Volume 2, Issue 2, February 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

The general comparison between AIMD and AIAD Congestion Control Algorithms

[1]P.Radha Krishna Reddy
M.Tech Student
JNTU-Anantapur,INDIA

[2]G.Sireesha
Assistant Professor
JNTU-Hyderabad,INDIA

[3] C.V.Chirangeevi Kumar
Assistant Professor
JNTU-Anantapur,INDIA

Abstract: ***Here we are going to see the general difference between two linear congestion control protocols {AIMD, AIAD} 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 good put performance in the more modern settings. Where AIMD fails, AIAD proves to be a reasonable alternative. From the early days of modern congestion control, ushered in by the development of TCP's and DEC bit's congestion control algorithm. There has been widespread agreement that linear additive-increase-multiplicative-decrease (AIMD) congestion 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 queuing behavior. For example, RED active queue management and Explicit Congestion Notification (ECN) can tolerate bursty flow behavior.***

Keywords: AIMD, MIMD, AIAD, congestion control, active queue management.

1. Introduction:

The first sophisticated transport congestion control algorithms, developed almost simultaneously for DEC bit [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 AIAD, 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 proposes either non-linear congestion control algorithms [4] or approaches that differ radically from TCP [5, 6]. Notable exceptions to this statement are [7] and [8, 9] which propose linear control algorithms other than AIMD.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 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.

Our simulations show that AIMD is the superior design choice in the traditional setting of TCP Reno loss recovery and FIFO drop-tail routers. AIMD is no longer superior. TCP SACK, active queue management techniques and fair queuing in routers enable the other linear alternatives to provide comparable and sometimes significantly better good put

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. AIAD and also provide good performance.

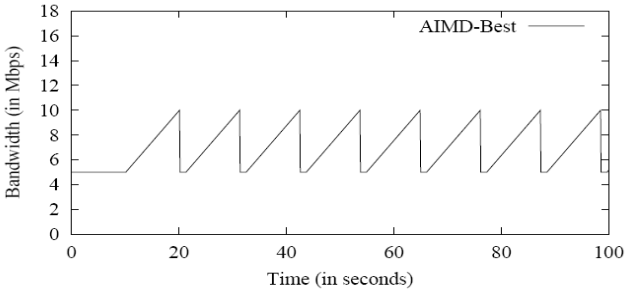
There are a few things to keep in mind when choosing a value for N.

1: The sender must not transmit too fast. N should be bounded by the receiver's ability to process packets.
2: N must be smaller than the number of sequence numbers (if they are numbered from zero to N) to verify transmission in cases of any packet (any data or ACK packet) being dropped.
3: Given the bounds presented in (1) and (2), choose N to be the largest number possible.

The receiver window is one frame wide; on a frame error the receiver discards the frame and all subsequent fames and sends no ACKs. Eventually the senders will timeout and resend the damaged frame and all subsequent frames. This can waste a lot of bandwidth if the error rate is high.

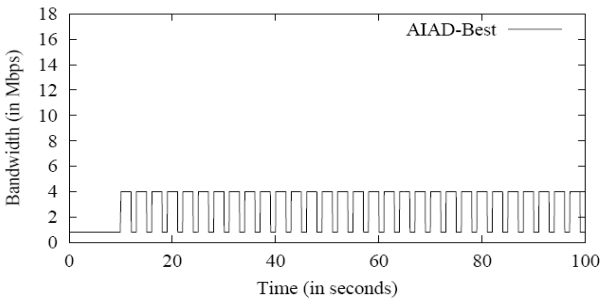
2 Is AIMD Clearly Superior?

Our first question is whether AIMD is clearly superior to the other choices in terms of the good put achieved. For each of the algorithms we can find scenarios in which it performs the best and the worst among the 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.



AIMD	AIAD
0.97	0.93

(a)

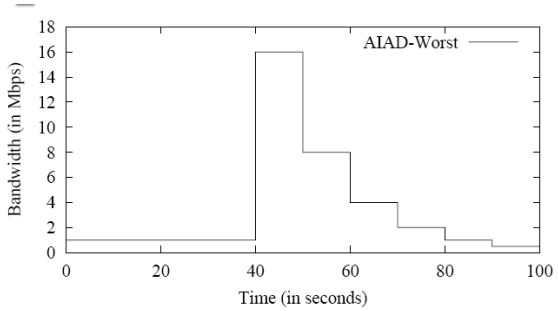


AIMD	AIAD
0.52	0.96

(b)

