Bimodal Congestion Avoidance and Control

Paul C. Attie, Adrian Lahanas, and Vassilis Tsaoussidis

***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 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 for both static and dynamic environments 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.**

***Index Terms*— AIMD, fairness, congestion control.**

I. Introduction

Internet congestion avoidance and control is largely based on the behavior of transport protocols. Competing flows continuously adjust their windows upwards or downwards, trying to follow the network dynamics with regard to the available bandwidth or the alarming signals of congestion. Since no centralized authority controls the actions of the transmitting sources, the competing flows make decisions independently, based on the system signal provided. By default, this signal is a packet drop due to congestion, and is detected through a timeout at the source. Alternatively, a sophisticated router may mark or drop packets when congestion boosts up. The primary concern of any congestion avoidance and control algorithm is to avoid a congestive collapse; a backward adjustment is needed when congestion boosts up. Furthermore, resolving contention efficiently requires fair bandwidth allocation and efficient bandwidth exploitation.

Internet applications are currently governed by the rules of Additive Increase Multiplicative Decrease (AIMD), an algorithm proposed by Chiu and Jain in [1] as the most efficient approach to resource management. Jacobson in [2] exploited experimentally the mechanism's potential in TCP [3], integrating AIMD with transmission tactics suitable for congestion avoidance and control. Since then, the algorithm has become the major component of TCP's [2] congestion avoidance and control and, in turn, has signified an operational point for the Internet. According to AIMD principles, congestion should trigger a drastic response from the senders (multiplicative decrease) to avoid a congestive collapse, a major concern in packet networks of the last decade. In addition, the algorithm is 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 have shown that additive increase multiplicative decrease guarantees convergence to fairness. That is, all flows will eventually converge to a fair-share. The flows will converge to fairness faster when the multiplicative decrease is larger, but then, bandwidth will be further underutilized; moreover, applications will experience severe transmission gaps. Hence, although smoothness is certainly a desirable behavior, 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, in order 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. Furthermore, a piece of information that 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 known, then the sources could avoid congestion by adjusting immediately after the fair-share was discovered, to a new state where the bandwidth allocation of each flow is exactly its fair-share. Obviously, such a system could utilize bandwidth fully and fairly. However, bandwidth availability is not only a matter of channel capacity but is also dependent upon the number of participating flows, and the

College of Computer Science,Northeastern University,Boston, Massachusetts 02115

transmitting behavior of the sources. Since applications may finish their tasks, or since new flows may enter the system, bandwidth availability needs to be persistently detected at every single 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 will remain unexploited.

Current systems do not distinguish between congestion due to increasing contention or due to increasing resource consumption, and hence lack a key component of the decision-making process. We provide here a simple method which enables this distinction; our algorithm explicitly calculates the fair-share and continuously monitors its dynamics. Furthermore, we go beyond the algorithmic improvements by proposing a congestion control scheme which is suitable for packet networks. Our proposal is based on 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 the increasing contention, but instead, the increasing (but fair) resource consumption. 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.

We also note that the modifications presented here do not favor efficiency at the cost of fairness; nor do they favor smoothness at the cost of responsiveness, or vice versa. Therefore, we do not attempt here to optimize the balancing trade of the additive increase parameter α and the multiplicative decrease parameter β within the frame of TCP limitations (i.e., efficiency) and application requirements (i.e., smoothness) but instead we attempt to 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 (see [4]).

We organize the paper as follows. Section II presents our congestion control algorithm. Section III presents a theoretical analysis of the algorithm, and establishes some correctness properties for it. Section IV discusses related work. Section V presents some experimental evaluation of our algorithm and discusses our experimental setup. Section VI presents some discussion of our results, and Section VII concludes.

II. The Algorithm

A. Technical Assumptions

We assume the model of Chiu and Jain[1]: 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 [1], 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. This is also similar to the synchronous rounds-based model of distributed computing [5, part 1].

B. Definitions, Goals and Metrics

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). In the context of our system, the number of steps matches the number of RTTs.

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.

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, they call for continuous oscillations as a means of discovering the available bandwidth (and, in the context of the present work, the varying fair-share).

Our metrics for the system performance are as follows:

Efficiency: the average fraction of the 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.

C. Overview

The key idea underlying the algorithm is a fast method for calculating the fair-share of each flow. This can be done 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 the classical AIMD algorithm [1] 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 the 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 will increase their bandwidth by the same amount d. d therefore becomes *common knowledge* [6], [7] amongst all the 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 $step$ that gives the execution of the algorithm during a single step (see Section II-B above for the definition of a 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 I. $congestion()$ 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. 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.

