Approximate Fairness through Differential Dropping

Rong Pan
Stanford University
rong@stanford.edu

Lee Breslau
AT&T Research
breslau@research.att.com

Balaji Prabhakar
Stanford University
balaji@stanford.edu

Scott Shenker
ICSI
shenker@icsi.berkeley.edu

ABSTRACT

Many researchers have argued that the Internet architecture would be more robust and more accommodating of heterogeneity if routers allocated bandwidth fairly. However, most of the mechanisms proposed to accomplish this, such as Fair Queueing [16, 6] and its many variants [2, 23, 15], involve complicated packet scheduling algorithms. These algorithms, while increasingly common in router designs, may not be inexpensively implementable at extremely high speeds; thus, finding more easily implementable variants of such algorithms may be of significant practical value. This paper proposes an algorithm called *Approximate Fair Dropping* (AFD), which bases its dropping decisions on the recent history of packet arrivals. AFD retains a simple forwarding path and requires an amount of additional state that is small compared to current packet buffers. Simulation results, which we describe here, suggest that the design provides a reasonable degree of fairness in a wide variety of operating conditions. The performance of our approach is aided by the fact that the vast majority of Internet flows are slow but the fast flows send the bulk of the bits. This allows a small sample of recent history to provide accurate rate estimates of the fast flows.

1. INTRODUCTION

Since the pioneering observations of Nagle [16], many researchers have argued that the Internet architecture would be more robust (in the face of ill-behaved flows) and more accommodating of heterogeneity (by no longer requiring adherence to a single congestion control algorithm) if routers allocated bandwidth fairly. This viewpoint is not universally shared,[1] but even among adherents of this approach the question of feasibility has been a major concern. This is because most of the proposed mechanisms, such as Fair Queueing [16, 6] and its many variants [2, 23, 15], involve complicated packet scheduling algorithms. These algorithms, while increasingly common in router designs, may not be inexpensively implementable at extremely high speeds; thus, finding more easily implementable variants of such algorithms may be of significant practical value and several such algorithms have been proposed (e.g., [13, 24]).

This paper focuses on the design of a router mechanism that combines approximately fair bandwidth allocations *to all users* with relatively low implementation complexity. We have three basic requirements of our design. First, it should achieve reasonably fair bandwidth allocations, and by "fair" we mean the max-min definition of fairness [6]. We make no pretense at being able to match the packet-by-packet fairness of Fair Queueing or other schemes that use intricate packet scheduling algorithms to ensure that perfect fairness is achieved on very short time scales. Instead, we aim for approximate fairness over longer time scales, on the order of several roundtrip times.

Second, we require that the design be easily amenable to high speed implementations. To this end we limit ourselves to FIFO packet scheduling algorithms with probabilistic drop-on-arrival; as in RED [8] and FRED [13], when a packet arrives either the packet is dropped or placed on a FIFO queue. The dropping decisions must be simple with $O(1)$ complexity (that is, the complexity must not increase with the number of flows or packets). The amount of state required to make these dropping decisions must be small; the point of reference we choose to define "small" is the amount of memory which is already devoted to the packet buffers.

Third, the algorithm must employ some form of active queue management (AQM). It need not mimic RED or any other form of AQM precisely, but it should embody AQM's fundamental principles of responding early (and gently) to congestion while absorbing small bursts of traffic.[2]

[1] We discuss other viewpoints in Section 7.

[2] There is a fourth requirement that, for lack of space, we do not discuss in this paper; this is the requirement that the scheme be able to *punish* unresponsive flows. That is, the scheme should not just allocate bandwidth fairly, but should be able to shut down (by dropping all their packets) flows that incur high drop rates for long periods of time. The rationale for this is persuasively described in [10]; [24] discusses the role of fair bandwidth allocation in implementing this. We are easily able to augment our design to meet this goal and, as we see in Section 5, one of the designs we present here already (but unintentionally!) achieves this goal.

To achieve these goals, we propose an algorithm called Approximate Fair Dropping (AFD). AFD uses a FIFO queue with probabilistic drop-on-arrival. These probabilistic dropping decisions are based not only on the past measurements of the queue size but also on the recent history of a flow.[3] AFD uses the history to estimate the flow's current sending rate, and then drops with a probability designed to limit that flow's throughput to the fair share. Thus, dropping is not applied uniformly across flows (as in RED) but is applied differentially based on an estimate of the flow's current sending rate. The exact form of differential dropping is chosen to approximate fair bandwidth allocations.

Our design might initially appear to be seriously misguided. The state required to keep enough history to accurately estimate each flow's rate is quite large, and grows linearly with the number of flows. This is clearly not a feasible approach. However, note that our design only needs to estimate the rates of flows whose packets are likely to be dropped. Thus, we need only keep enough state to estimate rates when those rates are comparable to (or larger than) the fair share rate. This is a crucial difference.

It is well known that the distribution of the sizes of flows (the total number of bytes transferred) has a long tail, and that most flows are small – the mice – but a large fraction of the bytes are sent by large flows – the elephants.[4] As we show using trace data in Section 6, and has been observed in [14], the distribution of flow *rates* has a similar property. While perhaps not as long-tailed as the size distribution, initial evidence suggests that the flow rate distribution (measured on the time scale of a second) is long-tailed. Most flows are *slow* (in bits/sec) – we will call them *turtles* – but most of the bytes are sent by fast flows – we will call them *cheetahs*.

In AFD, we only need state for the fast flows and can ignore the slow flows because they won't be dropped; the amount of state required is roughly proportional to the number of fast flows, which is manageable. However, the challenge is how to keep state only on the fast flows when one doesn't know, in advance, which flows are fast. A record of the very recent packet arrivals contains mostly packets from the fast flows (because they send most of the bytes), while most of the slow flows won't show up in the recent history. This is the approach we take.

While other algorithms with similar goals and building blocks have been proposed, our work is unique in two important ways. First, AFD fills a void in the space of performance and deployability. In particular, simulation results show that it has better fairness properties than other algorithms, such as FRED, that have similar complexity and deployability. At the same time, AFD does not have the deployment obstacles of CSFQ (and its variants) while providing similar performance in the fairness domain. We are not aware of any other algorithm that is both as deployable and as fair as AFD.

The second contribution of AFD lies in its technical novelty. While several components of its design are present in existing algorithms, it is the first algorithm to leverage the distribution of flow rates in conjunction with per-packet sampling in order to address the scalability requirements of per-flow state. By basing the design of AFD on this characteristic of network traffic, we have achieved an important advance in this class of algorithms. Furthermore, we demonstrate techniques that reduce AFD's complexity relative to other algorithms while not sacrificing performance.

This paper has 7 sections. We describe a conceptual version of our design in Section 2, and then use analysis to develop guidelines for the various parameters in Section 3. We then present the practical design in Section 4. We evaluate this design using simulation in Section 5, and use trace data and other measurements to estimate the state requirements for the design in Section 6. We conclude in Section 7 with a discussion of related work.

2. CONCEPTUAL DESIGN

We first present a high level conceptual design of AFD. Consider a link of speed C shared by n flows, each sending at rate r_i. Assume, for convenience, that all packets are the same size P. The total load on the link is $R = \sum_i r_i$. The fair share r_{fair} is given by the solution to $C = \sum_i \min(r_i, r_{fair})$. We can use differential dropping to accomplish our goal of fair bandwidth allocations if we use dropping probabilities for each flow, d_i, given by the relation $d_i = (1 - \frac{r_{fair}}{r_i})_+$.[5] The resulting throughput of each flow is bounded by the fair share: $r_i(1 - d_i) = \max(r_i, r_{fair})$.

The key design question is: how can we estimate r_i and r_{fair}. To estimate r_i we keep a *shadow buffer* of b recent packet headers. When a packet arrives, with probability $\frac{1}{s}$, where s is the sampling interval, we copy its header and insert it into the shadow buffer; thus, we sample roughly 1 in s packets. When a packet is inserted into the shadow buffer, we remove another packet at random to keep the total number of packets at b.[6] Thus, at all times the shadow buffer has a record of b recent packet arrivals. Note that the shadow buffer contains copies of the packet headers, and the insertion and deletion of packets from the shadow buffer is parallel to the main forwarding of packets. The shadow buffer is used to guide the dropping decisions in the following manner. When a packet (from, say, flow i) arrives (regardless of whether its header is copied into the packet buffer) we compute the number of packets from that flow in the shadow buffer; we call this the number of *matches*, m_i. We then estimate the rate of that flow to be $r_i^{est} = R\frac{m_i}{b}$ (recall that R is the aggregate arrival rate).

To avoid having to scan the shadow buffer each time a packet

[3]In this paper, we define a flow as a stream of packets with the same source and destination addresses. However, one could use any definition of flow discernible from an IP packet header. The particular choice is orthogonal to our work. Further, we do not address the impact of NAT boxes or hosts with multiple IP addresses on flow identification; these issues will affect any algorithm that attempts to allocate bandwidth on a per-flow basis.

[4]As we discuss later in Section 6, we use the term "long-tail" to describe distributions that decay slower than exponentially.

[5]We use the notation that $z_+ = \max(0, z)$.

[6]Random removal avoids synchronization problems. We could also remove packets in a FIFO manner; it turns out that this does not affect performance greatly.

arrives, we keep a table of the flows and their current packet counts; we call this the *flow table*. A hash table is one natural data structure for the flow table that has $O(1)$ lookup time. Each time a packet is inserted into the shadow buffer, the appropriate counter in the flow table is incremented, each time a packet is removed the appropriate count is decremented, and every time a packet arrives the match count is looked up in the flow table. Insertions and deletions can be somewhat slow (because we need only insert and delete every s packets) but the lookup has to done at linespeed (because it is done on every packet arrival).

Estimating the rate of a single flow requires only the recent activity of that flow and so is fairly straightforward. Estimating r_{fair} is more complicated because the definition of r_{fair} depends on all the r_i. We use an approach borrowed from [24]. Upon each packet arrival we apply the dropping probability $d_i = (1 - \frac{r_{fair}}{r_i^{est}})_+$, which can be rewritten as $d_i = (1 - \frac{m_{fair}}{m_i})_+$ with $m_{fair} = b\frac{r_{fair}}{R}$. Note that if we vary m_{fair} the total throughput, $\sum_i r_i(1 - d_i)$, varies from R when $m_{fair} = \infty$, to C when $m_{fair} = b\frac{r_{fair}}{R}$, to 0 when $m_{fair} = 0$. Thus, we can approximate r_{fair} by varying m_{fair} so that the link is fully utilized but not overloaded.

In our approach, we vary m_{fair} to ensure that the average queue length stabilizes around the target value of q_{target}. We set q_{target} to be $0.5(min_{th}+max_{th})$, where min_{th} and max_{th} are the threshold values defined in RED, to provide sufficient packet buffering while retaining low queueing delays. We borrow the AQM approach described in [12], which employs a proportional-integral controller. Since this form of AQM is very effective at keeping the links fully utilized, the resulting values of $m_{fair}\frac{R}{b}$ will be a good approximation to r_{fair}.

