can fluctuate substantially. Also, the traffic load model might change with time as new applications arise. Due to these reasons, in this paper we seek to evaluate the performance of the congestion control algorithms under a *wide* variety of bandwidth variations, some of which might even appear extreme at first sight. This is in contrast to most of the past congestion control evaluation and simulation studies that looked primarily at cases where the available bandwidth was constant.

In this paper, we use simulations in NS-2 [16] to evaluate how the four linear congestion control scheme would function in the various settings of loss recovery mechanisms and queue management schemes described above against the backdrop of a variety of bandwidth variations. We focus on the effect that each setting has on the relative performances of the algorithms.

The primary metric by which we evaluate these various schemes is the goodput (fraction of available bandwidth used to transmit distinct data packets) achieved in these various scenarios. In the scenarios with varying bandwidth, the key to achieving high goodput is the ability to *track* the available bandwidth, that is, the ability to keep up with its variations without significantly overshooting or undershooting. We also evaluate the fairness, delay, and loss rate properties of these congestion control schemes.

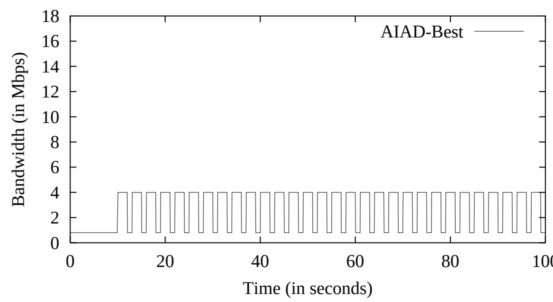
Our simulations show that AIMD is the superior design choice in the traditional setting of TCP Reno loss recovery and FIFO drop-tail routers. However, when we consider the modern developments mentioned above, AIMD is no longer superior. TCP SACK, active queue management techniques and fair queueing in routers enable the other linear alternatives to provide comparable and sometimes significantly better goodput performance. We observe that AIAD is always among the best linear alternatives, and can even achieve fairness as long as routers are not FIFO drop-tail. Moreover, we show, via analysis and simulations, that hybrid linear algorithms in which the linear increase and decrease need not be purely additive or purely multiplicative can alleviate the fairness issues of AIAD and also provide good performance.

The rest of the paper is organized as follows. In Section 2, we compare the tracking ability of the various window adjustment schemes. In Section 3, we lay the framework used in this paper to compare and contrast various congestion control algorithms. Section 4 presents the results of our study. In Section 5, we briefly revisit the issue of fairness of linear congestion control schemes. In Section 6, we present related work. Finally, Section 7 summarizes the paper.



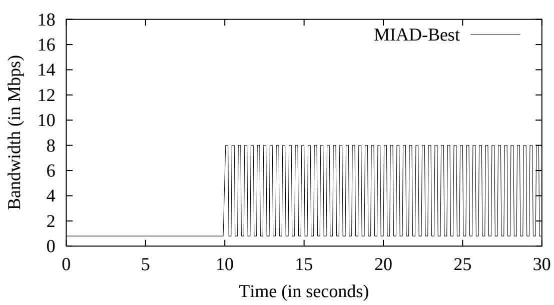
AIMD	AIAD	MIMD	MIAD
0.97	0.93	0.95	0.61

(a)



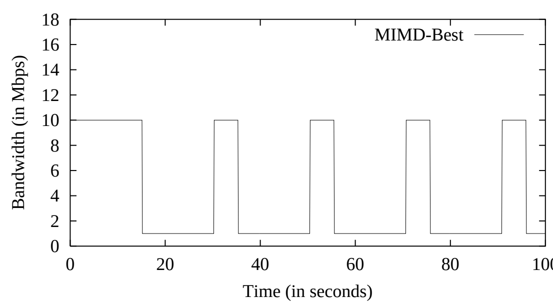
AIMD	AIAD	MIMD	MIAD
0.52	0.96	0.75	0.82

(b)



AIMD	AIAD	MIMD	MIAD
0.87	0.94	0.90	0.96

(c)



AIMD	AIAD	MIMD	MIAD
0.77	0.79	0.89	0.84

(d)

Figure 1: Good cases for each of the four canonical choices. The goodput seen by the various algorithms for the given variation is shown in the table below each plot. The goodput, measured as the fraction of average available bandwidth used to transmit unique packets, is a number in $[0, 1]$.

2 Is AIMD Clearly Superior?

Our first question is whether AIMD is clearly superior to the other choices in terms of the goodput achieved. If it was, then our subsequent analysis would be moot and the entire paper a misguided exercise. However, as we show in this section, for each of the four algorithms we can find scenarios in which it performs the best and the worst among the four algorithms. In fact, this suggests that special care needs to be taken in deciding which algorithm is the best.

To maximize the usage of the available bandwidth, a congestion control scheme needs to balance between (1) tracking rapid changes of the available bandwidth, and (2) minimizing packet losses. The faster a sender modifies its window size, the faster the sender can track changes of the available bandwidth. On the other hand, fast and large changes of the window size increase the probability of the sender overshooting the available bandwidth, which may result in a large number of packets being dropped. This has two negative implications. First, more losses mean that more packets are retransmitted, and thus a higher fraction of the available bandwidth is devoted to retransmitting old packets. Second, a burst of losses can hurt the loss recovery algorithm by forcing it to restart.

The canonical congestion control schemes we consider use either multiplicative or additive schemes to vary the window size. When the available bandwidth increases slowly, an additive increase will likely outperform a multiplicative increase since it is fast enough to track the changes and is less likely to overestimate the available bandwidth. In contrast, a multiplicative increase will likely perform better when bandwidth increases are large and abrupt. The same reasoning applies to bandwidth decreases and the window decrease algorithms. Intuitively, this is the reason why no single canonical congestion control scheme would be able to dominate across a wide range of scenarios. Next, we present results supporting this observation.

Figure 1 shows four patterns of bandwidth variations. We measure the goodput for the four linear algorithms in these four scenarios.[1] The scenarios are chosen such that in each case there is a different algorithm that achieves the highest throughput. We now describe the results in more detail.

Figure 1(a) shows a saw-tooth bandwidth pattern under which AIMD performs the best. Compared to AIMD, the window size under MIMD and MIAD increases too fast, while under AIAD decreases too slowly. As a result these schemes experience more losses, and consequently more retransmissions than AIMD. The significantly worse goodput of MIAD is due to the fact that MIAD experiences a larger number of timeouts.

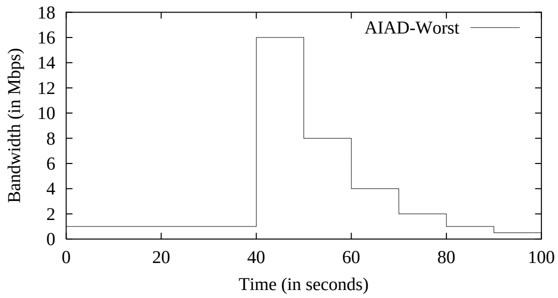
Figure 1(b) shows an example in which AIAD performs the best. The reason for this result is somewhat more subtle. When the available bandwidth drops, AIAD reduces the window size slower than the other disciplines (except MIAD). While this causes AIAD to lose slightly more packets, the decrease of the window size is not enough to offset the increase of the window size during the previous high bandwidth period. Thus, the window size of AIAD increases continuously over multiple high bandwidth periods. Moreover, since the duration of a low bandwidth period is short, TCP SACK recovers from packet losses without experiencing retransmission timeouts. In contrast, MIAD and MIMD cannot avoid timeouts as they constantly overshoot the available bandwidth. Finally, AIMD does not perform well because the window decrease during the low periods almost offsets the window increase during the high periods.

Compared to the previous experiment, in the scenario presented in Figure 1(c) the high and low periods are shorter and the high bandwidth value is larger. These changes are enough to give advantage to MIAD over AIAD. MIAD no longer overshoots when the bandwidth increases, while AIAD suffers from the fact that it cannot increase the window size fast enough during the high periods.

Figure 1(d) presents a scenario which makes MIMD the winner. This is again a square pattern bandwidth variation, but the periods are much longer. When the bandwidth decreases, AIAD and MIAD lose too many packets, and TCP SACK is no longer able to avoid retransmission timeouts. On the other hand, AIMD cannot exploit the available bandwidth as its window size increases too slowly during the high periods. This leaves MIMD as the only discipline which can adequately track the abrupt changes of the available bandwidth.

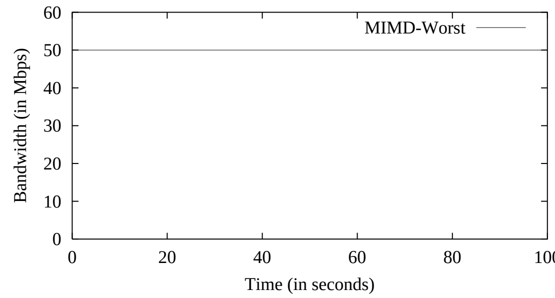
Not only we are able to construct scenarios that make any of the canonical congestion control schemes a winner, but we can also construct scenarios that make any of these congestion control schemes a *loser*. For instance, AIMD exhibits the worst performance in the experiments presented in Figures 1(b), 1(c) and 1(d) and MIAD performs the worst in Figure 1(a). For completeness, in Figure 2 we present two traffic scenarios in which AIAD and MIMD perform the worst. In Figure 2(a), both the additive increase algorithms are too slow to catch up with the sudden increase of the available bandwidth. Between AIAD and AIMD, AIAD cannot track the rapid decrease in the available bandwidth as well, and thus it ends up losing more packets than AIMD. Figure 2(b) shows an example where MIMD performs the worst. In this case, both the multiplicative increase algorithms overshoot the available bandwidth by significant amounts and time-out

[1] The simulation details are described in Section 3.3. For these simulations we use drop-tail FIFO routers with TCP SACK.



