Beyond AIMD: Explicit Fair-share Calculation

Paul C. Attie, Adrian Lahanas, Vassilis Tsaoussidis
College of Computer Science, Northeastern University

Abstract

We introduce an alternative approach to congestion avoidance and control, which has the potential to increase efficiency and fairness in multiplexed channels. Our approach, Bimodal Congestion Avoidance and Control, is based on the principles of TCP's Additive Increase Multiplicative Decrease. It is designed to better exploit the system properties during equilibrium, without trading off responsiveness for smoothness. In addition, it is capable of achieving convergence to fairness in only two congestion cycles. As a result, both efficiency and fairness are improved, responsiveness is not degraded, and smoothness is significantly improved when the system is in equilibrium. We provide a theoretical analysis and we discuss the potential of our approach for packet networks. Our experiments confirm that Bimodal Congestion Avoidance and Control as a component of the Transmission Control Protocol outperforms the traditional scheme.

1 Introduction

IInternet congestion avoidance and control is currently governed by the rules of Additive Increase Multiplicative Decrease (AIMD) [4]. Jacobson [8] exploited experimentally AIMD's potential in TCP [15], integrating AIMD with transmission tactics suitable for congestion avoidance and control. Since then, AIMD has become the major component of TCP's congestion avoidance and control mechanism [8]. In AIMD, congestion triggers a drastic response from the senders (multiplicative decrease) to avoid a congestive collapse, a major concern in packet networks. AIMD is also designed to be responsive to fluctuations of bandwidth availability due to varying contention; this is managed by a continuous probing mechanism through additive increase of resource consumption. Chiu and Jain [4] show that AIMD guarantees convergence to fairness: all flows eventually converge to a fair-share, i.e., an equal allocation of resources. Convergence to fairness is faster when the multiplicative decrease is larger, but then, bandwidth is further underutilized, and applications experience severe transmission gaps. Hence, although smoothness is desirable, it works against fairness: the smoother the adjustment, the longer convergence to fairness takes.

Key information for the sources to determine action is whether congestion is due to increasing contention (i.e., new flows joining), or due to increasing bandwidth consumption of the existing flows (additive increase). The former calls for rapid downward adjustments, to allow space for the new flows attempting to utilize the system's bandwidth. The latter calls for a moderate response, since the system limitations relevant to the number of participating flows have already been discovered. Thus, a relatively small decrease in bandwidth consumption followed by more additive probing is sufficient. Our algorithm is the first which makes this distinction between increasing contention and increasing bandwidth consumption. A piece of information which enables efficient congestion avoidance is the "fair-share" of the total bandwidth that each flow should be allocated, at any point during the system's execution. If the fair-share were discovered, then the sources could avoid congestion by adjusting immediately to a new state where the bandwidth allocation of each flow is exactly its fair-share. A system that can discover the fair-share at every point in its execution could utilize bandwidth fully and fairly. However, the fair-share is not only a matter of channel capacity but is also dependent upon the number of participating flows, and the transmitting behavior of the sources. Since applications may finish their tasks, or new flows may enter the system, bandwidth availability needs to be persistently detected at every step of operation; for example, once the fair-share is discovered, flows cannot simply adjust to that value and remain there for their lifetime, since, in that case, bandwidth that eventually becomes available when some flows leave the system remains unexploited, i.e., the increase in the fair-share of each flow remains undetected.

Current systems do not distinguish between congestion due to increasing contention or due to increasing resource consumption, and thus lack a key component of the decision-making process. We present a simple method which enables this distinction; our algorithm explicitly calculates the fair-share and continuously monitors its dynamics. We also go beyond algorithmic improvements by