We will defer a more complete comparison with other algorithms until Section 7, but we note that AFD builds upon features present in earlier algorithms, such as CSFQ [24], CHOKe [18] and FRED [13]. In addition, it is similar to the concurrently-developed RED-PD [14].[7] The result is an overall design that attempts to approximate fairness with reasonable overhead. FRED uses the number of packets from each flow in the packet buffer to guide its dropping decisions; thus, our design can be seen as an extension of FRED to use more history and a different dropping function (which is the same as used in CSFQ).

Does AFD achieve the goals we laid out in the Introduction? AFD's probabilistic drop-on-arrival can be implemented with a very simple forwarding path. The dropping decision has complexity $O(1)$ and involves few computations. In addition, AFD incorporates active queue management. Fairness and feasibility (in terms of the state required) are the two remaining questions. If the rate estimates are accurate, we expect AFD to allocate bandwidth fairly. While we delay the bulk of our simulation results, and all of the simulation details, until Section 5, we now show initial evidence that this expected fairness is actually achieved. Figure 1 shows the result of 50 CBR sources sending at five different rates (some above the fair share, some below) over a single link. The bandwidth allocations are quite fair; flows

[7]While the two schemes have some similarities, they are targeted at somewhat different goals. See Section 7.

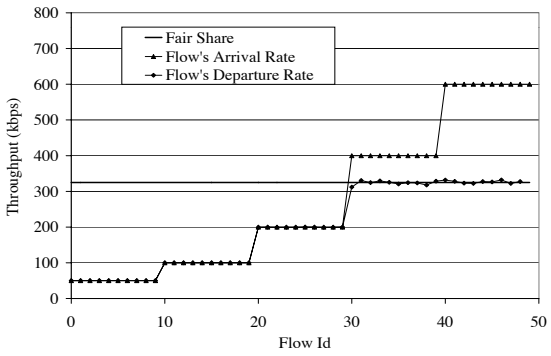


Figure 1: Offered Load and Throughput for 50 CBR Flows under AFD

sending below the fair share incur no drops, and flows sending above the fair share have their excess packets dropped.

Fairness depends on the accuracy of the rate estimations, and the rate estimations depend on the size of the shadow buffer. Without sufficient history the rate estimations will fluctuate wildly and lead to quite unpredictable (and unfair) results. Rate estimation improves with larger b (the size of the shadow buffer) and, to some extent, with larger s (because s increases the time interval represented by the contents of the shadow buffer). The question is how large do b and s have to be in order to achieve reasonable fairness. This is crucial because we want the state required for this algorithm to be small (at least compared to the packet buffers). The viability of our scheme comes down to whether fairness can be achieved without too much state, and we explore that issue in the next few sections.

In Section 3 we develop three guidelines for how to set b and s. After describing a more practical (as opposed to the conceptual version just presented) version of AFD in Section 4, we evaluate the fairness that results from such parameter settings in Section 5. Finally, in Section 6 we estimate the state requirements of AFD when using the parameter-setting guidelines.

3. SETTING THE BASIC PARAMETERS

In this conceptual model, there are two basic parameters that we need to set: the shadow buffer size b and sampling interval s. Our goal is to achieve a reasonable degree of fairness, and in this section we develop three *guidelines* on what values of b and s achieve that goal. To do so, we consider three scenarios and analyze them using very simple models. These models are unrealistic but our hope is that the resulting guidelines provide useful order-of-magnitude estimates for the parameters of interest.

3.1 Static Scenario

We first consider the static case of n sources sharing a single link of bandwidth C. All flows are Poisson sources sending at a constant rate r_i and all packets are the same size. The overall drop rate D is given by $D = \frac{(R-C)_+}{R}$ where, as above, $R = \sum_i r_i$. Let $P_i(m)$ be the probability that when a packet from flow i arrives, there are m packets from flow i in the

shadow buffer. In our simple model,

$$P_i(m) = \binom{b}{m}(1-\frac{r_i}{R})^{b-m}(\frac{r_i}{R})^m$$

For the purposes of this model, we assume that m_{fair} is fixed to be the appropriate value: $m_{fair} = b\frac{r_{fair}}{R}$. We denote the *ideal* dropping rate by $\tilde{d}_i$: $\tilde{d}_i = (1-\frac{r_{fair}}{r_i})_+$. $d_i(b)$ is the average dropping rate of the i'th flow for a given size of shadow buffer:

$$d_i(b) = \sum_{m=0}^{b} P_i(m)(1-\frac{m_{fair}}{m})_+$$

The relative error in the throughput of the i'th flow is

$$\Delta_i(b) = |\frac{r_i(1-d_i(b)) - r_i(1-\tilde{d}_i)}{r_i(1-\tilde{d}_i)}| = |\frac{\tilde{d}_i - d_i(b)}{1-\tilde{d}_i}|$$

Note that the sampling interval s drops out of all of these quantities. This is not surprising given that a random sampling of a Poisson process remains a Poisson process.

Our goal is to choose the size of the shadow buffer so that we achieve some reasonable degree of fairness. We seek to determine how large the shadow buffer must be so that the maximal error, $\max_i \Delta_i(b)$, is below some target error tolerance E. Analysis and numerical calculations (which we do not have space for here) indicate the maximal error occurs when $r_i \approx r_{fair}$. Consider a flow with $r_i = r_{fair}$; we can approximate the relative error, call it Δ, in the limit of large b for this flow as follows (noting that the ideal dropping rate for this flow is zero, $\tilde{d}_i = 0$, and so $\Delta(b) = d_i(b)$). Letting $\alpha = \frac{r_{fair}}{R}$, we have:

$$\Delta(b) = \sum_{m=0}^{b}\binom{b}{m}(1-\alpha)^{b-m}\alpha^m(1-\frac{b\alpha}{m})_+$$

$$= \sum_{m=\lceil b\alpha\rceil}^{b}\binom{b}{m}(1-\alpha)^{b-m}\alpha^m(1-\frac{b\alpha}{m})$$

For sufficiently large b, we can approximate the binomial form as a normal distribution with average αb and variance $b\alpha(1-\alpha)$. Using an integral formulation, we find:

$$\Delta(b) \approx \frac{1}{\sqrt{2\pi b\alpha(1-\alpha)}}\int_{b\alpha}^{\infty} dx\ e^{-\frac{(x-b\alpha)^2}{2b\alpha(1-\alpha)}}(1-\frac{b\alpha}{x})_+$$

$$< \frac{1}{\sqrt{2\pi b\alpha(1-\alpha)}}\int_0^{\infty} dy\ e^{-\frac{y^2}{2b\alpha(1-\alpha)}}\frac{y}{b\alpha}$$

$$= \sqrt{\frac{2(1-\alpha)}{\pi b\alpha}}\int_0^{\infty} dz\ ze^{-z^2} \approx .40\sqrt{\frac{(1-\alpha)}{b\alpha}}$$

Thus, we can bound the asymptotic expression for the relative error from above by the formula

$$\Delta(b) \le .40(b\alpha)^{-0.5}$$

A choice of b that will always meet the error tolerance of E (in this asymptotic limit) is:

$$b\frac{r_{fair}}{R} > (\frac{E}{0.40})^{-2}$$

Note that $b\frac{r_{fair}}{R} = m_{fair}$ where, as defined above, m_{fair} is the expected number of shadow buffer matches for a flow with $r_i = r_{fair}$. One can meet the error tolerance merely by making the shadow buffer big enough to make sure that $m_{fair} > (\frac{E}{0.40})^{-2}$. For our purposes, we choose $E \approx .15$. This might seem quite unambitious, but recall that we are calculating an upper bound on the worst-case error (that is, of flows sending at a constant rate right at the fair share); the errors for rates well above and well below the fair share are significantly smaller. Setting $E = 0.15$ results in $m_{fair} \approx 10$. This is our first guideline.

Guideline 1: $b \gtrapprox 10\frac{R}{r_{fair}}$ or, equivalently, $m_{fair} \gtrapprox 10$

3.2 Dynamic Scenario

Even if routers allocate bandwidth fairly on the time scale over which they measure usage, flows that respond slowly when excess bandwidth is available are at a disadvantage when competing with flows that respond more rapidly. We can illustrate this with a simple model involving flows with different roundtrip times (RTTs).

Consider flows competing for bandwidth with differing roundtrip times τ_i. All packets are the same size P. Each flow adjusts its window size w_i in a TCP-like fashion (with all the details of TCP omitted); the window (measured in terms of packets) increases by one in time τ_i when there is no packet loss during that RTT, and the window is halved if there is a packet loss (and we neglect the case of multiple packet losses, or of timeouts, in an RTT). The rates r_i of the flows are roughly $r_i = \frac{w_i P}{\tau_i}$. The router has a shadow buffer of size b and a sampling interval of s; the time interval T over which the shadow buffer measures usage is $T = \frac{bsP}{R}$. We assume that the fair share rate r_{fair} is constant and that the fair share expected number of matches $m_{fair} = b\frac{r_{fair}}{R}$ is roughly 10 (as suggested by guideline 1). Consider a flow i and let m_i denote the average number of its matches in the shadow buffer. Because the fair share is fixed, we can approximate the dynamics of each flow as independent.[8] We consider a series of time slots $t = 1, 2, \ldots$, each of length τ_i. For convenience, we assume that $T = k_i\tau_i$ for some integer k_i. The average number of matches m_i in the shadow buffer at some time t is roughly $m_i(t) = \frac{1}{s}\sum_{j=0}^{k_i-1} w_i(t-j)$. Thus, the window dynamics can be written as:

$$w_i(t+1) = w_i(t) + 1 \quad (m_i \le m_{fair})$$

$$w_i(t+1) = \frac{w_i(t)}{2} \quad (m_i > m_{fair})$$

$$m_i(t) = \frac{1}{s}\sum_{j=0}^{k_i-1} w_i(t-j)$$

[8]The error occurs because the time scale T has R in the denominator; if we replace that by C then the dynamics are completely independent in this simple model.