AIMD	AIAD	MIMD	MIAD
0.67	0.65	0.90	0.83

(a)



AIMD	AIAD	MIMD	MIAD
0.96	0.96	0.21	0.38

(b)

Figure 2: Bad cases for AIAD (Figure(a)) and MIMD (Figure (b)). Notice that the y-axes are on different scales.

often. MIAD performs better than MIMD because the additive decrease allows it to keep substantially higher number of packets outstanding than MIMD.[2]

In summary, there is no clear winner or loser among the four canonical congestion control schemes. As shown in this section, for each congestion control scheme, we can devise a bandwidth variation pattern that can make that scheme either a winner or a loser. However, the *relative* performance of the various schemes is different across scenarios. To better compare the behavior of these congestion control schemes, we develop an evaluation methodology based on competitive analysis. This is the subject of the next section.

3 Evaluation Methodology

In order to meaningfully compare the algorithms, we make two key guiding assumptions. Our first guiding assumption is that congestion control algorithms should not be designed for any particular scenario, no matter how realistic that scenario may be at the time. This is because, the load model in the Internet may change abruptly as new applications arise or as the nature of the infrastructure changes. Thus, congestion control algorithms should be evaluated across a wide variety of scenarios. Our second guiding assumption is that robustness is more important than optimality, that is, we demand that the congestion control algorithm perform reasonably in most situations and are more concerned with its worst-case performance than its best-case performance.

To embody these guiding assumptions in a concrete methodology, we borrow an approach similar to that used in the study of competitive algorithms [17]. Let A be the set of congestion control algorithms that we wish to compare. For our study, A consists of AIMD, AIAD, MIMD and MIAD. Let E be the set of possible environments or scenarios that these algorithms might be faced with, where a scenario is a particular variation in the available bandwidth.

For the particular quantity of interest – be it goodput, fairness, loss, or delay – let $s_a(e)$ denote the *score* of algorithm a in a given environment e. Let $s_{max}(e) = max_{a \in A}\{s_a(e)\}$ denote the best score achieved in scenario e among the algorithms in A. Let $d_a(e) = s_{max}(e) - s_a(e)$; $d_a(e)$ is a measure, for a given environment e, of how close a comes to matching the best performance among the algorithms in A. Out of these per-environment scores we define two aggregate scores. The rank C_a is the worst-case score among the various environments: $C_a = max_{e \in E}\{d_a(e)\}$. The rank measures the worst-case performance of the algorithm, and lower ranks represent more robust algorithms, those that never do particularly poorly. The other measure is the aggregate value D_a of the differences: $D_a = \sum_{e \in E} d_a(e)$. The lower the values of D_a and C_a are, the better the algorithm a is, for the given metric. Table 1 outlines the notation used in this section.[3]

[2]Although these scenarios look extreme and abrupt, it is possible to construct traffic models (e.g., periodic packet bursts or periodic outages) that result in variations roughly similar to the ones shown here.

[3]We could have also defined $d_a(e) = \frac{s_{max}(e)}{s_a(e)}$. We have tried this alternate definition and the results are similar to the one

Our approach is intended to capture the basic qualitative behavior of these algorithms. However, the approach does have a few limitations. If a wide enough range of algorithms are not explored, then the scores $d_a(e)$ do not really represent the deviations from what a good algorithm does on that environment. They would only represent the deviations from the particular algorithms in A. If a wide enough set of environments are not sampled, then similar problems arise. If one focuses on environments that are too extreme, then these dominate the worst case results and may alter the rankings. Thus, choosing the sets A and E is crucial to this method, and we now explain our choices.

Notation	*Description*
$s_a(e)$	The score of algorithm a in environment e
$s_{max}(e)$	$max_a\{s_a(e)\}$
$d_a(e)$	$s_{max}(e) - s_a(e)$
C_a	$max_a\{d_a(e)\}$, Rank of an algorithm a (measuring worst-case performance)
D_a	$\frac{\sum_{e \in E} d_a(e)}{\lVert E \rVert}$, metric for average performance

Table 1: Notation

	Environment	*Variation in per-flow available bandwidth (B_t)*
1	**cons-low (CL)**	constant at 1Mbps
2	**cons-high (CH)**	constant at 10Mbps
3	**sq-low (SQL)**	10Mbps→5Mbps→10Mbps. . ., at regular intervals (5s)
4	**sq-high (SQH)**	10Mbps→1Mbps→10Mbps. . ., at regular intervals (5s)
5	**rw-low (RWL)**	$B_{t+1} \in [B_t - \Delta, B_t + \Delta]$, $B_0 =$ 1Mbps, $\Delta = 0.5B_0$
6	**rw-high (RWH)**	same as above except that $B_0 =$ 10Mbps
7	**rd-low (RDL)**	$B_t \in [0, B_0]$, $B_0 =$ 1Mbps
8	**rd-high (RDH)**	$B_t \in [0, B_0]$, $B_0 =$ 10Mbps
9	**rw-additive (RWA)**	$B_{t+1} \in [0, B_t + \Delta]$, $B_0 =$ 1Mbps, $\Delta = 0.25B_0$
10	**rw-multiplicative (RWM)**	$B_{t+1} \in [0, \mu B_t]$, $B_0 =$ 1Mbps, $\mu = 5/3$
11	**real-cons-low (RCL)**	Pareto length flows arrive with Poisson inter-arrival times
12	**real-cons-high (RCH)**	Same as in real-cons-low, except that the mean inter-arrival time is very low
13	**real-sq (RSQ)**	Mean inter-arrival time varies in a square manner

Table 2: Environments

3.1 Choosing the Set E of Environments

In choosing the components of E, we aim to include enough environments to cover a wide variety of situations while still keeping the set small enough to be manageable. We deliberately choose some of the environments to be fairly extreme. The goal for these is not to be realistic, but to test the algorithms under unusually harsh conditions. We include a few environments that reflect reasonably realistic scenarios. Finally, we add few other environments that we hope would help reveal key aspects of the algorithms' behavior. The resulting composition for the set E is shown in Table 2.

Most of the scenarios are on a simple topology (described later) where the single congested link has varying available bandwidth. The basic bandwidth variations we consider are described below (we describe the variation in *per-flow* available bandwidth):

- Constant: The available bandwidth is constant. We included a cons-low (CL) environment where the constant bandwidth is low (1Mbps) and a cons-high (CH) environment where the constant bandwidth is high (10Mbps).

we consider.

- Square-Wave: The available bandwidth undergoes square wave oscillations, with the bandwidth variations occurring every 5 seconds. In the sq-low (SQL) scenario, the bandwidth varies between 5Mbps and 10Mbps. In the sq-high (SQH) scenario, the bandwidth varies between 1Mbps and 10Mbps.

- Random Walk (RWL, RWH): Here the bandwidth varies according to a random walk. If the bandwidth at time t is B_t then the next bandwidth is chosen uniformly from the interval $[B_t - \Delta, B_t + \Delta]$ where $\Delta = 0.5B_0$. For rw-low (RWL), $B_0 = 1$Mbps and for rw-high (RWH), $B_0 = 10$Mbps.

- Random (RDL, RDH): The bandwidth is chosen uniformly randomly from the interval $[0, B_0]$ where $B_0 = 1$Mbps for rd-low (RDL) and $B_0 = 10$Mbps for rd-high (RDH).

- Additive Random Walk (RWA): For the environment rw-additive (RWA), the available bandwidth is chosen from an additively constrained interval [18]. That is, the bandwidth at time $t + 1$ is chosen uniformly at random from the interval $[0, B_t + \Delta]$. Here, $B_0 = 1$Mbps, $\Delta = 0.25$.

- Multiplicative Random Walk (RWM): In the environment rw-multiplicative (RWM), the available bandwidth is picked from a multiplicatively constrained interval [18]. In other words, B_{t+1} is chosen uniformly at random from the interval $B_{t+1} \in [0, \mu B_t]$. Here $B_0 = 1$Mbps, $\mu = 5/3$.

- Realistic Cross Traffic: The variation in available bandwidth is determined by pareto-length flows arriving at the bottleneck router with Poisson inter-arrival times. For the environment real-cons-low (RCL), the mean inter-arrival time is 0.03s, while for real-cons-high (RCH) it is 0.01s. For the environment real-sq (RSQ), the mean varies in a square manner between 0.03s and 0.01s, where the variation in mean occurs every 5 seconds. These scenarios were chosen to reflect realistic load and cross-traffic models.

In all the cases 1 through 10 listed in Table 2, whenever B_t exceeds the capacity of the link, C, we set it to C.

3.2 Choosing the Set A of Algorithms

For $LC \in$ {AIMD, AIAD, MIMD, MIAD}, let$LC(a, b)$ denote a linear congestion control scheme with an increase parameter of a and a decrease parameter of b. For example, the window increase and decrease equations for MIMD(1.5, 0.5) are $W_{t+1} = 1.5W_t$ and $W_{t+1} = 0.5W_t$, respectively.

For each of the four linear control schemes, we would like to pick a *single* set of parameters that provides reasonable performance across *all* possible settings of loss recovery schemes, router algorithms and bandwidth variations. Such a choice would ensure two key properties: (1) the single algorithm in each case (for example, AIMD(1,0.1)) would best summarize the overall behavior of the entire family of linear control schemes that the algorithm belongs to (for example, AIMD). (2) the single algorithm would ensure near-optimal performance across all possible settings.