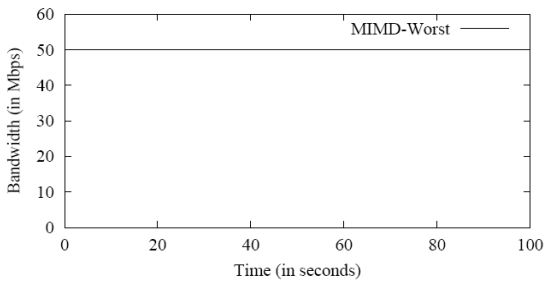
Figure 1: The good put seen by the various algorithms for the given variation is shown in the table below each plot.
The good put, measured as the fraction of average available bandwidth used to transmit unique packets, is a number in [0; 1]. The canonical congestion control schemes we consider use either 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.

Figure 1 shows two patterns of bandwidth variations. We measure the goodput for the two linear algorithms in these scenarios. The scenarios are chosen such that in each case there is a different algorithm that achieves the highest throughput. Figure 1(a) shows a saw-tooth bandwidth pattern under which AIMD performs the best. Com-pared to AIMD, the window size under AIAD decreases too slowly. As a result these schemes experience more losses, and consequently more re-transmissions than AIMD.



AIMD	AIAD
0.67	0.65

(a)



AIMD	AIAD
0.96	0.96

(b)

Figure 2: Bad cases for AIAD (Figure (a) and MIMD (Figure (b))). Notice that the y-axes are on different scales. 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. 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. 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. AIAD suffers from the fact that it cannot increase the window size fast enough during the high periods.

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.

For instance, AIMD exhibits the worst performance in the experiments presented in Figures 1(b), 1(c) and 1(d)). 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.

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. 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.

Let A be the set of congestion control algorithms that we wish to compare, a consists of AIMD, AIAD. 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.

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	$\frac{\sum_{e \in E} d_a(e)}{\|E\|}$, metric for average performance

Table 1: Notation

	Environment	Variation in per-flow available bandwidth (Bt)
1	Cons-low (CL)	Constant at 1 Mbps
2	Cons-High (CH)	Constant at 10 Mbps
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	Bt+1 € [Bt –Δ ,Bt + Δ],

	(RWL)	B0 = 1Mbps, Δ= 0:5B0
6	rw-high (RWH)	same as above except that B0 = 10Mbps
7	rd-low (RDL)	Bt Є [0,B0], B0 = 1Mbps
8	rd-high (RDH)	Bt Є [0,B0], B0 = 10Mbps
9	rw-additive (RWA)	Bt+1 Є [0,Bt + Δ], B0 = 1Mbps, Δ = 0:25B0
10	rw-multiplicative (RWM)	Bt+1 Є [0,μBt], B0 = 1Mbps, μ= 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

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 react 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).
- Square-Wave: the available bandwidth undergoes square wave oscillations, with the band-width 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 = 1Mbps and for rw-high (RWH), B0 = 10Mbps.
- Random (RDL, RDH): The bandwidth is chosen uniformly randomly from the interval $[0, B_0]$ where B_0 = 1Mbps for rd-low (RDL) and B_0 = 10Mbps for rd-high (RDH).
- Additive Random Walk (RWA): For the environment rw-additive (RWA), the available band-width 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 = 1Mbps, Δ = 0:25.
- Multiplicative Random Walk (RWM): In the environment rw-multiplicative (RWM), the avail-able 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} Є $[0, \mu B_t]$. Here B0 = 1Mbps, μ = 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 react 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 Є{AIMD, AIAD}, let LC(a,b) denote a linear congestion control scheme with an increase parameter of a and a decrease parameter of b.

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.