If we start off with $w_i(t=0)=1$, the first drop for flow i occurs when flow i sends, in a TCP saw-tooth, $m_{fair}+1$ packets, at which point, the TCP window size $w_i(t)$ equals $\frac{sm_{fair}}{k_i}+\frac{k_i}{2}$ (we assume $w_i(t)$ is an integer). Then, the window is halved to $w_i(t)=\frac{sm_{fair}}{2k_i}+\frac{k_i}{4}$. If this halving of the window brings the number of matches down below m_{fair} in the next time period then the increasing process starts again. If not, the window is halved again (and again) until $m_i(t)$ is below m_{fair}. Let's assume that the window is halved only once; in this case $w_i(t)$ becomes a repeating series of values starting at $w_i(t)=\frac{sm_{fair}}{2k_i}+\frac{k_i}{4}$ and increasing by 1 until the value $w_i(t)=\frac{sm_{fair}}{k_i}+\frac{k_i}{2}$ is reached. The average window over this periodic cycle of flow i, $\bar{w}_i$, is:

$$\bar{w}_i=\frac{3}{4}(\frac{sm_{fair}}{k}+\frac{k_i}{2})$$

The average window size that yields the fair share is $\frac{sm_{fair}}{k_i}$. $\bar{w}_i$ reaches that level when $sm_{fair}=\frac{3}{2}k_i^2$. Thus, if we set $m_{fair}=10$ we have, approximately,

$$k_i\approx\sqrt{\frac{20s}{3}}$$

For $s=1$, $k_i\approx 2.6$, and for $s=10$, $k_i\approx 8$.

This model suggests that the size of the shadow buffer should be large enough so that it can average the rate of the longest RTT flow over a TCP saw-tooth. A time scale smaller than that would forget periods during which larger RTTs send less than the fair share; otherwise, it would overact during periods when larger RTTs send more than the fair share.

Note that this model is vastly oversimplified – in particular, the impact of multiple drops per round-trip times and possible time-outs are not modeled and it is assumed that dropping starts as soon as the number of matches is over m_{fair} – and it probably significantly underestimates the hardship imposed on flows with long RTTs. We therefore choose to be conservative and increase (almost double) the ratio to be closer to $k_i\approx 5\sqrt{s}$. Moreover, we choose to accommodate roundtrips on the order of $150msec$. Recalling that $T=k_i\tau$ and $T=\frac{bsP}{R}$, we adopt as our guideline 2 the following inequality:

Guideline 2: $b\sqrt{s}\frac{P}{R}\gtrapprox 750msec$

3.3 Burst Scenario

Consider a flow that, after being quiescent, sends h packets (of size P) back to back. If h is smaller than $m_{fair}s$ then it is likely that none of the incoming packets will be dropped, and if h is significantly larger than this amount then the latter packets will probably be dropped. We want the quantity $m_{fair}s$ to be large enough to absorb the packet bursts typical of TCP and other window-based congestion control algorithms. If we take τ to be a typical RTT, and assume that a flow might send an entire window's worth of packets back-to-back (where the window size corresponds to the fair share), then we would want $m_{fair}sP>r_{fair}\tau$. For a link of capacity C with $R\approx C$, this becomes $bs>\frac{C\tau}{P}$.

We must also ensure that $m_{fair}sP$ is not so large that bursts of that size dominate the packet buffer. If the packet buffers are sized to be $C\tau$ where C is the link speed and τ is some nominal delay (on the order of 250msec) used to size the packet buffers, and as above we assume the dropping rate is low (so $R\approx C$), then we require that $m_{fair}sP\ll C\tau$ or, equivalently, $bs\ll\frac{C^2\tau}{Pr_{fair}}$.

The previous two guidelines presented lower bounds for b and s. Our third guideline yields both upper and lower bounds on the product bs.

Guideline 3: $\frac{C\tau}{P}<bs\ll\frac{C^2\tau}{Pr_{fair}}=\frac{C\tau}{P}\frac{b}{m_{fair}}$

We treat these three guidelines not as hard-and-fast rules but rather as general rules-of-thumb to guide how the parameters are set. There is no need for getting the parameters b and s exactly right. If they are too small, the fairness will be less than ideal; if they are too big, the network may be more vulnerable to bursts. But in both cases, as we see in Section 5, the degradation is gradual. We also note that compliance with these guidelines can be easily checked by operators. For the first guideline, one need only record the historical values of m_{fair} (which, as we describe in Section 4, the algorithm sets automatically) to ensure that it is well above 10. The second guideline is computable from the link capacity. The third can be checked by using the values of m_{fair}.

With these guidelines, we now see how the algorithm works in practice. In the next section we describe two practical versions of the algorithm, one that follows directly from our conceptual model and one that requires less state.

4. PRACTICAL DESIGNS

Our description of the conceptual design glossed over several aspects of the algorithm. First, the algorithm for adjusting m_{fair} is borrowed from [12]. We sample the queue length at a rate f_s; upon each sample we adjust the value of m_{fair} according to the following equation:

$$m_{fair}(t)=m_{fair}(t-1)+\alpha(q(t-1)-q_{target})$$

$$-\beta(q(t)-q_{target})$$

where $q(t)$ is the queue length at the t'th sample, $q(t-1)$ is the queue length at the previous sample, and q_{target} is the target queue size. This procedure not only stabilizes the average queue length around the target value and provides active queue management, but also estimates m_{fair} implicitly. We choose $f_s=160Hz$, which is the same as in [12], and α , β are 1.7 and 1.8, respectively. Our algorithm does not seem terribly sensitive to small changes in the parameter values, but we have not done a systematic study of these parameters.[9]

Second, we will use a hash table for the flow table. In so doing, we need not store entire packet headers in the shadow

[9]Our particular AQM design choice, borrowed from [12], seemed particularly easy to adapt to the AFD algorithm. However, we assume that there are countless ways to incorporate AQM in our differential dropping design.

buffer but merely enough bits to accurately identify the flow in the hash table. In practice we will store a k-bit hash of the source-destination addresses (or whatever flow signature is chosen). Third, the design must be modified to accommodate variable size packets. This entails keeping byte counts in the shadow buffer, and measuring matches in bytes rather than packets in the hash table. When removing packets from the shadow buffer (which are picked randomly), we seek to keep the total number of bytes in the shadow buffer constant (but tolerate some jitter). We call this design AFD-SB, with the SB standing for *shadow buffer*.

AFD-SB stores flow information in two places, the shadow buffer and the flow table. A natural way to reduce the state requirements of AFD would be to eliminate one of these. We can't eliminate the flow table, since it is required to make the packet lookups $O(1)$, so we seek instead to eliminate the shadow buffer. We argue in Section 6 that if a hash table is used, the hash table and shadow buffer require similar amounts of state; eliminating the shadow buffer would thereby reduce the state requirements by half. However, if we used CAM memory for the flow table the state reduction would be far greater (since hash tables need to be sized an order of magnitude larger than the number of flows to avoid collisions).

The question is whether we can perform the correct operations on the flow table without the shadow buffer. Incrementing the flow table upon packet insertions is straightforward. However, decrementing is hard. When we eliminated packets from the shadow buffer, we picked one at random (or in a FIFO order); all packets had equal chance to be eliminated. Choosing a *flow* at random with uniform probability in the flow table is easy, but it isn't clear how to pick a *packet* at random with uniform probability without linearly traversing the flow entries.

To solve this problem, we adopt an approximation. Note that one could choose a packet with uniform probability if one chose flows (from which to eliminate a packet) according to probabilities $\frac{m_i}{\sum_j m_j}$. However, choosing among all n flows in this manner requires a linear search. We propose picking k flows at random, where $k \ll n$ and choosing among them with the same probability function: $\frac{m_i}{\sum_{j \in S} m_j}$ where S is the set of flows chosen at random. We use $k = 5$ in our simulations.[10]

We call this AFD-FT (with the FT standing for *flow table*). We propose AFD-FT as an example of how one could introduce approximations into AFD in order to make it more easily implementable (by requiring less state). However, we expect that there are many other ways to implement the flow table; in particular, each router design will have its own hardware constraints and to maximize implementability it will be important to adapt AFD to each situation. We offer AFD-FT as an existence proof that AFD's performance is reasonably robust to approximations.

[10]To accommodate varying size packets, we decrement the chosen flow by the number of bytes in the arriving packet. If the chosen flow doesn't have enough bytes, we remember the deficit and subtract more upon the next packet.

We now proceed to explore the performance of our two designs – AFD-SB and AFD-FT – through simulation.

5. SIMULATION

5.1 Simulation Preliminaries

We used the *ns* [28] simulator to evaluate our two designs in a variety of scenarios. We compare our designs to two other schemes: RED [8] and FRED [13].[11] RED provides a baseline comparison with an algorithm that makes no attempt to allocate bandwidth fairly.[12] FRED provides a useful guidepost of how much fairer the allocations are when the dropping decisions are informed by the current packet buffer occupancy.[13] AFD keeps a longer history than FRED, and so the relative performance can be seen as an indication of how important this additional state is. FRED is easier to implement than AFD and requires less state, and so our simulation results should be taken as illustrating the fairness vs. complexity/state tradeoffs provided by the two algorithms.[14]

We envision AFD operating in an environment where flows use many forms of congestion control. In our simulations, we used the following different congestion control algorithms:

TCP: We use TCP-sack as provided in the *ns* release. The TCP flows sometimes have different RTTs, or different file sizes, as described below.

AIMD: We modify the increase parameter a and decrease parameter b in TCP's Additive-Increase/Multiplicative-Decrease algorithm. Standard TCP has $(a, b) = (1, 0.5)$. In addition, we use sources with the following parameters: $(1, 0.9)$, $(0.75, 0.31)$, $(2.0, 0.5)$. We refer to these as AIMD1, AIMD2 and AIMD3, respectively. The first and third are more aggressive than normal TCP, and the second is slightly less aggressive than TCP.

Binomial: Recently Bansal and Balakrishnan [1] introduced a more general family of increase/decrease algorithms. Increases are of the form $w + aw^{-k}$ and decreases are of the form $w - bw^{l}$ for constants (a, b, k, l). We used two versions: $(1.5, 1.0, 2, 0)$ and $(1.0, 0.5, 0, 1)$. The first is roughly comparable to TCP and the second is much more aggressive than TCP. We refer to these as Binomial1 and Binomial2.

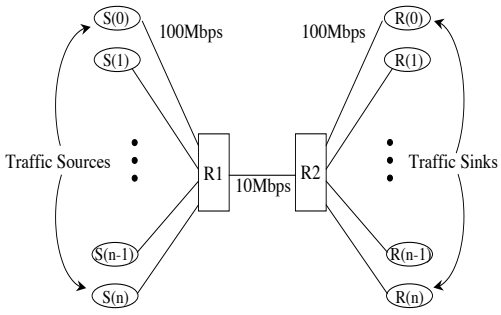
CBR: These are constant rate flows that do not perform congestion control.

[11]We have not included algorithms such as Fair Queueing [6] and CSFQ [24] in our performance evaluations. These algorithms are known to have good fairness properties and we do not claim that AFD performs better. The advantage of AFD over these algorithms is in complexity and deployability.