In addition, while choosing the parameters (a, b) of such a representative algorithm we try to ensure that the choice does not obscure the core qualities of increase and decrease of each linear control algorithm. For example, we do not want to pick the decrease parameter of the candidate AIMD algorithm to be very close to 1, lest it should look similar to an additive decrease. Clearly, this property needs to hold over the entire range of sizes of the congestion window spanned by the bandwidth variations that the algorithms are exposed to. Subjectively, we list out the following conditions to be satisfied by the linear control algorithms over all possible window sizes:

- Additive Increase (AI): The additive increase component should be such that at no instant of time should the window undergo an increment greater than about 10% the current size. This serves to distinguish an additive increase form a multiplicative increase.

- Additive Decrease (AD): The additive decrease component should be such that the decrement in the window should never be more 10%. This serves to differentiate it from a multiplicative decrease.

- Multiplicative Increase (MI): The multiplicative increase component should be large enough so that the increment in size of the window is larger than 10% always.

- Multiplicative Decrease (MD): The multiplicative decrease component should be so chosen that the window decrement is never less than 10%.

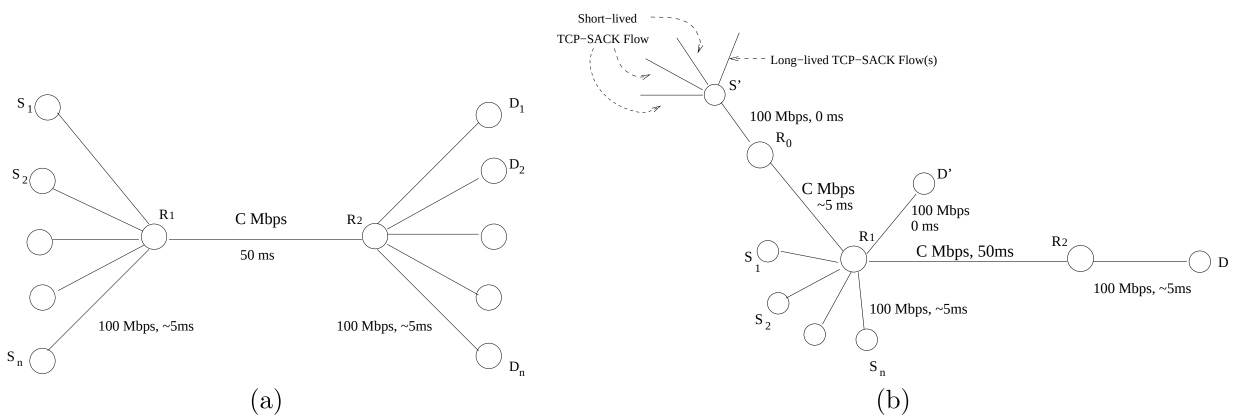


Figure 3: Topology for the simulations. Figure (a) shows the setting for simulations where a CBR source starting on S_1 was used to control the available bandwidth on the link $R_1 - R_2$. For the simulations involving realistic traffic patterns, the setting shown in Figure (b) was used.

From the way the bandwidth variations were chosen, it is not hard to see that the lowest window size to ever undergo an increment is about 12-15 (assuming a round-trip time of 120s. See Section 3.3 below). Similarly the lowest window size to ever undergo a decrement is about 25-20. Applying the above four conditions, we get the following permissible values for the parameters, approximately: AI ≤ 3, AD ≤ 3, MI > 0.1 and MD < 0.9. The results of the comparison between the various candidate alogithms in each of the four classes – AIMD, AIAD, MIMD, MIAD – are shown in the appendix. Based on these results, we choose the following four candidate linear control schemes for comparison: AIMD(1,0.8), AIAD(1,3), MIMD(1.125, 0.8) and MIAD(1.125, 3). Henceforth, we shall refer to these schemes as AIMD, AIAD, MIMD and MIAD, respectively.

3.3 Simulation Set-up

We use simulations in NS-2 to study the above congestion control schemes under the various combinations of loss recovery and router algorithms and against the environments described above. In each simulation we have n identical TCP test flows using the particular linear congestion control scheme under investigation, and we subject them to different variations in the available bandwidth.

The topology used for testing with variations 1 through 10 is shown in Figure 3(a). To implement variations in the available bandwidth in these scenarios, we choose to keep the bandwidth of the link constant and introduce CBR-like cross-traffic to consume varying amounts of bandwidth. If the link bandwidth is B and the cross-traffic consumes B_c then we say that the available bandwidth is $B_a = B - B_c$. The descriptions above of the bandwidth variation scenarios can be turned into recipes for how the cross-traffic rate should be varied. When testing with Drop-Tail and RED router mechanisms (at R_1), we employ a single rate controlled CBR source (between end-points S_1 and D_1) to realize the bandwidth variations [4]. When testing with DRR schedulers, however, we use a time-varying number of fixed rate CBR sources (between S_1 and D_1). This is because, if we have n simultaneous TCP flows being tested, a single CBR source would be limited to at most $\frac{1}{n+1}$th of the available bandwidth at any instant of time when DRR is used, and so the available bandwidth would not vary as desired. Hence, we vary the number of CBR flows to accurately implement the variation in available bandwidth. A simple calculation shows that to achieve an available bandwidth of B_a we need $n(C - B_a)/B_a$ CBR flows, where C is the capacity of the link $R_1 - R_2$ and n is the number of test TCP flows. The test TCP flows are between S_i and D_i. Also, we randomize the round-trip times slightly to avoid synchronization effects.

[4]The CBR source might incur a few losses and hence the available bandwidth might be slightly larger than expected.

While the topology shown in Figure 3(a) is well suited for tests in which we control available bandwidth directly, it is not amenable to the implementation of bandwidth variations 11 through 13. We use the topology shown in Figure 3(b) to implement variations 11 through 13. In what follows, we first describe the set-up in detail and then explain the reasons for the difference from that of Figure 3(a).

TCP-SACK flows implementing AIMD with pareto-distributed lengths and poisson inter-arrival times run between nodes S' and D. These constitute the cross traffic on the link R_1-R_2. The n test TCP flows are between nodes S_i $(i = 1, \ldots, n)$ and D. In addition, we have n "place-holder" long-lived flows between nodes S' and D'. The router R_0 employs DRR scheduling. The router at R_1 implements either DRR or RIO (explained in greater detail below). The TCP-SACK cross traffic is given priority over the test traffic at router R_1. In addition, the bandwidth between R_0 and R_1 equals that of link R_1-R_2.

While simulating the environments 11 through 13, we would like to ensure that the TCP flows constituting the cross-traffic on link R_1-R_2 are allocated their fair-share of the R_1-R_2 capacity irrespective of the congestion control algorithm employed by the test flows. This is ensured by using DRR at router R_0 and using n long-lived place-holder flows between S' and D'. The place-holder flows emulate the n test-flows so that when the pareto-distributed TCP flows enter link R_1-R_2, their aggregate occupies no more than the fair-share. However, we also want the TCP flows constituting the cross-traffic to not incur any more losses beyond link R_0-R_1 since they already are at their fair-share upon exiting this link. This is ensured by: (i) using RIO at router R_1 and marking the packets belonging to the cross traffic as high priority and (ii) using TCP-SACK for the cross traffic. When testing with the setting of Drop-tail buffers, we modify the parameters for the low priority packets at router R_0 to implement Drop-tail behavior on packets belonging to the test flows. When testing with the DRR setting, we use a DRR scheduler, instead of RIO, at router R_1. Notice that the flows belonging to the cross traffic do not incur any additional losses when DRR is employed at router R_1.

4 Results

As we discussed in the Introduction, we use four metrics in evaluating the performance of the different congestion control schemes:

- *goodput*: We measure goodput as the fraction of available bandwidth used to transmit unique packets. The goodput values all lie in $[0, 1]$.
- *delay*: The queueing delay is measured in milliseconds.
- *loss rate*: The loss rate is measured in terms of the percentage of packets lost (so a score of 5 indicates a 5% packet loss).
- *fairness*: The fairness metric is defined as $\frac{g_{max} - g_{min}}{g_{avg}}$ where g_{max}, g_{min} and g_{avg} are the maximum, minimum and average goodputs of the test flows respectively.[5]

We present the results by first describing how the rankings change when going from the TCP-Reno with its severe loss penalty to more gentle loss penalties. We then discuss the impact of different router drop policies and queueing behavior. While presenting the results, we show both C_a and D_a for goodput. This is because these two aggregate scores show rather different behavior. We only present the values of D_a for fairness, loss, and delay because the ordering of the D_a values for these quantities is very similar to the ordering of the C_a values. For delay and loss, we also show the raw values as the absolute magnitude of delays and losses cannot be easily inferred from the value of D_a.

In all our simulations we have 10 test flows. We have run our simulations with different number of test flows and the results are qualitatively similar. For reasons of space, we do not show the results for higher numbers of flows here.

[5]We have also employed the Chiu-Jain Fairness Index [3] for comparison, and the results are qualitatively identical.

4.1 The Impact of Loss Recovery Algorithms

As mentioned in Section 1, TCP Reno incurs a *severe* penalty when recovering from losses; TCP SACK is more adept at handling losses and therefore incurs a much more *gentle* penalty. We performed simulations of the four congestion control algorithms with these two types of loss recovery (TCP Reno and TCP SACK). We simulated these schemes on the various scenarios; in these tests the routers used FIFO, drop-tail routers with buffers sized to match the delay-bandwidth product of the network.

4.1.1 TCP Reno Loss Recovery

The results for TCP Reno loss recovery are shown in figure 4. Here, and in subsequent tables, we mark the algorithms with the best goodput (best values for both C_a and D_a) by underlining them. From the results, AIMD and AIAD deliver roughly similar goodput in all the test environments. On the other hand, the multiplicative increase (MI) algorithms perform poorly, as indicated by the large values of D_a and C_a, because they often significantly overestimate the available bandwidth and must invoke TCP Reno's expensive loss recovery routines.

