AMES: A Framework For Fair Bandwidth Sharing

Srinivasan Ramasubramanian, Sathya Parathasarathy
Dept. of Electrical and Computer Engineering
University of Arizona, Tucson, AZ
srini@ece.arizona.edu, sathya.parthas@gmail.com

Arun K. Somani
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA
arun@iastate.edu

***Abstract*—Fair bandwidth sharing has been an active research area since the early days of networking. Today's networks employ packet dropping as the primary mechanism for reacting to congestion, while transport layer protocols are designed to adapt their rates based on the observed packet losses. Due to the increasing transmission speed and a disproportionate decrease in the end-to-end delay, future networks are expected to have a high bandwidth-delay product as compared to the networks of today. The cost of a packet retransmission increases with the increase in the bandwidth-delay product and decrease in buffer sizes. Hence, there is a need to develop new techniques, for a future clean-slate network design, to achieve fair bandwidth sharing that do not primarily rely on packet dropping.**

To this end, this paper develops a framework for fair bandwidth sharing called Access Mechanism for Efficient Sharing (AMES), where every core router in the network employs link-specific queuing, round-robin scheduling, and reacts to congestion by restricting the adjacent routers from transmitting, rather than dropping packets. The core-routers do not maintain any per-flow information, nor perform any flow-specific operation. We demonstrate the link utilization and fairness characteristics of the AMES framework through extensive simulations and compare it to the Core Stateless Fair Queuing (CSFQ) architecture developed in the literature.

I. Introduction

Fair bandwidth sharing has been an active research area since the early days of networking. With the network scale as large as the Internet, it is believed that it is often impractical to achieve fair bandwidth sharing without any support from the underlying packet scheduling algorithms. Several packet scheduling algorithms have been developed in the literature to offer various levels of fair sharing of bandwidth available on a link.

A. Prior Work and Motivation

Initial proposals for fair bandwidth allocation in the Internet were based on *per-flow* queuing mechanisms, e.g. Generalized Processor Sharing [1], Fair Queuing [2], Worst-case Fair Weighted Fair Queuing (WF^2Q) [3], Self-Clocked Fair Queuing (SCFQ) [4], Deficit Round Robin [5], and several of their variations [6], [7], [8]. These approaches mandate the routers to maintain per-flow state information, hence are not widely accepted due to scalability concerns.

Recent efforts to provide a scalable packet switching framework for the Internet include Stateless Core (SCORE) architecture [9] that achieves the level of QoS as obtained in the IntServ architecture with the complexity of DiffServ implementation. Core Stateless Fair Queuing (CSFQ) [10] is a realization of fair queuing algorithm employed in SCORE, where every packet of a flow is tagged with the rate of the flow at the edge routers. Several variations to CSFQ have been proposed to enhance the performance under various scenarios [11], [12], [13], [14], [15], [16]. Similarly, rainbow fair queuing (RFQ) [17] and Aggregated Flow Fair Queuing (AFFQ) [18] employ layer number and number of active flows, respectively, in their header to achieve fair sharing. Despite their fairness properties, these algorithms face stronger resistance in wide acceptance as they require significant changes to the header information in the existing protocols. In addition, the information carried in the packet reflect either the transmission speed of the network or the number of flows sharing the link. Thus, the number of bits employed to encode this information will have significant effect on the performance of the protocols.

Packet dropping, bandwidth measurement, and rate adaptation complement fair packet scheduling for congestion control. Among packet dropping mechanisms, Random Early Detection (RED) and its variants [19], [20], [21], [22] are used widely in the Internet today. Routers assist in congestion detection through Explicit Congestion Notification (ECN) [23]. Several researchers have attempted to estimate the bandwidth available on a path [24], [25], [26], [27], [28], [29], [30]. Variations to basic TCP rate adaptation technique [31] have been proposed and their stability and performance have been analyzed [32], [33], [34], [35], [36], [37]. The bandwidth estimation techniques are rendered meaningless due to large variation in the measurements as the assumptions made by these techniques are not met by the packet switching layer. For example, the packet-pair technique [25] transmits two packets together and measures the difference in arrival times of the packets at the destination. Unfortunately, no packet switching framework other than per-flow queuing and deficit-round-robin (DRR) scheduling satisfies such a requirement.

Packet dropping is the most widely used mechanism for congestion control in the Internet. While TCP adapts its rate based on roundtrip time and dropped packets, UDP does not. Therefore, achieving fair sharing among TCP and UDP flows when they share a common link is a difficult task. If UDP flows are transmitting at a much higher rate than their fair share, the throughput of TCP flows that share a link with the UDP flows will suffer. In addition, when different flows transmit at different rates, random dropping of packets does not necessarily achieve fair sharing. Thus, packet dropping mechanisms based on flow characteristics (such as Flow Random Early Detection, FRED) were developed. However, such techniques require maintenance of per-flow information

necessitating increased computational power at the routers.

The processing power of the routers have increased tremendously in the recent past that most routers today can process packets at line-speed. The increased processing power has significantly reduced the queueing delay encountered by packets. In addition, advances in transmission technologies have led to increased transmission speeds, thereby reducing packet transmission delay. Thus, the primary component of the end-to-end delay in the future networks will be the propagation delay. Given that the backbone networks of today employ optical transmission technologies, the propagation delay on a link has reached the physical limit of the velocity of light in the propagation medium. Thus, increasing transmission speed at constant propagation delay per unit distance, resulting in an increasing bandwidth-delay product, is the key characteristic trend of future network evolution. Together with decreasing buffer size relative to the bandwidth-delay product of a link, the combination of packet dropping and loss-based rate adaptation at the transport layer may not be the optimal strategy to achieve fair bandwidth sharing.

Secondly, all existing approaches to fair packet scheduling thus far treat a packet (a network-layer characteristic) as the basic unit of service in the network. However, fair bandwidth sharing is desired at the flow level (a transport-layer characteristic). One approach to achieve flow-level fairness at the packet layer is to explicitly count the number of flows in the network and employ flow-specific operations at each router. For example, the ERICA [38] technique developed for fairness among available bit-rate (ABR) traffic in ATM networks falls into this category. In ERICA, every switch maintains a table that indicates if a cell from a virtual connection has been seen in a cycle. The sum of these entries within a time duration (cycle) indicates the number of flows traversing the switch. While a flow in an ATM network is easily identified by the virtual circuit (VC) identifier, identifying a flow in a TCP/IP network is difficult[1]. Irrespective of how a flow is identified, maintaining flow-specific information at every router is tedious and does not scale with network size.

