Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Computer Networks 44 (2004) 135–152
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

Fair adaptive bandwidth allocation: a rate control based active queue management discipline ☆

Abhinav Kamra [a,1], Huzur Saran [a], Sandeep Sen [a], Rajeev Shorey [b,*,2]

[a] Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi 110016, India
[b] IBM India Research Laboratory, Indian Institute of Technology, New Delhi 110016, India

Received 9 May 2002; received in revised form 7 July 2003; accepted 7 July 2003

Responsible Editor: Z.-L. Zhang

Abstract

Active queue management disciplines such as RED and its extensions have been widely studied as mechanisms for providing congestion avoidance, differentiated services and fairness between different traffic classes. With the emergence of new applications with diverse Quality-of-Service requirements over the Internet, the need for mechanisms that provide differentiated services has become increasingly important. We propose fair adaptive bandwidth allocation (FABA), a buffer management discipline that ensures a fair bandwidth allocation amongst competing flows even in the presence of non-adaptive traffic. FABA is a rate control based AQM discipline that provides explicit fairness and can be used to partition bandwidth in proportion to pre-assigned weights. FABA is well-suited for allocation of bandwidth to aggregate flows as required in the differentiated services framework. Since FABA can be extended to scenarios such as aggregate, hierarchical and weighted flows, it can serve as a useful method for enforcing service level agreements at the edge of the network. We study and compare FABA with other well known queue management disciplines and show that FABA ensures fair allocation of bandwidth across a much wider range of buffer sizes at a bottleneck router. Further, FABA is shown to give high values of fairness coefficient for diverse applications such as FTP, Telnet and HTTP. FABA uses randomization and has an O(1) average time complexity, and, is therefore scalable. The space complexity of the proposed algorithm is O(B) where B is the buffer size at the bottleneck router. We argue that though FABA maintains per active-flow state, through O(1) computation, reasonably scalable implementations can be deployed which is sufficient for network edges.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Adaptive flows; Non-adaptive flows; Congestion avoidance; Buffer management; Bandwidth; Fairness; Differentiated services; Time complexity; Space complexity; Active flows

☆ A preliminary version of this paper appeared in the Networking'2002 Conference held from 19–24 May 2002 in Pisa, Italy. See http://www.cnuce.pi.cnr.it/Networking2002/.

* Corresponding author.
E-mail addresses: kamra@cs.columbia.edu (A. Kamra), saran@cse.iitd.ernet.in (H. Saran), ssen@cse.iitd.ernet.in (S. Sen), rajeev@comp.nus.edu.sg (R. Shorey).

[1] Currently a graduate student in the Computer Science Department at Columbia University, New York, USA.

[2] Since August 2003 at Computer Science Department, National University of Singapore. On leave from IBM India Research Laboratory, New Delhi, India.

1389-1286/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S1389-1286(03)00347-5

1. Introduction

Adaptive protocols such as the transmission control protocol (TCP) employ end-to-end congestion control algorithms [11]. These protocols adjust their sending rate on indications of congestion from the network in the form of dropped or marked packets [8]. However, the presence of non-adaptive flows in the Internet, gives rise to issues of unfairness. Adaptive flows back off by reducing their sending rate upon detecting congestion while non-adaptive flows continue to inject packets into the network at the same rate. Whenever adaptive and non-adaptive flows compete, the non-adaptive flows, because of their aggressive nature, grab a larger share of bandwidth, thereby depriving the adaptive flows of their fair share. This gives rise to the need for active queue management (AQM) disciplines [14] at intermediate gateways which provide protection for adaptive traffic from aggressive sources that try to consume more than their "fair" share, ensure "fairness" in bandwidth sharing and provide early congestion notification.

With the emergence of newer applications, each having its own resource requirements, it has become necessary to deploy algorithms at intermediate gateways that provide differentiated services [18] to the various types of traffic. Often flows of similar nature form one aggregate so that all flows within an aggregate receive the same Quality-of-Service (QoS). Clark [7] suggests a model for differential bandwidth allocation between classes of flows. Flows within an aggregate can have further classifications resulting in a hierarchy [6] of flows. Efficient mechanisms to implement such classifications at intermediate gateways are of considerable interest.

Fig. 1 shows a typical Internet gateway with multiple incoming links, a single outgoing link of bandwidth C packets per second and a buffer size of B packets. The queue management discipline, operating at the enqueuing end, determines which packets are to be enqueued in the buffer and which are to be dropped. The scheduling discipline, at the dequeuing end, determines the order in which packets in the buffer are to be dequeued. Sophisticated scheduling algorithms, such as weighted fair queueing (WFQ) [3], can be used to provide differentiated services to the various types of traffic. However, for early detection and notification of congestion a queue management discipline is necessary. Combinations of queue management and scheduling disciplines can provide early congestion detection and notification and can be used in the differentiated services framework. Complex scheduling mechanisms such as WFQ require per flow queues and as such have high implementation overheads.

In this paper, we present fair adaptive bandwidth allocation (FABA), a queue management discipline which when coupled with even the simplest scheduling discipline, such as first-come-first-served (FCFS), achieves the following goals:

(i) fair bandwidth allocation amongst flows, aggregates of flows and hierarchy of flows,
(ii) congestion avoidance by early detection and notification,
(iii) low implementation complexity,

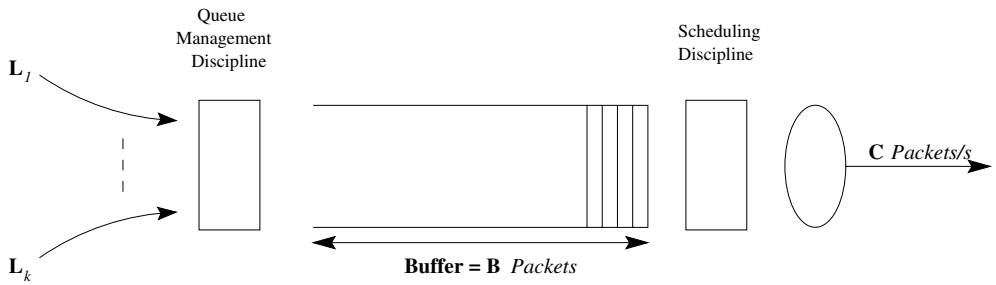


Fig. 1. A typical Internet gateway.

(iv) easy extension to provide differentiated services.

In an earlier work [17], we had proposed the selective fair early detection (SFED) algorithm. SFED is a rate control based buffer management algorithm. For the sake of completeness, we describe SFED in detail in this paper. FABA, proposed and described in this paper is an extension of SFED. We explain the details of SFED and FABA in subsequent sections.

FABA is an AQM algorithm that deals with both adaptive and non-adaptive traffic while providing incentive for flows to incorporate end-to-end congestion control. It uses a rate control based mechanism to achieve fairness amongst flows at a router. As in random early detection (RED) [5], congestion is detected early and notified to the source. FABA is easy to implement in hardware and is optimized to give $O(1)$ complexity for both enqueue and dequeue operations. We compare FABA with other queue management schemes such as RED, CHOKe [13], flow random early drop (FRED) [10], and observe that FABA performs at least as well as FRED and significantly better than RED and CHOKe. However, when buffer sizes are constrained, it performs significantly better than FRED. FABA is shown to give high values of fairness coefficient for diverse applications such as FTP, Telnet and HTTP.

The space complexity of FABA is $O(B)$ where B is the buffer size at the bottleneck router. Though FABA maintains per active-flow state, through $O(1)$ computation, reasonably scalable implementations can be deployed which is sufficient for network edges. Note that core routers in the Internet have limited memory and processing power. These routers need to be extremely fast since hundreds of thousands of flows pass through the routers at any point in time. Moving intelligence to the edge of the network results in more efficient network design and utilization, thereby lowering system costs. Several efforts in the recent past have addressed moving buffer management and scheduling from the core nodes to the edge nodes in the Internet. For example, Zhang et al. (see [12] and the references therein) propose the *Core-Stateless* architecture in which only edge nodes perform per flow management. In the recent past, papers have also investigated issues related to the role of edge devices in enabling service differentiation over the Internet. For example, Guerin et al. (see [20] and the references therein) investigate the problem of making QoS guarantees available in access devices such as edge routers.