D. Calculation of the fair-share

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 two main cases: the *equilibrium* case, in which no flows join or leave, and the *transient* case, in which flows continuously join and leave. With each case, we consider the calculation of the fair-share in which each flow receives the same share of the bandwidth, and the calculation of *proportional fairness*, in which each flow is allocated a share of the bandwidth according to a fixed "weight" associated with it. Table II introduces some notation, and Table III introduces some abbreviations. 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 with factor β. Equations 1 and 2 hold because the sum of the flow allocations at a congestion point is

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 I

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 ▷ 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
      $bc_f \leftarrow a_f$ ▷ 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
      $cc_f \leftarrow false$ ▷ 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
    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

Fig. 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
$j(t)$	the set of identifiers of flows that have joined, up to time t
$\ell(t)$	the set of identifiers of flows that have left, up to time t

TABLE II
NOTATION.

abbreviation	its definition	its meaning
j_c	$(j(e_c) - j(e_{c-1})) - \ell(e_c)$	the set of identifiers of flows that joined and did not leave in cycle c
ℓ_c	$(\ell(e_c) - \ell(e_{c-1})) - j(e_c)$	the set of identifiers of flows current at the beginning of c that left in c
$A(G,t)$	$\sum_{f \in G} a_f(t)$	total allocation of flows in set G at time t
$A(t)$	$\sum_{f \in \mathcal{F}(t)} a_f(t)$	total allocation of current flows at time t

TABLE III
ABBREVIATIONS.

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 all scenarios: transient and equilibrium, fairness and proportional fairness.

1) Calculation of the fair-share in the equilibrium scenario: 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).

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

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.

2) Calculation of the proportional fair-share in the equilibrium scenario: The situation is the same as described in Section II-D.1 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 $p_f B$.

Let f, g be arbitrary flows in F. Then, the number of steps in cycle c is equal to $(a_f(e_c) - a_f(b_c))/\alpha_f$, i.e., the total amount of allocation increase of flow f divided by the increase per step. Since the number of steps is also equal to $(a_g(e_c) - a_g(b_c))/\alpha_g$, we obtain

$$\bigwedge_{f,g \in F} (a_f(e_c) - a_f(b_c))/\alpha_f \; = \; (a_g(e_c) - a_g(b_c))/\alpha_g \quad (5)$$

Thus $(a_f(e_c) - a_f(b_c)) \; = \; (\alpha_f/\alpha_g)(a_g(e_c) - a_g(b_c))$. Fix g to be an arbitrary flow id in F. Then

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

$$\sum_{f \in F} (\alpha_f/\alpha_g)(a_g(e_c) - a_g(b_c)) =$$

$$(a_g(e_c) - a_g(b_c))(1/\alpha_g) \sum_{f \in F} \alpha_f =$$

$$(a_g(e_c) - a_g(b_c))/p_g.$$

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

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

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 II-D.1 for fairness.

3) Calculation of the proportional fair-share in the transient scenario: Since the proportional fairness case subsumes the (regular) fairness case, we only present the analysis for proportional fairness in the transient scenario. Each flow f has its own weight α_f, which is the additive increase parameter, as described in Section II-D.2 above. Flows join and leave dynamically.

Let f, g be arbitrary flows that are both present throughout cycle c. Then, the number of steps in cycle c is equal to $(a_f(e_c) - a_f(b_c))/\alpha_f$, i.e., the total amount of allocation increase of flow f divided by the increase per step. Since the number of steps is also equal to $(a_g(e_c) - a_g(b_c))/\alpha_g$, we obtain

$$\bigwedge_{f,g \in \mathcal{F}(b_c) \cap \mathcal{F}(e_c)} (a_f(e_c) - a_f(b_c))/\alpha_f = (a_g(e_c) - a_g(b_c))/\alpha_g \quad (6)$$

Thus $(a_f(e_c) - a_f(b_c)) \; = \; (\alpha_f/\alpha_g)(a_g(e_c) - a_g(b_c))$. From (2) and (3), we have $A(e_c) - A(b_c) = \beta B$. Hence

$$A(\mathcal{F}(e_c) - j_c, e_c) + A(j_c, e_c) - [A(\mathcal{F}(b_c) - \ell_c, b_c) + A(\ell_c, b_c)]$$

$$= \beta B,$$

and so

$$A(\mathcal{F}(e_c) - j_c, e_c) - A(\mathcal{F}(b_c) - \ell_c, b_c) =$$

$$\beta B + A(\ell_c, b_c) - A(j_c, e_c).$$

Assuming flow id's are unique within a cycle, we have, from the definitions, $\mathcal{F}(e_c) - j_c = \mathcal{F}(b_c) - \ell_c = \mathcal{F}(b_c) \cap \mathcal{F}(e_c)$. Hence

$$A(\mathcal{F}(b_c) - \ell_c, e_c) - A(\mathcal{F}(b_c) - \ell_c, b_c) =$$

$$\beta B + A(\ell_c, b_c) - A(j_c, e_c).$$

Now $A(\mathcal{F}(b_c) - \ell_c, e_c) - A(\mathcal{F}(b_c) - \ell_c, b_c) = \sum_{f \in \mathcal{F}(b_c) - \ell_c} (a_f(e_c) - a_f(b_c))$. Hence

$$\sum_{f \in \mathcal{F}(b_c) - \ell_c} (a_f(e_c) - a_f(b_c)) = \beta B + A(\ell_c, b_c) - A(j_c, e_c)$$