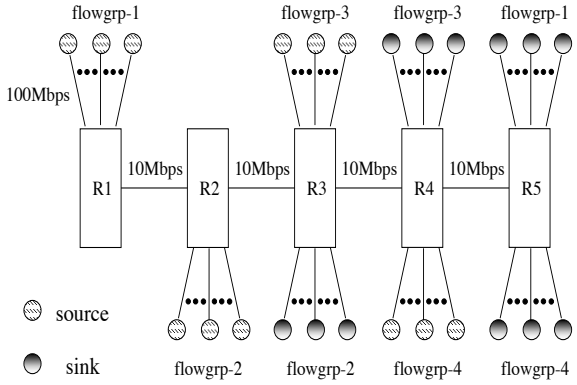
[12]We use RED in *gentle* mode with the parameter settings of $min_{th} = 25$ and $max_{th} = 125$ for a 10Mbps link. The thresholds are scaled proportionally for higher speed links.

[13]We simulated FRED with $min_q = 4$ and other parameters as in the RED simulations.

[14]Other algorithmic differences, such as the manner in which the drop probabilities are computed, differentiate AFD from RED.



(a) Single Link Topology



(b) Multiple Link Topology

Figure 2: Topologies

Most of our simulations are run on the topology depicted in Figure 2(a), and a few are run on the topology in Figure 2(b). The congested links have transmission latencies of 20msec, and unless otherwise stated the uncongested links have latencies of 2msec. The routers on the congested link(s) have buffers that hold 600 packets.[15] The simulations are run for 10 minutes of simulation time, with the first 100 seconds discarded for warmup. Unless otherwise stated, all data packets are 1000 bytes, and AFD is run with $b = 500$ and $s = 5$.

5.2 Simulation Results

Mixed Traffic: In Section 2 we showed that AFD provides equal bandwidth shares to CBR flows sending at different rates. We now consider the more challenging case of a mixture of flows using different congestion control algorithms. The basic scenario we present consists of 7 groups of flows, each with five flows, sharing the bottleneck link in Figure 2(a). The 7 groups are AIMD1, AIMD2, AIMD3, Binomial1, Binomial2, TCP, and TCP with a higher latency of 24.5msec on the 100Mbps access links (yielding a roundtrip latency, without queueing, of 138msec for this flow group, as compared to 48msec for the other flows). Figure 3(a) shows the throughput received by each group of flows (we average within each group so that the data is more easily presentable); the fluctuations within groups are quite small, as was seen in Figure 1 for the CBR simulation. The bar chart shows the bandwidth allocations for the flow groups in the following order: AIMD1, AIMD2, AIMD3, Binomial1, Binomial2, TCP, and TCP with increased latency.[16]

The throughput allocations for RED and FRED are substantially uneven, showing that this set of flows includes several rather aggressive congestion control algorithms. Therefore, this presents a reasonable test of the fairness properties of the various algorithms. AFD-SB and AFD-FT both have reasonably fair allocations. Figure 3(b) shows the average drop rate for each group of flows; RED has uniform drop rates, FRED has somewhat uneven drop rates, and both AFD algorithms have extremely uneven drop rates. This demonstrates that applying differentiated drop rates can lead to fairness for different flows.

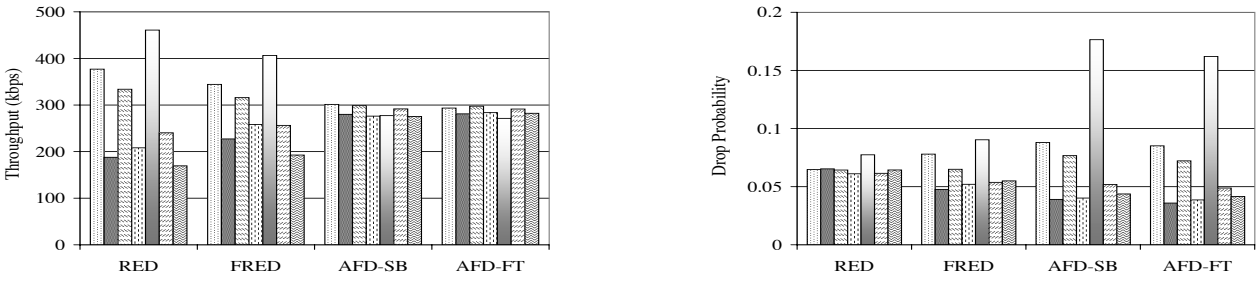
We stated at the outset that we did not intend to match the packet-by-packet fairness of Fair Queueing, and we have shown that AFD achieves reasonable fairness on long time scales. We nonetheless examine the behavior of the algorithm on shorter time scales. Figure 4 shows the cumulative distribution of the short term throughput of all 35 flows over 2, 5 and 10 second intervals normalized to the average throughput. When looking at the throughput over 10 second intervals we see that it is within 10% of the average 53.0% of the time and within 20% of the average 86.0% of the time. Over 5 second intervals, throughput is within 10% of the average 36.8% of the time and within 20% of the average 66.5% of the time. The corresponding percentages for 2 second intervals (which represents the amount of history in the shadow buffer) are 18.4% and 36.4%.

Although long term behavior is fair, not surprisingly, short term fairness suffers. This is undoubtedly due to the random nature of dropping decisions. When packets are dropped probabilistically and the drop probabilities are typically low, it is difficult to match the drop rates on short time scales. Consequently, fairness suffers on intervals over which the target and actual drop rates diverge. Nonetheless, the long term behavior is fair. Further, and perhaps more importantly, because the short term unfairness is random, there appears to be no way for a malicious flow to exploit or manipulate this unfairness to its advantage.

While the primary goal of AFD is fairness, it is also designed to provide AQM. The part of AFD that provides AQM is largely orthogonal to the fairness aspects of the algorithm. We have not fine tuned the AQM component and assume that better AQM components could be incorporated into AFD. Nonetheless, results (not shown here) indicate that

[15]The actual physical buffer size is not critical as the algorithm attempts to maintain the queue length near a target level, q_{target}. We have repeated simulations with a buffer size of 200 packets setting q_{target} to 75 in both cases. The throughput results are almost identical, indicating that large variations in the queue size above q_{target} are rare.

[16]Unless otherwise noted, we use the same ordering of flow groups in the remaining bar charts.



(a) Throughput

(b) Drop Rates

Figure 3: Mixed Traffic

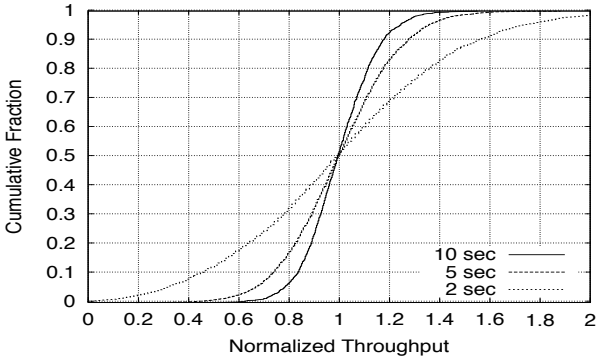


Figure 4: Cumulative distribution of per-flow throughput over 2, 5 and 10 second intervals

the link utilization, average queue size, and drop probability under AFD are all comparable to those of RED.

We now check how well this scenario compares with the guidelines. With $b = 500$ and 35 flows, the average share of the shadow buffer is roughly 14 packets which is in line with the recommendation of 10 packets in Guideline 1. It turns out that due to the fluctuations induced by the window based flow control, in which packets tend to be bunched, the average of m_{fair} is closer to 17. Guideline 2 sets bounds on $b\sqrt{s}\frac{P}{R}$; the guideline calls for this quantity to be larger than 750msec and our scenario has 890msec. Lastly, Guideline 3 calls for $300 < bs \ll 15,000$ if we choose $\tau = 250$msec. Our scenario has $bs = 2,000$.

High Speed Link: To ensure that the fairness in this initial simulation scales, we increased the speed of the congested link, the size of the shadow buffer, and the number of flows by a factor of 10. Figure 5 shows the throughput levels that result.

Increased Multiplexing: Next we tested what happens if the number of flows is increased by a factor of three (retaining the same mixture of flows and the same b). Results for AFD-SB are shown in Figure 6; results for AFD-FT are similar. Even though this scenario violates Guideline 1 (because now the fair number of matches is less than 5), the fair throughput levels are preserved. The conservative assumptions used in deriving the guidelines provide a margin for error.

Reduced Shadow Buffer: If we return to our canonical scenario and reduce the size of the shadow buffer, does fairness suffer? Figure 7 shows the throughput levels for AFD-SB (similar results hold for AFD-FT) for buffer sizes of $b = 250$ and $b = 125$ (the latter violates Guideline 1 significantly) in addition to the canonical case of $b = 500$. In this case, fairness degrades slightly as the shadow buffer size decreases, but AFD still outperforms the baseline algorithms.

While these cases establish that AFD-SB and AFD-FT can provide fair bandwidth allocations when faced with a diverse set of sources, there are cases when the AFD algorithms do not perform as well. We now present three such cases.

Long RTTs: First, consider the case where there are 4 flow groups, each with 10 TCP flows. The flow groups have different RTTs, as determined by the latencies of the connecting 100Mbps links. The latency of the central congested link remains the same (20msec), and the latencies of the connecting links are (followed in parentheses by the resulting total latency of the path, not counting queueing delays): 2msec (48msec), 7msec (68msec), 12msec (88msec), 17msec (108msec). The congested link has queueing delays of roughly 60msec which should be added to the total latency numbers to compute estimates of the RTTs. For each algorithm, the resulting bandwidth allocations for each flow group are shown in the bar chart in Figure 8(a) ordered from smallest to largest latency. The AFD designs provide much better fairness than RED or FRED. The throughput values differ from the fair share by less than 4%. We chose parameters that were designed (according to Guideline 2) to accommodate RTTs of up to 150msec, and this scenario is just a bit over that limit. If we increase the latency values to 2msec (48msec), 22msec (111msec), 42msec (175msec) and 62msec(238msec), the fairness is even further reduced, as shown in Figure 8(b). Here the throughputs vary from the fair share by as much as 20%. The largest RTT here (roughly 300msec including queueing delay) is larger than our b and s can accommodate (by guideline 2), and the re-

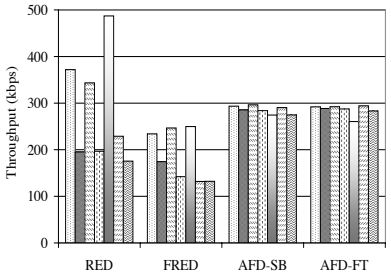


Figure 5: High-Speed Link

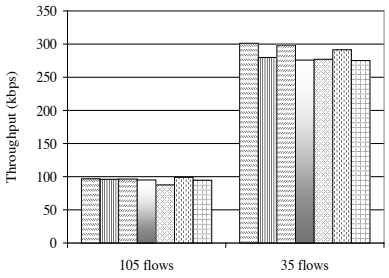


Figure 6: Increased Multiplexing

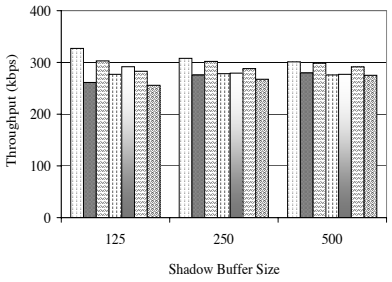
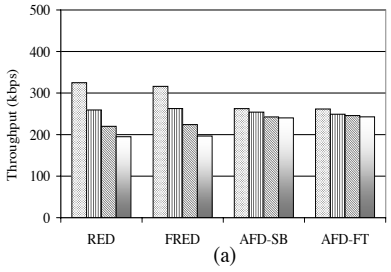