The proposed FABA algorithm is ideally suited for the network edge. This is due to the fact that service differentiation, packet/flow classification, QoS and admission control are simpler and more cost-effective to implement at the edge nodes than the code nodes. Moreover, scalability issues dominate the core nodes due to a very large number of flows passing through core routers. Scalability is less of an issue at edge devices, and, in addition, there are more resources (e.g., memory, processing power) available at the edge devices than the core routers. It is for this reason that FABA is an ideal buffer management algorithm for the edges, even though, in principle, it can easily be implemented at the core nodes; the price to be paid being the $O(N)$ space complexity, where N is the number of active flows at the core router. Since the number of active flows are much less than the total number of flows at a core node in a given time interval, it should be clear that FABA can be implemented even at the core nodes. Further, with appropriate aggregation (as in the differentiated services framework), FABA can be applied to core nodes.

We demonstrate that FABA performs well when (i) flows are assigned different weights, (ii) when used to allocate bandwidth between classes of traffic such as would be required in the differentiated services scenario, and (iii) when it is used for hierarchical sharing of bandwidth. FABA can be easily extended to provide differentiated services in a network requiring QoS. Further, since FABA can be extended to scenarios such as aggregate, hierarchical and weighted flows, it can serve as a useful mechanism for enforcing service level agreements (SLAs) (see [2] for a detailed explanation of SLAs).

The paper is organized as follows. In Section 2, we briefly describe the existing queue management policies. An overview of FABA is presented in Section 3. We describe our basic mechanism and how it is used to achieve the desired properties of a

queue management discipline. Section 4 explains our proposed algorithm in detail. We show how using randomization, our proposed algorithm can be optimized to give O(1) performance. We then describe its hardware implementation. We compare the performance of FABA with other gateway algorithms namely RED, CHOKe and FRED with the help of simulations conducted using the NS-2 [15] network simulator in Section 5. In Section 6, we show how FABA can be generalized for weighted, aggregate and hierarchical flows. We summarize our results in Section 7.

2. Related work

A number of approaches for queue management at Internet gateways have been studied earlier. Droptail gateways are used almost universally in the current Internet due to their simplicity. A droptail gateway drops an incoming packet only when the buffer is full, thus providing congestion notification to protocols like TCP. While simple to implement, it distributes losses among flows arbitrarily [9]. This often results in bursty losses from a single TCP connection, thereby reducing its window sharply. Thus, the flow rate and consequently the throughput for that flow drops. Tail dropping also results in multiple connections simultaneously suffering losses leading to global synchronization [10].

RED addresses some of the drawbacks of droptail gateways. An RED gateway drops incoming packets with a dynamically computed probability when the exponential weighted moving average queue size avg_q exceeds a threshold called min_th. This probability increases linearly with increasing avg_q till max_th after which all packets are dropped until the avg_q again drops below max_th. The RED drop probability also depends on the number of packets enqueued since the last packet drop. The goal of RED is to drop packets from each flow in proportion to the amount of bandwidth it is using. However, from each connection's point of view the packet loss rate during a short period is independent of the bandwidth usage as shown in [10]. This contributes to unfair link sharing in the following ways:

- Even a low bandwidth TCP connection observes packet loss which prevents it from using its fair share of bandwidth.
- A non-adaptive flow can increase the drop probability of all the other flows by sending at a fast rate, which increases the drop probability for all flows.
- The calculation of avg_q for every packet arrival is computationally intensive.

To ensure fairness amongst flows in terms of bandwidth received and to identify and penalize misbehaving users, it is necessary to maintain some sort of per-flow state [14]. Many approaches based on per-flow accounting have been suggested earlier. FRED [10] does per-flow accounting maintaining only a single queue. It suggests changes to the RED algorithm to ensure fairness and to penalize misbehaving flows. It puts a limit on the number of packets a flow can have in the queue. Besides it maintains the per flow queue occupancy. Drop or accept decision for an incoming packet is then based on average queue length and the state of that flow. It also keeps track of the flows which consistently violate the limit requirement by maintaining a per-flow variable called *strike* and penalizes those flows which have a high value for *strike*. It is intended that this variable will become high for non-adaptive flows and so they will be penalized aggressively. It has been shown through simulations [12] that FRED fails to ensure fairness in many cases.

CHOKe [13] is an extension to RED. It does not maintain any per flow state and works on the good heuristic that a flow sending at a high rate is likely to have more packets in the queue during the time of congestion. It decides to drop a packet during congestion if in a random toss, it finds another packet of the same flow.

In a significant paper by Guerin et al. [19], the authors establish how rate guarantees can be provided by simply using buffer management. They show that the buffer management approach is indeed capable of providing reasonably accurate rate guarantees and fair distribution of excess resources. The key observation in the paper is that the combination of buffer management and limited scheduling is capable of a broad range of trade-offs

between efficiency and complexity. Note, however, that the study in [19] focusses on non-TCP flows where each flow behaves as a Markov-modulated ON–OFF source and specifies a traffic profile (peak rate, token rate, and token bucket size).

3. An overview of FABA

Since FABA is an extension of SFED algorithm [17], for the sake of completeness, we begin with a description of SFED algorithm in this section.

SFED is an easy to implement rate control based AQM discipline which can be coupled with any scheduling discipline. SFED operates by maintaining a token bucket for every flow (or aggregate of flows) as shown in Fig. 2. The token filling rates are in proportion to the permitted bandwidths. Whenever a packet is enqueued, tokens are removed from the corresponding bucket. The decision to enqueue or drop a packet of any flow depends on the occupancy of its bucket at that time. The dependence of drop probability on the buffer occupancy is shown in Fig. 3. The probability profile is explained in Section 4.

It is to be noted that the notion of token buckets per flow (or aggregate of flows) at a gateway or network edge is not new (see Chapter 6 in [4] and the references therein), [3]. The novelty in this paper is to use token buckets for buffer management at the network edges. The buffer management algorithm proposed in this paper is efficient: it has O(1) time complexity whereas the space complexity is bounded by the buffer size.

SFED ensures early detection and congestion notification to the adaptive source. A sending rate

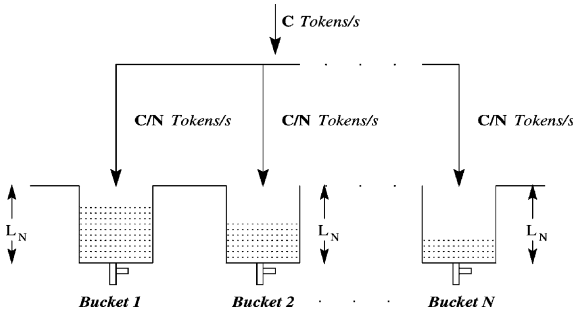


Fig. 2. FABA architecture for rate control using token buckets.

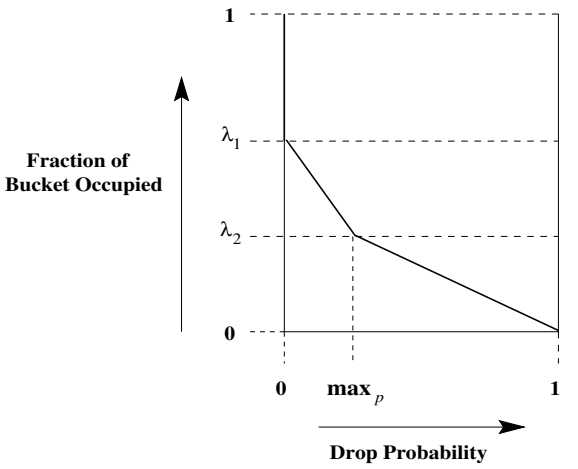


Fig. 3. Probability profile of dropping a packet for a token bucket.

higher than the permitted bandwidth results in a low bucket occupancy and so a larger drop probability thus indicating the onset of congestion at the gateway. This enables the adaptive flow to attain a steady state and prevents it from getting penalized severely. However, non-adaptive flows will continue to send data at the same rate and thus suffer more losses.

Keeping token buckets for flows is different from maintaining an account of per-flow queue occupancy. The rate at which tokens are removed from the bucket of a flow is equal to the rate of incoming packets of that flow, but the rate of addition of tokens in a bucket depends on its permitted share of bandwidth and not on the rate at which packets of that particular flow are dequeued. *In this way a token bucket serves as a control on the bandwidth consumed by a flow.*

Another purpose of a token bucket is to keep a record of the bandwidth used by its corresponding flow in the recent past. This is because the token bucket occupancy increases if packets of a flow do not arrive for some time. Similarly, the bucket occupancy drops if packets of that flow are enqueued at a higher rate than the token addition rate. This is important for protocols such as TCP which leave the link idle and then send a burst in alternate periods. No packet of such a flow should be dropped if its arrival rate averaged over a

certain time interval is less than the permitted rate. The height of the bucket thus represents how large a burst of a flow can be accommodated. As such, this scheme does not penalize bursty flows unnecessarily.

