Fair Adaptive Bandwidth Allocation: A Rate Control Based Active Queue Management Discipline

Abhinav Kamra, Huzur Saran, Sandeep Sen*, and Rajeev Shorey**

Department of Computer Science and Engineering,
Indian Institute of Technology,
Hauz Khas, New Delhi 110016, India
{saran, ssen, srajeev}@cse.iitd.ernet.in

Abstract. 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 active queue management 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. 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. 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 and ISPs.

1 Introduction

Active Queue Management (AQM) disciplines [4] are needed at intermediate routers since they provide protection for adaptive traffic from aggressive sources that try to consume more than their "fair" share, These schemes ensure "fairness" in bandwidth sharing and provide early congestion notification.

A typical Internet gateway is characterized by 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. Combinations of queue

* Supported in part by an IBM UPP award.

** Research Staff Member, IBM India Research Laboratory, New Delhi, Adjunct Faculty CSE Department, I.I.T, New Delhi.

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 838–849, 2002.

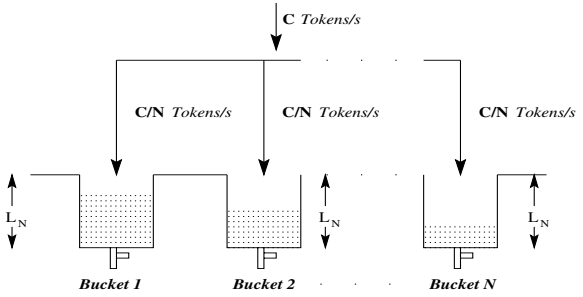

Fig. 1. FABA architecture for rate control using token buckets

management and scheduling disciplines can provide early congestion detection and notification and can be used in the differentiated services framework.

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, (ii) congestion avoidance by early detection and notification, (iii) low implementation complexity, (iv) easy extension to provide differentiated services.

In a recent work [6], the authors proposed 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.

FABA is an active queue management 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. Further, 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 [1], CHOKe [3], Flow Random Early Drop (FRED) [2], 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.

The paper is organized as follows. An overview of FABA is presented in Section 2. Section 2.1 explains our proposed algorithm in detail. We compare the performance of FABA with other buffer management algorithms namely RED, CHOKe and FRED with the help of simulations conducted using the ns2 [5] network simulator in Section 3. We summarize our results and discuss future work in Section 4.

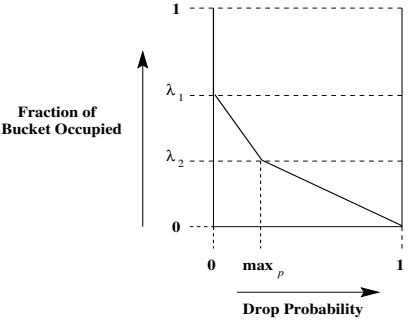

Fig. 2. Probability profile of dropping a packet for a token bucket

2 An Overview of FABA

Since FABA is an extension of SFED algorithm [6], for the sake of completeness, we begin with a description of SFED algorithm in this section. Selective Fair Early Detection (SFED) is an easy to implement rate control based active queue management 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 Figure 1. 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 Figure 2.

SFED ensures early detection and congestion notification to the adaptive source. A sending rate 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 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[1] (see Figure 1), the rate at which tokens are filled in a bucket is given by $R_N = C/N$ 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 [4,2,1], 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.

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. Similarly, 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. 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.

With reference to Figure 1 and Figure 2, we define the system constants and variables below.

Constants: B is the size of the buffer (in packets) at the gateway, α is the parameter that determines the total number of tokens, T, in the system. We assume $\alpha = 1$ throughout the paper and $T = \alpha B$, $\alpha > 0$. $\lambda_1, \lambda_2, \max_p$ are constants between 0 and 1 and are illustrated in Figure 2. The probability profile f_p maps the bucket occupancy to the probability of dropping a packet when it arrives.