TCP Reno DROPTAIL	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.15	0.06	8.12	332	0.11
AIAD	0.02	0.01	15.92	440	3.26
MIMD	3.34	0.54	0.32	21	15.46
MIAD	3.32	0.52	0.56	47	19.72

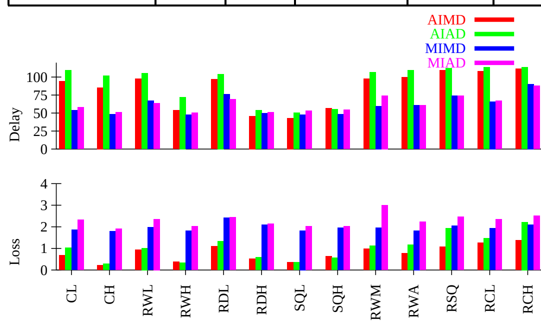


Figure 4: FIFO Drop-Tail buffers with TCP Reno loss recovery. The table compares the different algorithms on the basis of the four metrics. The algorithms achieving the best goodput performance are underlined. The figure on the bottom shows the raw delays and loss rates. In each column in the figure, the algorithms are presented in the order AIMD, AIAD, MIMD, MIAD.

AIMD provides better fairness than AIAD, although AIMD is not perfectly fair. In addition, AIMD suffers the fewest losses, with MIAD suffering the most. AIMD and AIAD have the highest delay values; this is because the MI algorithms are frequently timed-out, leaving the queue empty.

To summarize, *with TCP Reno loss recovery and FIFO drop-tail routers, AIMD and AIAD provide the best goodput performance. However, AIAD is not as fair.*

4.1.2 TCP SACK Loss Recovery

Figure 5 shows the results for TCP SACK loss recovery, again with FIFO drop-tail routers. The absolute values (which we don't show in our tables) of the goodputs are significantly higher than with TCP Reno. As expected, the loss and delay values are higher too. In comparative terms, MIMD achieves high goodput, and in fact, is marginally better than AIMD and AIAD. However, in terms of delay and loss rate, MIMD provides the worst performance.

Also, AIMD remains the only algorithm to achieve reasonable levels of fairness. In fact, the gentle loss recovery of TCP SACK makes the fairness properties of the non-AIMD algorithms worse. This is because without the timeout-induced restarts invoked by TCP Reno, flows that get more than their fair share can continue to exploit their advantage for longer periods of time.

TCP SACK *DROPTAIL*	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.25	0.12	0.67	23	0.00
AIAD	0.26	0.12	19.27	99	13.04
MIMD	0.12	0.03	9.72	115	40.70
MIAD	0.65	0.16	24.00	145	60.87

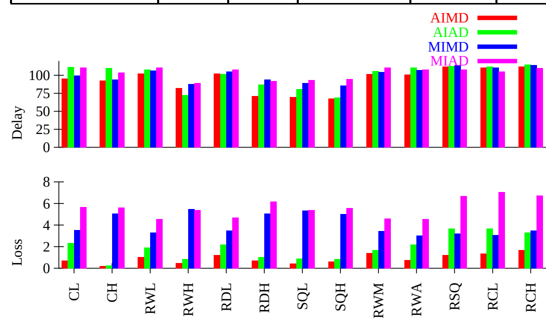


Figure 5: FIFO Drop-Tail buffers with TCP SACK loss recovery.

To summarize, *TCP SACK loss recovery reduces some of the distinguishing factors between the different schemes. All schemes, except MIAD, provide roughly comparable goodput performance. Moreover, AIMD provides the highest levels of fairness.*

4.1.3 An Aside: The Impact of Multiple Congested Bottleneck Links

The persistent high loss rate of AIAD might raise concerns about the validity of its goodput results in scenarios with multiple congested bottleneck links. It is possible that a high loss-rate at a congested down-stream link might affect the goodput of competing flows at up-stream links. To check if this is indeed the case, we perform a few simulations in which the test flows traverse multiple, congested, distinct bottleneck links.[6] We do not show details of these simulations here.

We observe that even in the situations with multiple congested bottlenecks, AIAD continues to have identical goodput as AIMD. This is mainly a consequence of AIAD's ability to keep more packets outstanding than AIMD at any instant. This ability more than offsets the negative impact of AIAD's higher loss rate on its goodput and helps AIAD utilize available bandwidth more effectively than AIMD under most situations.

4.2 The Impact of Router Queuing Behavior (and ECN)

A variety of router configuration settings affect the behavior of end-to-end congestion control schemes. Some of the important factors here include the drop policy (drop-tail or AQM), early congestion notification (ECN), fair scheduling (DRR) and buffer sizing. We consider each of these in turn.

4.2.1 Effect of Active Queue Management

A router employing a drop-tail policy does not help senders gauge incipient congestion. The only indication of congestion, an actual buffer overflow, is drastic and frequently results in burst losses and long queues at the routers. RED active queue management attempts to provide an earlier and more gradual indication of congestion to network endpoints.

The objective of the RED style of feedback is to reduce loss rates and delays by managing buffer occupancy at the router more actively. Thus, when the end-hosts use TCP Reno, having the routers employ RED active queue management (see Figure 6) does reduce loss rates and delays significantly across all the schemes. However, RED does not greatly alter the relative goodputs of the four algorithms, with AIMD and AIAD still achieving the best goodput results, just as with FIFO drop-tail buffers and TCP Reno end-points.

[6] We simulated a circular topology in which each test flow traverses two bottleneck links. Each of these links is in turn shared with different competing test flows.

TCP Reno RED	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.06	0.02	1.07	102	0.00
AIAD	0.08	0.04	5.22	115	3.94
MIMD	0.75	0.19	0.82	45	6.08
MIAD	1.40	0.30	2.44	55	8.22

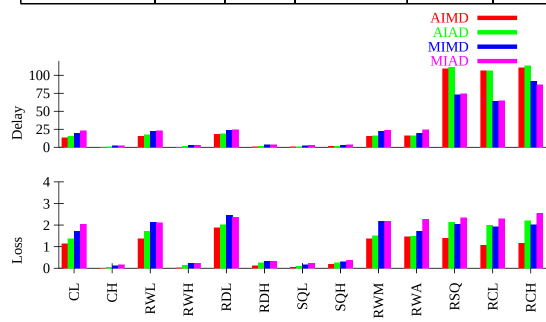


Figure 6: RED gateways with TCP Reno loss recovery.

RED significantly improves the fairness of all schemes (AIAD is the worst in this regard, but the difference is relatively small).[7] This increased fairness with RED can be explained as follows: With FIFO drop-tail routers (and also in the model considered by Chiu and Jain [3]), packet drops are synchronous and deterministic resulting in all flows experiencing similar loss epochs. RED, on the other hand, randomizes losses across flows and across time. Thus, RED effectively decouples the loss epochs of the flows. This helps punish flows with a greater number of packets outstanding at a rate higher than those with fewer packets outstanding. Hence, RED can achieve long-term fairness.

In summary, *with RED routers and TCP Reno loss recovery, AIMD and AIAD provide the best goodput, and they are both reasonably fair.*

TCP SACK RED	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.41	0.13	1.81	11	0.00
AIAD	0.25	0.10	9.85	39	6.28
MIMD	0.19	0.07	1.41	65	9.97
MIAD	1.64	0.41	16.37	71	22.26

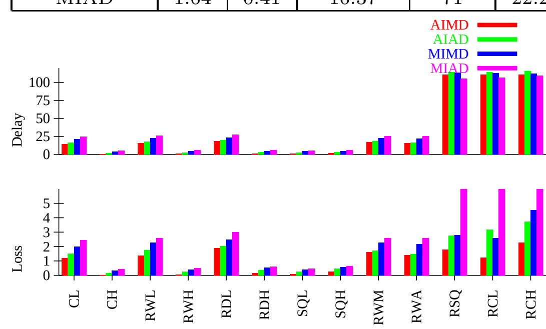


Figure 7: RED gateways with TCP SACK loss recovery.

When end-hosts use TCP SACK (Figure 7), RED routers cause a more significant shift in performance. As expected, the loss rates, delays and fairness are worse than with TCP Reno for all algorithms. In terms of goodput, MIMD provides the best performance with AIAD being not too far behind. The reason for MIMD and AIAD doing significantly better than AIMD when compared to FIFO drop-tail buffers is because RED is much more tolerant to bursty traffic patterns. Thus while TCP SACK times-out occasionally with FIFO drop-tail buffers when aggressive congestion control schemes like MIMD are employed (due to large bursts

[7] We have checked that this increased fairness persists when one looks at scenarios when the flows have differing RTTs; in such a case, the algorithms give roughly the same RTT-biased allocations as AIMD.

of packet losses), such time-outs are rare in RED. MIAD, however is much more aggressive and times-out more often, thus providing poor performance.

Thus, *with RED routers and TCP SACK loss recovery MIMD and AIAD achieve the highest goodput. MIMD is reasonably fair while AIAD is slightly unfair. AIMD continues to provide the lowest loss and delay and the highest fairness.*

4.2.2 Effect of Early Congestion Indications

RED routers can provide Explicit Congestion Notification (ECN) by marking a bit in the headers of forwarded packets to indicate incipient congestion. This marking is a more gentle form of feedback to end-systems since packets are not lost to providing congestion indications. Consequently, as the simulations below confirm, with both Reno and SACK loss recovery, the loss rates are significantly reduced (compared to RED and drop-tail) and delays are slightly increased. However, just as TCP SACK loss recovery exacerbates the fairness issues with FIFO drop-tail routers, added gentleness due to ECN helps aggressive flows hold onto additional bandwidth without incurring much penalty. This also results in slightly worse fairness with ECN, under either form of loss recovery.