Figure 7: Reduced Shadow Buffer



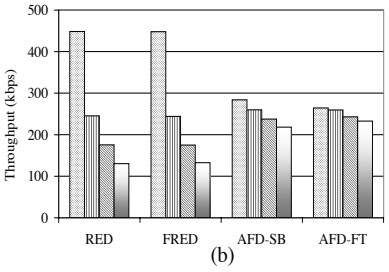


Figure 8: Long RTTs (a) Max Latency = 200msec (b) Max Latency = 300msec

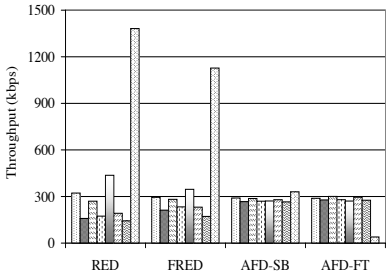


Figure 9: Unresponsive Flow

sulting decrease in fairness is evident.[17]

Unresponsive Flow: Next, we return to the latency values and traffic mix used in Figure 3. To this traffic mix we add a CBR flow sending at 15% of the link capacity. Figure 9 shows that the throughput allocations in AFD-SB are not perfectly fair, with the CBR receiving roughly 15% more than the fair share. The CBR under AFD-FT receives very little throughput at all, only about 15% of the fair share. This was not our intention when designing AFD-FT. However, we are not unhappy with this result; as we observed in Section 1, punishing unresponsive flows that tolerate persistent high drop rates is an important design goal (see [10] for the rationale). We have to (and can easily) add additional mechanism (which we do not discuss in this paper) to AFD-SB to accomplish this goal but AFD-FT does this by default. This occurs because the CBR flow occupies 15% of the shadow buffer (packets are selected to be inserted into the shadow buffer whether or not they are destined to be dropped), which is 5 times larger than the share of any other flow. The approximate packet deletion scheme in AFD-FT is biased against smaller flows. This in turn biases the dropping algorithm against larger flows. When the differences between flows is small this bias is negligible, but when the difference is a factor of five, the bias dominates the results. We believe this phenomenon is also responsible for AFD-FT performing better than AFD-SB in the RTT tests (see Figure 8).

Multiple Congested Links: The third challenging scenario for AFD is when there are multiple congested links. This scenario used the topology in Figure 2(b). There are four different source-destination pairs: three that traverse a single congested link and one that traverses the long path through 3 congested links. Each link has a latency of 5ms. In the first scenario we consider, each of these flow groups consists of 20 standard TCP flows, so that every congested link is traversed by the same number (i.e., 40) of flows. Figure 10(a) shows that the flows that traverse the long path (the fourth group shown for each algorithm in the bar chart) receive substantially less bandwidth than the others, roughly 83% of the fair share. While the throughputs are much fairer than RED or FRED, these results are troubling. If we double the number of flows in the group that traverses the link between R4 and R5, which reduces the fair share on that link, then the results become much fairer. These results are shown in Figure 10(b), with the first group of two bars showing the throughput for the flow groups traversing each of the first two congested links, respectively, and the second group of two bars showing the throughput of the group traversing R4-R5 and the throughput of the group traversing the long path, respectively. Bandwidth allocation for pairs of flow groups that share the same bottleneck link is much fairer for the AFD algorithms relative to the previous scenario. These results suggest, and simple anal-

[17] The analysis leading to guideline 2 suggests that $b\sqrt{s}\frac{P}{R}$ should be greater than 5 times the maximal RTT. In this case, that would call for $b\sqrt{s}\frac{P}{R} \approx 1.55$sec whereas in this scenario $b\sqrt{s}\frac{P}{R} = 890$msec.

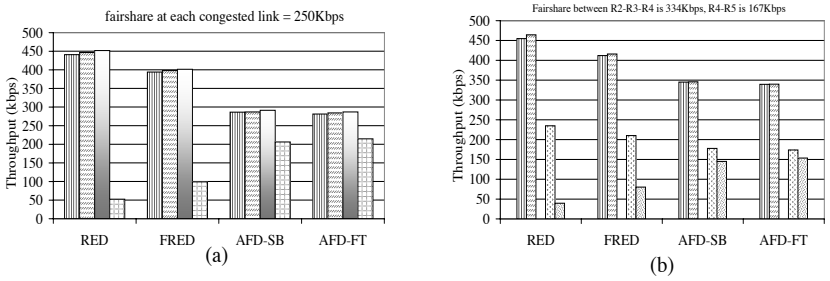


Figure 10: Multiple Congested Links (a) same Fair Share (b) different Fair Share

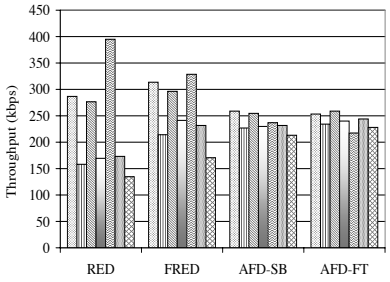


Figure 11: Two-Way Traffic

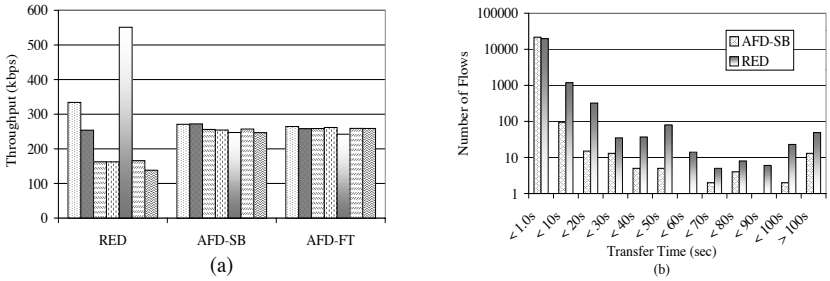


Figure 12: Web Traffic – (a) throughput of large flows (b) transfer time of small flows

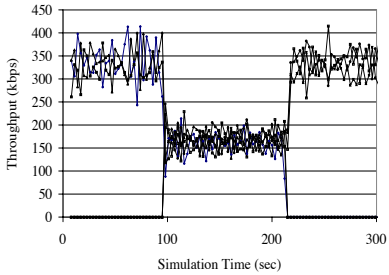


Figure 13: Changing Traffic

ysis confirms, that AFD does not perform well when there are multiple congested links with *exactly* the same fair share. Flows that are sending right at their fair share experience probabilistic dropping at multiple hops, and therefore receive less than flows that traverse only one congested link. However, if flows pass through multiple congested links that have different fair shares, then when sending at around the fair share the multi-hop flow only suffers drops at one link. We conjecture (not based on any special insight but just based on common sense) that traversing paths with multiple congested links with approximately the same fair shares will be an unusual case.

We tested many other scenarios in simulation, including different traffic mixes and variable size packets, that we do not present here. In all cases, the fairness properties of AFD were maintained. Before concluding the description of our simulations, we briefly mention 3 additional experiments.

Many Small Flows: In the simulations described above, all of our sources had an unlimited amount of data to send. In the next scenario, keeping our basic mixture of 7 flow groups (with infinite sources), we introduce approximately 22,000 short TCP flows representing, for example, shorter web transactions. We use a Pareto distribution of flow lengths. The average number of packets per flow is 15 with a shape parameter of 1.2. The resulting mix of traffic reflects the salient features of Internet traffic. Namely, many flows are coming and going, and most flows are small and short-lived. However, the majority of the bytes belong to long-lived and larger flows.

Using this simulation setup, we hope to answer two questions: does the presence of many short flows interfere with the fairness properties of the longer-lived flows, and does AFD allow the shorter flows to finish sooner? Figure 12(a) shows the throughput received by the infinite source flow groups under RED, AFD-SB and AFD-FT. AFD-SB and AFD-FT still provide very good fairness in this scenario. Figure 12(b) shows a histogram of the finishing times of the short-lived flows under AFD-SB and RED. Note that this is on a logarithmic scale, so that while the difference in heights between the bars representing the fastest finishing times is small, it represents a difference of approximately 2000 flows. In all the other histogram bins, RED has more flows. Thus, AFD-SB (and the same applies to AFD-FT) aids short flows by lowering their drop rates and allowing them to finish sooner. With RED, the short flows see the same ambient drop rate as all other flows.

Two-Way Traffic: When two-way traffic is present, the traffic can become burstier due to ACK-compression [27]. In this scenario, we have the 7 flow groups from our basic scenario sending in both directions. The congested link has a latency of 5ms. Figure 11 shows the throughput for each flow group. AFD-SB and AFD-FT both provide much better fairness than RED and FRED. AFD-FT punishes one particularly aggressive group of flows (for reasons we dis-

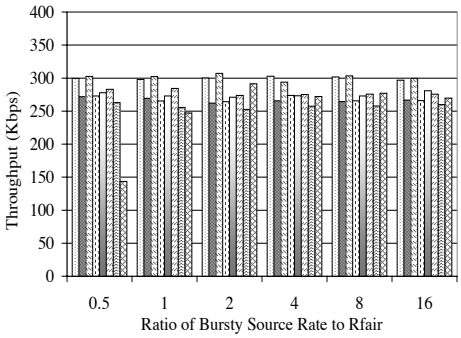


Figure 14: Bursty Traffic

cussed in the Unresponsive Flow section above). The additional queueing delay incurred in the reverse direction is enough to cause the performance of the TCP flow group with longer RTT (the 7th group) to suffer under AFD-SB (their RTT with queueing delay is roughly 180msec, which is larger than the 150msec our parameters were intended for).

Changing Traffic: We also tested the performance of AFD under changing traffic conditions. In this scenario, three flow groups (AIMD1, Binomial2, TCP) begin sending data at the start of the simulation. After 100 seconds, three additional groups (of the same kinds) start transmitting, doubling the offered load. At time 200 seconds, the first 3 groups stop sending. Figure 13 shows that the AFD-SB algorithm is able to adapt to the changing traffic, cutting the allocations in half when the offered load doubles, and increasing them again when the load is reduced. In both cases, the allocations are adjusted in less than 500msec.