proposing a congestion control scheme suitable for packet networks. Our scheme exploits the observation that a system in equilibrium (no flows joining or leaving) need not adjust rapidly backwards during congestion, since the cause of congestion in equilibrium is not increasing contention, but rather increasing (but fair) resource consumption. Hence, the same small decrease in the bandwidth allocation of each flow maintains fairness and avoids congestion. When not in equilibrium (i.e., during convergence) the sources indeed adjust with rapid decrease. Since the fair-share can be calculated, the adjustment need not be graduated but can be immediate. The combined tactics, during equilibrium and during convergence, lead to improved smoothness and faster convergence to fairness. Practically, since additive increase is a key bandwidth-detecting mechanism, flows need to adjust slightly below the level of the detected fair-share to allow for continuous bandwidth probing. Hence, from our perspective, upwards and downwards adjustments need to operate in association with the system state, i.e., determine action based on whether the system is in equilibrium (fair-share is known) or not (fair-share is unknown). Due to this property, we call our scheme *Bimodal Congestion Avoidance and Control*. Our scheme does not favor efficiency at the cost of fairness; nor smoothness at the cost of responsiveness, or vice versa. Therefore, we do not attempt to optimize the tradeoff of the additive increase parameter α and the multiplicative decrease parameter β within the framework of TCP limitations (i.e., efficiency) and application requirements (i.e., smoothness) but instead we improve efficiency and fairness of TCP without degrading its potential for congestion avoidance. A protocol can exploit bandwidth well *and* avoid congestion *only* if it is responsive. In this context, the goal of the present work is in marked distinction with the TCP-Friendly protocols which take a useful but somewhat confined perspective, since they favor smoothness at the expense of responsiveness [14].

Our paper has two contributions. First, the concept of *explicit fair-share calculation* as a novel conceptual framework for thinking about, designing, and analyzing congestion control algorithms. Second, our specific congestion control algorithm (Bimodal Congestion Avoidance and Control) based on explicit fair-share calculation, which outperforms traditional AIMD.

Related work. The impact of AIMD has been recently discussed from two perspectives. First, the perspective of improvements to original AIMD. Yang and Lam [16] discuss a control system which extends the system of Chiu & Jain towards asynchronous feedback. Lahanas and Tsaoussidis [11] propose AIMD-FC, a modification which increases both fairness and efficiency, prove properties of the modified algorithm algebraically, and show experimentally significant improvements over TCP. Second, the perspective of parameterization of a general algorithm which exploits the tradeoffs of smoothness and responsiveness but does not disturb the performance of the traditional TCP scheme. Such modifications attempt to achieve similar efficiency to TCP, trading responsiveness for smoothness. The efficiency is associated with the utilized bandwidth; at first, the system dynamics suggest that the higher the oscillation the less the efficiency. It also appears[1] that the higher the oscillation, the faster we approach fairness. Recent versions of AIMD-based algorithms that take advantage of this property are [16, 6, 3, 10]. It has been observed that streaming applications can benefit from modest oscillations since these reflect the smoothness of adjusting the transmission rate backwards. Such protocols are called TCP-friendly because they consume the same bandwidth as TCP($1, \frac{1}{2}$) [14].

2 The Algorithm

We assume the AIMD model of Chiu and Jain [4]: congestion is indicated by feedback from the network to the users (flows) in the form of a *congestion bit*: if the bit is set, this indicates congestion, and each flow then decreases its usage multiplicatively, while if the bit is not set, then each flow increases its usage additively. We assume the control system model (with synchronous feedback) of [4], where time is divided into small intervals (steps), and each flow sets its load at the beginning of each interval based on the congestion bit fed back to it during the previous interval. Although the AIMD model is simple, it captures two important design assumptions: simple binary feedback, and decentralized control [4].

In the context of our system behavior we define the following measurement units. A *cycle* is the phase starting immediately after a system congestion feedback of 1 (indicating congestion) and ending at the next event of congestion when the system congestion bit fed back is again 1. Hence, a cycle consists of one multiplicative decrease step followed by a number of additive increase steps. A *step* reflects each window adjustment towards convergence in response to the congestion bit fed back by the system (0 or 1). Hence, a step during additive increase involves an increment of one ($\alpha = 1$) resource unit per flow, and each increase step involves n packets more than the previous step, n being the number of flows. Since a step involves a round trip, the time taken for a step is exactly the RTT. Hence, any number of steps induces that same number of RTT's. We set five distinct goals:

1. To achieve high bandwidth utilization.
2. To converge to fairness faster.
3. To minimize the length of oscillations.
4. To maintain high responsiveness.

[1] Both statements were made initially in [4].