Fix g to be an arbitrary flow id in $\mathcal{F}(b_c) - \ell_c$. Then by equation 6, we have

$$\sum_{f \in \mathcal{F}(b_c) - \ell_c} (a_f(e_c) - a_f(b_c)) =$$

$$\sum_{f \in \mathcal{F}(b_c) - \ell_c} (\alpha_f/\alpha_g)(a_g(e_c) - a_g(b_c)) =$$

$$(a_g(e_c) - a_g(b_c))(1/\alpha_g) \sum_{f \in \mathcal{F}(b_c) - \ell_c} \alpha_f.$$

Hence $\sum_{f \in \mathcal{F}(b_c) - \ell_c} (a_f(e_c) - a_f(b_c)) \; = \; (a_g(e_c) - a_g(b_c))/p_g$, where $p_g = \alpha_g / \sum_{f \in \mathcal{F}(b_c) - \ell_c} \alpha_f$. And so,

$$(a_g(e_c) - a_g(b_c))/p_g = \beta B + A(\ell_c, b_c) - A(j_c, e_c)$$

and finally

$$(a_g(e_c) - a_g(b_c))/\beta = p_g B + (p_g/\beta)[A(\ell_c, b_c) - A(j_c, e_c)].$$

For the transient setting, we define the actual proportional fair-share of flow g to be $p_g B$, where $p_g = \alpha_g / \sum_{f \in \mathcal{F}(b_c) - \ell_c} \alpha_f$. This notion of proportional fair-share is only approximate, since it only considers the flows that are current throughout the cycle c, and ignores the flows that join and leave within cycle c. Since our notions of fair-share and proportional fair-share are defined on a per-cycle basis, this is unavoidable. A finer-grain notion of fair-share, defined on a per-step basis, could address this shortcoming. We intend to pursue this in future work.

The calculated fair-share is $(a_g(e_c) - a_g(b_c))/\beta$, and differs from the actual proportional fair-share $p_g B$ by an "error factor" of $(p_g/\beta)[A(\ell_c, b_c) - A(j_c, e_c)]$. When the bandwidth usage $A(j_c, e_c)$ of flows joining and the bandwidth usage $A(\ell_c, b_c)$ of flows leaving is approximately equal, this error will be small.

III. Analysis and Proof of Correctness

We analyze the properties of the algorithm in both the equilibrium scenario and the transient scenario. We assume the synchronous rounds-based model of computation given in [5, part 1], thus, in each round (i.e., step), the action $step$ in Figure 1 is executed for each current flow f. We also adopt a formal approach to modeling the semantics of the algorithm. Since the algorithm is *dynamic* in that flows are continuously joining and leaving, we adapt the approach of [8]. Thus, we define a *global state* s to be a pair $\langle s.\mathcal{F}, s.\mathcal{S} \rangle$ where $s.\mathcal{F}$ is a set of *flow identifiers*, and $s.\mathcal{S}$ maps each $f \in \mathcal{F}$ to a *flow-state* for the flow with identifier f (i.e., "flow f"). A flow-state is an assignment of values to the variables in Table I. The meaning of a global state s is that each $f \in s.\mathcal{F}$ is the identifier of a currently active flow, and $s.\mathcal{S}(f)$ gives the current state of flow f.

A computation π is a sequence of global states such that: (1) the first state of π assigns $fair\_share\_unknown$ to $mode_f$, $false$ to cc_f, and $false$ to con_f, for every flow f that is "initially" current, and (2) if s' immediately follows s in π, then s' results from s by: (2.1) executing $step(f, mode_f, con_f, cc_f, bc_f, fair_f)$ for each flow f that is current in both s and s', (hence, for each $f \in s.\mathcal{F} \cap s'.\mathcal{F}$, $s'.\mathcal{S}(f)$ results from $s.\mathcal{S}(f)$ by a single execution of $step(f, mode_f, con_f, cc_f, bc_f, fair_f)$), and (2.2) removing the identifiers of all flows that left in s, i.e., $s.\mathcal{F} - s'.\mathcal{F}$ = all the flows that left in s, and (2.3) adding the identifiers of all flows that joined in s, i.e., $s'.\mathcal{F} - s.\mathcal{F}$ = all the flows that joined in s.

A. Rate of convergence to efficient and fair allocation in the equilibrium scenario

The equilibrium scenario is when no flows join or leave. In this case, the fair-share is B/n, where n is the (static) number of current flows. and the proportional fair-share is $p_f B$. We establish that our algorithm computes the fair-share exactly in this case, under the assumption that the flows all additively increase and multiplicatively decrease at the same rate.

Theorem 1: Let F be the set of current flows in some global state s, and assume that from s onwards, no flows join or leave. Let $n = |F|$, i.e., the number of flows. Then, within at most two congestion cycles of s, $\wedge_f (fair_f = B/n)$ holds, and continues to hold forever after.

Proof: Consider an arbitrary computation π such that there are n current flows, and no flows join or leave (along π). Hence, we can express a global state s along π as $s.\mathcal{F} = \{1, \ldots, n\}$, and $s.\mathcal{S}(f)$ is a flow-state for flow $f, 1 \leq f \leq n$.