Based on the above observations, we seek to develop a framework to achieve fair bandwidth sharing among flows that may or may not employ rate adaptation techniques that has: (1) low scheduling complexity; (2) requires less buffering; (3) does not perform flow-specific operations; and (4) does not employ packet dropping as the primary mechanism for reacting to congestion. We believe that such a framework is critical to the clean-slate design of future large-scale networks with high bandwidth-delay product. In addition, such a framework is also highly desirable in enterprise data-center networks, as these networks are: (a) small-scale, (b) traffic sources belong to the enterprise, and (3) there is often no undesired traffic that is being routed.

[1]A flow may be classified based on: (a) source address; (b) source-destination address pair; or (c) source-destination address and port numbers.

B. Contribution and Organization

This paper develops a new packet switching framework called Access Mechanism for Efficient Sharing (AMES) that transmits, buffers, and forwards packets in *rounds*. The contributions of the paper are: (a) the concept of transmitting and buffering of packets in rounds; (b) a scheduling algorithm that transmits a round of packets at each output link, with support for variable length packets and different flow priorities; and (c) computing the bandwidth available for a flow along a path, based on the characteristics of the round at every router.

The remainder of the paper is organized as follows: Section II describes the network model, the concept of transmission of packets in rounds which forms the central point of the framework, and provide a detailed description of the AMES scheduling algorithm with illustrative examples to demonstrate the flexibility of the framework in achieving weighted fair sharing. The effectiveness of the framework is evaluated under various scenarios involving TCP and UDP traffic streams and employing a simple rate-based flow control algorithm employed at end hosts. The experimental setup and simulation results, comparing the performance of AMES architecture to that of CSFQ, are presented in Section III. Section IV concludes the paper.

II. AMES Network Architecture and Scheduling

We consider a network architecture consisting of an island of routers (edge and core routers) in which all routers employ the AMES architecture developed in this paper. An example network is shown in Figure 1. All routers in the network employ link-specific queueing. This implies that only edge-routers perform per-flow queuing. The core-routers do not maintain any per-flow state information, nor perform any flow-specific queuing.

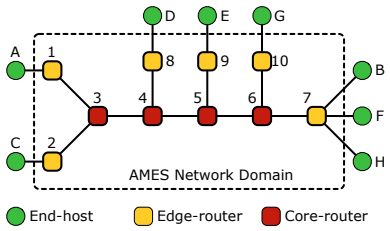


Fig. 1. A network with an island of routers employing the AMES framework.

A. AMES Concept

The central point of the AMES framework is the concept of transmitting packets in rounds. A router receives rounds of packets from every link. The routers are assumed to employ link-specific queuing at every output link. The packets from link i that are destined to an output link j are queued in the buffer corresponding to link i at output j. Packets that arrive at an output link from the same input link on different rounds must be *identifiable* when stored in the buffers. Therefore, the packets are stored in the buffers in rounds. The management of rounds during forwarding and buffering may be achieved

in several ways, one of which is discussed in Section II-B. At each output link, a deficit round-robin scheduler is employed. In each pass, the scheduler selects a round of packets from each eligible queue. The selected packets are then transmitted on the output link as a round of packets. The concept of rounds is employed at every router and end-host independently, hence does not require global coordination. An end-host always transmits one packet in a round.

Consider the example network shown in Figure 1. For the sake of clarity, assume for now that all flows transmit at link rate and have fixed-size packets. Consider four flows from A to B, C to D, E to F, and G to H. The snapshot of the network illustrating the transmissions at various routers are shown in Figure 2. The transmission of packets on links A–1 and 1–3 involves one packet in each round. Similarly, the transmission on links C–2 and 2–3 will contain one packet per round. The packets arriving at router 3 from these two links are buffered in their respective queues. The scheduler on link 3–4 selects a round of packets from each input queue and merges them into a round and transmits them on the outgoing link. The transmission on link 3–4 will therefore have two packets in each round. At router 4, the two flows are split across two output links 4–5 and 4–8 with the transmission on each link having one packet per round. At router 5, packets from link 4–5 is merged with that from link 9–5, thus resulting in two packets in a round on link 5–6. At node 6, packets from link 10–6 merges with that from link 5–6 to have three packets in a round on link 6–7.

Observe that the concept of rounds enables router 6 to identify the number of packets to transmit from the queue corresponding to link 5–6 for every packet transmitted from the queue corresponding to link 10–6. Thus, when all flows have equal packet sizes, the scheduler at the output link behaves like a round-robin scheduler selecting a round of packets from each queue. As the number of packets in each queue will have at most one packet per flow, the bandwidth at the output link is shared on a per-flow manner, without explicitly tracking the number of flows.

B. Management of rounds

The concept of round is required to identify a group of packets to be selected for transmission. One approach to manage rounds is to introduce round numbers in the packets. As the need for round is only to distinguish one round from another (or identify the round boundaries), it is sufficient to have only one bit to indicate the round. We employ ODD and EVEN rounds and the packets are transmitted in either one of them. Similarly, the packets are also buffered in either one of the rounds. However, the round in which the packet is transmitted may be different from the round in which the packet is buffered. Note that the rounds employed for buffering are maintained independently from that of transmission.

C. Detailed Working

We describe in detail the scheduling algorithm employed at each router, assuming variable sized packets. We describe the working of the algorithm at a specific output link at a router.

The processing of packets received on a specific link involves two steps as shown in Figure 3. The first step is to identify if there is a change of round in transmission. A change in the round number of the currently received packet from that of the previous packet received on the same link indicates a change in the transmission round. Whenever a change in transmission round is detected, the packets following them will have to be queued in a new round at the respective output. Note that an output queue may not have buffered any packet in the previous round. If only one-bit round number is employed, it has to be ensured that two rounds of packets are not merged during buffering. Therefore, the buffering round number is changed only when at least one packet was queued in the previous round.

1) **If change of round is detected, then invoke change of buffering round in all the corresponding queues.**
   Comment: A change in round value from the previously received packet on that link will indicate a change of round. The buffering round for a queue is changed when at least one packet has been buffered in the previous round.
2) **If the packet has to be forwarded, buffer the packet in the corresponding output queue.**

Fig. 3. Steps involved in processing packets received on a link.