5. To coexist fairly with the traditional AIMD-based protocols.

Although the sources discover their fair-share early on, the dynamics of real systems in practice prohibit a straightforward adjustment, but instead, call for continuous oscillations as a means of discovering the available bandwidth, and the varying fair-share. Our metrics for system performance are:

***Efficiency*:** the average fraction of total bandwidth utilized by the flows when the system is in equilibrium.

***Responsiveness*:** measured by the number of steps needed to close the gap between two (or more) flows.

***Smoothness*:** reflected by the length of the oscillations during multiplicative decrease.

2.1 Overview of the algorithm

Our key idea is a fast method for calculating the fair-share of each flow, which can be implemented by each flow autonomously. Consider an "equilibrium" situation in which there are n (≥ 1) flows present and no flows join or leave. Further suppose (for the time being) that the flows allocate bandwidth according to AIMD with additive increase parameter α (new bandwidth = old bandwidth + α) and multiplicative decrease parameter β (new bandwidth = old bandwidth * $(1-\beta)$). Suppose the network reaches congestion at some point, due to additive increase. Now, all flows will decrease their bandwidth multiplicatively, and then resume additive increase until the network congests again. Assume that all flows increase their bandwidth at the same rate. Then, from one congestion point to the next, all flows increase their bandwidth by the same amount d. d therefore becomes *common knowledge* [7] amongst all flows, and can be used by each flow to calculate its fair-share of the bandwidth. Thus, within at most two congestion cycles, provided no flows join or leave, every flow can calculate its fair-share and set its allocation directly to the fair-share, i.e., abandon the usual AIMD protocol. Thus, we converge to efficient and fair operation in two cycles.

The algorithm for flow f is given in Figure 1 as an action $step$ which gives the execution of the algorithm during a single step. The algorithm operates in two modes: a mode where the fair-share has been calculated (using d), and is therefore known, and a mode where the fair-share is unknown (e.g., due to new flows joining or leaving, the previously calculated value of the fair-share is now obsolete). The algorithm for flow f uses the variables given in Table 1. $congestion()$ is a system call that returns the current congestion bit. In the $fair\_share\_unknown$ mode, the algorithm behaves like AIMD, until two congestion cycles have passed, which is sufficient to recalculate the fair-share. The algorithm then sets the bandwidth allocation for flow f to $(1-\epsilon)$ times the calculated fair-share, and shifts to the $fair\_share\_known$ mode; ϵ is a small, tunable parameter. In the $fair\_share\_known$ mode, the algorithm continues to use additive increase and multiplicative decrease, but the multiplicative decrease factor is ϵ instead of β. Since ϵ is significantly smaller than β, smoothness is improved. The algorithm also monitors the point at which congestion occurs. If this point is too early, i.e., smaller than $(1-\epsilon)$ * calculated-fair-share, then that indicates that the actual fair-share decreased, due to some new flow(s) joining. Since the new flows are not in a fair state (i.e., have equal allocations), the fair-share must be recalculated, and so the algorithm changes mode to $fair\_share\_unknown$. If this point is too late, i.e., larger than $(1-\epsilon)$ * calculated-fair-share, then that indicates that the actual fair-share increased due to some flow(s) leaving. In this case, the remaining flows are still in a fair state (have equal allocations), and so all that is needed is to set the calculated fair-share to be the allocation of each flow at congestion, and then do a multiplicative decrease by ϵ. The algorithm remains in the $fair\_share\_known$ mode.

2.2 Theoretical basis of the algorithm

We now provide the theoretical basis for our method of calculating the fair-share. The fair-share calculation is always performed in the $fair\_share\_unknown$ mode, where the algorithm behaves like classical AIMD, with additive increase α and multiplicative decrease β. We consider the calculation of the fair-share in the case where each flow receives the same share of the bandwidth, and also in the case of *proportional fairness*, in which each flow is allocated a share of the bandwidth according to a fixed "weight" associated with it. Table 2 gives some notation. We define $A(t) = \sum_{f \in \mathcal{F}(t)} a_f(t)$, i.e., $A(t)$ is the total allocation of the current flows at time t. For technical convenience, we number the congestion cycles that occur during system execution as cycle 1, cycle 2, etc.