TCP Reno ECN	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.37	0.18	1.33	69	0.38
AIAD	0.30	0.16	3.48	154	0.61
MIMD	0.44	0.10	1.26	83	3.65
MIAD	0.37	0.13	13.49	55	5.23

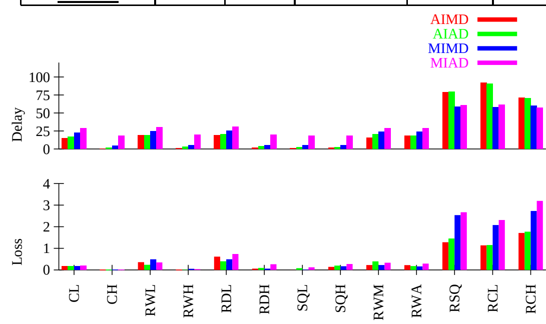


Figure 8: ECN with TCP Reno loss recovery.

When hosts employ TCP Reno loss recovery, using ECN (Figure 8) produces a dramatic change with all algorithms providing identical goodput performance. All algorithms provide identical performance in terms of goodput and loss rate. MIAD provide poor fairness, while AIAD provide somewhat poor performance in terms of delay.

Therefore, *with TCP-Reno loss recovery and routers employing ECN, all algorithms provide near-identical performance. Morover, MIAD is the only unfair algorithm.*

With TCP SACK loss recovery (Figure 9), the same general conclusions as above, hold along with two key side effects. Firstly, MIAD and MIMD's superiority in goodput increases considerably. Secondly, AIAD also performs as well, and is somewhat better than AIMD in terms of goodput when compared to the TCP-Reno case presented above.

In summary, *with ECN and TCP-SACK end-points, MIAD and MIMD have a moderate goodput advantage. AIAD also provides good performance (comparable to MIAD and MIMD) in terms of goodput. However, AIMD, being the least aggressive, shows relatively poor goodput performance.*

4.2.3 Effect of Smaller Buffers

In our previous experiments, we configured our routers to have queue sizes roughly equal to the delay-bandwidth product of the congested link. This rule-of-thumb has been adopted in the Internet to enable TCP's AIMD (with the default setting of (1,0.5) to fully utilize the network capacity. We performed simulations with reduced router buffer size (only 10% of the delay-bandwidth product) to see which congestion

TCP SACK ECN	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.41	0.18	1.13	21	0.41
AIAD	0.28	0.15	2.45	152	3.80
MIMD	0.13	0.05	7.49	51	7.81
MIAD	0.05	0.02	20.65	78	15.25

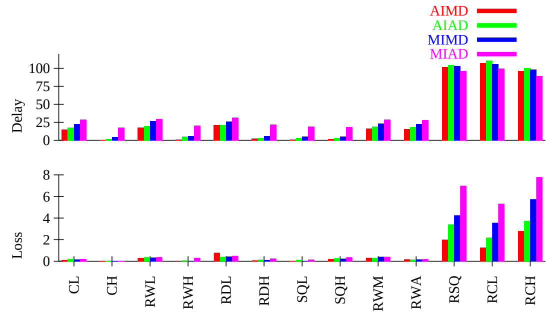


Figure 9: ECN with TCP SACK loss recovery.

control schemes can best cope with small buffers. Due to limited space, we do not include the detailed results here but merely summarize them. The fairness properties are largely unchanged by the small buffers. The goodput and loss rates of the MI algorithms, with either form of loss recovery, deteriorate significantly with reduced buffers due to excessive overshooting. With drop-tail routers, AIAD achieves better goodput than AIMD, largely because AIMD's relatively conservative back-off upon drops occasionally leaves the queue empty. Small buffers have little impact on the results using RED routers since the RED system is designed to ensure low average queue occupancy. In essence, *the use of smaller drop-tail router buffers makes more gradually adapting schemes such as AIAD look better. With RED, however, the performance ordering does not change much with smaller buffers.*

4.2.4 Effect of DRR

In our evaluation, we consider fairness as an important metric of a congestion control algorithm's performance. However, an alternative approach to providing fairness is to rely on router mechanisms such as DRR [15]. The question we seek to answer here is: which congestion control algorithms perform well when routers provide fairness explicitly?

TCP Reno DRR	*Goodput*		*Fairness*	*Delay*	*Loss*
	D_a	C_a	D_a	D_a	D_a
AIMD	0.05	0.02	2.35	168	0.02
AIAD	0.10	0.08	4.20	135	2.18
MIMD	1.37	0.23	1.34	71	14.65
MIAD	1.87	0.28	2.51	78	24.74

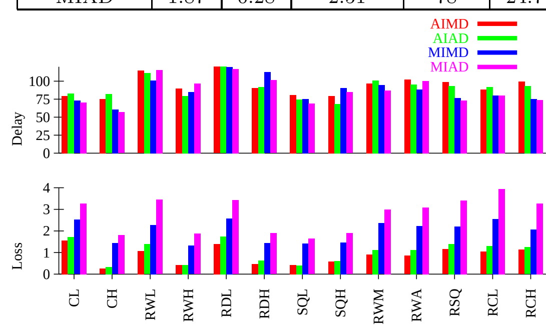


Figure 10: DRR with TCP Reno loss recovery.

In our simulations, we use a DRR scheduler at the bottleneck router to provide instantaneous per-flow max-min fairness. Figures 10 and 11 show the results with Reno and SACK loss recovery respectively.

TCP SACK *DRR*	*Goodput* D_a	*Goodput* C_a	*Fairness* D_a	*Delay* D_a	*Loss* D_a
AIMD	0.01	0.00	0.69	128	0.00
AIAD	0.15	0.07	2.28	189	5.33
MIMD	0.77	0.21	5.15	82	36.25
MIAD	1.41	0.29	4.16	44	40.67

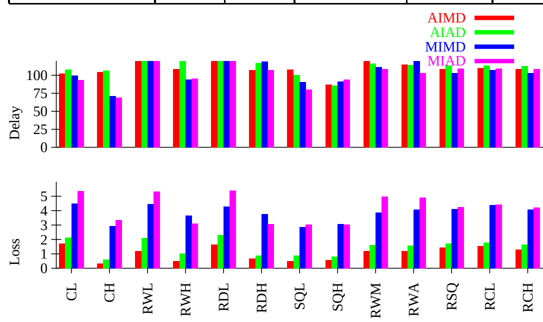


Figure 11: DRR with TCP SACK loss recovery.

Firstly, all the schemes are equally fair. Also, under either form of loss recovery, the less aggressive AI algorithms provide significantly better goodput performance than the MI algorithms. In addition, the AI algorithms are similar in terms of delay and loss rate.

Thus, *when routers provide fairness explicitly, AI algorithms provide significantly better goodput performance than the MI algorithms. All algorithms are equally fair.*

4.3 Discussion

If the goal of congestion control is to maximize fairness and minimize loss and delay while still achieving reasonable levels of goodput, then AIMD is the clear choice no matter what form of loss recovery or router queueing discipline is employed. In almost every setting AIMD achieved low delays and loss rates and high levels of fairness.

On the other hand, if the goal of congestion control is to maximize goodput while still achieving reasonable levels of fairness, loss rates and delay, then the situation is considerably different. Figure 12 may be helpful in understanding how the various algorithms compare under this goal. For each combination of the loss recovery algorithm and router algorithm, the table depicts the congestion control algorithms that achieve the highest goodput. Those algorithms that do not achieve reasonably fair performance are circled.

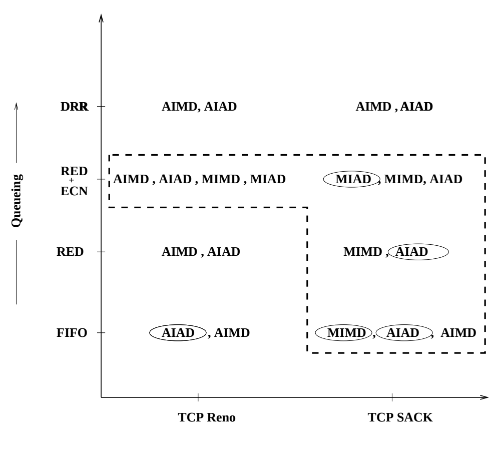


Figure 12: A plot showing the algorithms that achieved the highest goodput in each of the settings considered in the previous section. The encircled algorithms are unfair.

When TCP flows incur a large penalty for losses and routers employ packet drops to indicate congestion or when routers ensure perfect isolation between flows (scenarios outside the dotted box in Figure 12), the congestion control schemes of AIMD and AIAD are clearly superior. These schemes have a conservative increase and as a result a sending rate that is less bursty than MIMD and MIAD. In particular, *within the traditional setting of TCP Reno loss recovery and FIFO drop-tail routers, AIMD achieves the highest goodput.*

However, when the penalty due to losses is minimal either due to flows employing more tolerant loss recovery schemes or due to routers either using marked packets to indicate congestion (scenarios inside the dotted box in Figure 12), aggressive congestion control algorithms stand to gain in terms of goodput. In particular, AIMD being the most conservative, both in its increase and in its decrease, sometimes provides inferior performance in these settings. One the other hand, schemes with an aggressive increase (MIMD), or an aggressive decrease (AIAD) or both (MIAD) provide significantly better goodput in this situation.