The routers employ link-specific queueing at every output link. A counter that indicates the service given to each queue, denoted by $ServiceReceived_q$ for a queue q, is maintained at the output. If the value of this metric for a queue is greater than zero, then it indicates that this queue has been serviced adequately. Otherwise, the queue is eligible for service. Let $MinimumPacket_q$ denote the minimum packet size of the round of packet selected from queue q when it is eligible for service. Let $MinimumService$ denote the minimum service that is received by an eligible queue. This minimum service is also the minimum packet size among those packets that were selected from all the eligible queues.

Figure 4 shows the steps involved in AMES scheduling with comments on the need for each step. The scheduler operates in a round-robin fashion visiting every queue once. At the beginning of every new round-robin pass, the transmission round is changed. The change of transmission round is similar to that of change of buffering round. The value of the one-bit round field is toggled provided at least one packet was transmitted in the previous round.

We deduct the service provided by the scheduler in the previous round, denoted by $ServiceProvided$, from $ServiceReceived_q$ for every queue q. If $ServiceReceived_q > 0$, i.e., the queue is eligible for service, then a round of packets is selected from the queue and transmitted in the current transmission round. The amount of service received by the queue is incremented with the size of the minimum packet among those selected packets from the queue. During every pass, the minimum packet size across the packets selected from different queues is updated in the variable

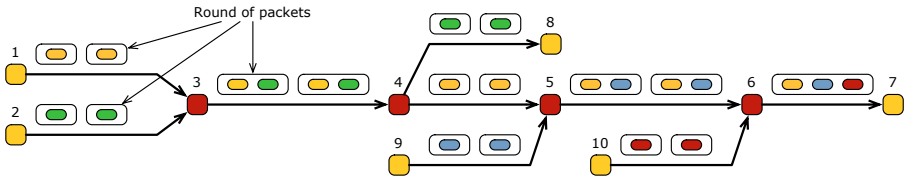


Fig. 2. Snapshot of the network operation illustrating the transmission of packets in rounds on various links.

1) **Change transmission round.**
Comment: The transmission round is toggled between ODD and EVEN. This operation is performed when at least one packet is transmitted in the previous round.

2) **Initialize the variables that are to be used in every pass of the round-robin scheduling.**
Comment: MinimumService = 0;

3) **For every queue** q **do**
   a) $ServiceReceived_q \leftarrow ServiceReceived_q - ServiceProvided.$
   Comment: Deduct the minimum service provided across all queues in the previous round.
   b) **If queue** q **is not empty and** $Service_q \leq 0$ **then**
      i) **Select a round of packet from the queue.**
      ii) **Transmit the selected packets in the current transmission round.**
      Comment: If the one-bit overhead approach is employed, then the round field in the packet is set to the current transmission round before transmission.
      iii) $ServiceReceived_q \leftarrow ServiceReceived_q + MinimumPacket_q.$
      Comment: $MinimumPacket_q$ is the minimum packet size among those packets that were selected for service.
      iv) **If** $(MinimumService = 0)$ **or** $(MinimumPacket_q < MinimumService)$ **then** $MinimumService = MinimumPacket_q.$
      Comment: Compute the minimum service provided across all queues during the round. The minimum service provided in current round is the minimum packet size across all the queues from which a round of packet has been selected for transmission.
   c) **Else if queue** q **is empty and** $ServiceReceived_q < 0$ **then**
      i) $ServiceReceived_q \leftarrow 0.$
   d) **Else if** $ServiceReceived_q > 0$
      i) **If** $(MinimumServiceReceived = 0)$ **or** $(ServiceReceived_q < MinimumServiceReceived)$ **then** $MinimumServiceReceived = ServiceReceived_q;$
      Comment: If the queue is not empty, but has a positive value of received service, then, the queue is not selected for service. However, if no other queues are selected for service as well, then the minimum received service among those backlogged queues will be assumed to have been serviced in the current pass.
4) **If** $(MinimumService > 0)$ **then** $ServiceProvided = MinimumService;$
else $ServiceProvided = MinimumServiceReceived.$
Comment: The minimum service provided in the previous round is the minimum size of the packet selected for transmission. However, if no packets were selected from the previous round, then the minimum Service that is positive among the queues is chosen.

5) **Go to step 1.**

Fig. 4. Steps involved in AMES scheduling algorithm.

$MinimumPacketInRound$. At the end of a pass, this variable indicates the minimum service provided by the scheduler in the current round ($ServiceProvided$), which is deducted in the following round from every queue.

Note that in the event that no queues were selected for transmission, then there may be some queues with positive service which may not be serviced at all. Such scenarios could occur when a small-sized packet arrives on a link and there are no other packets in the link following that. In such cases, a round of transmission would be given a minimum service equal to the size of this packet and would render a positive balance for the service received by other queues. In the successive round, no packets would be selected as the queue from which the short packet was received is empty and all other queues have positive values for service received. In order to avoid such a deadlock situation, the service provided in a round is updated with the minimum positive service received value from the list of backlogged queues in case no queues are eligible for transmission (as shown in Step 4 of Figure 4).

We illustrate the working of the AMES scheduling algorithm with an example. Consider the network in Figure 1. Consider three flows A–B, E–F, and G–F with packet sizes of 1000, 500, and 250 bytes, respectively. The snapshot of the network for this scenario is shown in Figure 5. Without loss

of generality, assume that the two flows arriving at router 5 has not received any service yet. Router 5 selects a round of packets from each of the queues for transmission. The service received by link 4–5 would be incremented by 1000, while that of 9–5 is incremented by 500. As every queue has been serviced a minimum of 500 bytes of service, this quantity is deducted in the next round. Link 4–5 would have a positive service received value of 500, hence will not be eligible for transmission. Hence, the successive pass of round-robin scheduling involves only a round of packet from link 9–5. The minimum service provided across all queues is 500 bytes which is again deducted, thus resulting in the service received values of both flows being set to zero again. This results in a transmission pattern on link 5–6 as shown in Figure 5.