Let $c \geq 1$ be an arbitrary cycle such that the congestion point at the end of cycle $c-1$ resulted in a multiplicative decrease by factor β. Equations 1 and 2 hold because the sum of the flow allocations at a congestion point is equal to the total available bandwidth B, by definition. Note that we do not assume that B is known to the flows.

$$A(e_{c-1}) = B \tag{1}$$

$$A(e_c) = B \tag{2}$$

Equation 3 holds by definition of multiplicative decrease.

$$A(b_c) = (1-\beta)A(e_c) = (1-\beta)B \tag{3}$$

The above equations apply to the cases of both fairness and proportional fairness.

con_f	the congestion bit sent back to flow f
$mode_f$	the current mode of flow f
a_f	the current bandwidth allocation (or window size) of flow f
$fair_f$	the calculated fair-share for flow f
cc_f	a count of the number of cycles passed since a mode change for flow f
bc_f	the bandwidth allocation for flow f at the beginning of the latest cycle

Table 1. The variables used in the algorithm.

Initially: $mode_f = fair\_share\_unknown \wedge con_f = false \wedge cc_f = false$.

$step(f, mode_f, con_f, cc_f, bc_f, fair_f)$
- **if** $mode_f = fair\_share\_unknown$ **then**
  - **if** $\neg con_f$ **then**
    - $a_f \leftarrow a_f + \alpha$ ▷ additive increase
  - **else** $\{con_f\}$ ▷ congested
    - **if** cc_f **then** ▷ ≥ 2 cycles since modechange
      - $fair_f \leftarrow (a_f - bc_f)/\beta$ ▷ calculate fair-share
      - $a_f \leftarrow fair_f * (1 - \epsilon)$
      - $bc_f \leftarrow a_f$ ▷ record allocation at beginning of new cycle
      - $mode_f \leftarrow fair\_share\_known$
    - **else**
      - $a_f \leftarrow a_f * (1 - \beta)$ ▷ multiplicative decrease by β
      - $bc_f \leftarrow a_f$ ▷ record allocation at beginning of new cycle
      - $cc_f \leftarrow true$
    - **endif**;
    - $con_f \leftarrow false$ ▷ reset congestion bit
  - **endif**
- **else** $\{mode_f = fair\_share\_known\}$
  - **if** $\neg con_f$ **then**
    - $a_f \leftarrow a_f + \alpha$ ▷ additive increase
  - **else** $\{con_f\}$ ▷ congested
    - **if** $a_f < fair_f$ **then** ▷ congested too early: fair-share decreased
      - $a_f \leftarrow a_f * (1 - \beta)$ ▷ multiplicative decrease by β
      - $cc_f \leftarrow false$ ▷ reset cc_f
      - $mode_f \leftarrow fair\_share\_unknown$ ▷ new flows are not in fair state
    - **else if** $a_f > fair_f$ **then** ▷ congested too late: fair-share increased
      - $fair_f \leftarrow a_f$ ▷ flows are in fair state
      - $a_f \leftarrow a_f * (1 - \epsilon)$ ▷ multiplicative decrease by ϵ
    - **else** $\{a_f = fair_f\}$ ▷ congestion due to additive increase
      - $a_f \leftarrow fair_f * (1 - \epsilon)$ ▷ adjust based on fair-share
    - **endif**;
    - $con_f \leftarrow false$ ▷ reset congestion bit
  - **endif**
- **endif**;
- $con_f \leftarrow congestion()$ ▷ get congestion bit feedback for current interval

Figure 1. The algorithm.

notation	meaning
B	the fixed total amount of bandwidth available
$\mathcal{F}(t)$	set of current flows at time t
$f, g, \ldots$	flow identifiers
$F, G, \ldots$	sets of flow identifiers
$a_f(t)$	the bandwidth allocation of flow f at time t
c	congestion cycle number
b_c	begin time of cycle c
e_c	end time of cycle c

Table 2. Notation.

Calculation of the fair-share Consider an arbitrary system state. There is a fixed set F of current flows, and each has some bandwidth allocation. Each flow additively increases its allocation by the same α at each step until congestion occurs. Let $n = |F|$, i.e., the size of F. We define the actual fair-share to be B/n, i.e., the available bandwidth divided by the number of current flows. We now show how the fair-share can be calculated by each flow independently.