For the simplest case, all buckets have equal weights and each token represents a packet [3] (see Fig. 2), the rate at which tokens are filled in a bucket is given by

$$R_N = C/N \text{ tokens per second,}$$

where C is the outgoing link capacity in packets per second and N is the number of *active* flows. We say that a flow is active as long as its corresponding bucket is not full. Note that this notion of active flows is different from that in the literature [5,10,14], namely, a flow is considered active when the buffer in the bottleneck router has at least one packet from that flow. However, we will see later, that despite the apparent difference in the definitions of an active flow, the two notions are effectively similar. See remark in Section 4.1 (5). The addition of tokens at the rate thus chosen ensure that the bandwidth is shared equally amongst all the flows and no particular flow gets an undue advantage.

A flow is identified as inactive whenever while adding tokens to that bucket, it is found to be full. The corresponding bucket is then immediately removed from the system. Its token addition rate is compensated by increasing the token addition rate for all other buckets fairly. Thus, the tokens of the deleted buckets are redistributed among other buckets.

The heights of all the buckets are equal and their sum is proportional to the buffer size at the gateway. Since the total height of all the buckets is conserved, during the addition of a bucket the height of each bucket decreases, and during the deletion of a bucket the height of each bucket increases. This is justified since if there are more flows, a lesser burst of each flow should be allowed, while if there are lesser number of flows, a larger burst of each should be permitted.

[3] For simplicity, we make the assumption that all packets have the same size.

During creation and deletion of buckets, the total number of tokens is conserved. This is essential since if excess tokens are created a greater number of packets will be permitted into the buffer resulting in a high queue occupancy while if tokens are removed from the system, a lesser number of packets will be enqueued which may result in lesser bursts being accommodated and also lower link utilization in some cases.

4. The algorithm

With reference to Figs. 1–3, we define the system constants and variables below.

Constants

- B: size of the buffer (in packets) at the gateway.
- α: this parameter determines the total number of tokens, T, in the system. We take $\alpha = 1$ throughout the paper.

$$T = \alpha \cdot B, \quad \alpha > 0.$$

- Probability profile f_p: it maps the bucket occupancy to the probability of dropping a packet when it arrives.

Global variables

- N: number of flows (equal to the number of buckets) in the system.
- L_N: maximum height of each bucket when there are N active connections in the system,

$$L_N = T/N.$$

- σ: variable to keep account of the excess or deficit of tokens caused at deletion and creation of buckets. During bucket addition and deletion, the total number of tokens in the system may temporarily deviate from its constant value T. σ is used to keep track of and compensate for these deviations.

Per-flow variables

- x_j: occupancy of the ith bucket in tokens, $0 \leqslant x_j \leqslant L_N$.

The generality of the algorithm allows any probability profile to be incorporated. An appro-

priate probability profile is however needed to detect congestion early and to notify the source by causing the packet to be dropped or marked (for ECN [8]). Fig. 3 shows the probability profile used in our simulations. It is similar to the gentle variant of RED drop probability profile [16]. We use the gentle variant of the RED probability profile as shown in Fig. 3:

$$f_p(x_i) = \begin{cases} 0, & \lambda_1 < x_i/L_N < 1, \\ \max_p\left(\dfrac{\lambda_1 - x_i/L_N}{\lambda_1 - \lambda_2}\right), & \lambda_2 < x_i/L_N < \lambda_1, \\ \max_p + (1 - \max_p)\left(\dfrac{\lambda_2 - x_i/L_N}{\lambda_2}\right), & \\ \quad 0 < x_i/L_N < \lambda_2. & \end{cases}$$

4.1. The SFED algorithm

The events that trigger actions at the gateway are:

1. Arrival of a packet at a gateway.
2. Departure of a packet from the gateway.

1. Upon arrival of a packet with flow id j. [4]
   - If the bucket j does not exist
     Create Bucket j (Section 4.1 (4)).
   - Drop packet with probability
     $p = f_p(x_j)$.
   - If packet not dropped
     $x_j = x_j - 1$.
     Enqueue packet in queue.
2. Departure of packet

The model shown in Fig. 2 is valid only for fluid flows where the flow of tokens into the buckets is continuous. However, the real network being discrete due to the presence of packets, the model may be approximated by adding tokens into the system on every packet departure since the packet dequeue rate is also equal to C. When there are no packets in the system, i.e., every bucket is full, no tokens will be added into the system as required. We explain the steps taken on every packet departure.

- $\sigma = \sigma + 1$
- if $(\sigma > 0)$
  distribute(σ). [5]

3. Token distribution

Tokens may be distributed in any manner to ensure fairness. One straightforward way is to distribute them in a round robin fashion as shown below.

- While $(\sigma \geqslant 1)$ find next bucket j
  $x_j = x_j + 1$
  $\sigma = \sigma - 1$
  if bucket j is full
  delete bucket j (Section 4.1 (5)).

4. Creation of bucket j

A bucket is created when the first packet of a previously inactive flow is enqueued thus increasing the flows to $N + 1$. The rate of filling tokens into each bucket is $R_{N+1} = C/(N + 1)$. A full bucket is allocated to every new flow. It is ensured that the total number of tokens in the system remain constant. The tokens required by the new bucket are compensated for by the excess tokens generated, if any, from the buckets that were shortened. Variable σ is maintained to compensate for the excess or deficit of tokens.

- Increment active connections to $(N + 1)$.
- Update height of each bucket to $L_{N+1} = T/(N + 1)$.
- $x_j = T/(N + 1)$.
- $\sigma = \sigma - L_{N+1}$.
- For every bucket $i \neq j$
  if $(x_i > L_{N+1})$
  $\sigma = \sigma + (x_i - L_{N+1})$
  $x_i = L_{N+1}$.

5. Deletion of bucket j

A bucket is deleted when a flow is identified to be inactive thus reducing the number of flows to $N - 1$. The rate of filling tokens into each bucket is increased to $R_{N-1} = C/(N - 1)$. It is obvious that the deleted bucket is full at the time of deletion. Variable σ is maintained to compensate for the excess tokens created so that the total tokens remain constant.

[4] Flow id is a 5-tuple of (Source IP address, Destination IP address, Source port number, Destination port number, Protocol) and characterizes a session uniquely.

[5] FABA attains higher efficiency by using randomization for this token distribution step.

- Decrement active connections to $(N-1)$.
- Update height of each bucket to $L_{N-1} = T/(N-1)$.
- $\sigma = \sigma + L_N$.

Remark. Note that by the nature of token distribution, if a packet is in position x from the front of the queue, the average number of tokens added to its bucket when the packet is dequeued is x/N. Therefore, when the bucket is full (and deleted), there is no packet in the buffer belonging to that flow. This is consistent with the existing definition of active flows.

We see that token distribution and bucket creation are $O(N)$ steps. Hence, in the worst case, both enqueue and dequeue are $O(N)$ operations. We now present the FABA algorithm which is $O(1)$ for both enqueue and dequeue operations. This extension makes the FABA algorithm scalable, and hence, practical to implement as compared to the SFED algorithm.

4.2. The FABA algorithm

We now propose the FABA algorithm that has $O(1)$ average time complexity for both enqueue and dequeue operations.

1. Arrival of a packet (flowid j)
   - If the bucket j does not exist
     Create Bucket j (Section 4.2 (4)).
   - If $x_j > L_N$ (to gather excess tokens)
     $\sigma += x_j - L_N$
     $x_j = L_N$.
   - Drop packet with probability
     $p = f_p(x_j)$.
   - If packet not dropped
     $x_j = x_j - 1$.
     Enqueue packet in queue.
2. Departure of packet

   We explain the steps taken on every packet departure.
   - $\sigma = \sigma + 1$.
   - *distribute*$(\sigma/Q + 1)$ where Q is the queue size of the bottleneck router (bounded by B) (Section 4.2 (3)).
3. Token distribution

   $\beta = \sigma/Q + 1$ buckets are accessed randomly. This means if we are short of tokens, i.e., $\sigma < 0$, then we grab tokens, else we add tokens. Any bucket which has more than L_N tokens is deleted.
   - for $i = 1$ to β do
     Choose a random bucket j
     if $\sigma > 0$
     $x_j = x_j + 1$
     $\sigma = \sigma - 1$
     else
     $x_j = x_j - 1$
     $\sigma = \sigma + 1$
     if $x_j > L_N$
     delete bucket j (Section 4.2 (5)).