A probability profile is needed to detect congestion early and to notify the source by causing the packet to be dropped or marked. Figure 2 shows the probability profile used in our simulations. It is similar to the gentle variant of RED drop probability profile (Figure 2).

$$f_p(x_i) = \begin{cases} 0 & \lambda_1 < \frac{x_i}{L_N} < 1 \\ max_p(\frac{\lambda_1 - x_i/L_N}{\lambda_1 - \lambda_2}) & \lambda_2 < \frac{x_i}{L_N} < \lambda_1 \\ max_p + & \\ (1 - max_p)(\frac{\lambda_2 - x_i/L_N}{\lambda_2}) & \\ & 0 < \frac{x_i}{L_N} < \lambda_2 \end{cases}$$

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

Global variables: N is the number of flows (equal to the number of buckets) in the system. L_N is the maximum height of each bucket when there are N active connections in the system, $L_N = T/N$. A variable σ keeps 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 is the occupancy of the *ith* bucket in tokens, $0 \leq x_j \leq L_N$.

2.1 The SFED Algorithm

The events that trigger actions at the gateway are (i) arrival of a packet at the gateway, (ii) departure of a packet from the gateway.
Arrival of a packet (flowid j): If the bucket j does not exist, we create bucket j. A packet is dropped with probability $p = f_p(x_j)$. If the packet is not dropped, $x_j = x_j - 1$, and the packet is enqueued.
Departure of packet: The model shown in Figure 1 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. The steps taken on each packet departure are: (i) $\sigma = \sigma + 1$, (ii) if $(\sigma > 0)$ distribute(σ). [2]
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 follows. As long as $(\sigma \geq 1)$, we find next bucket j, $x_j = x_j + 1$, $\sigma = \sigma - 1$, and if bucket j is full, we delete bucket j.
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. The following are the steps: (i) Increment active connections to $(N + 1)$, (ii) Update height of each bucket to $L_{N+1} = T/(N + 1)$, (iii) $x_j = T/(N + 1)$, (iv) $\sigma = \sigma - L_{N+1}$, (v) For every bucket $i \neq j$, if $(x_i > L_{N+1})$, $\sigma = \sigma + (x_i - L_{N+1})$, $x_i = L_{N+1}$.
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. The steps are:

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

(i) decrement active connections to $(N-1)$, (ii) update height of each bucket to $L_{N-1} = T/(N-1)$, (iii) $\sigma = \sigma + L_N$.
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 $\frac{x}{N}$. 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.

2.2 The FABA Algorithm

We now propose the FABA algorithm that has O(1) average time complexity for both enqueue and dequeue operations.
Arrival of a packet (flowid j): If the bucket j does not exist, create bucket j. If $x_j > L_N$ (to gather excess tokens), $\sigma += x_j - L_N$, $x_j = L_N$. Drop the packet with probability $p = f_p(x_j)$. If the packet is not dropped, $x_j = x_j - 1$, enqueue packet in the queue.
Departure of packet: The steps taken on a packet departure are (i) $\sigma = \sigma + 1$, (ii) Let $\beta = \max\left(1, \frac{\sigma}{Q+1}\right)$, where Q is the queue size after the packet departure, (iii) *distribute*(β).
Token distribution: $\beta = \frac{\sigma}{Q+1}$ buckets are accessed randomly. Now instead of always adding tokens, we try to keep the number of spare tokens (σ) as close to 0 as possible. This means if we are short of tokens, i.e., $\sigma < 0$, then we grab tokens from the buckets, else we add tokens. Any bucket that has more than L_N tokens is deleted. From a number between 1 to β (including 1 and β), choose a random bucket j. If $\sigma > 0$, $x_j = x_j + 1$ and $\sigma = \sigma - 1$, else, $x_j = x_j - 1$, $\sigma = \sigma + 1$. If $x_j > L_N$, we delete bucket j.

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.

Creation of bucket j: The steps associated with this are (i) increment active connections to $(N+1)$, (ii) update $L_{N+1} = T/(N+1)$, (iii) $x_j = T/(N+1)$, (iv) $\sigma = \sigma - L_{N+1}$.

In FABA, we do not gather the excess tokens that might result in the buckets when we decrease the height. Hence, this procedure becomes O(1). Instead these excess tokens are removed whenever the bucket is accessed next.

Deletion of bucket j: The steps associated with bucket deletion are (i) decrement active connections to $(N-1)$, (ii) update height of each bucket to $L_{N-1} = T/(N-1)$, (iii) $\sigma = \sigma + x_j$.

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.

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 leaky 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 $\frac{x}{N}$. Therefore, when the bucket is active, it is highly likely that there is at least one packet in the buffer belonging to that flow.

3 Simulation Results

We compare the performance of FABA with other active queue management 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 are done using Network Simulator (ns) [5]. We use FTP over TCP NewReno to model adaptive traffic and a Constant Bit Rate (CBR) source 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. The values of λ_1 and λ_2 are 1/2 and 1/4 for FABA. All values chosen for the algorithms correspond to those recommended by the respective authors.