Equation 4 holds by our assumption that all flows increase their bandwidth allocation at the same rate (additive increase by α at each step).

$$\bigwedge_{f,g \in F} a_f(e_c) - a_f(b_c) \;=\; a_g(e_c) - a_g(b_c) \qquad (4)$$

Let g be an arbitrary flow id in F. Thus,

$$n(a_g(e_c) - a_g(b_c)) \;=\; \sum_{f \in F} (a_f(e_c) - a_f(b_c))$$

From (2) and (3), we have $A(e_c) - A(b_c) = \beta B$. Now $A(e_c) = \sum_{f \in F} a_f(e_c)$, $A(b_c) = \sum_{f \in F} a_f(b_c)$. Hence

$$\sum_{f \in F} (a_f(e_c) - a_f(b_c)) \;=\; \beta B.$$

From the above two displayed equations, we obtain $\beta B = n(a_g(e_c) - a_g(b_c))$. And so

$$(a_g(e_c) - a_g(b_c))/\beta \;=\; B/n.$$

We show below that our algorithm calculates $(a_g(e_c) - a_g(b_c))/\beta$ as the fair-share. Thus, the calculated fair-share is equal to the actual fair-share. Thus, flow g can calculate the fair-share by simply recording its beginning and ending allocations on the second cycle after initialization, or after a mode change. The first cycle is needed to generate the multiplicative decrease by β.

Calculation of the proportional fair-share The situation is the same as described above, except that each flow f has its own weight α_f, and increases its allocation by α_f at each step. Thus, the additive increase parameter is different for each flow, in general, but the multiplicative decrease parameters (β, ϵ) are the same for all flows. We define the *actual proportional-fair-share* to be the allocation of available bandwidth to each flow f in proportion to α_f. Let $p_f = \alpha_f / \sum_{g \in F} \alpha_g$. Then, the actual proportional fair-share of flow f is $p_f B$. We show, in the technical report [2], that

$$(a_g(e_c) - a_g(b_c))/\beta \;=\; p_g B.$$

Thus, the calculated fair-share $(a_g(e_c) - a_g(b_c))/\beta$ is equal to the actual proportional fair-share. Note that if the weights α_f are all equal, then $p_g = 1/n$ and this agrees with the result of Section 2.2 for fairness.

The technical report version of this paper [2] gives formal proofs that our algorithm computes the fair-share correctly for all the above scenarios, i.e., the computed fair-share in execution corresponds to the theoretical expressions for fair-share given above, in all the scenarios.

Efficiency of Bimodal Congestion Avoidance and Control In equilibrium, our algorithm uses ϵ for the multiplicative decrease factor. Thus, in equilibrium, the efficiency is $1 - (\epsilon/2)$. Since ϵ is much smaller than β, this compares favorably with the equilibrium efficiency of $1 - (\beta/2)$ for traditional AIMD.

3 Experimental Results

We incorporated our algorithm into TCP [15] and validated its performance on NS-2 [1]. TCP controls the sending rate with the *congestion window* [8] parameter. When resources are available TCP increases the congestion window by *one* Maximum Segment Size (MSS); upon congestion and after three duplicate acknowledgments, TCP multiplies the congestion window by a factor of 1/2 (this is TCP($1, \frac{1}{2}$)). In the absence of errors the average long term efficiency of the AIMD mechanism of TCP is 75% [13, 5].

We used TCP-SACK [12] in our experiments. Due to its fast recovery and its capability for multiple retransmissions within one RTT, this version matches better our theoretical assumptions. In our experiments, multiple flows share a high-bandwidth bottleneck link (see Figure 2); the fair-share (the Delay×Bandwidth share per flow) was set relatively high to provide an environment which adequately tests the potential of the algorithms. For example, AIMD is not activated when the fair-share is only one packet, or otherwise when contention is too high and bandwidth is limited, efficiency is not really an issue.

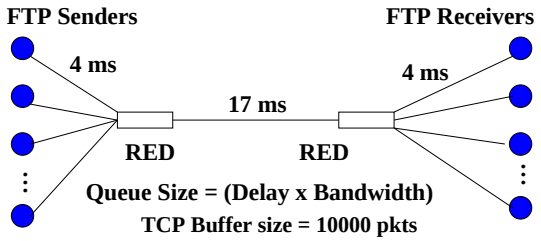

Figure 2. Multiple flows experimental set-up for AIMD evaluation.