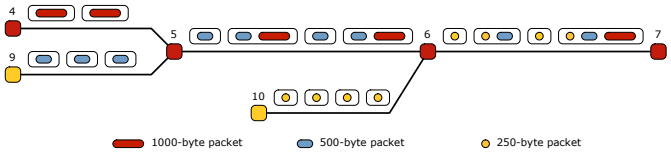


Fig. 5. Sharing of links by flows with different packet sizes.

Similarly, at router 6, packets from link 10–6 merge with those from link 5–6 and it is fairly easy to convince oneself of the pattern shown for link 6–7 in Figure 5. The pattern of four rounds that repeats itself has four packets from flow E–F and two packets from flow C–D for every packet of flow A–B, thus leading to a fair sharing of link bandwidth even with variable packet sizes. Observe that the working of this algorithm is similar to that of Deficit Round Robin (DRR) except that the routers do not employ flow-specific queuing. The transmission of packets in rounds may be viewed as if the service offered at a scheduler is normalized with respect to the minimum size of all packets selected in a pass.

D. Complexity of scheduling

The scheduling of packets at an output link mimics a round-robin scheduler over the set of queues (which equal the number of links). The number of packets to select from a queue for a round is determined by the change of round bit in the packet. Thus, when a packet from a queue has been transmitted, the next packet to be transmitted is either from the same queue (if the round has not changed), or the next queue. Thus, the complexity of the AMES scheduling algorithm is O(1).

E. Weighted fairness

Due to the nature of transmission of packets as rounds, support for weighted sharing of link bandwidth is easily achieved in this framework. Assume that every flow in the network has a weight that is indicated in the packet header. The service given to a flow in a round is then computed as the ratio of the packet size to the weight of the flow. Hence, the service received by a queue when a round of packets is selected is computed as the minimum value of this ratio among the selected round of packets. In the scheduling algorithm shown in Figure 4, step 3(b)iii is modified as:

$$ServiceReceived \leftarrow ServiceReceived + \min_{p} \left(\frac{L_p}{W_p} \right)$$

where, p refers to the packets selected in that round, L_p denotes the length of packet and W_p denotes the weight of each packet (or the flow to which the packet corresponds to). Consider the example with three flows, A–B, E–F, and G–H with packet lengths of 1000, 500, and 250 bytes, respectively, and weights of 4, 2, and 1, respectively. The snapshot of the network is shown in Figure 6. It is fairly easy to convince oneself that the scheduling at every router would involve selection of a round of packets from each of its queue[2].

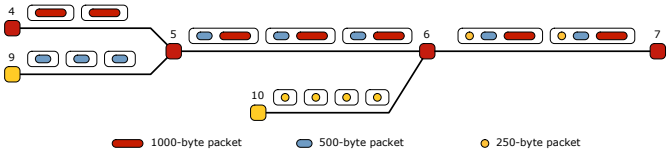


Fig. 6. Sharing of links by three flows with packet sizes of 1000, 500, and 250 bytes with weights of 4, 2, and 1, respectively.

F. Congestion control

We assume that the core-routers do not drop packets while the edge routers may, if necessary, as we will evaluate the performance of end hosts that may employ TCP/UDP which may not be forced to stop transmission by the edge routers. Packet dropping at the edge routers may be avoided if edge-routers can prevent end-hosts from transmission when necessary.

Every core-router has link-specific buffers at the output and if the buffer occupancy of any of the buffers corresponding to an input link i exceeds a threshold value, a control message is sent back to the router connected on link i that stops the transmission from that router. We employ a binary-valued feedback that indicates whether the previous router may or may not transmit. When the occupancy of all the buffers corresponding to an input link falls below the threshold, a control message is sent to the corresponding router on that link to resume transmission. These control messages are exchanged only among the adjacent routers and the congestion control techniques works independently from one router to another. The threshold for every buffer is set in a manner such that the buffer space over the threshold can accommodate all the packets that would be received from the previous router since the transmission of the control message. The excess buffer size is computed as $2\tau G$ bytes, where τ denotes the one-way propagation delay between the adjacent routers and G refers to the transmission speed (in bytes/sec) of the previous router.

G. Bandwidth measurement

One of the key features of the AMES framework is the ability to measure the available bandwidth on the path. In the example traffic considered in Figure 2 with four flows having

[2]Every flow receives equal amount of service after being normalized to their respective weights, hence every round has one packet from each flow.

fixed-size packets, it may be observed that the maximum number of packets in a round along the path indicates the number of flows with which a particular flow shares the bandwidth in the most congested link. In the case of weighted sharing, the objective is to compute the sum of weights of all the flows on the most congested link. Every router computes the average of ratio of the number of bytes transmitted in a round r, denoted by $BytesTransmitted_r$, to the service provided in that round, denoted by $ServiceProvided_r$, as:

$$\text{TotalWeight} = \frac{\sum_{r=1}^{R} BytesTransmitted_r}{\sum_{r=1}^{R} ServiceProvided_r} \quad (1)$$

The average value over a window of last R rounds of transmission is maintained at every router. Consider the example in Figure 5 where three flows with packet lengths of 1000, 500, and 250 bytes with equal weights share the link. The sum of the weight of flows computed at router 4 on link 4–5 would have a value of 1, similar to that for link 9–5. The transmission on link 5–6 has a pattern of two rounds with one round having 1500 bytes and the other with 500 bytes and the serviced provided by router 5 in each of the rounds is 500 bytes. Thus, the sum of the weights of the flow is then computed as 2, which also indicates the number of flows sharing the link. A similar computation at router 6 yields a value of 3. Thus, the value indicates the maximum number of flows that share a link along the path.

In the example considered in Figure 6, where three flows with packet lengths of 1000, 500, and 250 bytes and weights of 4, 2, and 1, respectively, the computation yields a value of 1 on links 4–5 and 9–5, 3 on link 5–6, and 7 on link 6–7. Thus, a flow with a weight of w traversing a path with a maximum TotalWeight observed on a link as W can estimate its fair bandwidth share as w/W of the link bandwidth.

End-to-end congestion control algorithms may be developed by computing the available bandwidth along the path, by measuring the maximum TotalWeight encountered along the path. IPv6 provides per-hop metrics that can be updated along every packet and may be employed to obtain the above metric in IP-based networks. If the links have different transmission rates, the above metric may be converted to transmission time required for every byte at each router. The end-hosts may then control its traffic based on their weight and the maximum value of the above metric computed.