Let s be an arbitrary global state along π in which $\wedge_{f \in F} mode_f = fair\_share\_unknown$ holds, and let s occur in cycle $c - 1$. Let t be the global state at the beginning of cycle c. (i.e., after the "round" in which $\wedge_f con_f$ becomes true terminates). Let u be the global state at the end of cycle c, and v be the global state at the beginning of cycle $c+1$. In the following proof, each assertion follows from inspection of Figure 1 and from earlier assertions. Then,

$$\bigwedge_{f \in F} (cc_f \wedge \neg con_f \wedge bc_f = a_f(b_c))$$

holds in all states from t up to but not including u. Hence

$$\bigwedge_{f \in F} (cc_f \wedge con_f \wedge bc_f = a_f(b_c))$$

holds in u. Hence

$$\bigwedge_{f \in F} \quad (\quad \begin{array}{l} mode_f = fair\_share\_known \wedge \\ fair_f = (a_f(e_c) - a_f(b_c))/\beta \wedge \\ a_f = fair_f * (1 - \epsilon) \quad) \end{array}$$

holds in v. Thus, $v(fair_f)$ is equal to the fair-share B/n, by the analysis in Section II-D.1. So,

$$\bigwedge_{f \in F} \quad (\quad \begin{array}{l} mode_f = fair\_share\_known \wedge \\ fair_f = B/n \wedge \\ a_f = (B/n) * (1 - \epsilon) \quad) \end{array}$$

holds in v. So,

$$\bigwedge_{f,g \in F} (a_f = a_g)$$

holds in v. Thus,

$$\bigwedge_{f,g \in F} (a_f = a_g)$$

holds in all states from v until before the next congestion cycle.

Thus, the next congestion cycle begins when each allocation a_f has value B/n, since the allocations are all equal. Subsequently, each allocation is reduced by a factor of $(1-\epsilon)$, with the allocations still remaining equal. The pattern persists through all subsequent congestion cycles. a_f then changes, in a linear manner, from $(B/n)*(1-\epsilon)$ just after congestion occurs, to B/n just before the next occurrence of congestion. ■

B. Rate of convergence to efficient and proportional fair allocation in the equilibrium scenario

Theorem 2: Let F be the set of current flows in some global state s, and assume that from s onwards, no flows join or leave. Let $p_f = \alpha_f / \sum_{g \in F} \alpha_g$. Then, within at most two congestion cycles of s, $\wedge_{f \in F}(fair_f = p_f B)$ holds, and continues to hold forever after.

Proof: Let f be any flow id in F. As in the proof of Theorem 1, we establish that within two cycles, $fair_f = (a_f(e_c) - a_f(b_c))/\beta$ is true, and continues to hold forever. By the results of Section II-D.2, we have that $fair_f = p_f B$ is true within two cycles of s, and continues to hold forever thereafter. ■

C. Transient Behavior

Theorem 3: 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 with factor β. Then, $\wedge_{f \in \mathcal{F}(b_c) - \ell_c}(fair_f = p_f B + (p_f/\beta)[A(\ell_c, b_c) - A(j_c, e_c)])$ holds at the end of cycle c.

Proof: Let f be any flow id in $\mathcal{F}(b_c) - \ell_c$. As in the proof of Theorem 1, we can show that $fair_f = (a_f(e_c) - a_f(b_c))/\beta$ holds at the end of cycle c. By the results of Section II-D.3, we have that $fair_f = p_f B + (p_f/\beta)[A(\ell_c, b_c) - A(j_c, e_c)]$ holds at the end of cycle c. ■

D. Efficiency at Equilibrium

Theorem 4: Assume that there are n current flows, and that no flows join or leave. Then, after at most two congestion cycles, the efficiency becomes $1-(\epsilon/2)$ and remains at that value forever after.

Proof: As noted in Section III-A after the first two congestion cycles, a_f increases linearly from $(B/n)*(1-\epsilon)$ just after congestion occurs, to B/n just before the next congestion occurs. If the time between these two points is T, then the total data transmitted per flow during the congestion cycle delineated by these two points (call it cycle c) is $(B/n)*(1-\epsilon)*T + (B/n)*\epsilon*T/2$. The maximum possible data transmitted per flow during cycle c is $(B/n)*T$. Thus, the efficiency in cycle c is

$$\frac{(B/n)*(1-\epsilon)*T + (B/n)*\epsilon*T/2}{(B/n)*T}.$$

This simplifies to $[(1-\epsilon) + \epsilon/2]$, i.e, $1-(\epsilon/2)$. Since the flow pattern in cycle c repeats indefinitely, this value for the efficiency holds forever after. ■