   Note that σ may be negative but it is balanced from both sides, i.e., when it is negative we try to increase it by grabbing tokens from the buckets and when it is positive we add tokens to the buckets. The quantity β is the upper bound on the work done in the token distribution phase. But generally, and as also observed in our experiments, β is always close to 1.
4. Creation of bucket j
   - Increment active connections to $(N+1)$.
   - Update $L_{N+1} = T/(N+1)$.
   - $x_j = T/(N+1)$.
   - $\sigma = \sigma - L_{N+1}$.

   In FABA, we do not gather the excess tokens that might result when we decrease the height of the buckets. Instead, the excess tokens are removed whenever the bucket is next accessed. Hence, this procedure becomes $O(1)$.
5. Deletion of bucket j
   - Decrement active connections to $(N-1)$.
   - Update height of each bucket to $L_{N-1} = T/(N-1)$.
   - $\sigma = \sigma + x_j$.

   Every operation in the FABA algorithm is $O(1)$ in the amortized sense. This can be seen from the following observation. If K packets have been enqueued till now, then, the maximum number of tokens added to the buckets over successive dequeues is also K, implying $O(1)$ amortized cost for token distribution. All other operations such as bucket creation, deletion etc., are constant time operations.

Complexity of locating a bucket for a flow

The complexity of locating a bucket for a given flow, either 5-tuple or an aggregate is worth mentioning. For a packet arrival with flow id j, we check whether the corresponding bucket for that 5-tuple j exists or not. If a bucket does not exist, we create a new token bucket corresponding to the flow id j. Note that as in many other buffer management schemes proposed in the literature, we use a hashing function to match the incoming flow id with the existing flow ids (i.e., those flows whose buckets have already been created).

Operating in byte mode

FABA can be easily implemented in the byte mode where each token represents a byte. Whenever a packet is enqueued tokens equal to the size of the packet are subtracted from the corresponding bucket and whenever a packet is dequeued corresponding number of tokens are added to σ.

4.3. Space complexity of FABA algorithm

If the buffer size of the bottleneck router is B, then it is easy to see that the space complexity of FABA algorithm is $O(B)$. This can be argued as follows: the number of token buckets is equal to the number of active flows passing through the router and in the worst case, there are B active flows at the bottleneck buffer. We have already argued that by the nature of token distribution, if a packet is in position x from the front of the queue, the average number of tokens added to its bucket when the packet is dequeued is x/N. Therefore, when the bucket is active, it is highly likely that there is at least one packet in the buffer belonging to that flow.

5. Simulation results

We compare the performance of FABA with other AQM algorithms in different network scenarios. RED and CHOKe are $O(1)$ space algorithms and make use of the current status of the queue only to decide the acceptance of an arriving packet, whereas FRED keeps information corresponding to each flow. This makes FRED essentially $O(N)$ space. FABA also keeps one variable per active-flow. This extra information per active-flow is made use of to provide better fairness. All simulations in this paper are performed using the Network Simulator (NS) [15]. We use FTP application over TCP NewReno to model adaptive traffic and a Constant Bit Rate (CBR) source (over UDP) to model non-adaptive traffic. The packet size throughout the simulations is taken to be 512 Bytes. For RED and FRED, min_*th* and max_*th* are taken to be 1/4 and 1/2 of the buffer size, respectively. The value of $\max_p$ is 0.02 for RED, FRED and FABA. For FABA, the values of λ_1 and λ_2 are 1/2 and 1/4 respectively and α is equal to 1 throughout the paper. In the next subsection, we describe how the parameters of FABA affect the performance of the system and how their values should be selected.

5.1. Selection of parameters of the token bucket

In this section, we investigate what are the appropriate values for the parameters of the token bucket such as α, λ_2 and λ_1. We perform NS simulations and arrive at a good choice of these parameters. We study how these parameters affect the performance of the system and how their values should be selected.

We recall that α is the ratio of the total number of tokens in the system, T, divided by the buffer size at the gateway, B. λ_1 and λ_2 are the parameters of the token bucket. Fig. 3 shows the probability profile used in our simulations. It is similar to the gentle variant of RED drop probability profile [16]. We use the gentle variant of the RED probability profile as shown in Fig. 3. Note again that the probability profile $f_p(\cdot)$ is given by,

$$f_p(x_i) = \begin{cases} 0, & \lambda_1 < x_i/L_N < 1, \\ \max_p \left(\dfrac{\lambda_1 - x_i/L_N}{\lambda_1 - \lambda_2} \right), & \lambda_2 < x_i/L_N < \lambda_1, \\ \max_p + (1 - \max_p) \left(\dfrac{\lambda_2 - x_i/L_N}{\lambda_2} \right), & \\ \quad 0 < x_i/L_N < \lambda_2. & \end{cases}$$

In Fig. 4, we plot λ_2 versus the average bandwidth achieved by the CBR flow. λ_2 is varied from

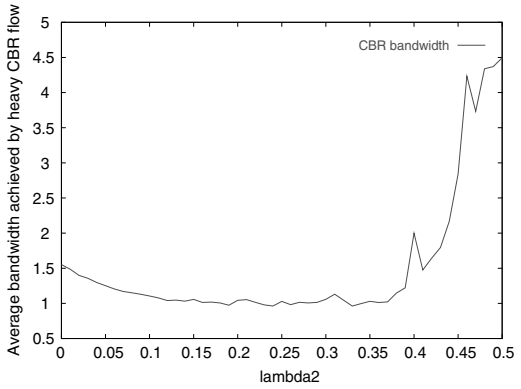


Fig. 4. λ_2 versus average bandwidth achieved by heavy CBR flow.

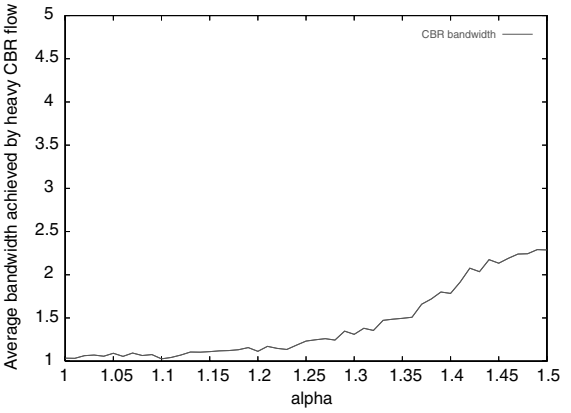


Fig. 5. α versus average bandwidth achieved by heavy CBR flow.

0 to 0.5. The simulation is performed for λ_2 varying from 0 to 0.5 in steps of 0.005 units, thus there are a total of 100 simulation points in Fig. 4. The value of λ_1 is twice the value of λ_2; λ_1 is therefore varied from 0 to 1. The network topology, parameters and the number of flows are the same for all the experiments. There are a total of 9 TCP flows competing with a 5 Mbps CBR flow. The bottleneck link capacity is 10 Mbps.

As λ_2 is varied from 0 to 0.5, we see that the bandwidth grabbed by the heavy CBR flow follows a function as seen in Fig. 4. We explain this as follows. For values of λ_2 close to 0.5 (i.e., when λ_1 is close to 1), the probabilistic packet dropping starts very early for each flow since the probability profile starts close to the top of each bucket. As soon as the TCP flows start sending at a significant steady rate, they see packet drops and thereby reduce their sending rates. This specifically affects TCP flows due to their bursty nature. On the other hand, when λ_2 is close to 0, the packets from TCP flows get enqueued as long as there are tokens in the respective buckets. However, when the tokens are sparse, the flow experiences bursty drops since there is not enough probabilistic packet dropping to indicate incipient congestion to the flow.

In both the extreme cases, the CBR flow is not affected much, whereas the TCP performance degrades. Thus, only when λ_2 is in between the extreme values of 0 and 0.5, that a flow can be notified of the incipient congestion early enough while still being able to absorb burts of packets. From Fig. 4, we infer that a value of λ_2 between 0.2 and 0.3 is appropriate for the token bucket since the average bandwidth achieved by the non-conformant CBR flow is the lowest in this range. We have kept λ_2 equal to 0.25 in the paper.