III. Performance Evaluation

The performance of the AMES framework is evaluated using the Scalable Simulation Framework [39] with UDP and TCP traffic under a variety of traffic patterns. In addition, we also evaluate the framework using a new transport protocol that periodically probes the path for available bandwidth and adapts the rate accordingly. Each of these experiments are aimed at studying specific characteristics of the framework. The network and flows considered are shown in Figure 7. The flows in the network are divided into four groups, A–B, C–D, E–F, and G–H with hosts in groups A, C, E, and G acting as sources and those in groups B, D, F, and H acting as destinations. Each link has a bandwidth of 10 Mbps and a propagation delay of 1 ms. The core-routers have buffer thresholds of 15 KBytes per queue while the edge routers have a buffer of 30 KBytes per queue. The core-routers have additional buffer space so that they do not drop packets.

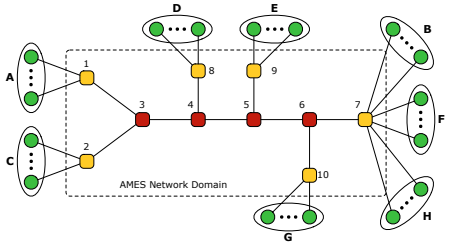


Fig. 7. Network and flows considered for performance evaluation.

We compare the performance of the AMES framework with that of CSFQ [10] as it is the closest to AMES in terms of realizing a stateless architecture. In the CSFQ architecture, every queue was assigned a buffer of 100 KBytes with a buffer threshold of 15 KBytes. However, unlike AMES, CSFQ will drop packets when the buffer occupancy exceeds the threshold (indication of congestion), based on the probabilities computed according to the flow rate estimates as described in [10].

All experiments were run for 50 seconds. All flows were started at time 0 for AMES. For CSFQ, the UDP flows in groups E–F and G–H were started at 0.1 s. We observed that when the UDP flows were started at time 0, the TCP flows that share a common link with these flows were significantly affected. For each experiment, we plot the average bandwidth received by a flow in each group.

Scenario 1 – Multiple congested links and flows with equal priorities: This scenario is aimed at demonstrating the fair sharing capabilities of the AMES framework where groups of flows join and leave at different points along the path. The traffic for this scenario is as shown in Table. I.

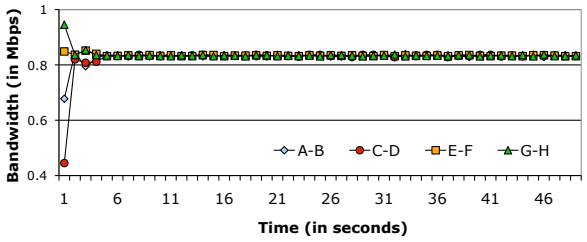
The first experiment under this scenario involves UDP traffic in group A–B that transmit at the fair rate of 0.833 Mbps while the other UDP flows in groups E–F and G–H transmit at 4 Mbps each. Figures 8(a) and (b) show the average bandwidth received by a flow in different groups under the AMES and CSFQ architectures, respectively, computed over one second intervals. It is observed that all flows share the congested link(s) in a fair manner and the UDP flows that transmit at their fair rates are not penalized by the ill-behaving flows. As CSFQ achieves the fair sharing using probabilistic dropping and AMES achieves using deterministic rounds, the former exhibits a higher variation in the bandwidth received by flow.

The second experiment under this scenario considers TCP traffic in group A–B while all other traffic types are left unchanged. Figures 9(a) and (b) show the average bandwidth obtained by a flow in a group with the AMES and CSFQ architectures, respectively. With the AMES framework, the flows converge to their respective fair share immediately, except for the initial few seconds required for stabilization. However, with CSFQ, we observe that the probabilistic dropping packets results in an unfair bandwidth assignment for TCP flows. In

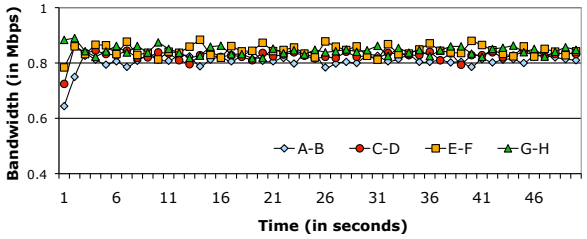
TABLE I
TRAFFIC FOR SCENARIO 1.

Flow group	Number of flows	Flow type	Packet length distribution	Mean packet length (in bytes)
$A-B$	4	UDP/TCP-Reno*	Fixed	1000
$C-D$	8	TCP-Reno	Fixed	2000
$E-F$	4	UDP	Fixed	500
$G-H$	4	UDP	Fixed	250

* This scenario has two experiments. The first experiment has all flows as UDP while the second has all flows as TCP-Reno.

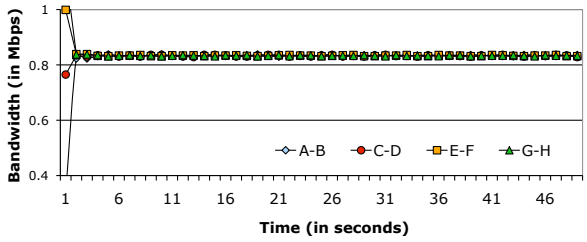


(a) AMES

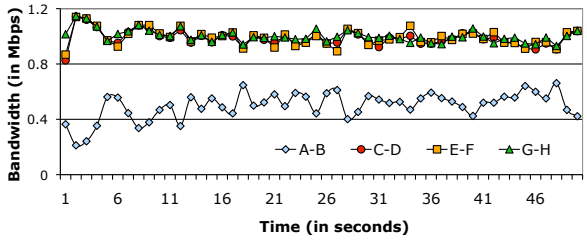


(b) CSFQ

Fig. 8. Average bandwidth received by a flow in different groups for scenario 1 with flows in group A–B as UDP.



(a) AMES



(b) CSFQ

Fig. 9. Average bandwidth received by a flow in different groups for scenario 1 with flows in group A–B as TCP.

CSFQ, congestion control is achieved by packet dropping, which forces the TCP streams to adapt their rates. However, packet dropping does not affect UDP flows as the UDP flows transmit at much higher rates compared to TCP. Even dropping a small number of packets from the TCP flows causes the rate adaptation techniques to take aggressive action. On the other hand, AMES achieves congestion control by not allowing a router to transmit. As the routers in AMES architecture do not drop packets, TCP flows do not enter into congestion avoidance state, thus do not reduce their rates. UDP flows do not get their packets dropped as well, however are throttled to fair share rate by forcing the routers to not transmit.