IV. Related work

The impact of AIMD has been recently discussed mainly from two perspectives. First, from the perspective of the improvements of the original AIMD. Towards this end, Yang and Lam [9] discuss a control system which extends the system of Chiu further towards asynchronous feedback. Lahanas and Tsaoussidis in [10] propose a modification to increase both fairness and efficiency, namely AIMD-FC. They prove the properties of the modified algorithm algebraically, and show experimentally significant improvements with TCP. Second, from the perspective of the parameterization of a general algorithm which exploits the tradeoffs of smoothness and responsiveness but does not disturb much the performance of the traditional TCP scheme. That is, such modifications, which are called TCP-Friendly, attempt to achieve similar efficiency with TCP, trading responsiveness for smoothness. The question of 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. Some recent versions of AIMD-based algorithms that take advantage of this property are [11], [9], [12], [13]. More precisely, it has been observed that streaming applications could benefit from modest oscillations since these reflect the smoothness of adjusting the transmission rate backwards. Such protocols are characterized as TCP-Friendly because they consume the same amount of bandwidth as TCP($1, \frac{1}{2}$) does [4]. Theoretically, and in the context of Figure 2, these algorithms try to push the Efficiency Line closer to the Bandwidth Limit Line at the expense of the rate of convergence to fairness. For example, the GAIMD [9] algorithm with $\beta = \frac{1}{8}$, converges to fairness in $O(B \log_{1.14} B)$ steps, where B is the link bandwidth; the SIMD [13] with $\beta = 1/16$ converges in $O(B \log_{1.06} B)$ steps; the SQRT [12] algorithm with

[1] Both statements have been made initially in [1].

$\beta = \sqrt{1/B}$, converges in $O(B^2)$; so does the IIAD [12]. The performance of AIMD algorithm (with parameters $\alpha = 1$ and $\beta = \frac{1}{2}$) and in general of TCP is studied in [14], [15]. The efficiency of the AIMD algorithm is described in [14] by the formula: $\frac{3}{4}\, B\, (\frac{MSS}{RTT})$ where B is the link bandwidth, *MSS* is the TCP packet size and *RTT* is the Round Trip Time. The analysis suggests a 75% efficiency of the protocol when the system is in equilibrium. Several, congestion control algorithms along with a cost analysis of their capacity to discover the available bandwidth, have been recently presented in [16].

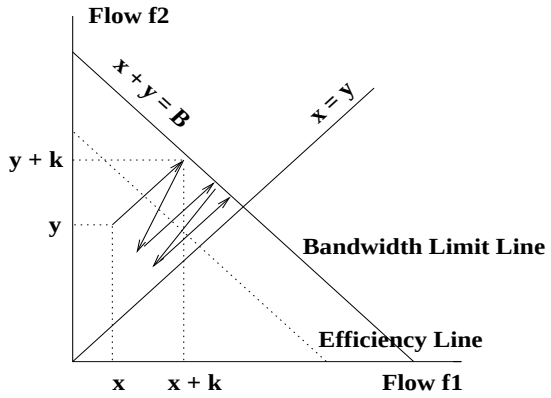

Fig. 2. Vectorial representation of two-fbw convergence to fairness. Figure is based on [1].

V. Experimental Methodology and Testbed

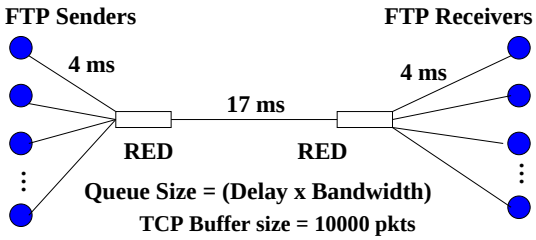

Fig. 3. Multiple fbws experimental set-up for AIMD evaluation.

We have incorporated our algorithm into TCP [3] and have validated its performance on NS-2 [17]. TCP controls the sending rate by a parameter called *congestion window* [2]. When resources are available TCP increases the congestion window by *one* Maximum Segment Size (MSS); upon congestion and in the presence of three duplicate acknowledgments, TCP multiplies the congestion window by a factor of 1/2 (this TCP is also known as TCP(1,$\frac{1}{2}$)). Recall that in the absence of errors the average long term efficiency of the AIMD mechanism of TCP is 75% [14], [15].

The TCP version of choice in our experiments was TCP-SACK [18]. Due to its Fast Recovery and its capability for multiple retransmissions within one RTT, this version matches better the assumptions of our theoretical work. However, there is an additional component in TCP's congestion control, namely the timeout mechanism; and there is an additional component in the initial window expansion phase, namely, the Slow Start mechanism.

In our experiments, multiple flows share a high-bandwidth bottleneck link (see Figure 3); the fair-share (the Delay×Bandwidth share per flow) was set relatively high in order to provide the environment for the algorithms to exploit their potential. 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.

We evaluate three distinct scenarios: Our first scenario is characterized by stationarity in terms of the number of participating flows. The scenario matches well the theoretical assumptions. 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 ought to demonstrate capabilities to consume the available resource fast. Both our second and third experiments aim at alleviating reasonable concerns regarding the algorithm's behavior in dynamic (and hence more realistic) environments. We note that our experiments do not cover the whole spectrum of experimental evaluation, which can be a subject of study in its own right; we provide here substantial evidence on the algorithm's practical impact, along with our theoretical perspective. Further experimental studies may be driven by specific network and flow characteristics, or protocol, application and device properties.

A TCP flow runs at each end node and an FTP application generates the traffic for each source. The task of the application is to send 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 the 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 [19]: $F(x) = (\sum x_i)^2/n(\sum x_i^2)$, where x_i is the goodput achieved by each flow.