We evaluate three scenarios: Our first scenario has a fixed number of participating flows. We study comparatively the behavior of the algorithms and we present experiments with both default and RED gateways. Our second scenario involves progressive contention due to periodic increase of the number of flows. The subject matter we investigate with this experiment is the mechanism's potential for efficient congestion avoidance and control, i.e., not only its convergence behavior. In our third experiment we evaluate the system's responsiveness: bandwidth becomes available and protocols should demonstrate the capability to consume the available resources quickly. Both our second and third experiments aim at alleviating concerns regarding the algorithm's behavior in dynamic (and hence more realistic) environments.

A TCP flow runs at each end node and an FTP application generates traffic for each source. The application sends data for 60 seconds. The RED queue buffers were set equal to the Delay×Bandwidth product. We measured the number of packets that arrive at the receivers; since the time of the experiments is fixed we report this number as *Goodput* in the figures (average of 30 experiments with minimal statistical deviation). Goodput is a metric for system efficiency. In line with our theoretical findings and in order to measure the convergence behavior of the participating flows, we use the Fairness Index used in [9]: $F(x) = (\sum x_i)^2/n(\sum x_i^2)$, where x_i is the goodput achieved by each flow.

Experimental Results for a Stationary Environment We present results with $\epsilon = 1/2$ (Figures 3, 4), $\epsilon = 1/3$ (Figures 5, 6), and $\epsilon = 1/8$ (Figures 7, 8). For $\epsilon = 1/2$, efficiency (in terms of bandwidth utilization) does not improve. However, due to the algorithms ability to calculate the fair-share, fairness is improved (see Figure 4). We demonstrate notable improvement in efficiency when ϵ is smaller. In Figure 5 where ϵ is $1/3$ goodput is increased up to 5% and in Figure 7 where ϵ is $1/8$ goodput is improved up to 10%. The corresponding improvements in fairness can be seen in Figures 6 and 8 respectively.

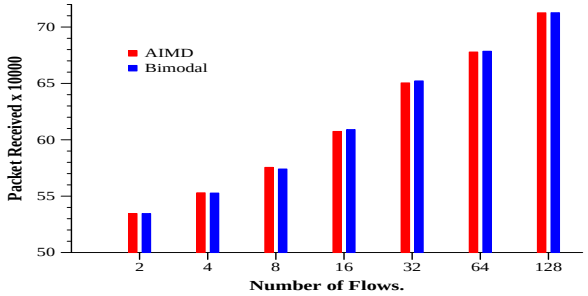

Figure 3. Goodput Performance on a 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

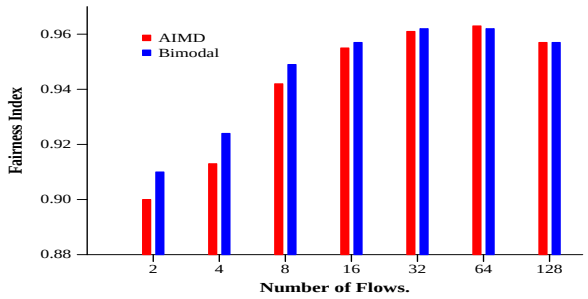

Figure 4. Fairness with the 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

Experimental Results for Graduated Contention Increase A reasonable question regarding the performance of Bimodal Congestion Avoidance and Control arise when contention increases unpredictably, and then adjusting to a precalculated fair-state may be risky. However, when contention increases (hence the actual fair-share decreases), calculation of the fair-share is not based on historical data but on the most recent evaluation of the number of steps, and on the values of β, ϵ. Hence, the capability of the algorithm to perform efficient congestion avoidance when contention increases is not compromised. Once the situation is

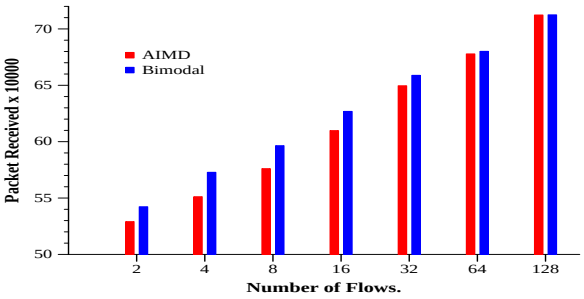

Figure 5. Goodput Performance on a 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{3}$, $\beta = \frac{1}{2}$.