In the second experiment (see Fig. 5), we plot α versus the average bandwidth achieved by the CBR flow. α is varied from 1 to 1.5 for the same simulation setup as above. The value of λ_2 is kept equal to 0.25. As can be observed from the figure, the CBR flow is able to grab more bandwidth at higher values of α. This is because when α is greater than 1, the number of tokens in the system are greater than the buffer capacity. It is therefore possible that the buffer is full and there a large number of tokens in the buckets. The conformant flows (e.g., the TCP flows) therefore experience packet drops due to buffer overflow. This, however, does not induce CBR to reduce its sending rate whereas the TCP flows cut down their sending rate. This is the reason that the CBR flow is able to grab more bandwidth share.

It is to be noted that keeping α less than 1 would not be useful since it would be equivalent to keeping a buffer size of αB, which is less than B, and, no more than αB packets can be enqueued in the buffer.

From Fig. 5, we infer that the parameter α can be selected in a range between 1 and 1.2, and a value of α greater than 1.2 would unduly penalize

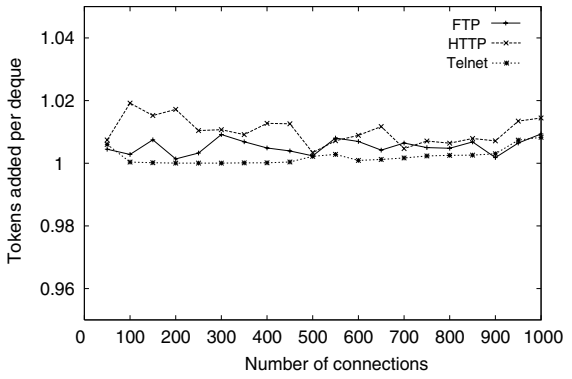


Fig. 6. Time complexity with the number of connections.

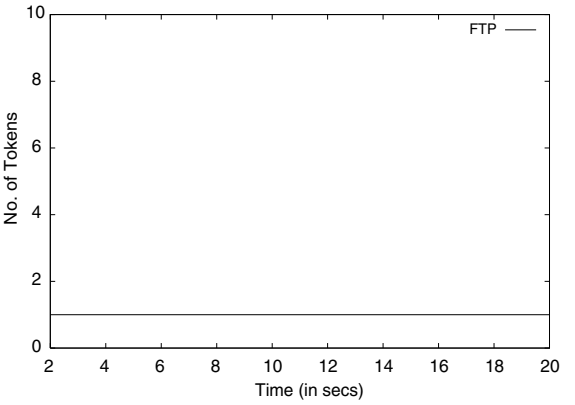


Fig. 7. Time complexity for 500 FTP connections.

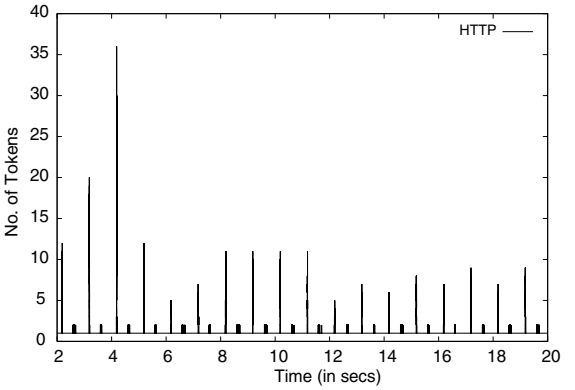


Fig. 8. Time complexity for 500 HTTP connections.

the conformant TCP traffic. [6] A value of α equal to 1 is therefore appropriate.

5.2. Time complexity of the proposed algorithm

The time complexity of FABA has two parts, namely creation or deletion of buckets and the number of tokens distributed to the buckets for every packet departure. The first two operations take constant time but the number of tokens distributed (step 2 of packet departure under *FABA*) is $\sigma/Q+1$. Over a sequence of packet departures, the amortised cost of a packet departure is O(1) (see Fig. 6) and therefore, it may be more appropriate to discuss the worst case for a single distribution step. Unless there are a large number of buckets created or deleted consecutively, the quantity $\sigma/Q+1$ is no more than two. This is borne out by Figs. 7 and 8 where the average number of tokens distributed is 1 almost everywhere.

As a further explanation, note that although the number of tokens distributed to the bucket at a single step, i.e. $\tau=\sigma/Q+1$ may be greater than 1, this can be charged to the τ packets that had departed earlier and that has each added a token to σ. Therefore, the total number of tokens distributed is no more than the total number of packets departing the system in the long run, yielding an O(1) amortized cost for token distribution.

5.3. Fair bandwidth allocation

The Internet traffic can broadly be classified into adaptive and non-adaptive traffic. An adaptive flow reduces its sending rate in response to indications of congestion in its network path. We show the performance of different queue management disciplines in providing fairness when adaptive traffic competes with non-adaptive traffic. The simulation scenario is shown in Fig. 9. The bottleneck link capacity is 10 Mbps. A CBR flow sends at 5 Mbps while 9 TCPs share the bottleneck link with the CBR flow. We denote by P_{bd}, the

[6] Throughout the paper, we use the terms conformant traffic and TCP-friendly traffic interchangeably; both refer to the TCP flows.

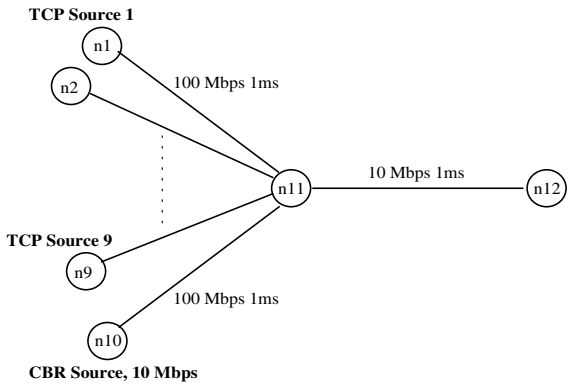


Fig. 9. Simulation scenario for Fig. 10.

bandwidth delay product of a single TCP connection, which is 78 packets in this example. The buffer size at the bottleneck link is set to $10P_{bd}$ since there are a total of 10 flows competing. In Fig. 10, we see how much bandwidth the heavy CBR flow can grab with different buffer management schemes. Since the bottleneck link capacity is 10 Mbps, the fair share of the CBR flow is 1Mbps. However, since CBR is always sending data at a higher rate and the TCP rates are not constant, the CBR flow gets at least its fair share of throughput i.e., 1 Mbps. We observe that FABA performs better in terms of fairness since it does not allow the bulky CBR flow to grab much more than its fair share of the link capacity.

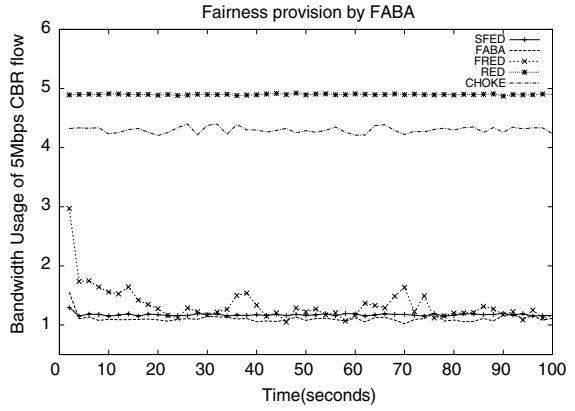


Fig. 10. Bandwidth allocated to heavy 10 Mbps CBR flow by different schemes.

5.4. Performance with varying buffer sizes

For a buffer management scheme to perform well, there should be enough buffering capacity available at the bottleneck gateway. We now see how well FABA performs with a variation in the buffer space. In the above simulation, the bandwidth delay product of all the flows combined is $P_{total} = 780$ packets. Table 1 shows the average bandwidth obtained by the 9 TCP flows combined as a fraction of their fair share with varying buffer space at the bottleneck link.

From Table 1, it is clear that FABA consistently performs better than RED, CHOKe and FRED across a wide range of buffer sizes. further, we observe that FABA performs almost as good as SFED. Since FABA has lower time complexity than SFED, it is only appropriate to study the performance of FABA rather than SFED.