A. Stationary Environment

We present results with $\epsilon = 1/2$ (figures 4, 5), $\epsilon = 1/3$ (figures 6, 7), and $\epsilon = 1/8$ (figures 8, 9). As we pointed out already, 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 5). We demonstrate notable improvement in efficiency when ϵ is smaller. In figure 6 where ϵ is $1/3$ goodput is increased up to 5% and in figure 8 where ϵ is $1/8$ goodput is improved up to 10%. The corresponding improvements in fairness can be seen in figures 7 and 9 respectively.

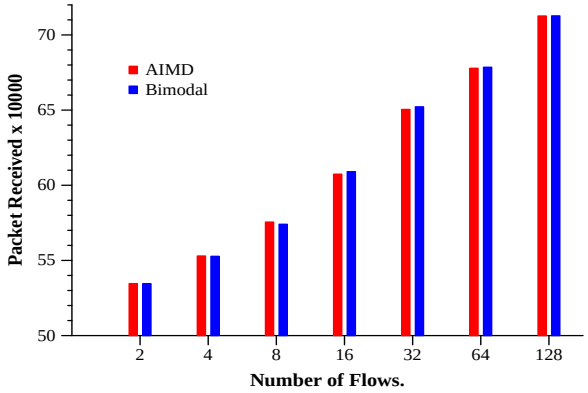

Fig. 4. Goodput Performance of TCP with AIMD and Bimodal congestion control algorithm on a 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{2}, \beta = \frac{1}{2}$.

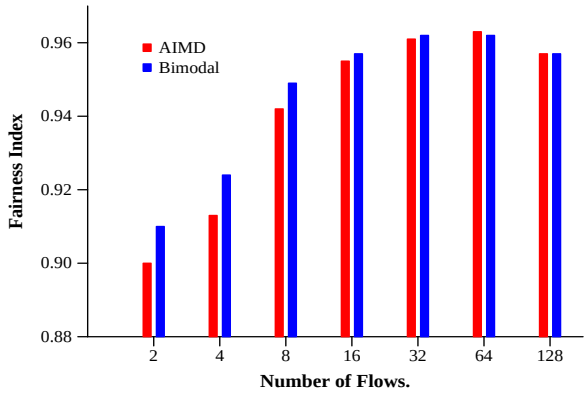

Fig. 5. Fairness with the 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

B. Graduated Contention Increase

A reasonable question regarding the potential of bimodal congestion avoidance and control may arise from the dynamics of Internet applications. Occasionally, contention may increase 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 de-

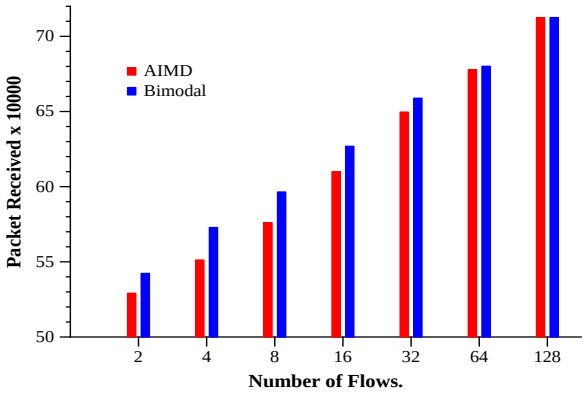

Fig. 6. Goodput Performance of TCP with AIMD and Bimodal congestion control algorithm on a 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{3}, \beta = \frac{1}{2}$.

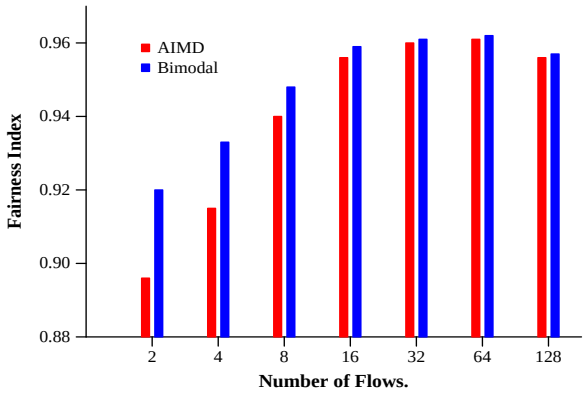

Fig. 7. Fairness with the 100Mbps link and a RED Gateway. $\epsilon = \frac{1}{3}$, $\beta = \frac{1}{2}$.

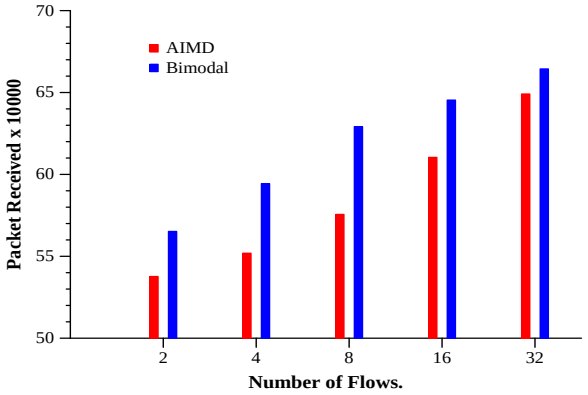

Fig. 8. Goodput Performance of TCP with AIMD and Bimodal congestion control algorithms on a 100Mbps link and RED Gateway. $\epsilon = \frac{1}{8}, \beta = \frac{1}{2}$.