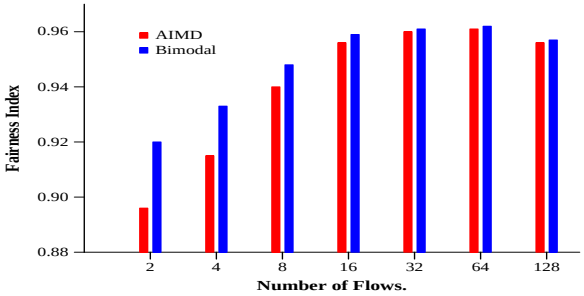

Figure 6. Fairness with the 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{3}$, $\beta = \frac{1}{2}$.

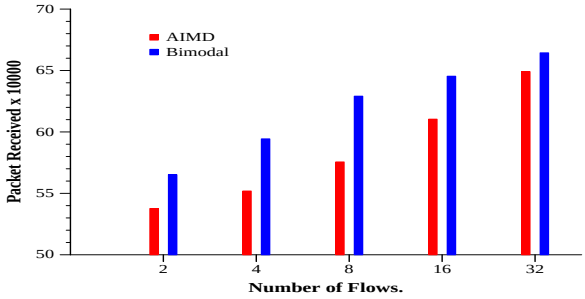

Figure 7. Goodput Performance on a 100Mbps link and RED Gateway. $\epsilon = \frac{1}{8}$, $\beta = \frac{1}{2}$.

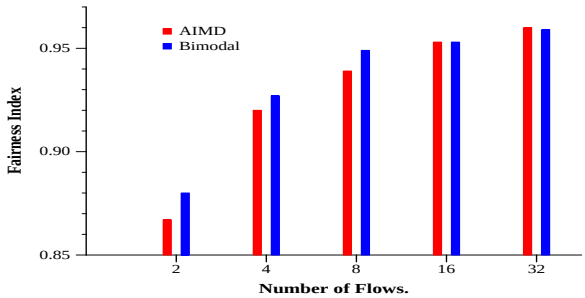

Figure 8. Fairness with the 100Mbps link and RED Gateway. $\epsilon = \frac{1}{8}$, $\beta = \frac{1}{2}$.

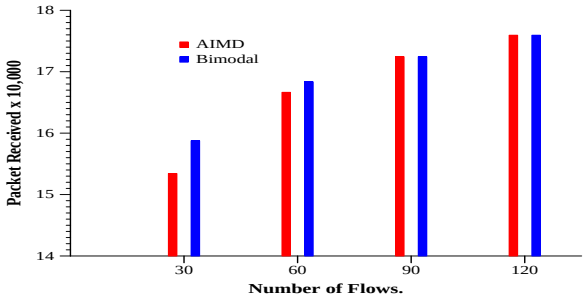

Figure 9. Goodput Performance on a 100Mbps link and a RED Gateway. The number of flows is increased by 30 every 15 seconds. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

detected[2] multiplicative decrease is drastic, making space for the new flows as in standard AIMD. Results of the experiments are presented in Figures 9 and 10.

Experimental Results for Graduated Bandwidth Increase We consider a deterministic scenario where some applications finish their tasks earlier than others i.e., not a bandwidth provisioning scenario. We measure the capability of the algorithm to exploit available bandwidth efficiently and fairly. With ϵ equal to β, a difference in goodput is not really expected. Unlike goodput, fairness shows an improvement. We present the results in Figures 11 and 12 respectively.

4 Discussion and Conclusions

We presented a congestion control algorithm that explicitly calculates the fair share and converges to it in two congestion cycles. Our algorithm outperforms standard AIMD. One issue is that the more the gain we have in goodput, by using a smaller ϵ, the less free space is left for incoming flows when contention increases. Although the system converges in two cycles, leaving no free space in order to maximize bandwidth utilization will impact the packet overhead due to retransmission. Investigating the optimal value of ϵ in conjunction with the dynamics of specific environments is a subject of future work.

Our algorithm is executed only by the sender. The sender

[2]Note that contention increase cannot be detected by traditional AIMD.