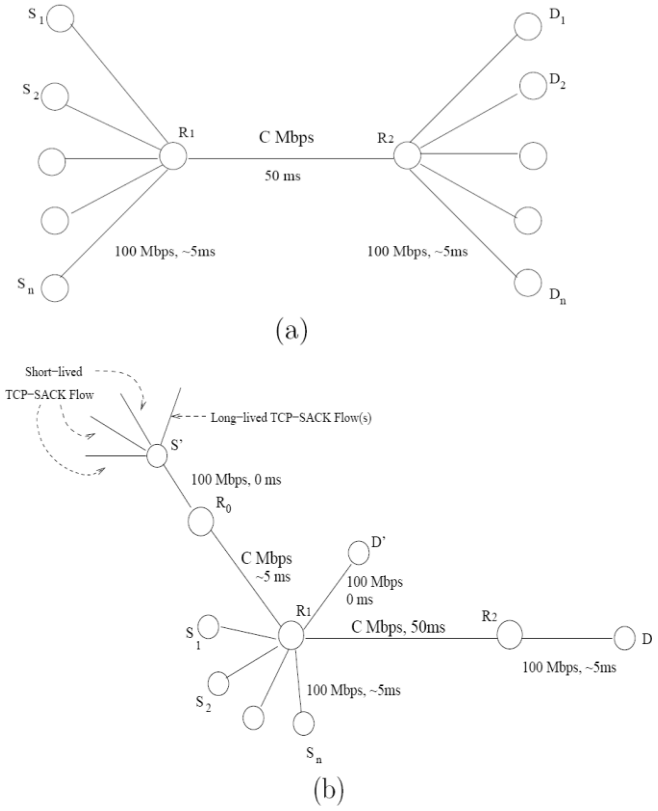


Figure 3: Topology for the simulations. Figure (a) shows the setting for simulations where a CBR source starting on S1 was used to control the available bandwidth on the link R1-R2. For the simulations involving realistic traffic patterns, the setting shown in Figure (b) was used.

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. 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 $\leq$ 3, AD$\leq$3, MI > 0:1 and MD < 0:9. The results of the comparison between the various candidate algorithms in classes {AIMD, AIAD} are shown in the appendix. Based on these results, we choose the following candidate linear control schemes for comparison: AIMD(1,0.8), AIAD (1,3).

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 $B_a = B\text{-}B_c$. 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 R1), we employ a single rate controlled CBR source (between end-points S1 and D1) to realize the bandwidth variations 4. When testing with DRR schedulers, however, we use a time-varying number of fixed rate CBR sources (between S1 and D1). This is because, if we have n simultaneous TCP flows being tested, a single CBR source would be limited to at most 1 n+1th 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 R1 □ R2 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. 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 S0 and D. These constitute the cross traffic on the link R1-R2. The n test TCP flows are between nodes S_i (i = 1…, n) and D. In addition we have n place-holder long-lived flows between nodes S0 and D0. The router R0 employs DRR scheduling. The router at R1 implements either DRR or RIO (explained in greater detail below). The TCP-SACK cross traffic is given priority over the test traffic at router R1. In addition, the bandwidth between R0 and R1 equals that of link R1-R2. While simulating the environments 11 through 13, we would like to ensure that the TCP flows constituting the cross-traffic on link R1-R2 are allocated their fair-share of the R1-R2 capacity irrespective of the congestion control algorithm employed by the test flows. This is ensured by using DRR at router R0 and using n long-lived place-holder flows between S0 and D0. The place-holder flows emulate the n test-flows so

that when the pareto-distributed TCP flows enter link R1-R2, 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 R0-R1 since they already are at their fair-share upon exiting this link. This is ensured by: (i) using RIO at router R1 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 R0 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 R1. Notice that the flows belonging to the cross traffic do not incur any additional losses when DRR is employed at router R1.

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 queuing 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 $(g_{max}-g_{min})/g_{avg}$ where g_{max}, g_{min} and g_{avg} are the maximum, minimum and average goodputs of the test flows respectively.

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 queuing 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.

4.1 The Impact of Loss Recovery Algorithms

TCP Reno incurs a severe penalty when recovering from losses; TCP SACK is more adept at handling losses and therefore incurs a much gentler penalty. We performed simulations of the 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 4Here, and in subsequent tables, we mark the algorithms with the best goodput (best values for both Ca and D_a) by underlining them. From the results, AIMD and AIAD deliver roughly similar goodput in all the test environments.

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. AIMD and AIAD have the highest delay values. 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.

5 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. 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 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.

6 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.

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 queuing. 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 ap- plications," 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 noti_cation," 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 queuing 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 queuing 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.

[20] Exploring Congestion Control Aditya AkellaSrinivasan Seshan Scott Shenker Ion Stoica5 May 2002 CMU-CS-02-139

[21]G. Sireesha Analysis for ARQ Protocols on Multi Channels by using MIMD Congestion Control Algorithm 2011.