5.5. Performance of FABA with different applications

We now study the fairness property of the FABA algorithm. We use the well known definition of fairness index [1]. If f is the fairness index, r_i is the throughput of connection i, and the total number of connections is N, then

$$f = \frac{\left(\sum_{i=1}^{N} r_i\right)^2}{N\left(\sum_{i=1}^{N} r_i^2\right)}.$$

We plot the fairness coefficient versus the number of connections and study three different applications—HTTP, Telnet and FTP, all of which use TCP as the transport layer protocol. The results in this section enable us to examine how our proposed algorithm scales with the number of connections. Note that HTTP and Telnet are examples of non-persistent traffic whereas FTP is a persistent application (or traffic). The FTP simulations have been performed using the standard NS-2 FTP application [15]. All the FTP connections spawned the entire duration of the simulation. Similarly, each Telnet source has been simulated using the standard NS-2 application/Telnet traffic generator. The average packet inter-arrival time for a single Telnet connection is

Table 1
Bandwidth with nine TCP flows at various buffer sizes as a fraction of their fair share

	$\frac{1}{10}P_{\text{total}}$	$\frac{1}{5}P_{\text{total}}$	$\frac{1}{2}P_{\text{total}}$	P_{total}	$2P_{\text{total}}$
RED	0.551	0.564	0.559	0.557	0.556
CHOKe	0.622	0.631	0.659	0.685	0.688
FRED	0.814	0.873	0.945	0.961	0.962
SFED	0.923	0.975	0.982	0.990	0.994
FABA	0.888	0.953	0.975	0.987	0.993

equal to 5 ms which corresponds to a sending rate of approximately 100 KBps.

In the simulation, we have a set of HTTP clients. Each client initiates several HTTP connections one by one, each connection being 5 s long. Hence, with a 100 second simulation time, each client performs 20 transfers. At the end, we collect the throughput obtained by each client and calculate the fairness coefficient. Note that a TCP source with HTTP application is simulated by a short-lived FTP application. The simulation topology is shown in Fig. 11.

Since the Internet traffic is predominantly HTTP, it is useful to study how well a buffer management algorithm performs with HTTP traffic. Fig. 12 shows how the fairness coefficient varies as the number of HTTP clients is increased. It can be seen that the fairness index is the largest (close to 1) with our proposed algorithm and is better than other AQM mechanisms. This is true even when the number of HTTP connections are large (equal to 1000).

We plot the fairness index versus the number of FTP connections in Fig. 13 and the fairness index versus the number of Telnet connections in Fig. 14. It can be seen that FABA performs consistently better than the other AQM mechanisms

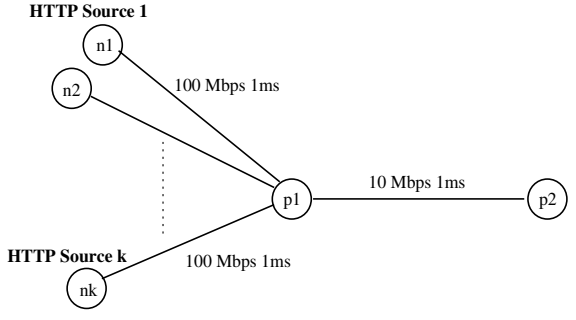


Fig. 11. The simulation topology.

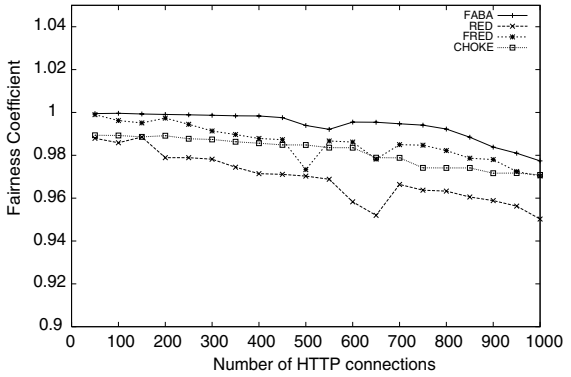


Fig. 12. Fairness coefficient versus number of HTTP connections for different AQM schemes.

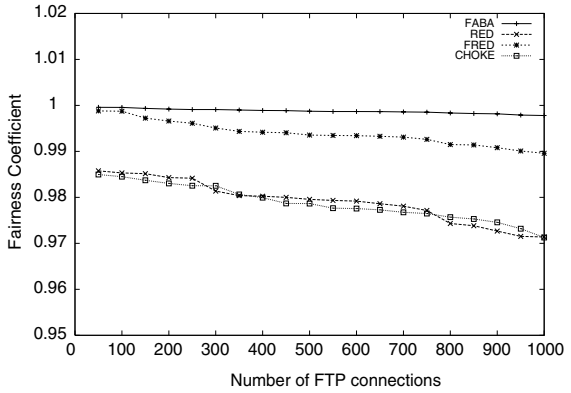


Fig. 13. Fairness coefficient versus number of FTP connections for different AQM schemes.

across a wide range of applications and the number of connections.

5.6. Protection for fragile flows

Flows that are congestion aware, but are either sensitive to packet losses or slower to adapt to

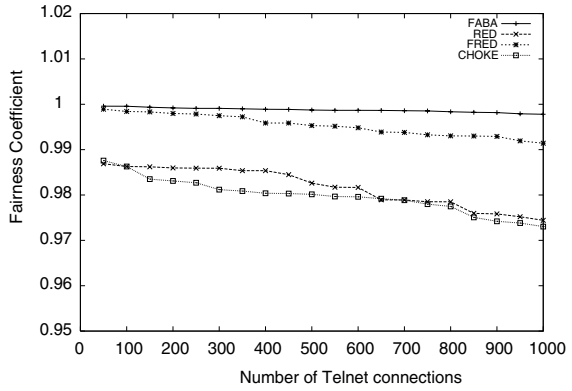


Fig. 14. Fairness coefficient versus number of Telnet connections for different AQM schemes.

more available bandwidth are termed fragile flows [10]. A TCP connection with a large round trip time (RTT) and having a limit on its maximum congestion window fits into this description.

We now evaluate the performance of FABA and other queue management disciplines with a traffic mix of fragile and non-fragile sources. The simulation scenario is shown in Fig. 15. The TCP source originating at node $n4$ is considered a fragile flow due to its large RTT of 40 ms, while the RTTs for the other flows is 6 ms. The CBR flow sends data at a rate of 40 Mbps. Since there are 4 flows in the system and the bandwidth of the bottleneck link is 45 Mbps, ideally each flow should receive its fair share of 11.25 Mbps. We vary the maximum allowed window size, w, for the fragile flow and observe the throughput obtained by this flow. The maximum achievable throughput is then given by $\gamma_{\max} = S(w/\mathrm{RTT})$ where S is the packet size and RTT is the round trip time. The maximum throughput is thus a linear function of

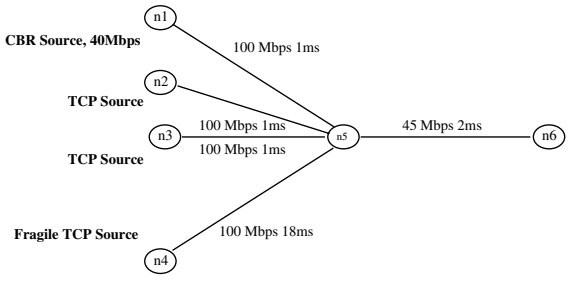


Fig. 15. Simulation scenario with a fragile flow.

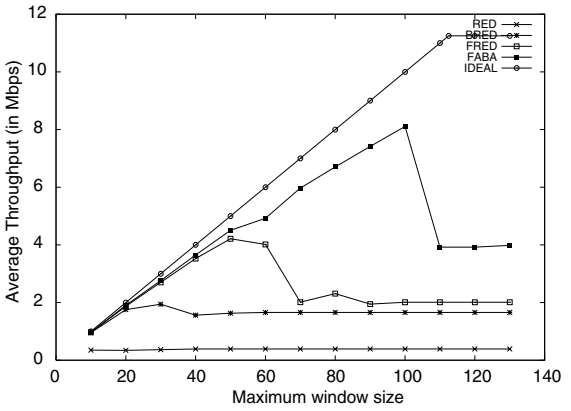


Fig. 16. Performance of the fragile flow with increasing receiver window constraint (gateway buffer size = 96 packets).

the maximum window size. Ideally, the maximum throughput should never exceed 11.25 Mbps, i.e., it should increase linearly until 11.25 Mbps, and should then stabilize at 11.25 Mbps. This ideal bandwidth share is shown in Fig. 16.