Bursty Flows: We also examined the performance of AFD in the presence of on-off sources. These sources exhibit maximally bursty behavior by sending a burst at the speed of the access link (100Mbps) for a brief period and then going idle. The burstiness of these on-off sources is varied by adjusting their burst times while holding the burst rate and idle period constant. An on-off source can be characterized by the ratio of its average sending rate, $\frac{r_{burst} \times t_{burst}}{t_{burst}+t_{idle}}$, to the fair share rate, r_{fair}.

The results of these experiments are shown in Figure 14. Each group of bars in the figure represents one experiment. In each experiment, there are 5 flows in each of the original 7 groups of flows (i.e., those used in the Mixed Traffic experiments above), and a single on-off source. The ratio of the average sending rate of the on-off flow to the fair share rate is indicated on the X-axis (ranging from .5 to 16.) For each experiment, the first seven bars show the average throughput for the flows in each of the 7 flow groups. The eighth bar shows the throughput of the on-off source.

There are three things to note in this figure. First, the bursty flows are not punished for their burstiness. When the average rate of the on-off source is one-half the fair share, its packets are not dropped. When its average sending rate is equal to the fair share, it experiences some packet drops and its throughput is approximately 90% of the fair share rate.[18] The second point to note is that as the sending rate of the bursty flow increases (up to 16 times the fair share rate), it does not receive more than the fair share. Finally, as the sending rate of the bursty flow increases, the fairness among the other flow groups is maintained. When the on-off source sends at 16 times the fair share rate, all the other flow groups receive between 95% and 100% of the fair share. Hence, these results indicate that AFD performs well in the face of very bursty flows, neither punishing nor rewarding flows for their burstiness while preserving the basic fairness among all flows.

5.3 Discussion of Simulation Results

We now review the general conclusions that can be drawn from our set of simulations. In most scenarios, as long as the guidelines are followed AFD-SB and AFD-FT provide very good approximations to fair bandwidth allocations. The level of fairness is far superior to RED (which does not attempt to provide fairness) and FRED (which uses a very restricted amount of state to guide the dropping decisions). The performance of AFD-SB and AFD-FT seemed fairly robust to varying parameters and conditions. In those scenarios where b was not large enough, or the RTTs were too large, the level of fairness degraded gracefully. Thus, AFD appears to not need precise parameter tuning, and fails softly when the parameters are far out of alignment.

As we saw with the unresponsive CBR flow, and in many of our other simulations not shown here, AFD-FT responds quite punitively when flows are not responsive. We judge this a good thing, but ultimately this is a policy question. With AFD-SB we can turn this policy on or off depending on the desires of the network operator;[19] in AFD-FT this policy is hardwired in. Moreover, the choice of $k = 5$ in our version of AFD-FT (where k is the number of other flows that must be compared before a packet is eliminated from the flow table) will probably need to be made larger when the number of persistent fast flows is larger. AFD-FT dealt quite well with the many small flows, but if the number of fast flows is on the order of thousands the number of comparisons will probably have to be on the order of 10 or more. We have not yet investigated how this parameter will scale.

6. STATE REQUIREMENTS AND RATE DISTRIBUTIONS

These simulations show that when using the guidelines to set the parameters, AFD provides reasonably fair bandwidth allocations. However, can this algorithm be feasibly implemented? All the operations on the forwarding path are $O(1)$, so the main barrier to implementation would be if AFD required an impractical amount of state. This feasibility requirement is economic, not technical. Clearly routers

[18] Recall that deviations from the fair share are most likely to occur for flows sending at or near the fair share rate.

[19] Again, we don't present the algorithm which implements this policy in AFD-SB, but it is straightforward and requires only that the shadow buffer and flow table keep separate track of matches that were dropped and matches that were not dropped.

could be built with vast amounts of additional storage; however, that would come at a steep price, and we are looking to achieve fairness without greatly increasing the complexity or the cost of future routers. We use, as our standard, the requirement that the additional state required by AFD should be small compared to the memory already required by the packet buffer. We compute the state requirements, with the aid of two measurements, in Section 6.1. In Section 6.2 we discuss how these state requirements depend crucially on the rate distribution and, in Section 6.3, present some trace data on current rate distributions.

6.1 Calculation of State Requirements

We compute the state requirements of our two designs – AFD-SB and AFD-FT – separately. We start with AFD-SB, whose state consists of two parts: the shadow buffer and the hash table.

Guideline 1 (Section 3.1) states that the shadow buffer should hold roughly $10\frac{R}{r_{fair}}$ packets. This estimate assumed that all packets were the same size. To adjust this for variable packet sizes, we first define $\tilde{R}$ as the packet arrival rate in terms of packets/sec, and $\tilde{r}_{fair}$ as the arrival rate of a flow sending at the fair share rate; also, let P_{max} and P_{ave} denote the maximal and average packet sizes. The proper sizing of the shadow buffer in terms of packets, when you have variable sized packets, is then $b = 10\frac{\tilde{R}}{\tilde{r}_{fair}}$. The worst case, the lowest value, for $\tilde{r}_{fair}$ is when a fair share flow is sending maximal sized packets: $\tilde{r}_{fair} \geq \frac{r_{fair}}{P_{max}}$. The expression for $\tilde{R}$ is simply $\tilde{R} = \frac{R}{P_{ave}}$. To represent each packet in the shadow buffer we use, roughly 6 bytes.[20] Assuming that $P_{max} = 1500$ bytes and $P_{ave} = 300$ bytes,[21] we then find that b_{bit}, the shadow buffer size in bits, should be roughly:

$$b_{bit} = 10 * 48 * \frac{\frac{R}{300}}{\frac{r_{fair}}{1500}} = 2400\frac{R}{r_{fair}}$$

Similarly, we can estimate the storage required for a hash table to keep track of the number of bytes sent by each flow. We describe our trace data in Section 6.3, but for now we use the fact that, in the various traces we have seen and for the shadow buffer sizes we are proposing, the number of flows in the hash table is typically less than a fourth the number of packets in the shadow buffer (see Figure 15). Let's assume that we need 2 bytes per entry for counting (assuming we count matches at the granularity of 40 byte units and ignore roundoff errors), and that we use a hash table 12 times larger than the expected number of flows to reduce the chance of collisions in the hash table. The size of the hash table h_{bit}, in bits, is then:

[20] We need not store the full header in the shadow buffer, merely the hash of the source-destination addresses. 6 bytes is a sufficiently large hash to comfortably accommodate roughly 10^8 flows with small chance of collision.

[21] See [3] for measurements of average packet size. We've used a fairly conservative estimate, as the average packet size reported in [3] is over 400 bytes. The average packet size over our three traces that we report on in Section 6.3 is almost 500 bytes.

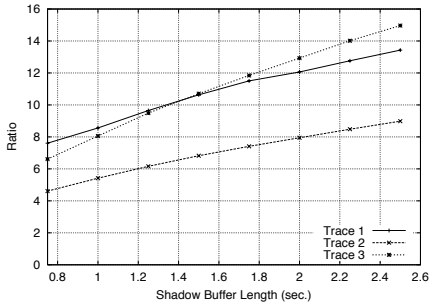


Figure 15: Ratio of Packets to Flows in Shadow Buffers

$$h_{bit} = 10 * 16 * 12 * \frac{1}{4}\frac{\frac{R}{300}}{\frac{r_{fair}}{1500}} = 2400\frac{R}{r_{fair}}$$

Thus, to a first approximation, these two data structures require similar amounts of memory. Many router vendors recommend having on the order of 250msec's worth of memory in the packet buffers. We compare the memory required by AFD to the memory already recommended for the packet buffer, which is roughly $\frac{C}{4}$ where C is the speed of the link (in bps). The ratio, call it ρ_{SB}, of AFD-SB's state requirement to the size of the packet buffers is given by:

$$\rho_{SB} = \frac{4800\frac{R}{r_{fair}}}{\frac{1}{4}C} = \frac{19.2kbits/sec}{(1-D)r_{fair}} \approx \frac{19.2kbits/sec}{r_{fair}}$$

where the last approximation is because $C = R(1-D)$ and we suspect the aggregate dropping rate will typically be low. Thus, the fraction of extra memory, ρ_{SB}, required by AFD-SB depends on the typical size of the fair share rate on a link.

We now calculate the state requirements of AFD-FT. If the flow table is implemented using a hash table, then we merely use the state requirements above (only the hash table part) and so the state requirements are roughly half that of AFD-SB. If the flow table is implemented using CAM then we no longer need a table 12 times the number of flows (as we did with the hash table). The CAM-based design needs roughly 64 bits per flow (48 bits for the hashed flow-id and 16 bits for a counter). Modifying the calculations above to reflect these changes, we find that the AFD-FT ratio is:

$$\rho_{FT} \approx \frac{3.2kbits/sec}{r_{fair}}$$

which is one-sixth of the AFD-SB requirements.

Note that these estimates of the ratios ρ depend only on the fair share of a link and not directly on its bandwidth. For slow links, the fair share will be quite small (and thus ρ could be 1 or larger), but the amount of memory required for these slow links is insignificant; if the fair share is 19.2kbps, so $\rho = 1$, on a T3 link, the required extra memory for AFD-

SB is roughly 1.5Mbytes. Thus, we care mostly about these ratios ρ on faster links where the absolute amount of memory devoted to the packet buffers is quite large. We assume that faster links will have larger fair shares, and these ratios ρ estimating the relative memory requirements of AFD will be smaller on such links.

We are not aware of many studies of the typical fair shares on current Internet links.[22] One can't infer the fair share merely by taking a packet trace; without fair bandwidth allocation one can't tell which flows are constrained by that link, and which flows are constrained elsewhere. One would have to find a link whose router has Fair Queueing (or some equivalent algorithm) on which to take traces (and even these traces would be misleading because of the lack of fair bandwidth allocations elsewhere along in the network). Lacking a solidly grounded method for measuring typical fair shares, we turned to an available dataset that has some bearing on the question.

One way to obtain a very rough estimate of the fair share on a path is to measure the end-to-end throughput obtained by a TCP connection that traverses that path. This is a very imperfect measure, since TCP throughput varies as a function of RTT and not all traffic is TCP-friendly. We obtained a dataset of measured end-to-end throughputs of transfers between NIMI measurement sites.[23] 47 sites participated in this measurement by periodically transferring a 1MB file with another randomly chosen NIMI site. The dataset contains roughly $35,000$ transfers. For lack of space we do not show the distribution here, but in over 90% of the transfers the rate is greater than 250kbps. These measurements are not focused on high-speed links. However, most of the NIMI boxes are at universities and other sites that are well-connected to the Internet, and none of them are behind very slow modems, so the measurements are probably indicative of moderate speed links. If we take 250kbps as a rough estimate of the fair share then $\rho_{SB} \approx \frac{1}{13}$ and $\rho_{FT} \approx \frac{1}{78}$. That is, the extra memory required by AFD-SB is less than a tenth of the memory already devoted to packet buffers. We view these estimates as being a very conservative lower bound, in that the fair shares on very fast links may be much larger, and thus the ratios ρ_{SB} and ρ_{FT} much smaller. As one moves up the network hierarchy towards the backbone links, it is likely that a larger fraction of flows are bottlenecked somewhere else in the network. That means that the fair share available to flows unconstrained elsewhere could be quite large. If backbone links had $r_{fair} \approx 2Mbps$ then $\rho_{SB} \approx \frac{1}{100}$ and $\rho_{FT} \approx \frac{1}{600}$. However, the level of fair shares remains an open question, and we hope in the future to find better ways to estimate the fair share on high-speed links.