A key observation that stands out from the above evaluation is the fact that AIAD is among the leading goodput performers in *all* the scenarios we have considered. Moreover, AIAD achieves reasonable levels of fairness as long as the routers are not FIFO drop-tail. This suggests that as we deploy more of either the modern loss recovery mechanisms or the router queue management schemes, AIAD is definitely a viable choice for congestion control. In fact, if we could alleviate the fairness issues of AIAD in the FIFO drop-tail case, it would be the best overall choice in terms of efficiency and fair bandwidth allocation.

So far we have only considered the four *pure* linear schemes. To address the issue of fairness we now consider *hybrid* schemes.

5 Hybrid Congestion Control Algorithms

In this section, we try to address two different issues. First, we ask how we might solve the fairness problems of AIAD with drop-tail FIFO routers. Second, we ask what are the advantages or disadvantages of *hybrid* linear congestion control algorithms; these are algorithms in which the linear increase and decrease need not be purely additive or purely multiplicative. It turns out that hybrid algorithms are solutions to the AIAD fairness problem, so we address both issues at once. We start by revisiting the Chiu-Jain [3] analysis and deriving necessary conditions for fairness.

5.1 Hybrid Algorithms and Fairness

Linear congestion control algorithms are governed by the following update equations:

$$w(t+1) = \begin{cases} a_I + b_I w(t) & \text{upon success} \\ a_D + b_D w(t) & \text{upon loss} \end{cases}$$

In what follows, we consider a system with synchronous congestion signals and static bandwidth as [3] does. We argue that, unless $b_I, b_D = 1$ (which is true for $AIAD$) or $a_I, a_D = 0$ (which is true for $MIMD$), most combinations of the four parameters yield a fair congestion control algorithm.

We assume the system reaches a steady state with a nonzero window size, in which the updates to a flow's window would follow a periodic cycle of increases and decreases (it is easy to see from the analysis below that this excludes MIAD). At the end of each such cycle, the size of the window is identical to that at the start. For simplicity, we assume that this sequence consists of k_I increases followed by k_D decreases (this sequence repeats itself indefinitely in steady state) as shown in figure 5.1. We would like to stress that our argument applies to more general steady states.

$$\xleftrightarrow{k_D}\ \xleftrightarrow{k_I}\ \xleftrightarrow{k_D}\ \xleftrightarrow{k_I}$$

. . I D D D . . . . DI I I . . . . I D D D . . . . DI I I . . . . I D . .

Figure 13: A window update sequence in steady state.

Using the fact that after sequence of k linear increases, a window of size w_0 becomes $a_I\left(\sum_{j=0}^{k-1} b_I^j\right) + b_I^k w_0$ and that after a sequence of k linear decreases it becomes $a_D\left(\sum_{j=0}^{k-1} b_D^j\right) + b_D^k w_0$, we obtain the following equation for the window in steady state:

$$\begin{aligned} w &= a_D\left(\sum_{j=0}^{k_D-1} b_D^j\right) + b_D^{k_D}\left[a_I\left(\sum_{j=0}^{k_I-1} b_I^j\right) + b_I^{k_I} w\right] \\ &= a_D\left(\sum_{j=0}^{k_D-1} b_D^j\right) + a_I b_D^{k_D}\left(\sum_{j=0}^{k_I-1} b_I^j\right) + b_D^{k_D} b_I^{k_I} w \\ &= \alpha + \beta w \end{aligned}$$

where , $\alpha = a_D\left(\sum_{j=0}^{k_D-1} b_D^j\right) + a_I b_D^{k_D}\left(\sum_{j=0}^{k_I-1} b_I^j\right)$ and $\beta = b_D^{k_D} b_I^{k_I}$.

If the algorithm were completely multiplicative ($a_I, a_D = 0$), then $\alpha = 0$, implying that $\beta w = w$; since we assume $w \neq 0$ this means that $\beta = 1$ and so any value of w is allowable. This means that not all flows need to have the same window size. On the other hand, if the algorithm were completely additive ($b_I, b_D = 1$), then $\beta = 1$ and $\alpha = 0$, again allowing any value of w.

For the other cases, unless the parameter values were precisely tuned, we have $\beta \neq 1$ and $\alpha \neq 0$ and so there is a *single* steady-state value of w to which all flows would converge. What this means is that we can use hybrid algorithms to achieve fairness. We present two such algorithms in the next section.[8]

5.2 Two Hybrid Algorithms

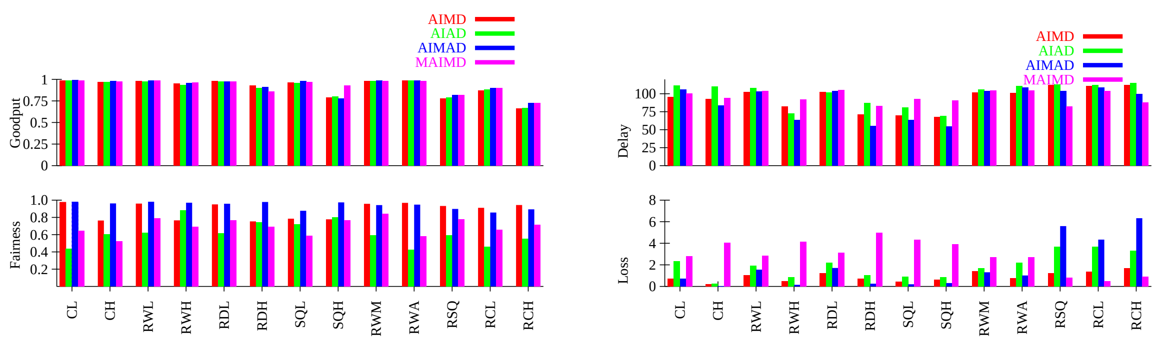


Figure 14: Figure showing the performance of hybrid algorithms. In each column, the algorithms are presented in the order AIMD, AIAD, AIMAD and MAIMD. We choose the setting with TCP SACK loss recovery and FIFO drop-tail routers, because AIAD had the worst fairness in this scenario.

In the Introduction, we mentioned that there were very few papers that proposed linear control schemes that were different from AIMD. One of the notable exceptions is [7], which argues against purely linear increase and proposes using a hybrid increase with both multiplicative and additive terms. In [7], the particular parameter values are $a_I = 1$, $b_I = 1.1$ and $b_D = 0.5$ (In the evaluation below, we use $b_D = 0.8$ since we found this to be a marginally better choice than $b_D = 0.5$). We call this the MAIMD linear control scheme, and note that it is a slight perturbation of the standard AIMD$(1, 0.5)$ with a small multiplicative component.

[8] Chiu and Jain [3] establish the following conditions for convergence to fairness:

(I) $a_D = 0$ and $0 \leq b_D < 1$

(II) $a_I > 0$, $b_I \geq 1$

Though these conditions are sufficient, they are *not* necessary. Hence, contrary to their conclusions, our analysis allows an additive component in decrease.

On the other hand, our previous results suggest that AIAD is, apart from fairness issues, a desirable candidate for linear congestion control. We propose an AIMAD control algorithm which adds only a bit of multiplicative decrease: $a_I = 1$, $a_D = -1$ and $b_D = 0.9$ (Notice that the multiplicative component allows for an aggressive additive decrease). We now examine the salient features as well as the drawbacks of these schemes. As an example, we present the results for TCP SACK loss recovery with FIFO drop-tail routers in Figure 14.

AIMAD achieves significantly better fairness than AIAD in most of the environments with FIFO drop-tail routers and TCP SACK loss recovery. This improvement in fairness does not come at the cost of worse goodput. In addition, AIMAD, as expected, has lower loss rates and lower delays than AIAD. Note also that AIMAD, provides fairness comparative to AIMD. In addition, AIMAD's delay and loss rates are not higher than those of AIMD.

Similarly, MAIMD (with FIFO drop-tail routers and TCP SACK loss recovery) improves the worst-case performance of AIMD and has reasonable goodput when compared with AIMD in all the environments. However, due to its aggressive increase, the loss rate and fairness are not as good as those of AIMD. Specifically, the fairness of MAIMD is slightly worse in environments where the raw link capacity is very high.[9] Finally, the delays of AIMD and MAIMD are similar.

In general, we observe that AIMAD performs at least as well as AIAD with respect to all the four metrics in *every* situation discussed in this paper. On the other hand, MAIMD's goodput performance is comparable to AIMD's *only* when used with TCP SACK loss recovery. This is true across all the router configurations that we consider. Due to space constraints, we do not present the detailed results here.

6 Related Work

In the past, there have been few research studies exploring linear alternatives to TCP's congestion control algorithms. Of these, two separate studies that bear similarity to our work are [8] and [9].

In [8], the authors present a study of the tracking abilities of various congestion control algorithms in networks that provide fairness explicitly. In this study, the increase component of the congestion control algorithms can be additive, multiplicative or non-linear. The decrease component is chosen to be multiplicative. The paper shows via analysis that in such a fair network, MIMD is more responsive to congestion notifications than the other schemes, including AIMD. Thus [8] concludes that MIMD can track changes in bandwidth more effectively. This work differs from ours in two key aspects: firstly, additive decrease schemes are outside the purview of the analysis in [8]; secondly, the impact of loss recovery is not factored into the analysis. As a result, the conclusions in [8] are different, qualitatively, from our observations about the impact of fair-queueing.[10] (presented in Section 4.2.4): We have shown that while under Reno-style loss recovery AI schemes are clearly dominant, under SACK-style loss recovery all algorithms except MIAD provide identical goodput performance.

In contrast, [9] employs simulations to study the relative performance of AIMD and AILD, where, AILD has the same linear increase as AIMD, but the decrease is defined by the following equation: $w_{t+1} = w_t - \beta f$, where β is a constant and f is the loss ratio. Also, the available bandwidth is kept constant. The authors observe that in networks that use RED-like gateways, AILD is both efficient and fair and outperforms AIMD. However, the definition of linear decrease adopted in this study does not produce a linear control scheme (in the sense we've defined in our paper) since the drop rate f is a nonlinear function of the aggregate load.