As can be observed from Fig. 16, FABA provides bandwidth allocation for the fragile flow almost as good as in the ideal case. For a small maximum window size, every algorithm is able to accommodate the bursts of the fragile flow without any drops, but with increasing maximum window size, packet drops result in drastic reduction of the fragile flow throughput. A packet drop is fatal for a fragile source as it is slow in adapting to the state of the network. We see that the throughput becomes constant after a while since the window size of the fragile source is not able to increase beyond a threshold. Therefore, no matter how large the maximum window size is increased beyond this threshold, the throughput does not increase and approaches a constant value. This constant value is much less than its fair share 11.25 Mbps due to the less adaptive nature of fragile flows.

5.7. Comparison of FABA with FRED: TCP flows with varying RTTs

In this section, we compare FABA with FRED and study two parameters, (i) the TCP goodput and (ii) the fairness index. We perform simulations with four TCPs with different round trip times.

Table 2
Goodput of four TCP Flows with different RTTs

	RTT = 10 ms	RTT = 20 ms	RTT = 50 ms	RTT = 200 ms	Fairness index
FRED	3401.94	2910.18	3641.26	3518.34	0.99325
FABA	4059.04	4109.35	4012.97	3605.87	0.99743

The RTTs of the TCP flows are 10, 20, 50 and 200 ms respectively. The four flows pass through a single bottleneck router (i.e., the congested router) as shown in Fig. 11.

From Table 2, we observe that the fairness index [1] of FABA is slightly higher than that of the FRED algorithm, although the difference is small. Note, however, that, for each RTT value, the TCP goodput obtained from FABA is greater than the goodput obtained from FRED. Upon calculation of the average over all 4 flows from the values listed in Table 2, we see that the total average TCP goodput in FRED is equal to 3367.93 (standard deviation is 277.50) wheras the total average TCP goodput in FABA is equal to 3941.81 (standard deviation is 199.77). One reason that the TCP throughput in FABA is higher than the TCP throughput in FRED is that there are fewer packet drops and fewer bursty drops in FABA. The reduction in the fraction of packet drops in FABA results in fewer TCP timeouts and hence a better TCP throughput.

In summary, from the simple experiment in this section, we conclude that FABA offers at least as good fairness and possibly better throughput than FRED. A detailed comparison of FABA with other buffer management algorithms in terms of QoS parameters such as packet delays and losses is a subject of future work.

6. Extensions of FABA

The FABA algorithm can be easily modified for scenarios such as weighted flows, differentiated services of aggregate flows and hierarchical flows.

6.1. FABA with weighted flows

FABA can be easily adapted for bandwidth sharing among flows with different weight requirements. Assuring fairness now means allocating bandwidth to each flow in proportion to its weight. By setting the bucket height and the share of token addition rate for each flow in proportion to its weight, the desired bandwidth allocation can be obtained.

We redefine the parameters for the weighted version of FABA as

- γ_i: weight of the ith flow,
- w_i: normalized weight of ith flow,

$$w_i = \gamma_i / \sum \gamma_i, \quad i \in \text{active connection},$$

- height L_i of bucket $i = w_i(\alpha \cdot B)$,
- token filling rate R_i for bucket $i = w_i(C)$.

6.2. FABA with aggregate flows

In the recent past, much work has gone into providing differentiated services [18] to flows or flow aggregates. Connections with similar properties and requirements can be aggregated and considered as a single flow. This simplification can then be used to advantage in providing QoS.

Aggregation can be done in many ways depending on the requirements. One way to aggregate connections is by the nature of traffic they carry. For example, all streaming audio and UDP traffic can be aggregated into one flow, all FTP, HTTP (i.e., Web traffic) in another while all Telnet, rlogin and similar interactive traffic in a separate flow.

FABA can be used effectively for bandwidth allocation among aggregate of flows. Multiple connections can be aggregated as a single flow in FABA and weighted according to their requirements. A set of connections, aggregated as a single flow has a single token bucket. This flow is then allocated bandwidth in proportion to its normalized weight, while the constituent connections share this allocated bandwidth.

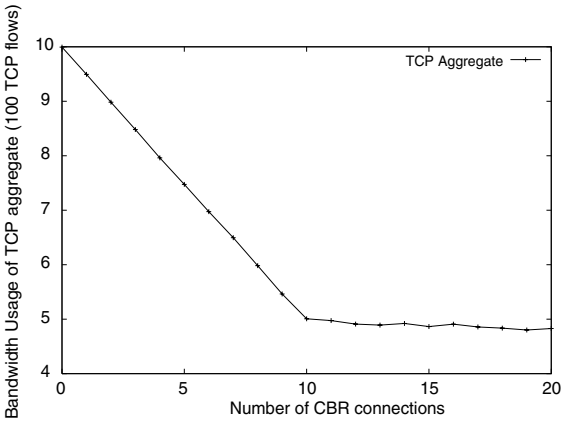


Fig. 17. Performance of FABA with flow aggregation.

Fig. 17 shows how the bandwidth allocated to an aggregate of 100 TCP connections changes as the sending rate of CBR connections increases. All the connections passing through the bottleneck link of 10 Mbps capacity are divided into 2 aggregate flows. Flow 1 consists of 100 TCP connections, while flow 2 consists of CBR connections each sending at 500 Kbps. The number of CBR connections in flow 2 is increased from 0 through 20. The collective bandwidth of CBR flows increases to 5 Mbps which is its fair share and then remains at nearly that value.

6.3. FABA with hierarchical flows

Besides simple aggregation of flows, it is also desired to have fair bandwidth sharing amongs hierarchical flows, i.e., multiple levels of flow distinction. For example, we would like to distinguish flows first on the basis of the traffic content and then amongst each kind of traffic, on the basis of different source and destination IP addresses or different source and destination port numbers (hence different applications). This is an example of a two level of hierarchy of distinguishing flows.

A hierarchical classification of flows can be represented by a tree structure where a leaf node represents an actual flow, while an interior node represents a class of flows. Each node has a weight attached to it such that the weight of each interior node is the sum of the weights of its children. If one of the flows is not using its full share of bandwidth, then the "excess" bandwidth should be allocated to its sibling flows, if possible. Otherwise this "excess" is passed on to its parent class for redistribution among its siblings and so on.

This sort of scenario is easily implementable using FABA. In such a hierarchy tree of flow classification, we assign a bucket each for the different leaves of the tree. The key distinction here is that at a flow addition or deletion the adjustment of tokens and heights would be done at a level closest to that flow and this redistribution will move upwards from the leaves to the root. The first level of redistribution of tokens of a flow would be at the same level, i.e., among its siblings. The overflow of tokens from this subtree will then be passed over to its sibling subtrees in the hierarchy and so on.

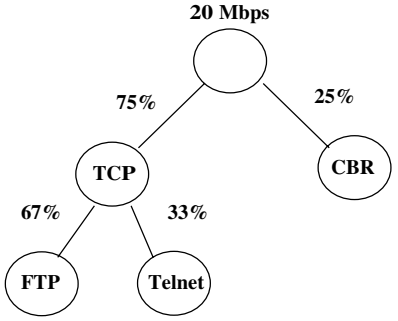


Fig. 18. The bandwidth sharing hierarchy tree.

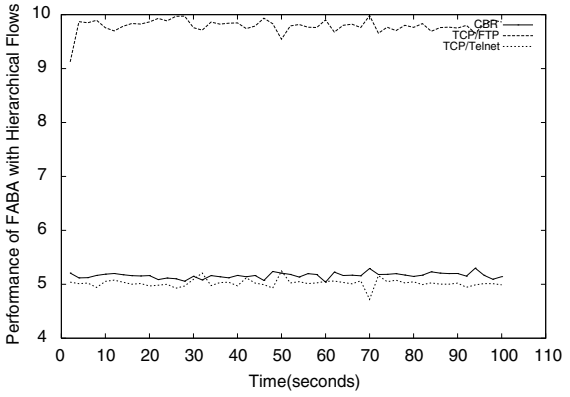


Fig. 19. Performance of FABA with hierarchical flows.