Scenario 2 – Multiple congested links and flows with different priorities: The experiment in this scenario demonstrates the weighted fair sharing properties of the AMES framework. The traffic for this scenario is shown in Table II. Figures 10(a) and (b) show the average bandwidth received by a flow in different groups with AMES and CSFQ, respectively. With AMES, the TCP flows in group A–B receive twice the share of bandwidth compared to the other TCP and UDP flows. With CSFQ, the TCP flows in group A–B suffer from packet dropping despite their higher weight. Notice that, CSFQ attempts to compute the average input rate from every incoming link, and appropriately adjust the probabilities of a packet being dropped. While this approach certainly reduces the packet drop probability for the TCP flows, the bandwidth received by TCP flows is still significantly reduced in the presence of UDP traffic.

In the second experiment, the traffic in groups E–F and G–H were changed to TCP flows and similar performance was observed with AMES (not shown here due to space constraints). Figure 11 shows the average bandwidth received by a flow in different groups with CSFQ. It is observed that changing the traffic in groups E–F and G–H to TCP results in a higher throughput for the TCP flows in group A–B, compared to when the traffic in E–F and G–H were UDP. However, the TCP flows in group A–B still do not obtain twice the bandwidth as the other flows.

Scenario 3 – Ability to increase throughput when available: The goal of the following experiments is to study the achievability of max-min fair sharing in a limited context. The traffic for this experiment is as shown in Table III. The traffic set-up is similar to that considered in scenario 1, except that the number of flows in group C–D is reduced to 4. As the four flows in group A–B are constrained to a bandwidth of 1/12 of link bandwidth due to sharing at link 6–7, flows in group C–D may acquire up to one-sixth of the link bandwidth each.

With AMES, the results of this experiment were similar

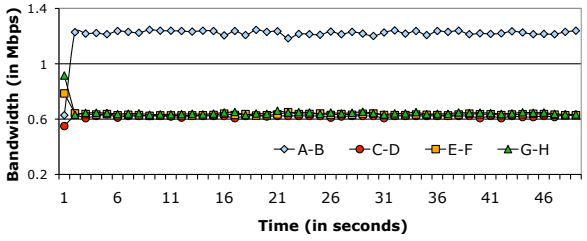
TABLE II
TRAFFIC FOR SCENARIO 2.

Flow group	Number of flows	Flow type	Packet length distribution	Mean packet length (in bytes)	Weight
$A-B$	4	TCP-Reno	Fixed	1000	2
$C-D$	8	TCP-Reno	Fixed	2000	1
$E-F$	4	UDP/TCP-Reno*	Fixed	500	1
$G-H$	4	UDP/TCP-Reno*	Fixed	250	1

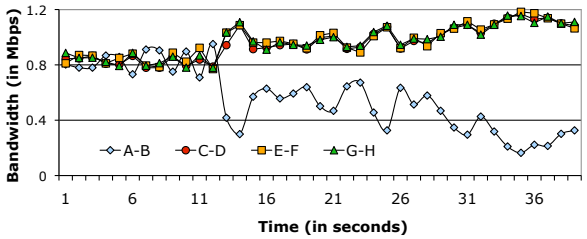
* This scenario has two experiments. The first experiment has all flows as UDP while the second has all flows as TCP-Reno.

TABLE III
TRAFFIC FOR SCENARIO 3.

Flow group	Number of flows	Flow type	Packet length distribution	Mean packet length (in bytes)	Weight
$A-B$	4	TCP-Reno	Fixed	1000	1
$C-D$	4	TCP-Reno	Fixed	2000	1
$E-F$	4	UDP	Fixed	500	1
$G-H$	4	UDP	Fixed	250	1



(a) AMES



(b) CSFQ

Fig. 10. Average bandwidth received by a flow in different groups for scenario 2a.

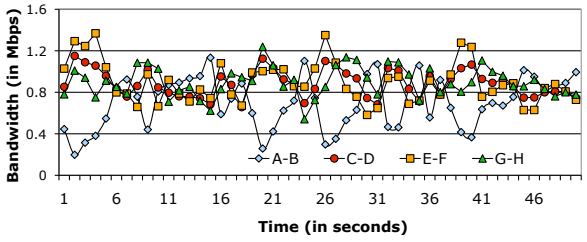


Fig. 11. Average bandwidth received by a flow in different groups for scenario 2.

to those obtained for scenario 1 as shown in Figure 8(a). The flows in group C–D received a bandwidth of only 1/12 th of the link bandwidth. The reason for such a behavior is the buffering of packets at router 3. Although TCP flows from group A–B start to transmit at their fair share, the packets are buffered at router 3. The buffer corresponding to links 1–3 and 2–3 have four packets in a round. Hence, whenever router 3 transmits, it transmits two rounds of packets from buffer of link 1–3 and one round of packets from buffer of link 2–3 as the flows from these groups have packet sizes of 1000 and 2000, respectively. Hence, as long as the buffer corresponding to link 1–3 clears, the flows in group A–B share the link equally with those in group C–D, thus receiving a bandwidth that is 1/8th of the bandwidth on link 3–4 and 4–5. However, the rate at which the flows from group A–B are served at router 6 is only 1/12 th of the link bandwidth and transmission at an input rate of 1/8 th the link bandwidth results in building up of buffer at router 6, then at router 5, and eventually at router 4. As a result, router 3 is restricted from transmitting on link 3–4, thus building up the buffer at router 3. Because of this reason, whenever a round of packets is scheduled from link 2–3, there is always a packet present in the buffer corresponding to link 1–3 as well, thus forcing them to share the bandwidth on link 3–4 equally. The brief transmission period on link 3–4 is not sufficient for the buffer corresponding to link 1–3 to be cleared up. As flows in group A–B can receive only a rate of 1/12 th of the link bandwidth, the flows in group C–D are also restricted to rate of 1/12 th of link bandwidth as they share the link 3–4 equally. This experiment demonstrates that a flow that is restricted downstream may affect other flows with which it shares the bandwidth upstream.