This paper does not address detailed implementation issues. However, since we've used the storage requirements of the packet buffer as a yardstick for the state requirements of AFD, we would be remiss if we did not point out that this comparison is somewhat misleading. Packet buffers make heavy use of slower (and cheaper) DRAM with a smaller amount of faster (and more expensive) SRAM. We can do likewise with the shadow buffer state. However, the flow table will probably require SRAM or CAM. Thus, we cannot conclude that AFD imposes a cost that is a fraction ρ of the packet buffer cost. We leave the detailed implementation issues for future work (by others); our goal here is to provide a rough estimate of the state requirements.

The estimates above are very rough in nature. On any particular choice, such as the 250msecs for packet buffers or the numbers of bits for counters, one could argue that we are off by a factor of two. But we believe these calculations give us an order-of-magnitude estimate of the state requirements of our design, and suggest that AFD, in either of its incarnations, is likely able to achieve reasonable levels of fairness without exorbitant amounts of extra state.

We now argue that the distribution of rates greatly aids the AFD approach.

6.2 Impact of the Rate Distribution

Our state calculations depended critically on the value of r_{fair}, and in particular on the ratio $\frac{r_{fair}}{R}$. One intuitive way of understanding this ratio is the following. Let's call a flow a *cheetah* if it is sending above or near the fair share, and let n_c denote the number of such flows. We call the other flows, the slow ones, *turtles*. Let $\gamma = \sum_{turtles} \frac{r_i}{R}$ denote the fraction of the offered load that comes from these slow flows. The equation defining the fair share $\sum_i \min(r_i, r_{fair}) = C$ becomes: $R\gamma + r_{fair} n_c = C$. We can write this as

$$\frac{R}{r_{fair}} = \frac{n_c}{(1-D) - \gamma}$$

where D is the overall drop rate. The quantity $\frac{R}{r_{fair}}$ is minimized when both the number of cheetahs is small *and* the fraction of bandwidth consumed by the turtles is quite small: in short, when most of the bandwidth is being sent by a very few fast flows. It is in this regime that AFD requires very little state to perform well.

To make this more precise, we now look at how r_{fair} varies under different rate distributions in a simple analytical model. We consider a continuum of flows whose rates are given by a density function $h(r)$. We will consider four continuum distributions that all have the same average rate (of 1 in arbitrary units) and normalize the distribution by n (representing the number of flows), so the total load offered by each distribution is n. For each distribution we have $n = \int_0^\infty h(r)dr$ and $1 = \frac{1}{n}\int_0^\infty h(r)rdr$. We assume that the router allocates bandwidth fairly, and that flows respond by restraining their flow to the fair share if their assigned rate (by the distribution) is larger than the fair share. As we vary the capacity C between 0 and n we compute the fair share $r_{fair}(C)$ from the constraint: $C = \int_0^\infty h(r)\min(r, r_{fair}(C))dr$. The question is how $r_{fair}(C)$ compares for the various distributions we consider.

We consider four distributions, in order of increasing variance. For a *point* distribution, where all flows are the same rate, the fair share is merely $r_{fair}^{point} = \frac{C}{n}$. For a distribution where rates are uniformly distributed between 0 and 2, the fair share is $r_{fair}^{uni} = 2(1 - \sqrt{1 - \frac{C}{n}})$. For an exponential dis-

[22] The *available bandwidth*, which is a somewhat different concept, was studied in [19].

[23] NIMI is the National Internet Measurement Infrastructure; see [20] for a more detailed description. See [26] for details of the measurement process.

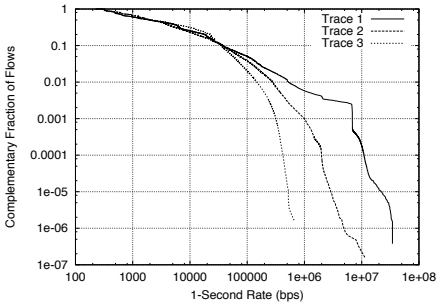


Figure 16: Complementary Distribution of 1-Second Rates

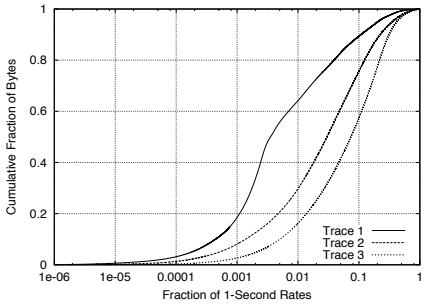


Figure 17: Cumulative Distribution of 1-Second Rates

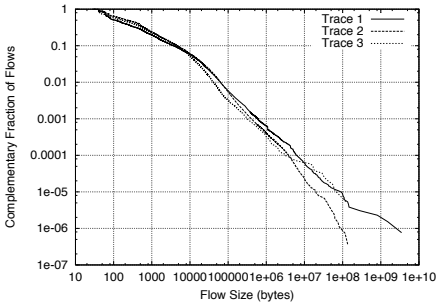


Figure 18: Complementary Distribution of Flow Sizes

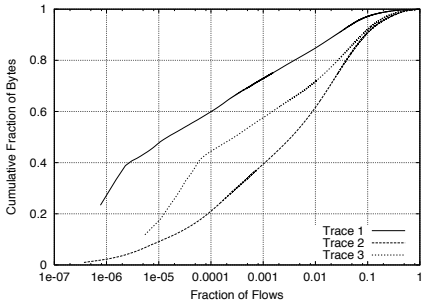


Figure 19: Cumulative Distribution of Flow Sizes

tribution, $h(r) = ne^{-r}$, the fair share is $r_{fair}^{exp} = -\ln(1-\frac{C}{n})$. Finally, for a power-law distribution $h(r) = \frac{n}{2}r^{-3}$ for $r \geq \frac{1}{2}$ (and 0 otherwise; this restriction is to avoid the divergence at the origin), we have $r_{fair}^{pl} = \frac{1}{4}(1-\frac{C}{n})^{-1}$. The point of this exercise is that as the tail of the distributions got larger (that is, the number of very fast flows increases) the fair share as a function of C became larger. While the fair share diverges for both the exponential and the cubic power-law distributions as $\frac{C}{n}$ approaches 1, the fair share is dramatically larger in the power-law case.

Guideline 1 calls for the buffer size to be roughly $b = 10\frac{R}{r_{fair}}$. If the rate distribution is such that r_{fair} is larger (for a given R) then the shadow buffer size, and all the state requirements, become smaller. Note that if the fair share remains the same then the state requirements increase linearly with the speed of the link (and the offered load R). However, if we keep the same offered load and let the fair share increase to its natural level as we increase C, and study $b(C) = 10\frac{C}{r_{fair}(C)}$, then we find two cases. For the point and uniform distributions, $b(C)$ increases; this is what we expect, more state is required on faster links. For the exponential and power-law distributions, however, $b(C)$ decreases; as the speed of the link increases the state requirements of the algorithm go down! Note this is not a decrease in the ratio ρ, this is a decrease in the absolute amount of state.

We have no way of judging with any certainty what the fair share rates will be in the future on high-speed links. However, the arguments above suggest that AFD will operate best when the distribution of flow rates has a long tail. We now turn to empirical evidence to see if this is currently the case.

6.3 Trace Data

We obtained traces of traffic from three separate locations: a 100Mbps link at a national laboratory, a T3 peering link between two providers, and a FDDI ring connecting a modem bank at a dial up provider to a backbone network. We refer to these traces as Trace 1, Trace 2, and Trace 3, respectively. Trace 1 consists of approximately 22 million packets collected during a two hour interval. Trace 2 consists of 34 million packets over a 45 minute span. Trace 3 collected approximately 6 million packets in 70 minutes.

Guideline 2 calls for keeping state of about 1 second in order to estimate the rates. For a variety of different time intervals, Figure 15 shows the average ratio between the number of packets in the shadow buffer and the number of distinct flows represented in the shadow buffer; this ratio was used in Section 6.1 to estimate the state requirements for AFD.

For each of the traces we estimated individual flow rates based on the packets seen in the last second; see [14] for similar data and related discussions. Figure 16 shows the complementary distribution of flow rates for each of the three traces. Trace 3 is not obviously inconsistent with a slowly decaying exponential distribution, but the other two clearly have long tails.[24] This is even more pronounced on

[24]We use the term *long tailed* to refer to distributions that de-

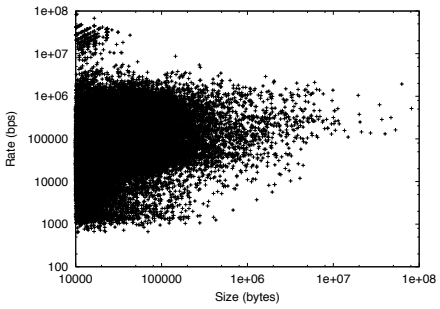


Figure 20: Rates vs Size

a log-linear plot, which for space reasons we do not show here. Figure 17 shows the cumulative distributions of the 1-second flow rates. Note 10% or less of the flows represent roughly 60% of the bytes in the worst case, and roughly 90% of the bytes in the best case. This is in contrast to the fastest 10% of the flows carrying 33% of the bytes for an exponential distribution. Thus, based on the very preliminary evidence presented here and in [14] we conjecture that rate distributions are usually long-tailed. In the language of Section 6.2, we expect that the number of cheetahs will typically be small, but they will represent the bulk of the bytes.

Similar statements have long been made about the distribution of flow sizes. Figures 18 and 19 show the analogous data for flow sizes as measured by the total number of bytes transferred. These distributions are significantly more skewed than the rate distributions. Figure 20 shows the total flow size plotted against the flow's rate (as measured by total bytes over completion time) for flows in Trace 2 that are larger than 10,000 bytes (the other traces have similar results). No correlation is visually apparent, and when we compute the correlation it is quite small; the correlations between rate and size are .0075, .0031 and .00088 for the three traces, and are .11, .22 and $-.0045$ when we correlate the logarithms of the rate and size. This shows that the distinction between cheetahs and turtles is different from the historical distinction between elephants and mice. The long-tailed distribution of rates is clearly not directly driven by the long-tailed distribution in sizes. Their underlying mechanisms are presumably different.