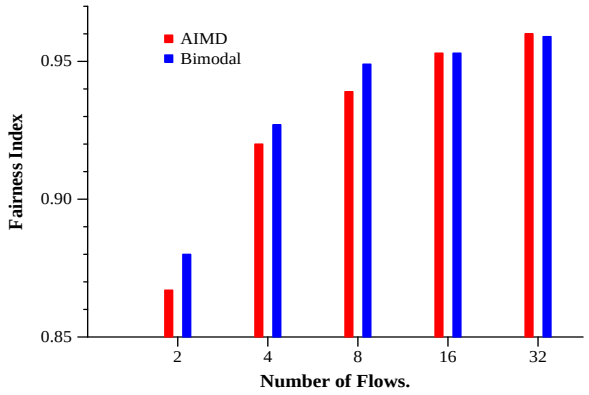

Fig. 9. Fairness with the 100Mbps link and RED Gateway. $\epsilon = \frac{1}{8}$, $\beta = \frac{1}{2}$.

tected [2] multiplicative decrease is drastic, making space for the new flows as in standard AIMD. Results of the experiments are presented in figures 10 and 11.

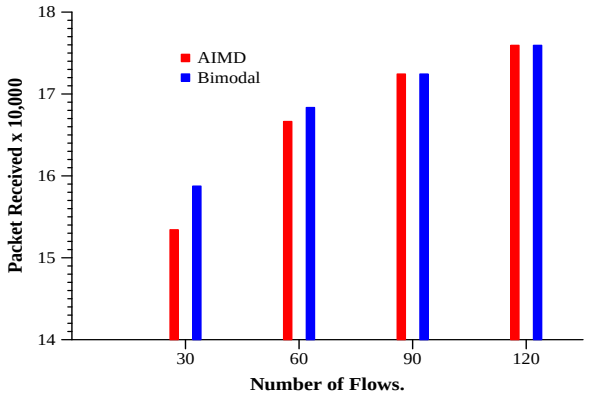

Fig. 10. Goodput Performance of TCP with AIMD and Bimodal congestion control algorithm on a 100Mbps link and a RED Gateway. The number of flows is increased by 30 every 15 seconds. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

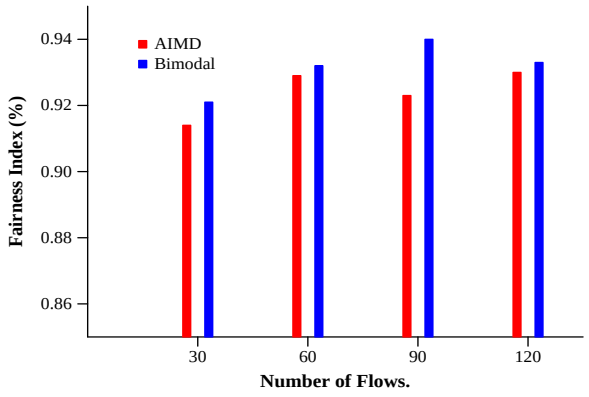

Fig. 11. Fairness with the 100Mbps link and a RED Gateway. The number of flows is increased by 30 every 15 seconds.

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

C. Graduated Bandwidth Increase

We consider a deterministic scenario where 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 12 and 13 respectively.

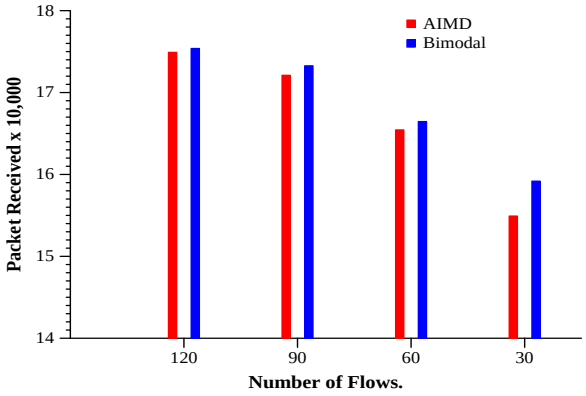

Fig. 12. Goodput Performance of TCP with AIMD and Bimodal congestion control algorithm on a 100Mbps link and a RED Gateway. The number of flows is halved every 15 seconds. $\epsilon = \frac{1}{2}$, $\beta = \frac{1}{2}$.

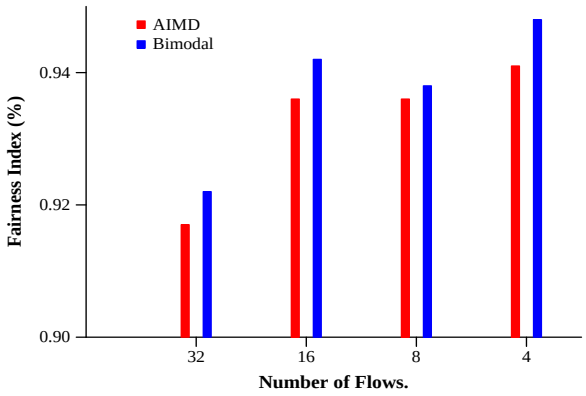

Fig. 13. Fairness with the 100Mbps link and a RED Gateway. The number of flows is halved every 15 seconds.