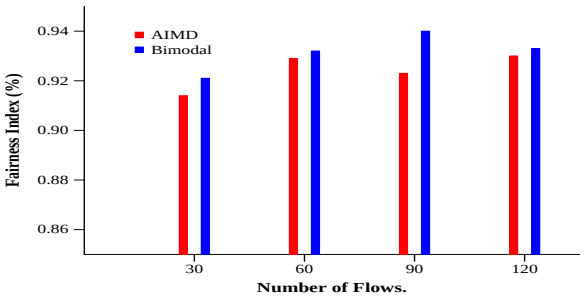

Figure 10. Fairness with 100Mbps link and a RED Gateway. The number of flows is increased by 30 every 15 seconds.

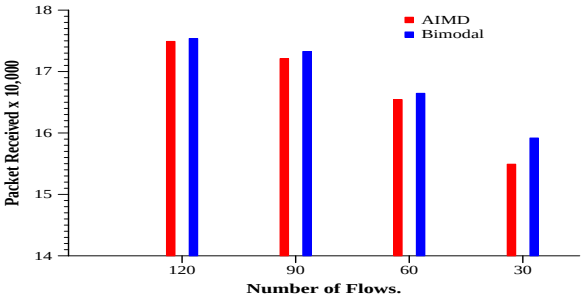

Figure 11. Goodput Performance on a 100Mbps link and a RED Gateway. The number of flows is halved every 15 seconds. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

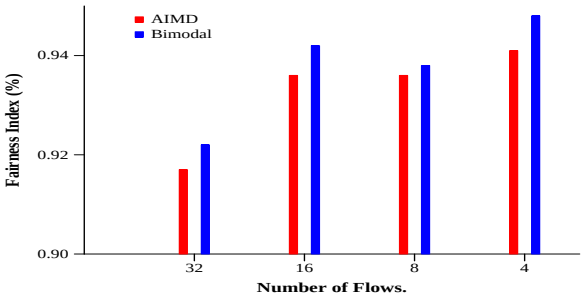

Figure 12. Fairness with the 100Mbps link and a RED Gateway. The number of flows is halved every 15 seconds.

maintains the current flow state and mode, and the router need not maintain any extra information. Thus, our algorithm works with standard IP routers, since it only relies on a 1 bit feedback in the form of congestion/no congestion, which in practice is generated by a packet drop or acknowledgment. Requiring the sender to maintain a small amount of information (two window sizes and three bits) per flow is not an impediment, in comparison to the amount of data a typical sender transmits. Also, the per flow state is maintained in a distributed fashion by the senders, and is not concentrated at performance-critical routers, where the aggregation of per-flow state for, e.g., thousands of flows, would cause severe performance degradation.

We distinguish our algorithm from the TCP-friendly algorithms, which favor smoothness at the cost of fairness. Our algorithm calculates the fair share explicitly, and so fairness is not compromised in our approach.

References

[1] The Network Simulator - NS-2. Technical report, Web Page: http://www.isi.edu/nsnam/ns/.

[2] P. C. Attie, A. Lahanas, and V. Tsaoussidis. Bimodal congestion avoidance and control. Technical report, College of Computer Science, Northeastern University, June 2002. Web Page: http://www.ccs.neu.edu/home/attie/papers/bimodal.ps.

[3] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proc. IEEE INFOCOM*, 2001.

[4] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1), June 1989.

[5] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proceedings of the ACM SIGCOMM 2000*, May 2000.

[7] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3), 1990.

[8] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM*, August 1988.

[9] R. Jain, D. M. Chiu, and H. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems. Technical Report DEC-TR-301, Digital Equipment Corporation, 1984.

[10] S. Jin, L. Guo, I. Matta, and A. Bestavros. TCP-friendly SIMD Congestion Control and Its Convergence Behavior. In *Proceedings of the ICNP'2001*, November 2001.

[11] A. Lahanas and V. Tsaoussidis. Additive Increase Multiplicative Decrease - Fast Convergence (AIMD-FC). In *Proc. Networks 2002, Atlanta, Georgia*.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. *RFC 2018*, April 1996.

[13] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27:20–26, July 1997.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of the ACM SIGCOMM*, 1998.

[15] J. Postel. Transmission Control Protocol. *RFC 793*, 1981.

[16] Y. Yang and S. Lam. General AIMD Congestion Control. In *IEEE International Conference on Network Protocols*, 2000.