Figure 12 shows the average bandwidth received by a flow in different groups with CSFQ. Observe that the TCP flows of group C–D is able to achieve higher throughput as the TCP flows of group A–B are affected by the UDP flows on link 6–7. Although the fair share of a flow belonging to group A–B is 0.833 Mbps, only half the bandwidth is obtained with CSFQ.

Scenario 4 – Sharing with explicit rate adaptation. We evaluate the achievability of max-min fair allocation when a transport protocol that can take advantage of the AMES framework is employed. In this experiment, we assume that every end-host employs a token-bucket scheme, where the token rate is periodically determined by probing for available bandwidth along the path. The source may accumulate tokens, however the bucket is emptied periodically to avoid prolonged

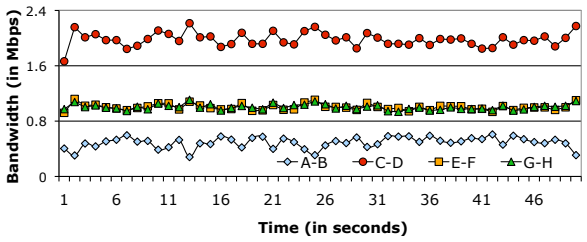


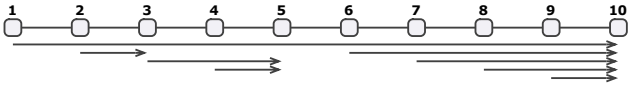
Fig. 12. Average bandwidth received by a flow in different groups for scenario 3 under CSFQ.

accumulation. Consider a sequence of ten routers as shown in Figure 13(a). The link transmission speed is assumed to be 1 Gbps with a propagation delay of 0.16 ms. The buffer size at each node is set at 200 KB while the threshold is 160 KB. Every node keeps a history of the last 10 transmission rounds on a link for bandwidth estimation. The available bandwidth along path is measured every 32 ms. The network is monitored in *intervals* of 64 ms. The traffic start times are shown in Figure 13(b). The max-min fair allocation rates for the flows at different intervals are shown in Table 13(c).

The bandwidth received by nodes 1, 2, 3, and 6 obtained using simulation are plotted against time intervals in Figure 14. By comparing the charts against Table 13(c), it is observed that the measured flow rates converge to the max-min fair rates. The convergence time for different flows depend on the traffic distribution. For example, the flow from node 2 must receive a fair share of 0.67 Gb/s from interval 31, however, its fair share is reduced to 0.33 Gb/s. This is because of the flow from node 1. Although node 1 reduces its rate from 0.5 Gb/s to 0.33 Gb/s during interval 31, some of its previously transmitted packets are buffered at node 2. The buffered packets have one packet in each round. Hence, for some time, node 2 will share its link bandwidth equally with node 1. As the fair rate allocation for flow at node 1 is 0.33 Gb/s, node 2 also receives only 0.33 Gb/s. However, once the backlog is cleared[3] at node 2, it receives its fair share of 0.67 Gb/s.

The simple rate-control protocol based on token-bucket scheme shows good promise to take advantage of the available bandwidth measurement offered by the AMES framework. We believe that employing a credit-based scheme, similar to the one developed for ATM networks [40], may exploit the bandwidth measurement even better. Unlike current rate adaptation mechanisms that increase and decrease their rates (such as in TCP and [40]), we may compute the number of outstanding packets at any given time in the network based on the round-trip time and the available bandwidth on the path. The latter is a metric that is unknown to a flow and is measurable using the AMES framework in a straight-forward manner.

[3]A backlog is cleared due the periodic emptying of token bucket at the source. Not allowing token accumulation has its own impact on the achieved fairness and so does prolonged accumulation.



(a) Traffic scenario

Traffic (s-d)	(1-10)	(2-3)	(3-5)	(4-5)	(6-10)	(7-10)	(8-10)	(9-10)
Start time (in intervals)	1	11	21	31	41	51	61	71

(b) Traffic starting times

Source	Time (in intervals)							
	1	11	21	31	41	51	61	71
1	1.00	0.50	0.50	0.33	0.33	0.33	0.25	0.20
2	0	0.50	0.50	0.67	0.67	0.67	0.75	0.80
3	0	0	0.50	0.33	0.33	0.33	0.375	0.40
6	0	0	0	0	0.67	0.33	0.25	0.20

(c) Max-min fair rates (in Gb/s)

Fig. 13. Example traffic scenario, their starting times, and max-min fair rates for illustrating achievable sharing with rate adaptation. An interval is 64 ms.

IV. Conclusion

This paper develops a framework for fair bandwidth sharing, called Access Mechanism for Efficient Sharing (AMES), that is aimed at a clean-slate design of future large-scale networks, where high bandwidth-delay product will be an inherent characteristic. The key characteristics of the framework are: (1) employs link-specific queuing; (2) requires small buffers; (2) employs round-robin scheduling algorithm; and (3) restricts adjacent routers from transmitting upon congestion, rather than drop packets. We studied the effectiveness of the framework in achieving fair sharing under different traffic patterns involving TCP and UDP. We observe that by employing a simple rate-based flow control scheme that can take advantage of the available bandwidth measurement along a path, the framework achieves close to max-min fair sharing. The performance of the framework is observed to be sensitive to the buffer threshold employed at the core-routers and the flow control mechanism employed at the end hosts.

Acknowledgment

We would like to thank Abishek Gopalan, PhD student at the University of Arizona, for his help in obtaining the performance results for the CSFQ architecture. We would like to thank the anonymous reviewers for their valuable input that helped us improve the quality of the paper.

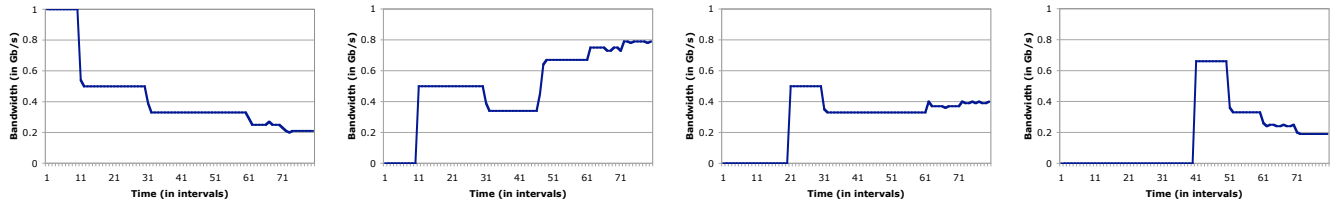
References

[1] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[2] A. Demers, S. Keshav, and S. Shenker., "Analysis and simulation of a fair queueing algorithm," *Journal of Internetworking Research and Experience*, pp. 3–12, October 1990.