VI. Discussion

Our algorithm is executed only by the flow sender. In particular, although the sender needs to maintain the current flow state and mode, the router need not maintain any extra information. Thus, our algorithm works with standard IP routers, since it relies only on a 1 bit feedback in the form of congestion/no congestion, which in practice is generated by a packet acknowledgment or packet drop. Requiring the sender to maintain a small amount of information (two window sizes and three bits) per flow

is clearly not an impediment, especially in comparison to the amount of data a typical sender generates and transmits. Furthermore, the per flow state is maintained in a distributed fashion by the senders, and is not concentrated at the performance-critical routers, where the aggregation of per-flow state for, e.g., thousands of flows, would cause severe performance degradation of the routers.

We distinguish our algorithm from the class of TCP-friendly algorithms. TCP friendly algorithms favor smoothness at the cost of fairness. Our algorithm calculates fair share explicitly, and so fairness is not compromised in our approach. Furthermore, since our algorithm restricts its flows to use only their fair share, it can be used in conjunction with any other transport protocols (e.g., standard AIMD) without monopolizing for itself most of the available bandwidth. This is in contrast with the TCP-friendly protocols, which attempt to grab all available bandwidth. Of course, our algorithm will not work well in conjunction with other protocols that aggressively (and unfairly) grab a disproportionate share of the bandwidth for their flows.

VII. Conclusions and Future Work

We have presented a congestion control algorithm that explicitly calculates the fair share and converges in two congestion cycles to the fair share. Our algorithm clearly outperforms standard AIMD.

One issue with our algorithm is that the more the gain we have in goodput (i.e., by using a smaller ϵ) the less the free space left for incoming flows when contention increases. Although in any case the system will indeed converge in two cycles, leaving no free space in order to maximize bandwidth utilization will have an impact on 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. We demonstrated, however, that even with the most conservative settings of ϵ, Bimodal Congestion Avoidance and Control can achieve up to 10% gain in goodput and fairness. Of course, the gain is proportional to the fair-share, i.e., when the fair-share is larger, the gain is larger, which makes our approach particularly suitable for high-speed networks.

Future work also includes modifying our algorithm for the asynchronous scenario by integrating the RTT into the fair-share calculation. For example, we could increase the bandwidth allocation of a flow when the RTT of its packets decreases, and decrease the allocation when the RTT increases. Also, in the asynchronous scenario, the effect of delays introduced by buffering becomes important. We will investigate how to take such delays into account in our algorithm. We will also investigate an integration of the ideas presented here with a receiver-oriented feedback approach, e.g., as embodied in TCP-real [20].

References

[1] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," *Journal of Computer Networks and ISDN*, vol. 17, no. 1, pp. 1–14, June 1989.

[2] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of the ACM SIGCOMM '88*, August 1988, pp. 314–329.

[3] J. Postel, "Transmission Control Protocol," *RFC 793*, September 1981.

[4] 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.

[5] N. A. Lynch, *Distributed Algorithms*, Morgan-Kaufmann, San Francisco, California, USA, 1996.

[6] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi, *Reasoning about Knowledge*, The MIT Press, Cambridge, Mass., 1995.

[7] Joseph Y. Halpern and Yoram Moses, "Knowledge and common knowledge in a distributed environment," *Journal of the ACM*, vol. 37, no. 3, pp. 549–587, 1990.

[8] P. C. Attie and N.A. Lynch, "Dynamic input/output automata: a formal model for dynamic systems," in *CONCUR'01: 12th International Conference on Concurrency Theory*. Aug. 2001, LNCS, Springer-Verlag.

[9] Y. Yang and S. Lam, "General AIMD Congestion Control," in *Proceedings of the IEEE International Conference on Network Protocols*, November 2000.

[10] A. Lahanas and V. Tsaoussidis, "Additive Increase Multiplicative Decrease - Fast Convergence (AIMD-FC)," in *Proceedings of the Networks 2002, Atlanta, Georgia*, August 2002.

[11] 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.

[12] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," in *Proceedings of the IEEE INFOCOM'01*, 2001.

[13] 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.

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

[15] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, August 1999.

[16] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, "Optimization Problems in Congestion Control," in *IEEE Symposium on Foundations of Computer Science*, November 2000, pp. 66–74.

[17] "The Network Simulator - NS-2," Tech. Rep., Web Page: http://www.isi.edu/nsnam/ns/.

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

[19] R. Jain, Dah Ming Chiu, and H. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems," Tech. Rep. DEC-TR-301, Digital Equipment Corporation, 1984.

[20] V. Tsaoussidis and C. Zhang, "TCP-Real: Receiver-oriented Congestion Control," *The Journal of Computer Networks*, 2002.