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 (Section 2.2) is $\frac{\sigma}{Q+1}$. Over a sequence of packet departures, the amortized cost of a packet departure is $O(1)$, 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 $\frac{\sigma}{Q+1}$ is no more than two (see [7]). We have seen that the average number of tokens distributed is 1 almost everywhere.

3.1 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

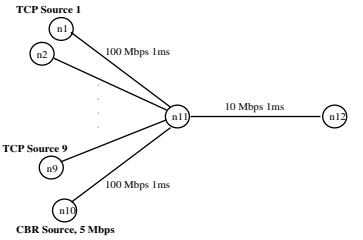

Fig. 3. The Simulation Topology

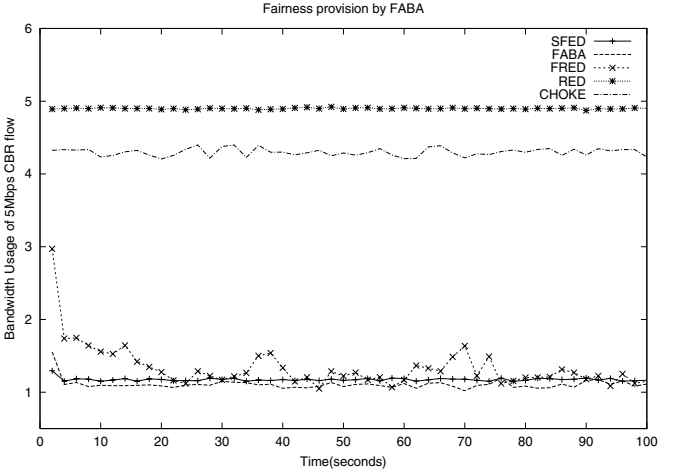

Fig. 4. Bandwidth allocated to a heavy (10 Mbps) CBR flow by different schemes

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 Figure 3. 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 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 Figure 4, 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.

Table 1. Bandwidth with 9 TCP flows at various buffer sizes as a fraction of their fair share

	$\frac{1}{10}P_{total}$	$\frac{1}{5}P_{total}$	$\frac{1}{2}P_{total}$	P_{total}	$2P_{total}$
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

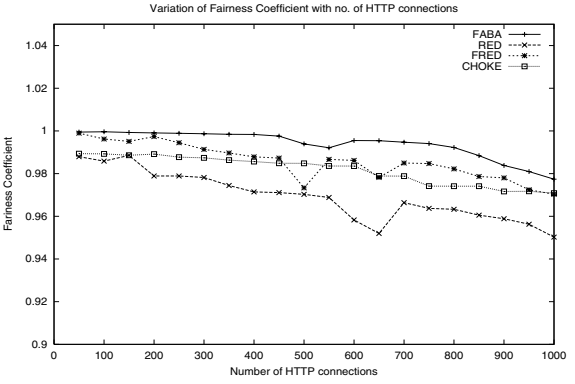

Fig. 5. Fairness coefficient versus number of HTTP connections for different AQM schemes

3.2 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 I 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 I, 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.

3.3 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. If f is the fairness index, r_i is the throughput of connection i, and the total number of connections is N, then $f = \frac{\left(\sum_{i=1}^{N} r_i\right)^2}{N\left(\sum_{i=1}^{N} r_i^2\right)}$.

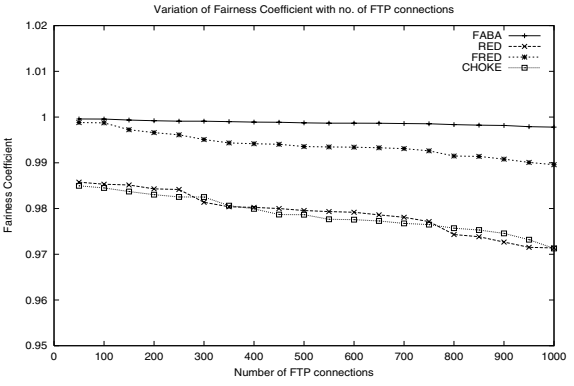

Fig. 6. Fairness coefficient versus number of FTP connections for different AQM schemes

We plot the fairness coefficient versus the number of connections and study three different applications (HTTP, TELNET, 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.