[3] J. C. R. Bennett and H. Zhang, "WF^{2}Q: Worst-case fair weighted fair queuing," in *Proceedings of IEEE INFOCOM*, March 1996, pp. 120–128.

[4] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proceedings of IEEE INFOCOM*, April 1994, pp. 636–646.



(a) Node 1 (b) Node 2 (c) Node 3 (d) Node 6

Fig. 14. Bandwidth received by nodes 1, 2, 3, and 6 over time (measured in intervals of 64 ms).

[5] M. Shreedhar and George Varghese, "Efficient fair queueing using deficit round robin," in *SIGCOMM*, 1995, pp. 231–242.

[6] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: A novel drr implementation to achieve better latency and fairness at O(1) complexity," in *Tenth IEEE International Workshop on Quality of Service*, May 2002, pp. 77–86.

[7] S. Y. Cheung and C. S. Pencea, "Bsfq: Bin sort fair queueing," in *Proceedings of IEEE INFOCOM 2002*, June 2002, vol. 3, pp. 1640–1649.

[8] D. Stiliadis and A. Varma, "Rate-proportional servers: a design methodology for fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 164–174, 1998.

[9] I. Stoica, "Stateless core: A scalable approach for quality of service in the Internet," *PhD Dissertation, Carnegie Mellon University*, December 2000.

[10] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 33–46, February 2003.

[11] I. Stoica, H. Zhang, and S. Shenker, "Self-verifying csfq," in *Proceedings of IEEE INFOCOM 2002*, June 2002, vol. 1, pp. 21–30.

[12] M. Nabeshima, "Adaptive csfq: a new fair queuing mechanism for score networks," in *Proceedings of High Performance Switching and Routing – Workshop on Merging Optical and IP Technologies*, May 2002, pp. 37–41.

[13] Z. Mingyu, G. Guanqun, and Y. Yuan, "Fair bandwidth allocations through queue management in core-stateless networks," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, November 2001, vol. 4, pp. 2248–2252.

[14] M. J. Karam and Tobagi, "Rate and queue controlled random drop (rqrd): a buffer management scheme for internet routers," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, November–December 2000, vol. 1, pp. 316–322.

[15] T. Kurimoto and T. Shimizu, "Performance evaluation of new fair queuing algorithm based on csfq architecture," in *10th IEEE Workshop on Local and Metropolitan Area Networks*, November 1999, pp. 78–84.

[16] T. Nandagopal, V. Venkitaraman, R. Sivakumar, and V. Bharghavan, "Delay differentiation and adaptation in core stateless networks," in *Proceedings of IEEE INFOCOM*, March 2000, vol. 2, pp. 421–430.

[17] Z. Cao, Z. Wang, and E. Zegura, "Rainbow fair queueing: fair bandwidth sharing without per-flow state," in *Proceedings of IEEE INFOCOM*, March 2000, vol. 2, pp. 922–931.

[18] S. S. Kanhere and H. Sethu, "Fair, efficient and scalable scheduling without per-flow state," in *Proceedings of IEEE International Conference Performance, Computing, and Communications*, April 2001, pp. 181–187.

[19] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[20] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: Stabilized RED," in *Proceedings of INFOCOM*, 1999, vol. 3, pp. 1346–1355.

[21] M. Christiansen, K. Jaffay, D. Ott, and F. D. Smith, "Tuning RED for web traffic," in *SIGCOMM*, 2000, pp. 139–150.

[22] W. Feng, D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," in *Proceedings of INFOCOM 99*, 1999, vol. 3, pp. 1320–1328.

[23] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, October 1994.

[24] R. Jain, "A delay based approach for congestion avoidance in interconnected heterogeneous computer networks," *Computer Communications Review, ACM SIGCOMM*, pp. 56–71, 1989.

[25] S. Keshav, "A control-theoretic approach to flow control," *Proceedings of the conference on Communications architecture and protocols*, pp. 3–15, 1993.

[26] V. Paxson, "End-to-end Internet packet dynamics," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, 1999.

[27] R. Carter and M. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Performance Evaluation*, vol. 28, no. 8, pp. 297–318, October 1996.

[28] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput," 2002.

[29] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *INFOCOM*, April 2001, pp. 905–914.

[30] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *SIGCOMM*, 2000, pp. 283–294.

[31] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, vol. 18, 4, pp. 314–329, 1988.

[32] L. S. Brakmo, S. W. O'Malley, and L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, 1994, pp. 24–35.

[33] J. C. Hoe, "Improving the start-up behavior of a congestion control sheme for TCP," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, 26–30 1996, vol. 26,4, pp. 270–280, ACM Press.

[34] P. Mehra, A. Zakhor, and C. De Vleeschouwer, "Receiver-driven bandwidth sharing for tcp," in *Proceedings of IEEE INFOCOM*, March–April 2003, vol. 2, pp. 1145–1155.

[35] S. Deb, S. Shakkottai, and R. Srikant, "Stability and convergence of tcp-like congestion controllers in a many-flows regime," in *Proceedings of IEEE INFOCOM*, March–April 2003, vol. 2, pp. 884–894.

[36] P. Tinnakornsrisuphap and A. M. Makowski, "Limit behavior of ecn/red gateways under a large number of tcp flows," in *Proceedings of IEEE INFOCOM*, March–April 2003, vol. 2, pp. 873–883.

[37] Y. Gao and J. C. Hou, "A state feedback control approach to stabilizing queues for ecn-enabled tcp connections," in *Proceedings of IEEE INFOCOM*, March–April 2003, vol. 3, pp. 2301–3211.

[38] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore, "The ERICA switch algorithm for abr traffic management in atm networks," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 87–98, February 2000.

[39] Scalable Simulation Framework, "http://www.ssfnet.org."

[40] H. T. Kung and R. Morris, "Credit-based flow control for atm networks," *IEEE Network*, vol. 9, no. 2, pp. 40–48, March–April 1995.