TCP Reno + DROPTAIL					
AIMD			AIAD		
Alg	D_a	C_a	Alg	D_a	C_a
(1, 0.5)	0.19	0.06	**(1, 1)**	0.10	0.03
(1, 0.65)	0.06	0.02	(1, 2)	0.05	0.02
(1, 0.8)	0.07	0.03	(1, 3)	0.06	0.03
(2, 0.5)	0.83	0.20	(2, 1)	0.56	0.19
(2, 0.65)	0.56	0.15	(2, 2)	0.62	0.21
(2, 0.8)	0.56	0.22	(2, 3)	0.59	0.22
(3, 0.5)	1.87	0.31	(3, 1)	1.11	0.25
(3, 0.65)	1.49	0.29	(3, 2)	1.15	0.25
(3, 0.8)	1.28	0.26	(3, 3)	1.17	0.27

TCP SACK + DROPTAIL					
AIMD			AIAD		
Alg	D_a	C_a	Alg	D_a	C_a
(1, 0.5)	0.26	0.12	**(1, 1)**	0.50	0.11
(1, 0.65)	0.16	0.11	(1, 2)	0.47	0.11
(1, 0.8)	0.25	0.13	(1, 3)	0.52	0.12
(2, 0.5)	0.11	0.04	(2, 1)	0.41	0.11
(2, 0.65)	0.16	0.04	(2, 2)	0.44	0.11
(2, 0.8)	0.10	0.03	(2, 3)	0.39	0.11
(3, 0.5)	0.13	0.02	(3, 1)	0.31	0.11
(3, 0.65)	0.07	0.01	(3, 2)	0.15	0.11
(3, 0.8)	0.03	0.01	(3, 3)	0.35	0.11

TCP Reno + RED					
AIMD			AIAD		
Alg	D_a	C_a	Alg	D_a	C_a
(1, 0.5)	0.60	0.15	**(1, 1)**	0.32	0.06
(1, 0.65)	0.59	0.15	(1, 2)	0.29	0.08
(1, 0.8)	0.38	0.12	(1, 3)	0.14	0.08
(2, 0.5)	0.50	0.09	(2, 1)	0.39	0.09
(2, 0.65)	0.26	0.06	(2, 2)	0.41	0.08
(2, 0.8)	0.29	0.07	(2, 3)	0.31	0.10
(3, 0.5)	0.47	0.06	(3, 1)	0.76	0.12
(3, 0.65)	0.31	0.04	(3, 2)	0.58	0.11
(3, 0.8)	0.27	0.06	(3, 3)	0.45	0.08

TCP SACK + RED					
AIMD			AIAD		
Alg	D_a	C_a	Alg	D_a	C_a
(1, 0.5)	0.68	0.16	**(1, 1)**	0.36	0.11
(1, 0.65)	0.47	0.14	(1, 2)	0.28	0.11
(1, 0.8)	0.32	0.11	(1, 3)	0.30	0.10
(2, 0.5)	0.42	0.09	(2, 1)	0.31	0.09
(2, 0.65)	0.30	0.08	(2, 2)	0.20	0.05
(2, 0.8)	0.06	0.03	(2, 3)	0.06	0.02
(3, 0.5)	0.36	0.07	(3, 1)	0.49	0.11
(3, 0.65)	0.26	0.07	(3, 2)	0.29	0.09
(3, 0.8)	0.01	0.01	(3, 3)	0.10	0.04

Bibliography:

Mr. P. Radha Krishna Reddy received his B.Sc(CS) from Sri Venkateswara University-Tirupati. M.Sc in Computer science from Sri Venkateswara University-Tirupati, and pursuing M.Tech in Computer Science and Engineering from Vagdevi Institute of Technology and Sciences, JNTU-Anantapur.

Ms. G.Sireesha received her B.Tech in Computer science and Engineering from Royal Institute of Technology and Science, JNTU, Hyderabad, M.Tech in Computer science (Parallel computing) from Aurora's Engineering College, JNTU, Hyderabad.She is working as a Assistant Professor in Computer Science and Engineering in Guru Nanak Institute of Engineering & Technology, JNTU-Hyderabad.

Mr C.V.Chirangeevi Kumar received his B.Tech(CSE) from JNTUH-Hyderabad. M.Tech in Computer Science and Engineering from JNTUH-Hyderabad. And working as a Assistant Professor in Computer Science and Engineering from past 3 years. Now he is working in Vaagdevi Institute of Technology and Sciences, JNTU-Anantapur.

Mr M. Naresh Babu, working as asst prof since from last 2 years completed B.Ttech in CSE from JNTUH and M.Tech in CSE from JNTUA.