To the best of our knowledge, our study is the first to compare all linear congestion control schemes under a wide variety of router configurations, different loss recovery schemes and a wide range of variations in available bandwidth.

There have been other studies on congestion control algorithms which propose slowly adaptive alternatives to congestion control that are TCP-friendly and provide identical throughput as the current TCP under a given steady state loss rate. For example, [4] proposes non-linear slowly-adaptive window adjustment

[9]When the alternate router configurations are considered, MAIMD's fairness is very similar to that of AIMD under either form of loss recovery.

[10]Notice, from the appendix, that MIMD(1.125, 0.5) provides performance similar to AIMD and AIAD when DRR and TCP-SACK are employed. Thus our results do not negate those presented in [8]. Our results only show that MIMD is not the absolute best choice.

algorithms and [6] proposes rate-based schemes for congestion control. The issues pertaining to the dynamic behavior of such schemes have been partially addressed in [19]. Though the work presented in our paper does not consider such non-linear algorithms and rate-based schemes, we hope it provides sufficient intuition as to how these schemes should be evaluated in the long run.

7 Summary

This paper was an attempt to revisit the original design decision to focus exclusively on AIMD linear congestion control. We examined the impact of modern developments in loss recovery and in router algorithms on the choice of the linear congestion control scheme. We tested the four basic linear congestion control algorithms in a wide variety of settings.

We affirm that in the traditional context of TCP Reno loss recovery and FIFO drop-tail routers, AIMD is clearly an aptly made choice. However the same cannot be said when we include these more modern developments. AIMD is no longer a compelling choice for congestion control with the other congestion control algorithms providing better performance than AIMD by varying degrees. In particular, we have shown that *AIAD is a reasonable alternative choice for a modern congestion control scheme*. In fact, AIAD also provides reasonable fairness as long as routers do not employ FIFO drop-tail queueing. Adding a small multiplicative component to the additive decrease of AIAD is enough to ensure that fairness is guaranteed, even when FIFO drop-tail buffers are employed, without compromising goodput.

References

[1] K. K. Ramakrishnan and Raj Jain, "A binary feeback scheme for congestion avoidance in computer networks," *ACM Transactions on Computer Systems*, vol. 8, no. 2, pp. 158–181, May 1990.

[2] Van Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988, Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[3] D. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, June 1989.

[4] Deepak Bansal and Hari Balakrishnan, "TCP-friendly congestion control for real-time streaming applications," Technical Report MIT-LCS-TR-806, MIT, Cambridge, Massachusetts, May 2000.

[5] R. J. Gibbens and F. P. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, pp. 1969–1985, 1999.

[6] M. Handley, J. Padhye, S. Floyd, and J. Widmer, "TCP friendly rate control (TFRC):protocol specification," Internet Draft, Internet Engineering Task Force, July 2001, Work in progress.

[7] S. Gorinsky and H. Vin, "Additive increase appears inferior," Tech. Rep. TR2000-18, Department of Computer Sciences, The University of Texas at Austin, May 2000.

[8] S. Gorinsky and H. Vin, "Analysis of binary adjustment policies in fair heterogeneous networks," Tech. Rep. TR2000-32, Department of Computer Sciences, The University of Texas at Austin, November 2000.

[9] Narayanan Venkitaraman, Tae eun Kim, Kang-Won Lee, Songwu Lu, and Vaduvur Bhargavan, "Design and evaluation of congestion control algorithms in the future internet," *Poster at ACM SIGMETRICS*, 1999.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," Request for Comments 2018, Internet Engineering Task Force, Oct. 1996.

[11] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.

[12] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[13] Sally Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, Oct. 1994.

[14] Alan Demers, Srinivasan Keshav, and Scott Shenker, "Analysis and simulation of a fair queueing algorithm," in *SIGCOMM Symposium on Communications Architectures and Protocols*, Austin, Texas, Sept. 1989, ACM, pp. 1–12, also in Computer Communications Review, 19 (4), Sept. 1989.

[15] M. Shreedhar and George Varghese, "Efficient fair queueing using deficit round robin," *ACM Computer Communication Review*, vol. 25, no. 4, pp. 231–242, Oct. 1995.

[16] "The network simulator - ns-2. http://www.isi.edu/nsnam/ns/," .

[17] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[18] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, "Combinatorial optimization in congestion control," in *Proceedings of the 41th Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, 12–14 Nov. 2000, pp. 66–74.

[19] Deepak Bansal, Hari Balakrishnan, Sally Floyd, and Scott Shenker, "Dynamic behavior of slowly-responsive congestion control algorithms," in *Proceedings of ACM SIGCOMM*, San Diego, California, 2001.

Appendix

TCP Reno + DROPTAIL											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.19	0.06	(1, 1)	0.10	0.03	(1.125, 0.5)	0.75	0.10	(1.125, 1)	0.27	0.08
(1, 0.65)	0.06	0.02	(1, 2)	0.05	0.02	(1.125, 0.65)	0.36	0.10	(1.125, 2)	0.15	0.05
(1, 0.8)	0.07	0.03	**(1, 3)**	0.06	0.03	**(1.125, 0.8)**	0.17	0.06	**(1.125, 3)**	0.09	0.03
(2, 0.5)	0.83	0.20	(2, 1)	0.56	0.19	(1.19, 0.5)	1.30	0.17	(1.19, 1)	0.68	0.11
(2, 0.65)	0.56	0.15	(2, 2)	0.62	0.21	(1.19, 0.65)	0.61	0.10	(1.19, 2)	0.51	0.09
(2, 0.8)	0.56	0.22	(2, 3)	0.59	0.22	(1.19, 0.8)	0.82	0.19	(1.19, 3)	0.58	0.11
(3, 0.5)	1.87	0.31	(3, 1)	1.11	0.25	(1.25, 0.5)	1.30	0.20	(1.25, 1)	1.30	0.25
(3, 0.65)	1.49	0.29	(3, 2)	1.15	0.25	(1.25, 0.65)	1.37	0.29	(1.25, 2)	1.33	0.37
(3, 0.8)	1.28	0.26	(3, 3)	1.17	0.27	(1.25, 0.8)	0.96	0.14	(1.25, 3)	1.31	0.31

Table 3: TCP Reno + DROPTAIL

TCP SACK + DROPTAIL											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.26	0.12	(1, 1)	0.50	0.11	(1.125, 0.5)	0.21	0.12	(1.125, 1)	1.56	0.34
(1, 0.65)	0.16	0.11	(1, 2)	0.47	0.11	(1.125, 0.65)	0.16	0.11	(1.125, 2)	0.76	0.26
(1, 0.8)	0.25	0.13	**(1, 3)**	0.52	0.12	**(1.125, 0.8)**	0.38	0.12	**(1.125, 3)**	0.96	0.26
(2, 0.5)	0.11	0.04	(2, 1)	0.41	0.11	(1.19, 0.5)	0.25	0.12	(1.19, 1)	2.03	0.58
(2, 0.65)	0.16	0.04	(2, 2)	0.44	0.11	(1.19, 0.65)	0.17	0.11	(1.19, 2)	1.19	0.30
(2, 0.8)	0.10	0.03	(2, 3)	0.39	0.11	(1.19, 0.8)	0.55	0.21	(1.19, 3)	1.29	0.32
(3, 0.5)	0.13	0.02	(3, 1)	0.31	0.11	(1.25, 0.5)	0.36	0.12	(1.25, 1)	2.07	0.57
(3, 0.65)	0.07	0.01	(3, 2)	0.15	0.11	(1.25, 0.65)	0.14	0.11	(1.25, 2)	1.17	0.35
(3, 0.8)	0.03	0.01	(3, 3)	0.35	0.11	(1.25, 0.8)	0.55	0.16	(1.25, 3)	0.91	0.28

Table 4: TCP SACK + DROPTAIL

TCP Reno + RED											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.60	0.15	(1, 1)	0.32	0.06	(1.125, 0.5)	0.21	0.06	(1.125, 1)	0.25	0.06
(1, 0.65)	0.59	0.15	(1, 2)	0.29	0.08	(1.125, 0.65)	0.13	0.05	(1.125, 2)	0.23	0.07
(1, 0.8)	0.38	0.12	**(1, 3)**	0.14	0.08	**(1.125, 0.8)**	0.06	0.04	**(1.125, 3)**	0.12	0.05
(2, 0.5)	0.50	0.09	(2, 1)	0.39	0.09	(1.19, 0.5)	0.56	0.08	(1.19, 1)	0.92	0.13
(2, 0.65)	0.26	0.06	(2, 2)	0.41	0.08	(1.19, 0.65)	0.44	0.09	(1.19, 2)	0.78	0.10
(2, 0.8)	0.29	0.07	(2, 3)	0.31	0.10	(1.19, 0.8)	0.63	0.14	(1.19, 3)	0.70	0.10
(3, 0.5)	0.47	0.06	(3, 1)	0.76	0.12	(1.25, 0.5)	0.93	0.13	(1.25, 1)	1.51	0.16
(3, 0.65)	0.31	0.04	(3, 2)	0.58	0.11	(1.25, 0.65)	0.91	0.16	(1.25, 2)	1.43	0.15
(3, 0.8)	0.27	0.06	(3, 3)	0.45	0.08	(1.25, 0.8)	0.94	0.16	(1.25, 3)	1.16	0.14