Fig. 19 shows the performance of FABA for a scenario for which the hierarchy tree for bandwidth sharing is shown in Fig. 18. The bottleneck link bandwidth is 20 Mbps. 25% of the bandwidth is allocated to the 400 CBR flows, each sending at 50 Kbps. One-third of the rest is allocated to the Telnet TCP traffic which consists of 400 Telnet connections sending at an average rate of 15 Kbps each. The remaining bandwidth is allocated to bulk FTP traffic consisting of 200 FTP connections.

In Fig. 19, we see the total bandwidth allocated to the three type of flows as time proceeds. The fair share of the CBR flows is one-fourth of the 20 Mbps bandwidth available, which is 5 Mbps. The Telnet Traffic should get a fair share of 1/3 of 75%, which is 5 Mbps. The remaining 10 Mbps should be allocated to the 200 FTP connections. It can be clearly seen that each type of traffic gets bandwidth close to its fair share.

7. Conclusion and future work

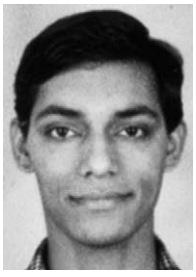
Motivated by a need for efficient AQM algorithms for the internet, we have proposed FABA, a rate control based AQM discipline that is well suited to network edges or gateways (e.g., the ISPs). FABA achieves a fair bandwidth allocation amongst competing flows with a mix of adaptive and non-adaptive traffic. It is a congestion avoidance algorithm with low implementation overheads. FABA can be used to partition bandwidth amongst different flows in proportion to preassigned weights. It is well suited for bandwidth allocation among flow aggregates as well as for bandwidth sharing within a hierarchy as required in the differentiated services framework. FABA can serve as a useful method for enforcing SLAs in the network. We showed that FABA is $O(1)$ in the amortized sense (and also in the worst case as noticed empirically), whereas the space complexity of FABA is $O(B)$ where B is the size (in packets) of the bottleneck buffer. Through simulations, we have compared FABA with other well known congestion avoidance algorithms and have seen that FABA gives superior performance. FABA is shown to give high values of fairness coefficient for diverse applications (e.g., FTP, Telnet, HTTP). The performance is superior even for a large number of connections (of the order of 1000) passing through the network edges; FABA is therefore a scalable algorithm.

A number of avenues for future work remain. It will be interesting to analytically model FABA. Note, however, that due to the closed-loop nature of the TCP traffic, any meaningful analysis of FABA becomes extremely complicated. In the literature, TCP interaction with even the simplest buffer management algorithm (e.g., RED) is hard to model [21]. We are currently exploring simple analytic techniques to study the performance of FABA. A preliminary analysis of SFED have been attempted in [17]. There arises a need to study FABA with different topologies (other than the dumbell topology assumed in this paper) and with a very large number of connnections (of the order of hundreds of thousands of flows). We are currently exploring the tradeoffs between time and space complexity for the FABA algorithm.

References

[1] L.L. Peterson, B.S. Davie, Computer Networks: A Systems Approach, Morgan Kaufmann, Los Altos, CA, 1996.
[2] D.C. Verma, Policy-Based Networking: Architecture and Algorithms, New Riders Publishing, Indianapolis, IN, USA, 2000.
[3] S. Keshav, An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network, Addison-Wesley, Reading, MA, 1997.
[4] J.F. Kurose, K.W. Ross, Computer Networking: A Top Down Approach Featuring the Internet, Addison-Wesley, Reading, MA, 2001.
[5] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993) 397–413.
[6] S. Floyd, V. Jacobson, Link sharing and resource management models for packet networks, IEEE/ACM Transactions on Networking 3 (4) (1995) 365–386.
[7] D.D. Clark, Explicit allocation of best-effort packet delivery service, IEEE/ACM Transactions on Networking 6 (1998) 362–373.
[8] S. Floyd, TCP and explicit congestion notification, Computer Communications Review 24 (5) (1994).
[9] S. Floyd, V. Jacobson, On traffic phase effects in packet switched gateways, Computer Communications Review (1991).

[10] D. Lin, R. Morris, Dynamics of random early detection, in: Proceedings of ACM SIGCOMM, 1997.
[11] V. Jacobson, M.J. Karels, Congestion avoidance and control, in: Proceedings of ACM SIGCOMM, 1988.
[12] I. Stoica, S. Shenker, H. Zhang, Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high speed networks, in: Proceedings of ACM SIGCOMM'98.
[13] R. Pan, B. Prabhakar, K. Psounis, CHOKe: a stateless active queue management scheme for approximating fair bandwidth allocation, in: Proceedings of IEEE INFOCOM'2000, Tel-Aviv, Israel, 26–30 March 2000.
[14] B. Suter, T.V. Lakshman, D. Stiliadis, A. Choudhury, Efficient active queue management for internet routers, in: Proceedings of INTEROP, Engineering Conference, 1998.
[15] S. McCanne, S. Floyd, ns-Network Simulator. Available from <http://www-nrg.ee.lbl.gov/ns/>.
[16] Available from <http://www.icir.org/floyd/red/gentle.html>.
[17] A. Kamra, S. Kapila, V. Khurana, V. Yadav, H. Saran, S. Juneja, R. Shorey, SFED: a rate control based active queue management discipline, IBM India Research Laboratory Research Report # 00A018, November 2000. Available from <http://www.cse.iitd.ernet.in/srajeev/publications.htm>.
[18] K. Nichols, Definition of differentiated services behavior aggregates and rules for their specification, Internet Draft, Differentiated Services Working Group, IETF.
[19] R. Guerin, S. Kamat, V. Peris, R. Rajan, Scalable QoS provision through buffer management, in: Proceedings of ACM SIGCOMM'1998.
[20] R. Guerin, L. Li, S. Nadas, P. Pan, V. Peris, Cost of QoS support in edge devices: an experimental study, in: Proceedings of IEEE Infocom'99, New York, March, 1999.
[21] P. Tinnakornsrisuphap, A.M. Makowski, Limit Behavior of ECN/RED gateways under a large number of TCP flows, in: Proceedings of IEEE INFOCOM 2003, San Francisco, CA, USA, April, 2003.



Abhinav Kamra is a Ph.D. student at the Department of Computer Science at Columbia University, New York. He works with Prof. Vishal Misra. He recieved his Bachelor of Technology (B.Tech) degree in Computer Science and Engineering from Indian Institute of Technology, Delhi in 2001 and his M.S. degree in Computer Science from Columbia University in 2003. His research interests include Game Theory, Performance Modeling, Design and Analysis of Algorithms for Communication Networks.

Huzur Saran received a B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Delhi, in 1983, and a Ph.D. degree in computer science from the University of California Berkeley in 1989. He has been on the faculty of the Department of Computer Science and Engineering at the Indian institute of Technology, Delhi since 1990. He has been Visiting Professor at the Information Systems Lab at Stanford and has on numerous occasions been a consultant to Bell Labs/AT & T Research, IBM India Research Labs. His research interests include wireless networking, Network performance analysis and modelling, scheduling and traffic management and efficient algorithms.

Sandeep Sen obtained his Ph.D. from Duke University, USA in 1989 and he is currently a Professor in the Department of Computer Science and Engineering in IIT Delhi. His primary research interests is in the area of Algorithms and Complexity.

Rajeev Shorey is a research staff member at the IBM India Research Laboratory, New Delhi since March 1998. He received the Bachelor of Engineering (B.E.) degree in Computer Science from the department of Computer Science and Automation, Indian Institute of Science, Bangalore, India in 1987. He received the M.S. and Ph.D. degrees in Electrical Communication Engineering from the Indian Institute of Science, Bangalore, India, in 1990 and 1996 respectively. Since March 1998, he is a Research Staff Member at the IBM India Research Laboratory, Indian Institute of Technology, New Delhi, India.

His research interests include wireless LANs and wireless PANs, Internet protocols, performance modeling and analysis of wireline and wireless networks. He has published numerous papers in international journals and conferences. He has to his credit one IBM US patent and around 8 US patents that are pending, all in the area of networking. He serves on the technical program committee of several international conferences in networking, namely, IEEE Infocom 2004 and IEEE ICC 2004. In the past, he was serving in the technical program committee for Infocom 2003 and Infocom 2002, Globecom 2002 and Globecom 2001 and ICC 2003. He is an adjunct faculty in the department of Computer Science and Engineering, Indian Institute of Technology, New Delhi where he actively teaches and guides undergraduate and graduate students. He is a senior member of IEEE.