In the simulation, we have a set of HTTP clients. Each client initiates several HTTP connections one by one, each connection being 5 seconds long. Hence, with a 100 second simulation time, each client performs 20 HTTP transfers. At the end, we collect the throughput obtained by each client and calculate the fairness coefficient. The simulation topology is shown in Figure 3 with the difference that all the flows are now TCP flows.

Since the Internet traffic is predominantly HTTP, it is useful to study how well a buffer management algorithm performs with HTTP traffic. Figure 5 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 Figure 6. The fairness index versus the number of TELNET connections follows a behaviour similar to that seen in Figure 6. FABA performs consistently better than the other AQM mechanisms across a wide range of connections.

3.4 Protection for Fragile Flows

Flows that are congestion aware, but are either sensitive to packet losses or slower to adapt to more available bandwidth are termed fragile flows [2]. A TCP connection with a large round trip time (RTT) and having a limit on its maximum congestion window fits into this description.

We study FABA and other AQM algorithms with a traffic mix of fragile and non-fragile sources. The simulation scenario is shown in Figure 7. The TCP

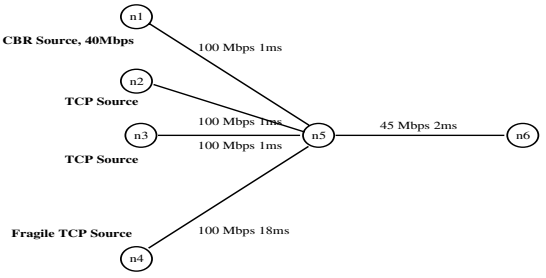

Fig. 7. Simulation scenario with a fragile flow

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 $\gamma_{max} = S(w/RTT)$ where S is the packet size and RTT is the round trip time. The maximum throughput is thus a linear function of 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 Figure 8.

As can be observed from Figure 8, 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 thus 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.

4 Conclusion and Future Work

We have proposed Fair Adaptive Bandwidth Allocation (FABA), a rate control based active queue management discipline that is well suited for the network edges or Internet 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 pre-assigned weights. It is well suited for bandwidth allocation among flow

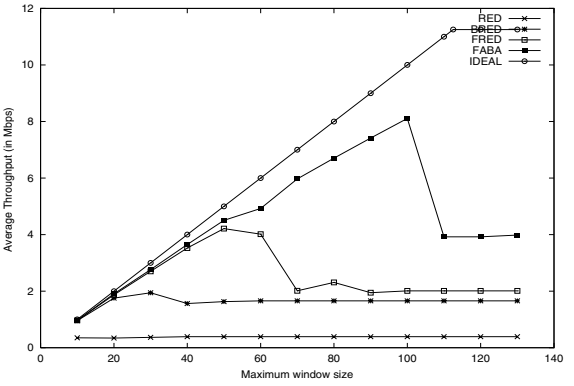

Fig. 8. Performance of the fragile flow with increasing receiver window constraint (gateway buffer size = 96 packets)

aggregates as well as for bandwidth sharing within a hierarchy as required in the differentiated services framework (see [7]). 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 (FTP, TELNET, HTTP).

A number of avenues for future work remain. It will be interesting to analytically model FABA. We need to study FABA with different topologies 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. Floyd, S., Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, Vol. 1, No. 4, August 1993.
2. Lin, D., Morris, R.: Dynamics of Random Early Detection, In Proceedings of ACM SIGCOMM, 1997.
3. Pan, R., Prabhakar, B., Psounis, K.: CHOKe: A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation, In Proceedings of IEEE INFOCOM'2000, Tel-Aviv, Israel, March 26-30, 2000.
4. Suter, B., Lakshman, T.V., Stiliadis, D., Choudhury, A.: Efficient Active Queue Management for Internet Routers, Proc . INTEROP, 1998 Engineering Conference.
5. McCanne, S., Floyd, S.: ns-Network Simulator, http://www-nrg.ee.lbl.gov/ns/
6. Kamra, A., Kapila, S., Khurana, V., Yadav, V., Saran, H., Juneja, S., Shorey, R.: SFED: A Rate Control Based Active Queue Management Discipline, IBM India Research Laboratory Research Report # 00A018, November, 2000. http://www.cse.iitd.ernet.in/∼srajeev/publications.htm
7. Kamra, A., Saran, H., Sen, S., Shorey, R.: *Full version of this paper*, http://www.cse.iitd.ernet.in/∼srajeev/publications.htm