We are not aware of work that analyzes the mechanisms responsible for the distribution of flow rates. We assume that the rates on a link, at least for longer-lasting flows, reflects the available bandwidth at that flow's bottleneck link. However, it is not clear to us why the distribution of these bottleneck rates should have a long tail. Much more work remains to be done to characterize these rate distributions, and explain their origin.

cay slower than exponentially. Power-law distributions are examples of this, but so are Weibull and other distributions. We do not intend to enter the rather lengthy and subtle debates about whether or not a particular distribution is a power law or Weibull or some other form. We merely observe that it obviously decays slower than an exponential.

7. CONTEXT, RELATED WORK, AND DISCUSSION

This paper starts with the assumption that routers should allocate bandwidth fairly. While this is a familiar and oft-told story, for context we once again briefly review the rationale for fairness. We then discuss related work and conclude with some comments on AFD's underlying design principles.

Congestion control is one of the Internet's most fundamental architectural problems. In the current Internet, most routers do not actively manage per-flow bandwidth allocations, and so the bandwidth received by a flow depends on the congestion control algorithms used by other competing flows. To achieve relatively equitable bandwidth allocations, all flows must be TCP-compatible [4] (also known as TCP-friendly); that is, they must use a congestion control algorithm that results in bandwidth allocations similar to TCP's. This approach requires uniformity and cooperation, both of which might be problematic.

By restricting the world of congestion control algorithms to those that are TCP-compatible, some applications may be impaired. This would be the case if some applications required radically different forms of congestion control that are inherently unfriendly to TCP. However, recent advances in *Equation-Based Congestion Control* (EBCC) [11] and other TCP-compatible algorithms [22, 21] gives hope that a very wide variety of application requirements could be fulfilled within the sphere of TCP-compatibility. Thus, the uniformity requirement, while potentially a problem, may in fact be tolerable.

As for cooperation, any host can obtain more bandwidth simply by using a more aggressive congestion control algorithm. Thus, the current approach relies on end-hosts (and their users) voluntarily adopting the TCP-compatible guidelines. There has been some initial work to penalize flows that violate the rules [9], but so far the goal of reliably identifying *ill-behaved* flows has proved elusive.[25]

In response to these problems, there has been a long history (dating back to Nagle [16]) of proposing that routers play a more active role in allocating bandwidth. If the routers ensure fair bandwidth allocations[26] then end-hosts are no longer required to adhere to any particular form of congestion control and the problems of uniformity and cooperation mentioned above no longer exist.[27] The proposals to accomplish this fair bandwidth allocation, such as Fair Queueing [6] and related algorithms [2, 15, 23], all involve complicated packet scheduling algorithms and require per-

[25]There has been more success with identifying high-bandwidth flows, and unresponsive flows, but that still leaves flows a large leeway to cheat.

[26]To give users an incentive to use some form of responsive congestion control we add the requirement that flows are punished if they incur persistent high drop rates. See [10, 24] for discussions of this topic.

[27]Some claim that such fair allocation mechanisms open the door to denial-of-service attacks through flow-spoofing; while we do not believe such arguments are sufficient to nullify the desirability of fair bandwidth allocations, and that this paper is not the place to delve into such arguments at length, we did want to note the existence of objections to the fair bandwidth allocation paradigm.

flow state. High-speed implementations of these algorithms are just now becoming available. If such implementations continue to scale to the highest speeds without causing undue costs, then the contents of this paper are probably moot. However, it isn't yet clear that these implementations will scale *inexpensively* to increasingly higher speeds. If the cost of adding this extra functionality is significant, designs with lower complexity but similar functionality may be preferable in commercial designs. This was our goal in designing AFD, to provide fairness at a significantly reduced level of complexity.

Core-Stateless Fair Queueing (CSFQ) [24] and the subsequently proposed variations [5, 25] share the same goal. While they achieve high degrees of fairness with scalable mechanisms in the core routers, these schemes require a change in the packet format (to accommodate another field) and the careful configuration of routers into core, edge, and peripheral regions. Thus, while CSFQ delivers a high degree of fairness, it faces significant deployment hurdles.

There are several other proposals for low-complexity approximations of fairness, such as FRED [13], SRED [17], SFB [7], CHOKe [18] and RED-PD [14]. These present a spectrum of possible designs, differing in the extent to which they carefully manage the bandwidth allocation. The extremes of the spectrum are complete fairness (Fair Queueing) on one end and unmanaged allocation (RED) on the other; in between, some algorithms carefully manage the bandwidth of a few flows (*e.g.* RED-PD) while others attempt to manage all flows at the same time, but do a less careful job on each one (*e.g.* AFD). Different choices along this spectrum embody different expectations about the set of congestion algorithms deployed. One possible view is that in the future almost all flows will use a TCP-compatible congestion control algorithm, and that there will only be a very few malicious (or broken) flows that are substantially more aggressive. In this scenario, routers only need to detect these few outlying flows and restrain their usage to an appropriate level. RED-PD is an example of an algorithm well-suited to this task. Another possible view is that flows will use a very wide variety of congestion control algorithms, not necessarily TCP-compatible, and that routers will need to allocate bandwidth to flows as the common case. AFD is designed for this scenario. Which of these two approaches – identifying a few outlying flows and treating them as special or treating allocation as the common case – depends on how the future of congestion control unfolds.[28] Our purpose in this paper is not to argue that one vision is more likely than another, merely to design an algorithm that would be suitable if the second scenario comes to pass.

The spectrum of approximate fairness designs also presents us with different tradeoffs between increased fairness and reduced complexity. The desirability of one spot on the spectrum versus another depends greatly on the nature of Internet traffic and on router design constraints, both of which change over time. We based AFD on the assumption that while traffic characteristics and hardware capabilities will evolve, there will be three enduring truths. First, FIFO packet scheduling will be significantly easier to implement than certain non-FIFO scheduling algorithms; it uses cheaper hardware that scales to faster speeds. Second, the distribution of rates on high-speed links will be long-tailed. We don't expect this to necessarily be a power-law or any other particular form, but we do expect that the majority of flows will be slow but the fast (high-rate) flows will send the bulk of the bytes. Third, we assume that memory – fast, slow, and CAM – will continue to decrease in price relative to the special logic needed to implement non-FIFO packet scheduling. AFD's main implementation burden is additional memory; we assume that the marginal cost of equipping routers with AFD will shrink (relative to packet scheduling designs) as these commodity products become cheaper over time. If these three assumptions hold, then we believe AFD may be a cost-effective scheme for providing approximately fair bandwidth allocations.

[28] That future will depend in part on the choices made for router allocation mechanisms, so the dependencies are circular.

8. REFERENCES

[1] Bansal, D., and Balakrishnan, H., "Binomial Congestion Control Algorithms " *Proceedings of Infocom '01.*

[2] Bennett, J. and Zhang, H., "Hierarchical Packet Fair Queueing Algorithms", *SIGCOMM Symposium on Communications Architectures and Protocols*, pp. 143–156, Aug. 1996.

[3] `http://www.caida.org/analysis/AIX/plen_hist/`

[4] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S.,Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L.,Ramakrishnan,K., Shenker,S., Wroclawski, J., Zhang, L., "Recommendations on queue management and congestion avoidance in the internet", *IETF RFC (Informational) 2309*, April 1998.

[5] Cao, Z., Wang, Z. and Zegura, E., "Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State", *Proceedings of INFOCOM'00* March 2000.

[6] Demers, A., Keshav, S. and Shenker, S., "Analysis and simulation of a fair queueing algorithm", *Journal of Internetworking Research and Experience*, pp 3-26, Oct. 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[7] Feng, W., Shin, K., Kandlur, D. and Saha, D., "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness", *Proceedings of INFOCOM'2001 (to appear)*, April, 2001.

[8] Floyd, S. and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transaction on Networking*, 1(4), pp 397-413, Aug. 1993.

[9] Floyd, S., and Fall, K., "Router Mechanisms to Support End-to-End Congestion Control", *LBL Technical report*, February 1997.

[10] Floyd, S., and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, August 1999.

[11] Floyd, S., Handley, M., Padhye, J., and Widmer, J., "Equation-Based Congestion Control for Unicast Applications", *Proceedings of ACM SIGCOMM'2000*, August 2000.

[12] Hollot, C.V., Misra, V., Towsley, D. and Gong, W., "On Designing Improved Controllers for AQM Routers Supporting TCP Flows", *Proceedings of Infocom '01.*

[13] Lin, D. and Morris, R., "Dynamics of random early detection", *Proceedings of ACM SIGCOMM'97*, pp 127-137, Oct. 1997.

[14] Mahajan, R., Floyd, S. and Wetherall, D., "Controlling High-Bandwidth Flows at the Congested Router", *9th International Conference on Network Protocols*, November 2001.

[15] McKenney, P., "Stochastic Fairness Queueing", *Proceedings of INFOCOM'90*, pp 733-740.

[16] Nagle, J., "On packet switches with infinite storage", *Internet Engineering Task Force*, RFC-970, December, 1985.

[17] Ott, T., Lakshman, T. and Wong, L., "SRED: Stabilized RED", *Proceedings of INFOCOM'99*, pp 1346-1355, March 1999.

[18] Pan, R., Prabhakar, B. and Psounis, K., "CHOKe - A Stateless Active Queue Management Scheme For Approximating Fair Bandwidth Allocation", *Proceedings of INFOCOM'00* March 2000.

[19] Paxson, V., "End-to-End Internet Packet Dynamics", *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277-292, 1999.

[20] Paxson, V., Mahdavi, J., Adams, A. and Mathis, M., "An Architecture for Large-Scale Internet Measurement", *IEEE Communications Magazine*, vol. 36, no. 8, pp. 48-54, August 1998.

[21] Rejaie, R., Handley, M. and Estrin, D., "An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet", *Proceedings of INFOCOM'99*, March, 1999.

[22] Rhee, I., Ozdemir, V. and Yi, Y., "TEAR: TCP emulation at receivers – flow control for multimedia streaming", *Technical Report, Department of Computer Science, NCSU*, April, 2000.

[23] Shreedhar, M., and Varghese, G., "Efficient Fair Queueing using Deficit Round Robin", *ACM Computer Communication Review*, vol. 25, no. 4, pp. 231–242, October, 1995.

[24] Stoica, I., Shenker, S. and Zhang, H., "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks", *Proceedings of ACM SIGCOMM'98.*

[25] Venkitaraman, N., Mysore, J., Srikant R.,and Barnes, R. "Stateless Prioritized Fair Queueing", *Internet Engineering Task Force* July 2000.

[26] Zhang, Y., Paxson, V., and Shenker, S., "The Stationarity of Internet Path Properties: Routing, Loss, and Throughput", ACIRI Technical Report, May 2000.

[27] Zhang, L., Shenker, S. and Clark, D., "Observations on the Dynamics of a congestion control Algorithm: The Effects of Two-Way Traffic", *Proceedings of ACM SIGCOMM'91.*

[28] ns - Network Simulator (Version 2.1b6).