Table 5: TCP Reno + RED

TCP SACK + RED											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.68	0.16	(1, 1)	0.36	0.11	(1.125, 0.5)	0.52	0.09	(1.125, 1)	0.34	0.10
(1, 0.65)	0.47	0.14	(1, 2)	0.28	0.11	(1.125, 0.65)	0.20	0.04	(1.125, 2)	0.33	0.08
(1, 0.8)	0.32	0.11	**(1, 3)**	0.30	0.10	**(1.125, 0.8)**	0.01	0.01	**(1.125, 3)**	0.15	0.05
(2, 0.5)	0.42	0.09	(2, 1)	0.31	0.09	(1.19, 0.5)	0.58	0.11	(1.19, 1)	0.82	0.16
(2, 0.65)	0.30	0.08	(2, 2)	0.20	0.05	(1.19, 0.65)	0.26	0.05	(1.19, 2)	0.64	0.19
(2, 0.8)	0.06	0.03	(2, 3)	0.06	0.02	(1.19, 0.8)	0.09	0.28	(1.19, 3)	0.50	0.16
(3, 0.5)	0.36	0.07	(3, 1)	0.49	0.11	(1.25, 0.5)	0.11	0.61	(1.25, 1)	0.96	0.21
(3, 0.65)	0.26	0.07	(3, 2)	0.29	0.09	(1.25, 0.65)	0.33	0.08	(1.25, 2)	0.75	0.19
(3, 0.8)	0.01	0.01	(3, 3)	0.10	0.04	(1.25, 0.8)	0.66	0.16	(1.25, 3)	0.68	0.17

Table 6: TCP SACK + RED

Based on the guidelines presented in Section 3 for the choice of the parameters, we evaluate: AI $\in \{1, 2, 3\}$, AD $\in \{1, 2, 3\}$, MI $\in \{1.125, 0.19, 0.25\}$ and MD $\in \{0.5, 0.65, 0.8\}$. This results in 9 instantiations of each of the four linear control schemes. The instantiations in each set are compared with each other compared in scenarios with the different combinations of loss recovery schemes and buffering mechanisms that we discuss in this paper against the backdrop of bandwidth variations that we employed. We use the same method as that outlined in Section 3 to compare the 9 instantiations in each of the four classes.

Recall that when picking the best candidate, we do not seek to separately choose the potentially distinct best candidates in each setting. Instead, we look for a uniform choice of parameters that has reasonable performance across all settings. This way we can pick one instantiation that: (1) summarizes the overall performance of the family of linear control schemes well (2) ensure near-optimal performance in a variety settings.

Next, we present the full set of results for each of the eight scenarios discussed in this paper (drop-tail, RED, DRR and ECN router mechanisms with each of Reno and SACK-style loss recovery) in Tables 3-10. We present the values of C_a and D_a for the goodput of each instantiation since goodput is our primary metric of comparison. We do not show the results for fairness, delay or loss, as the algorithms are not significantly different in terms of these metrics. In each table, the underlying scenario is listed at the top (e.g., TCP Reno + DROPTAIL). Also, the candidates in each of the four classes are compared in the column bearing the corresponding heading.

It is not hard to see that the instantiation we pick for each scheme (shown in bold font, underlined) shows reasonably good performs across all the scenarios. For all the other candidate instantiations, there is at least one scenario resulting in very poor performance and others resulting in sub-optimal performance.

Thus our final choices are – AIMD(1, 0.8), AIAD(1, 3), MIMD(1.125, 0.8) and MIAD(1.125, 3).

TCP Reno + DRR											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.17	0.06	(1, 1)	0.20	0.05	(1.125, 0.5)	0.15	0.04	(1.125, 1)	0.20	0.07
(1, 0.65)	0.31	0.10	(1, 2)	0.15	0.06	(1.125, 0.65)	0.13	0.04	(1.125, 2)	0.22	0.08
(1, 0.8)	0.21	0.12	**(1, 3)**	0.12	0.08	**(1.125, 0.8)**	0.22	0.09	**(1.125, 3)**	0.12	0.03
(2, 0.5)	0.17	0.03	(2, 1)	0.30	0.07	(1.19, 0.5)	0.78	0.11	(1.19, 1)	0.1.04	0.15
(2, 0.65)	0.17	0.03	(2, 2)	0.24	0.05	(1.19, 0.65)	0.69	0.10	(1.19, 2)	0.84	0.11
(2, 0.8)	0.18	0.04	(2, 3)	0.17	0.04	(1.19, 0.8)	0.76	0.16	(1.19, 3)	0.92	0.14
(3, 0.5)	0.21	0.04	(3, 1)	0.33	0.06	(1.25, 0.5)	0.93	0.15	(1.25, 1)	1.54	0.20
(3, 0.65)	0.19	0.03	(3, 2)	0.26	0.04	(1.25, 0.65)	0.92	0.15	(1.25, 2)	1.42	0.18
(3, 0.8)	0.28	0.05	(3, 3)	0.23	0.04	(1.25, 0.8)	1.24	0.24	(1.25, 3)	1.34	0.16

Table 7: TCP Reno + DRR

TCP SACK + DRR											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.11	0.05	(1, 1)	0.07	0.02	(1.125, 0.5)	0.02	0.01	(1.125, 1)	0.14	0.03
(1, 0.65)	0.15	0.07	(1, 2)	0.07	0.03	(1.125, 0.65)	0.09	0.03	(1.125, 2)	0.20	0.06
(1, 0.8)	0.12	0.06	**(1, 3)**	0.06	0.03	**(1.125, 0.8)**	0.60	0.14	**(1.125, 3)**	0.08	0.04
(2, 0.5)	0.08	0.02	(2, 1)	0.14	0.03	(1.19, 0.5)	0.08	0.03	(1.19, 1)	0.58	0.11
(2, 0.65)	0.16	0.09	(2, 2)	0.07	0.02	(1.19, 0.65)	0.57	0.28	(1.19, 2)	0.51	0.14
(2, 0.8)	0.09	0.02	(2, 3)	0.11	0.04	(1.19, 0.8)	1.51	0.35	(1.19, 3)	0.49	0.14
(3, 0.5)	0.08	0.04	(3, 1)	0.14	0.04	(1.25, 0.5)	1.02	0.27	(1.25, 1)	0.90	0.17
(3, 0.65)	0.09	0.03	(3, 2)	0.14	0.04	(1.25, 0.65)	1.41	0.36	(1.25, 2)	0.82	0.20
(3, 0.8)	0.03	0.01	(3, 3)	0.09	0.03	(1.25, 0.8)	1.64	0.40	(1.25, 3)	0.90	0.18

Table 8: TCP SACK + DRR

TCP Reno + ECN											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.73	0.15	(1, 1)	0.43	0.12	(1.125, 0.5)	0.90	0.13	(1.125, 1)	0.81	0.15
(1, 0.65)	0.49	0.15	(1, 2)	0.35	0.12	(1.125, 0.65)	0.44	0.07	(1.125, 2)	0.18	0.06
(1, 0.8)	0.28	0.12	**(1, 3)**	0.44	0.15	**(1.125, 0.8)**	0.08	0.02	**(1.125, 3)**	0.06	0.02
(2, 0.5)	0.48	0.09	(2, 1)	0.37	0.06	(1.19, 0.5)	1.00	0.13	(1.19, 1)	1.65	0.27
(2, 0.65)	0.25	0.06	(2, 2)	0.25	0.05	(1.19, 0.65)	0.59	0.09	(1.19, 2)	1.03	0.22
(2, 0.8)	0.15	0.05	(2, 3)	0.09	0.05	(1.19, 0.8)	0.20	0.07	(1.19, 3)	0.62	0.10
(3, 0.5)	0.41	0.08	(3, 1)	0.81	0.16	(1.25, 0.5)	0.75	0.13	(1.25, 1)	1.93	0.35
(3, 0.65)	0.20	0.04	(3, 2)	0.28	0.07	(1.25, 0.65)	0.31	0.06	(1.25, 2)	1.87	0.30
(3, 0.8)	0.11	0.03	(3, 3)	0.16	0.04	(1.25, 0.8)	0.14	0.05	(1.25, 3)	1.43	0.28

Table 9: TCP Reno + ECN

TCP SACK + ECN											
AIMD			AIAD			MIMD			MIAD		
Alg	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a	*Alg*	D_a	C_a
(1, 0.5)	0.75	0.18	(1, 1)	0.41	0.12	(1.125, 0.5)	0.77	0.13	(1.125, 1)	0.14	0.06
(1, 0.65)	0.49	0.14	(1, 2)	0.28	0.13	(1.125, 0.65)	0.36	0.07	(1.125, 2)	0.11	0.04
(1, 0.8)	0.36	0.13	**(1, 3)**	0.39	0.13	**(1.125, 0.8)**	0.11	0.03	**(1.125, 3)**	0.07	0.02
(2, 0.5)	0.51	0.10	(2, 1)	0.15	0.06	(1.19, 0.5)	0.78	0.13	(1.19, 1)	0.33	0.09
(2, 0.65)	0.27	0.05	(2, 2)	0.11	0.04	(1.19, 0.65)	0.38	0.06	(1.19, 2)	0.07	0.01
(2, 0.8)	0.14	0.04	(2, 3)	0.15	0.04	(1.19, 0.8)	0.13	0.06	(1.19, 3)	0.08	0.01
(3, 0.5)	0.46	0.08	(3, 1)	0.05	0.02	(1.25, 0.5)	0.75	0.13	(1.25, 1)	0.32	0.04
(3, 0.65)	0.21	0.05	(3, 2)	0.10	0.04	(1.25, 0.65)	0.30	0.06	(1.25, 2)	0.24	0.06
(3, 0.8)	0.01	0.01	(3, 3)	0.17	0.11	(1.25, 0.8)	0.01	0.00	(1.25, 3)	0.05	0.01

Table 10: TCP SACK + ECN